MLPro Documentations

Release 1.0.0

Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya et al

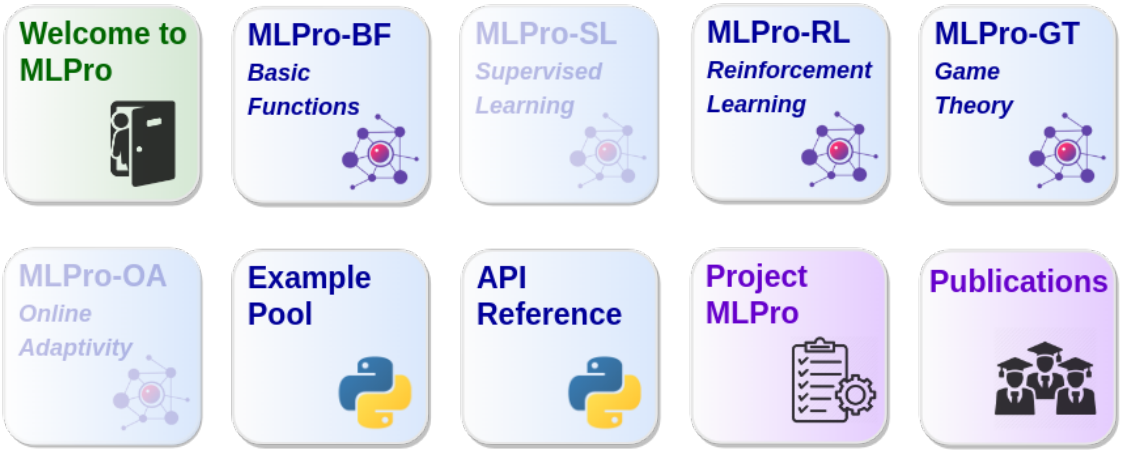
Mar 30, 2023

WELCOME TO MLPRO

Welcome to MLPro - the integrative middleware framework for standardized machine learning in Python!

MLPro

- enables hybrid ML applications in just one framework
- provides complete process landscapes and powerful ML templates on a scientific level
- integrates a growing number of *proven ML packages*
- enables comparable and reproducible results in publications
- is open source and even commercially usable (Apache License 2.0)



MLPro is also present on…

Python Package Index Anaconda.org GitHub

CHAPTER

ONE

INTRODUCTION

MLPro is a comprehensive and integrative middleware framework for standardized machine learning (ML) applications in Python. The objective is to provide processes and templates for a wide range of relevant ML sub-areas without having to forego the use of already established and proven ML frameworks such as Scikit-learn, TensorFlow, PyTorch, Optuna, etc. Rather, the latter is seamlessly integrated into the process landscapes of MLPro. By using MLPro, researchers, developers, engineers, and students can focus on their essential core tasks without having to worry about the integration/interaction of different frameworks or having to re-implement existing algorithms. MLPro is architecturally designed for extensibility and recombinability, which in particular enables the creation of hybrid ML applications across different learning paradigms.

1.1 Key Features

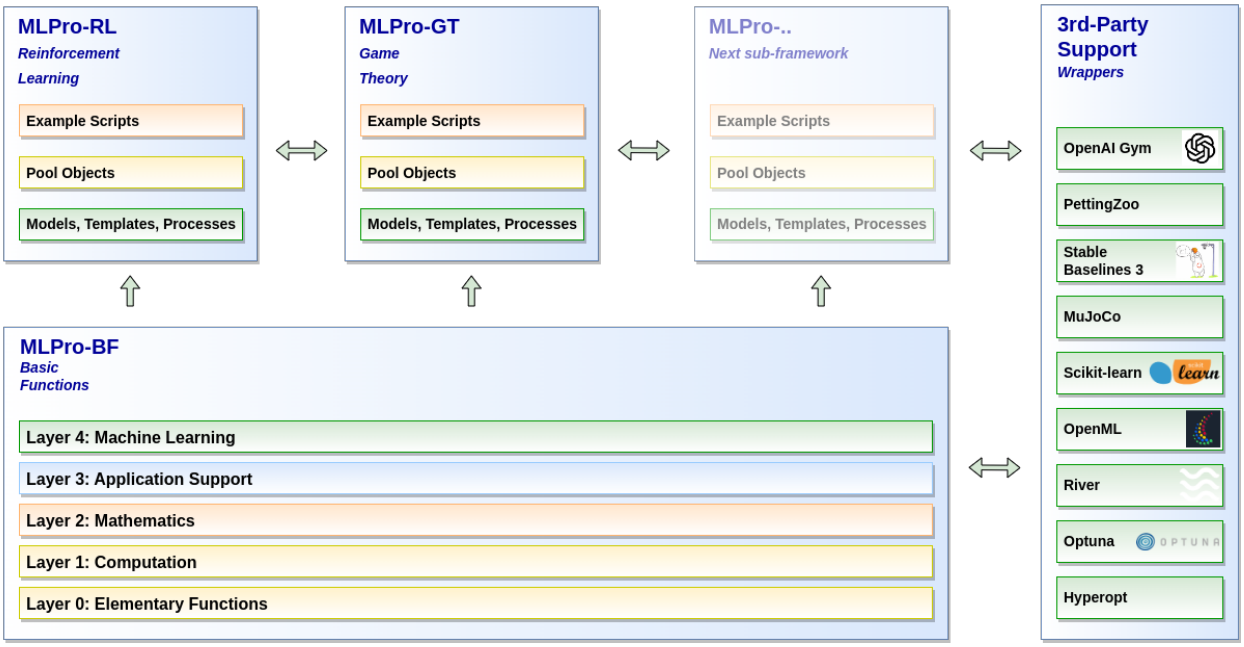
The most important key features of MLPro are…

1. **Sub-Frameworks for various ML topics**
   - *MLPro-RL for Reinforcement Learning*
   - *MLPro-GT for Game Theory*
   - *MLPro-SL for Supervised Learning*
   - *MLPro-OA for Online Adaptivity*
2. **Powerful substructure of overarching basic functions**
   - *MLPro-BF for Basic Functions*
3. **Extensive pool of examples**
   - *Example Pool*
4. **Integration of proven 3rd party packages**
   - *List of Wrappers*
5. **Open source, open design**

1.2 Architecture

MLPro consists of a continuously growing number of sub-frameworks covering different areas of machine learning. These include one or more fundamental process models (e.g. the Markovian Decision Process in reinforcement learning) and appropriate service and template classes. Furthermore, each sub-framework contains a specific pool of reusable classes for algorithms, data sources, training subjects, etc. Numerous sample programs for self-study complete the scope.

The sub-frameworks mentioned are in turn based on an overarching layer of basic functions. This is a common and obvious approach. What is special about MLPro, however, is the scope and internal structure of this base layer. A spectrum of elementary functions for logging and plotting through multitasking and numerics to the basics of machine learning is covered in a hierarchy of sub-layers that build on one another. This is also the key to the far-reaching recombinability of higher functions of MLPro. In fact, with each new feature, we consider how deeply we can incorporate it into MLPro. The deeper the level, the more universal the usability, and thus the range within MLPro.



1.2.1 Standardized Machine Learning

A special feature of MLPro is that machine learning standards are already defined in the basic functions. Templates for adaptive models and their hyperparameters as well as for executable ML scenarios are introduced in the top layer of MLPro-BF. Furthermore, standards for training and hyperparameter tuning are defined. These basic ML elements are reused and specifically extended in all higher sub-frameworks. On the one hand, this facilitates the creation of new sub-frameworks and, on the other hand, the recombination of higher functions from MLPro in your hybrid ML applications.

Learn more: *Basics of Machine Learning*

1.2.2 Example Pool

Numerous executable example programs (we call them "howtos") illustrate the essential functions of MLPro. They are also used for validation and are therefore an integral part of our automatic unit tests. With this, we ensure two things: the operability of all howtos and thus also the operability of the demonstrated functionalities (tdd - test-driven development).

Learn more: *Example Pool*

1.2.3 Third Party Support

MLPro integrates an increasing number of selected ML packages into its process landscapes. This is done at different levels of MLPro using so-called wrapper classes that are compatible with the corresponding MLPro classes.

Learn more: *Wrappers*

1.2.4 Real-World Applications in Focus

MLPro was designed not only for simulations but for use in real-world applications. To this end, various basic functions have been implemented that make diagnostics easier and make optimal use of the available system resources. These are for example

- Detailed logging
- Precise time management of simulated and real processes on a microsecond time scale
- Creation of detailed training data files (ASCII/CSV)
- Multithreading/multiprocessing

In addition, powerful templates for state-based systems are provided. They allow the standardized implementation of your systems, which can then be controlled, for example, by adaptive controllers based on reinforcement learning or game theory. Furthermore, a wrapper for the popular physics engine MuJoCo is provided, which can be used for the simulation and visualization of externally designed system models. The MLPro templates are also prepared for connection to industrial components like controllers, sensors, and actuators.

Learn more

- *Elementary Functions*
- *Computation*
- *State-based Systems*

1.3 Development

MLPro is developed at the South Westphalia University of Applied Sciences, Germany in the Department for Electrical Power Engineering in the Lab for Automation Technology and Learning Systems and is freely available to all interested users from research and development as industry and economy.

The development team consistently applies the following principles:

- **Quality first**
  We aim to provide ML functionalities at the highest possible level. We put these up for discussion in scientific *publications*. Open feedback and suggestions for improvement are always welcome.

- **Design first**
  In MLPro, new functions are not created in the code editor but in a class diagram. We provide the latter in the *API Reference*. A colour system documents the respective development status.
- **Clean Code Paradigm**
  We firmly believe that a clearly structured and legible source code has a significant influence on both the acceptance and the life cycle of software. Anyone who opens any source code of MLPro knows immediately what we mean :-)

1.3.1 Customer Extensions

Of course, frameworks like MLPro are made to reuse their functions in own applications. That's why we put a lot of effort into design and documentation to create powerful and understandable templates and related example programs. The following notes are intended to help software developers to interpret and use them correctly.

There are essentially three types of classes in the MLPro framework:

- **Property classes**
  These are classes that standardize certain properties and pass them on to child classes through inheritance. These classes are primarily found in the lower layers of MLPro and are not intended for direct use in your own applications. Nevertheless, they can of course be used in your own classes to maintain compatibility and integrity with MLPro. Examples can be found in *Basic Functions, Layer 0* among others.
- **Process classes**
  These classes provide higher level application functions such as training or running a model. They are primarily found in the higher sub-frameworks for machine learning. Customer extensions are not provided here.
- **Template classes**
  In order to be able to implement your own algorithms, models, data objects, systems, etc. in compliance with the MLPro standards, numerous template classes are provided on different levels. These in turn contain special **custom methods** that are intended for your own adjustments. These are explicitly identified in the *API Reference* both in the description of the classes and methods and in the associated class diagram. It should be noted here that custom methods are often inherited from parent classes (e.g. property classes). It is therefore recommended to follow the inheritance lines of template classes.

Learn more

- *Example Pool*
- *API Reference*

CHAPTER

TWO

GETTING STARTED

2.1 Installation from PyPI

MLPro is listed in the Python Package Index (PyPI) and can be installed using the package installer for Python (pip) in two variants:

- **Without any dependencies**

  The following command installs the latest version of MLPro.

  ```
  pip install mlpro
  ```

  Additional packages may need to be installed manually (depending on the functionalities you intend to use).

- **Full installation with all dependencies**

  There is also an option to automatically install MLPro and all depending packages in validated versions (see Subsection *Dependencies* below). This option will ensure that all the functionalities of MLPro, including wrappers and examples, work appropriately out of the box.

  ```
  pip install mlpro[full]
  ```

2.2 Installation from Anaconda

MLPro is also available on Anaconda and can be installed with the following command:

```
conda install -c mlpro mlpro
```

2.3 Dependencies

The table below shows all packages that MLPro has dependencies on. Additionally, the versions with which MLPro is compatible are listed. Since we cannot influence incompatible changes on dependent packages, we unfortunately cannot rule out the possibility of problems occurring with different versions. We review and update the list with each new release.

Which packages are actually required depends on the functionalities of MLPro that are used.

Package	Version
dill	0.3.6
numpy	1.23.5
torch	1.13.1
matplotlib	3.6.3
transformations	2022.9.26
stable-baselines	1.7.0
gym	0.21.0
scipy	1.8.1
petting-zoo	1.22.3
pygame	2.1.2
pymunk	6.4.0
multiprocess	0.70.14
river	0.14.0
scikit-learn	1.2.0
optuna	3.0.5
hyperopt	0.2.7
pyglet	1.5.27
mujoco	2.3.1.post1
lxml	4.9.2

2.4 First Steps

The easiest way to become familiar with the concepts and functions of MLPro is to browse through the numerous *example programs*. We can also recommend taking a closer look at the *key features* of MLPro and following the links.

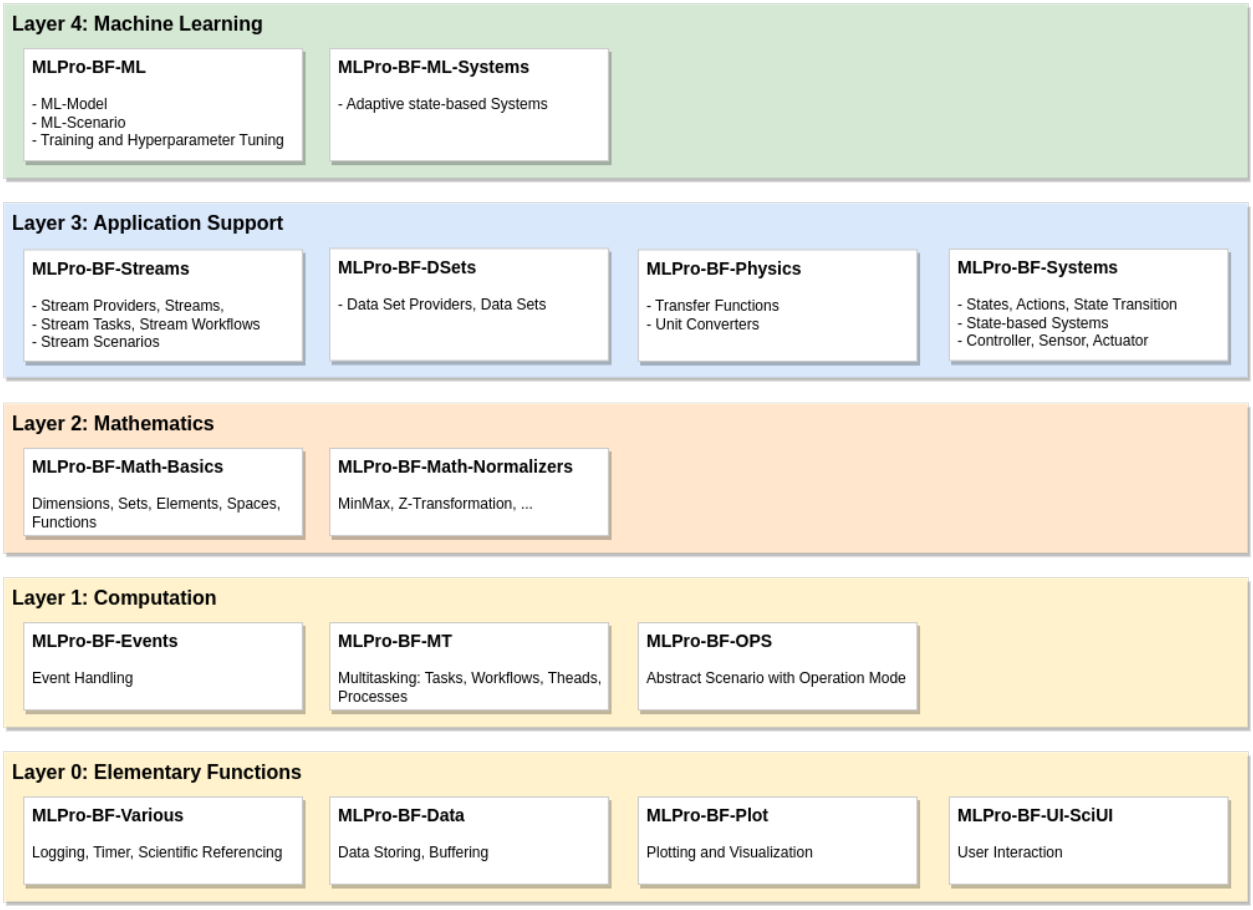
CHAPTER

THREE

MLPRO-BF - BASIC FUNCTIONS

3.1 Overview

MLPro has an extensive substructure of comprehensive basic functionalities, which are combined in the sub-framework MLPro-BF. This is organized in a total of five layers that build on one another, as shown in the following figure:



The lowest *Layer 0: Elementary Functions* provides a collection of functions for logging, time measurement in simulated or real processes, data management/plotting, etc. It also contains a framework for interactive GUI applications.

The next *Layer 1: Computation* consists of various functionalities related to computation topics, such as event handling and multitasking. Furthermore, it introduces an abstract runtime scenario with an operation mode (simulation or real operation). This is one of the key concepts in MLPro to support real applications. It is reused and specialized at higher levels.

On top of this, *Layer 2: Mathematics* introduces elementary mathematical objects like dimensions, sets and elements, metric spaces, and functions. Furthermore, numeric algorithms for data normalization etc. are included.

Layer 3: Application Support prepares the connection to real applications. It introduces powerful systematics for stream data processing/visualization and state-based systems that are, in turn, prepared for communication with real hardware components like sensors and actuators.

The top *Layer 4: Machine Learning* of MLPro-BF specifies fundamental standards for machine learning. All higher ML-related sub-frameworks reuse and specialize them. Topics like hyperparameters, adaptive models, and their training and tuning in ML scenarios are handled here.

3.2 Layer 0 - Elementary Functions

This lowest layer of MLPro-BF provides elementary functionalities for the following topics:

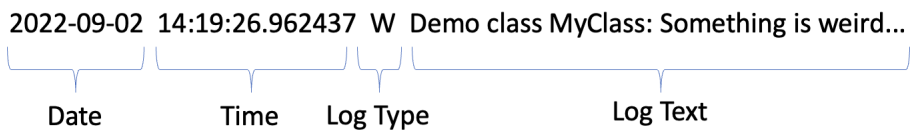
3.2.1 Logging

MLPro makes extensive use of a textual logging mechanism provided by the property class **Log**. This class centralizes and standardizes log outputs by adding appropriate methods to child classes by inheritance. It can be accessed as follows:

```
from mlpro.bf.various import Log
```

The log structure is divided into:

- Date : Current date in format year-month-day
- Time : Current time in format hour:minute:second.microsecond
- Log Type : The type of logging
- Log Text : The text that is being logged to the console



An example of logging functionality is shown in the figure below.

```
2022-09-02  14:19:26.962434  I  Demo class MyClass: Let me tell you what's going on...
2022-09-02  14:19:26.962437  W  Demo class MyClass: Something is weird...
2022-09-02  14:19:26.962500  E  Demo class MyClass: And here something failed...
2022-09-02  14:19:26.962505  I  Demo class MyClass: But don't worry. Everything is fine. It's just a demo:)
2022-09-02  14:19:26.962508  S  Demo class MyClass: This method terminated successfully!
```

In the figure above, it is shown that each log type has its own text color. Each log type is assigned to a color.

Here is the list of the color:

- Information (I) : White
- Warning (W) : Yellow
- Error (E) : Red
- Success (S) : Green

In the initialization, the logging level of object needs to be defined. By default, it will log all the informations; Information, Warning, Error, and Success. The user can specify which information that needs to be logged. The user can also change the logging level after the initialization with `switch_logging(p_logging)`.

The following are the identifier for `p_logging`:

- Log.C_LOG_ALL : Information, Warning, Error, Success
- Log.C_LOG_WE : Warning, Error
- Log.C_LOG_E : Error
- Log.C_LOG_NOTHING : Nothing

To log an information, function `log(p_type, *pargs)` needs to be called. The `p_type` is the type of logging. In the `*pargs`, the user can put the information in tuple.

Here is the list for `p_type`:

- Log.C_LOG_TYPE_I : Information
- Log.C_LOG_TYPE_W : Warning
- Log.C_LOG_TYPE_E : Error
- Log.C_LOG_TYPE_S : Success

```
from mlpro.bf.various import Log

class MyRandomClass(Log):

    def __init__(self, p_logging=True):
        # The constructor of class Log initializes the internal logging
        super().__init__(p_logging=p_logging)

    def random_method(self):
        self.log(self.C_LOG_TYPE_I, "Hi, I am information!")
        self.log(self.C_LOG_TYPE_W, "Hi, I am warning!")
        self.log(self.C_LOG_TYPE_E, "Hi, I am error!")
        self.log(self.C_LOG_TYPE_S, "Hi, I am success!")


if __name__ == "__main__":
    # Initilaization
    my_random_class = MyRandomClass(p_logging=Log.C_LOG_ALL)

    # Switch the logging level to Log.C_LOG_WE
    my_random_class.switch_logging(Log.C_LOG_WE)

    # Switch the logging level to Log.C_LOG_E
    my_random_class.switch_logging(Log.C_LOG_E)

    # Switch the logging level to Log.C_LOG_NOTHING
    my_random_class.switch_logging(Log.C_LOG_NOTHING)
```

Cross Reference

- *Howto BF-001: Logging*
- *API Reference*

3.2.2 Time Measurement

MLPro provides an internal timing mechanism that is introduced by class property **Timer**. This class uses the built-in python package, namely **datetime**, to deal with the time management system. This class also has a simple lap management, in which each time the maximum number of laps `C_LAP_LIMIT` is reached, then the lap counter restarts to 0. Timer class can be accessed as follows:

```
from mlpro.bf.various import Timer
```

The time measurement can cover two different modes, such as:

- **C_MODE_REAL** : real-time mode
- **C_MODE_VIRTUAL** : virtual time mode

The following are the functionalities of the timer:

- `reset` : to reset timer
- `get_time` : to get the actual time
- `get_lap_time` : to get the actual lap time
- `get_lap_id` : to get an id of an actual lap
- `add_time` : to add actual time, which is specifically for virtual mode
- `finish_lap` : to end the current lap

Cross Reference

- *Howto BF-002: Timer*
- *API Reference*

3.2.3 Data Management

Data management in a framework is extremely important, which mostly refers to the organization, storage, and retrieval of data within the framework. In MLPro, our team also provides such functionalities as saving data, loading data, storing data, creating a buffer, and plotting data. This involves defining a data model that describes the structure and relationships between data elements, implementing mechanisms for storing and retrieving data, and managing data consistency and integrity. A well-designed data management system is essential for the efficient and effective processing of data within the framework.

On this page, the data management within the MLPro framework is explained. However, an explanation of the plotting functionality is provided in the *next subchapter* of this documentation. The related data management classes can be accessed as follows:

```
from mlpro.bf.data import *
```

In general, there are three main functionalities of data management in MLPro:

1) **Saving and Loading Data**
   In MLPro, the users allow saving any classes using dill in the format of binary data to a specific path with a defined file name (or optionally keep the file name empty, then the name is auto-generated by MLPro). Consequently, the users also allow the loading of the saved classes from the binary data into MLPro's compatible classes. A class can be added by those functionalities by inheriting **Saveable** and **Loadable** classes from **mlpro.bf.various**.

2) **Data Storing**

The second possibility is to store a bunch of data in MLPro's **DataStoring** class with three different layers, as follows:

- **Layer 1 - Data Names** : the labels or the feature names of the data.
- **Layer 2 - Frames** : the frames can be added to each label or feature name. If none, then the frame id can be set to '0' all the time.
- **Layer 3 - Values** : the values can be added to each frame with a specific label or feature name.

Therefore, to add value to the data storage, the users can use `DataStoring.memorize(p_variable, p_frame_id, p_value)`, in which `p_variable, p_frame_id, p_value` refer to the feature name, the frame id, and the added value respectively.

For better understanding : *Howto BF-003: Store and plot data*

3) **Buffering**

The other data handling functionality is buffering by MLPro's **Buffer** class. The buffer is an important component in machine learning and online learning areas, where a number of data have to be stored, updated frequently, and also used for data sampling purposes. The following shows what the users can do with the buffer and how to define the buffer:

- Redefining `Buffer._gen_sample_ind(self, p_num: int)` to set up a method of sampling data from the buffer or optionally simply using **BufferRnd** class with random sampling functionality.
- Instantiating a buffer with a defined maximum buffer size.
- Storing the target values with specific feature names using **BufferElement**.
- Adding the buffer element to the buffer.
- Optional functionalities:
  - Checking whether the buffer is full.
  - Obtaining all data from the buffer.
  - Sampling from the buffer.
  - Clearing the buffer.

For better understanding : *Howto BF-004: Buffers*

Cross Reference

- *Howto BF-003: Store and plot data*
- *Howto BF-004: Buffers*
- *API Reference*

3.2.4 Plotting and Visualization

In order to be able to visualize processes on different levels in a standardized way, MLPro provides a property class **Plottable**. This inherits to higher classes (custom) methods for initializing and updating plot output during execution. There are three different views:

- 2-dimensional plot output
- 3-dimensional plot output
- n-dimensional plot output

Additional parameters can be set using the **PlotSettings** class.

MLPro and in particular the Plottable class is intended for plot outputs with the standard package Matplotlib in connection with the output backend TkAgg. In this combination, a good user experience is made possible. In principle, however, other packages can also be used for visualization.

Cross Reference

- *API Reference BF-PLOT - Plotting and Visualization*
- *Stream Plotting*

3.2.5 Scientific Reference

MLPro integrates scientific referencing in any class using a class **ScientificObject**. This class provides elementary functionality for storing scientific references. For example, when the users create a custom reinforcement learning policy or a custom environment, then the users can simply inherit ScientificObject class and add a scientific reference to the related elements. This class can be accessed as follows:

```
from mlpro.bf.various import ScientificObject
```

MLPro provides various forms of scientific references, which are:

- C_SCIREF_TYPE_NONE : None
- C_SCIREF_TYPE_ARTICLE : Journal Article
- C_SCIREF_TYPE_BOOK : Book
- C_SCIREF_TYPE_ONLINE : Online
- C_SCIREF_TYPE_PROCEEDINGS : Proceedings
- C_SCIREF_TYPE_TECHREPORT : Technical Report
- C_SCIREF_TYPE_UNPUBLISHED : Unpublished

After selecting the type of reference, the users can add more details, such as authors, titles, volume, DOI, and many more.

The type and detail of the related scientific reference in a class can be initialized, as follows:

```
from mlpro.bf.various import ScientificObject

class MyClass(ScientificObject):

    def __init__(self):
        self.C_SCIREF_TYPE = self.C_SCIREF_TYPE_ARTICLE
        self.C_SCIREF_AUTHOR  = "Max Mustermann"
        self.C_SCIREF_TITLE   = "Analysis of MLPro"
        self.C_SCIREF_JOURNAL = "My Journal"
        self.C_SCIREF_YEAR    = "2023"
        self.C_SCIREF_MONTH   = "01"
        self.C_SCIREF_DAY     = "01"
        self.C_SCIREF_VOLUME  = "01"
        self.C_SCIREF_DOI     = "10.XXXX"
```

Shortly, the MLPro team is planning to add a citing functionality. Therefore, the users can obtain the citation of the specific class in the form of BibTeX.

Cross Reference

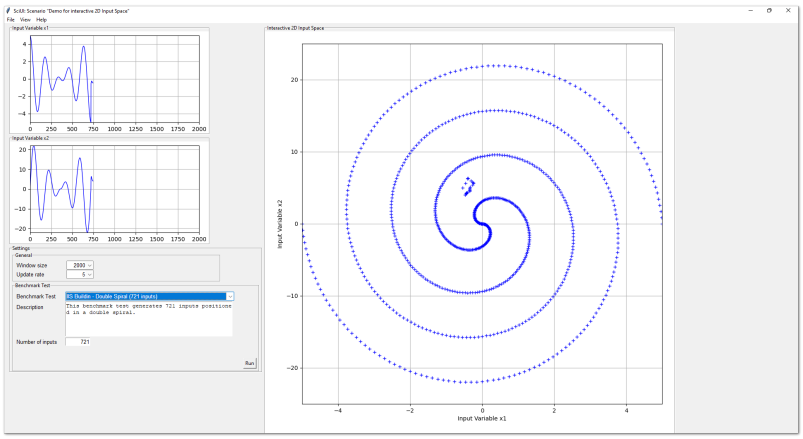
- *API Reference*

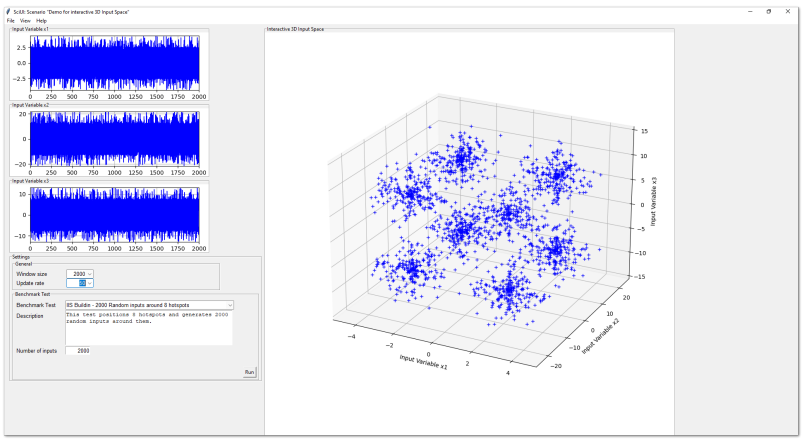
3.2.6 User Interaction

MLPro-BF provides a framework called SciUI (Scientific User Interface) for creating and running interactive ML applications for graphical validation, presentation and education purposes.

Key features are:

- Platform-indepe ndend framework for creation of own UI scenarios
- Ready-to-run application „SciUI“ auto-detects and starts own scenarios
- Focus on real-time visualization and interaction
- Based on Python standards Tkinter and MatPlotLib





Cross Reference

- *Howto BF-UI-001: SciUI - Reuse of interactive 2D/3D Input Space*
- Howto BF-UI-002: SciUI - Reinforcement Learning Cockpit

3.3 Layer 1 - Computation

This layer introduces various techniques for the efficient execution of higher functionalities of MLPro. In particular, it is dedicated to these topics:

3.3.1 Event Handling

Event handling is a widely used standard technique in software development. And that's how it found its way into MLPro. The mechanism is inherited by higher classes in the form of the property class **EventHandler**. This class allows event handler methods to be registered and events to be triggered, which in turn call registered handlers. An object of type **Event** is passed to each handler. This contains further information about the context of the event.

The event handling functionality is used extensively in MLPro. A large number of higher classes use this mechanism. Based on this, even an event-oriented adaptation mechanism is cultivated in *Layer 4 - Machine Learning*.

Cross Reference

- *Howto BF-EH-001: Event Handling*
- *API Reference BF-EVENTS - Event Handling*

3.3.2 Multitasking

In MLPro, the two essential techniques **Multithreading** and **Multiprocessing** for the asynchronous execution of program parts are summarized under multitasking. Both techniques allow the use of several or even all cores of the CPU(s), which can significantly increase the execution speed of the respective program. The basic prerequisite for this is, of course, that significant parts of the program can actually be executed in parallel.

While with multithreading program parts are executed in parallel within the same process, with multiprocessing they are started in separate processes. The latter uses computing resources more effectively but requires more administrative effort on the part of the operating system. Therefore, it is hard to say in advance which of the two techniques is the better choice for a specific implementation. What can be said, however, is that multithreading is the more straightforward technique to use with good acceleration of parallel programs. Multiprocessing requires a little more detailed knowledge of the internal mechanisms of the runtime environment. It unfolds its strengths, in particular with long-running parallel program parts, since the administrative overhead is not so significant here. The pros and cons of multithreading and multiprocessing can be excellently discussed. For this purpose, reference is made to relevant sources on the Internet.

The details of the multitasking implemented in MLPro are explained in more detail below:

Range

In the context of multitasking, the range describes the degree of parallelism of a program function. There are three different degrees:

- Synchronous/serial
- Asynchronous/parallel using multithreading
- Asynchronous/parallel using multiprocessing

Asynchronous execution of methods

In order to enable the asynchronous execution of methods of a class, MLPro provides the property class **Async**. In particular, this includes the method **_start_async()**, which allows the execution of another method in a separate thread or process.

Tasks und Workflows

A fundamental and consistently used concept in MLPro is that of tasks and workflows. A task is a class that can be executed in one of the three possible ranges mentioned above. Tasks can in turn be grouped into workflows. Any **directed graphs** of tasks can be set up and processed massively parallel via corresponding predecessor relationships. Successor tasks are informed about the termination of their predecessors via event technology. The **Task** and **Workflow** classes of the same name have once again been implemented as property classes. They are consistently reused in higher functions through inheritance. Some examples are *Stream Processing* and the *Adaptive Workflows*.

Gap under MacOS

At the time of the creation of MLPro, a technical problem related to multiprocessing occurred on Apple computers under MacOS. This is documented at Race condition when using multiprocessing BaseManager and Pool in Python3. It is recommended to check MLPro functionalities in multiprocessing mode on MacOS-based computers very carefully and, if in doubt, to use multithreading.

Cross Reference

- *Howto BF-MT-001: Multitasking - Parallel Algorithms*
- *Howto BF-MT-002: Multitasking - Tasks and Workflows*
- *API Reference BF-MT - Multitasking*

3.3.3 Operations

In this module, classes are made available that are required for the operative execution of higher functions of MLPro. In particular, the **ScenarioBase** class is introduced here, which serves as an abstract template for various scenarios, such as *Stream Scenarios* or *ML Scenarios*. In this respect, the scenario in MLPro is one of the fundamental concepts, which already introduces the following properties at this low level:

- Runtime mode (simulation or real operation)
- Execution of cycles
- Internal Timer-Management
- Persistence

All scenario classes in MLPro are ultimately template classes for implementing your own concrete applications. Therefore, special attention should be paid to the custom methods that are already introduced here.

Cross Reference

- *API Reference BF-OPS - Operations*

3.4 Layer 2 - Mathematics

Mathematics is an integral part of many areas, particularly in the fields of data analysis, machine learning, and system simulation. MLPro-BF-MATH provides the basic functions of mathematics, which become the foundation for various computational tasks. The basic functions of mathematics simplify the users of defining, conserving, and processing numeric data at a low level.

First, four main components are introduced in MLPro-BF-MATH to define a bunch of numeric data, such as:

a) **Dimension**

An object specifies properties of a dimension, including the name, the unit, the data type, the boundaries, and many more. Dimension in MLPro is not limited to real, natural, and integer numbers, but can also handle a big data object (e.g. images, point cloud, etc.).

b) **Set**

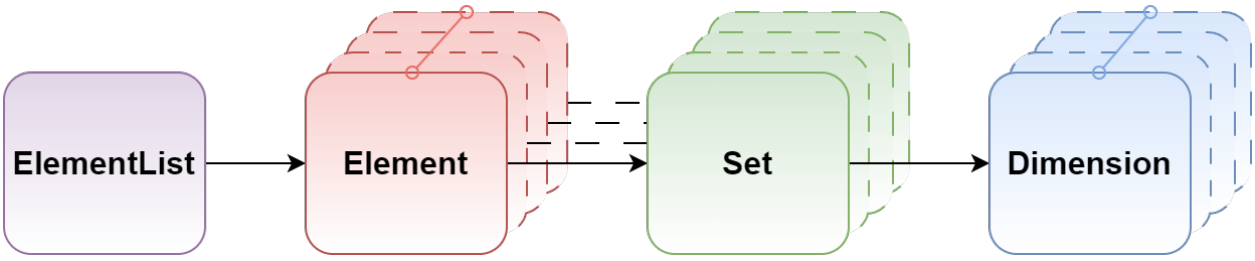
Set is a collection of dimensions that can also be described as a multivariate set in a mathematical sense. There are interesting functionalities, for instance, adding a new dimension to the set, getting information on the actual dimension in the set, and spawning a new object with a subset of dimensions.

c) **Element**

Element of a multivariate set. Each element represents a set, where the values of each component (Dimension object) can now be set.

d) **ElementList**

ElementList consists of a list of Element objects.



Second, MLPro also provides two objects to measure the distance between two elements, such as **MSpace** and **ESpace**. **MSpace** represents a metric space. The distance calculation is based on the metric of the space. **ESpace** represents a Euclidean space. The distance calculation is based on the Euclidean norm.

Third, MLPro has a **Function** class with mapping functionality. This class is intended for an elementary bi-multivariate mathematical function that maps elements of a multivariate input space to elements of a multivariate output space. With this class, it is possible to map a multivariate abscissa/input element to a multivariate ordinate/output element.

Lastly, there are more advanced functions of mathematics, as follows:

3.4.1 Normalizer

Normalization is a process of scaling different parameters to a common scale. The way the parameters are scaled depends on the type of normalization being performed. For e.g. parameters are scaled within a range of -1 to 1 in case of a minmax normalization. MLPro's normalizer classes can be used to normalize data based on MinMax Normalization and Z-transformation. These normalizer classes can be imported by incorporating following lines in your script.

```
from mlpro.bf.math.normalizers import NormalizerMinMax
from mlpro.bf.math.normalizers import NormalizerZTransform
```

Both normalizers store the parameters required for normalization based on the data provided for normalization. MLPro also provides the possibility to set/update the parameters when required, based on data instances or direct parameters for e.g boundaries for MinMax normalizers.

Both the normalizers provide following operations:

- Normalize : Normalize a given data element based on the set parameters.
- Denormalize : Denormalize a given data element based on the set parameters.
- Update Parameters : Upadte the normalization parameters based on data characteristics such as boundaries or statistical properties.
- Renormalize : MLPro's normalizers also provide the possibility to renormalize the previously normalized data elements on new normalization parameters.

Cross Reference

- *Howto BF-MATH-010: Normalizers*
- *API Reference*

Cross Reference

- *Howto BF-MATH-001: Dimensions, Spaces and Elements*
- *API Reference*

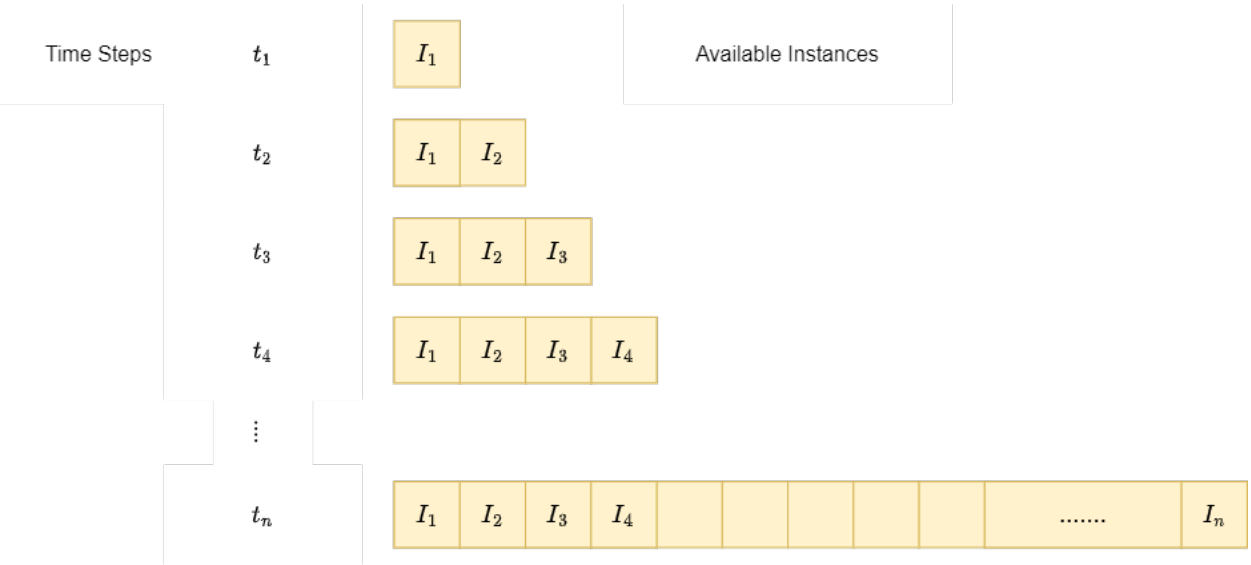
3.5 Layer 3 - Application Support

This layer deals with the provision of basic technologies to support real-world applications.

3.5.1 Stream Processing

Streams

A data stream is a live data source that delivers instances sequentially. Unlike offline datasets, data instances cannot be scanned on demand in case of streams. Data instances are only available at the order they arrive. For example, think of a live RADIO signal that delivers new data with time, where complete access to entire data is not possible.



As shown in the figure above, at every timestep, new information is available. However, the number of instances delivered at each instance and availability of historical instances depends on the type of stream and the processing task respectively.

In industrial scenarios, with more and more complex systems, the amount of live data delivered by the systems increases rapidly. This high amount of live data can be leveraged to take optimal decisions for processes. This real-time data is a data stream because of its live nature and processing such real-time data is a relevant field of data-mining and machine learning known as Stream Processing and Online Machine Learning respectively.

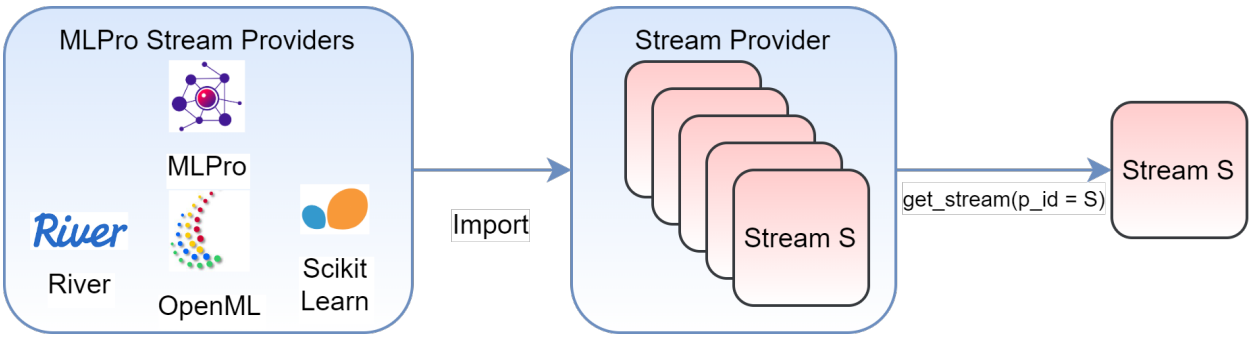
After loading the stream provider (MLPro's native stream provider for example), the list of available streams can be loaded as following:

```
# Import the stream provider class
from mlpro.bf.streams.native import NativeStreamProvider
# Create an object of the stream provider
mlpro = NativeStreamProvider()
# Get a list of streams
mlpro.get_stream_list()
```

Learn more

Streams Handling

MLPro's stream module provides a stream handling and bundle of stream processing functionalities. The stream handling architecture in MLPro is as shown in the following figure:



The figure shows a collection of stream provider apis, which in turn, contain a list of corresponding stream objects. Currently, MLPro supports stream provider api for MLPro's native streams and three external data providers:

- OpenML
- River
- Scikit-Learn

Stream Provider

Access to real-time data stream is not always possible for the purpose of testing and evaluations. MLPro's streams module provides Stream Provider functionality. A stream provider in MLPro is a data resource that provides stream objects for various operations.

MLPro's streams module provides native stream providers, that generate stream objects with user-defined parameters such as number of features and labels and pre-defined statistical properties such as feature boundaries. Currently MLPro's native stream provider supports random streams with random feature and label values. Along with native stream provider MLPro also supports data resources from popular external data resources including OpenML, ScikitLearn and River. MLPro's stream provider object accesses datasets from these resources and provide them as stream objects that imitate the sequential behaviour.

A stream provider in MLPro can be imported by including:

```
# import mlpro native stream provider
from mlpro.bf.streams.native import NativeStreamProvider
# import openml stream provider
from mlpro.wrappers.openml import WrOpenMLStreamProvider
# import river stream provider
from mlpro.wrappers.river import WrRiverStreamProvider
# import scikit learn stream provider
from mlpro.wrappers.sklearn import WrSKLearnStreamProvider
```

After loading the stream provider (MLPro's native stream provider for example), the list of available streams can be loaded as following:

```
# Import the stream provider class
from mlpro.bf.streams.native import NativeStreamProvider
# Create an object of the stream provider
mlpro = NativeStreamProvider()
# Get a list of streams
mlpro.get_stream_list()
```

Stream

In MLPro, a stream is a special iterator object that delivers new data instances with each iteration. A stream cannot be read directly for all the instances, instead an instance is only available when requested by a workflow. An instance in MLPro consists of feature and label data for that specific instance.

From a stream provider a specific stream of interest can be accessed with a stream id:

```
mystreamobject = mlpro.get_stream(p_id = '1')
```

After accessing the stream from the stream provider, a new instance can be accessed from the data stream by iterating over it.

Stream Instance

An instance in MLPro is a data element available at each time step, when processing a stream. An instance consists of a unique id, feature data and label data.

```
# Accessing an instance from stream
instance = next(iter(mystreamobject))

# Accessing the stream ID
id = instance.get_id()

# Accessing feature data
feature_element = instance.get_feature_data()
feature_data = feature_element.get_values()

# Accessing label data
label_element = instance.get_label_data()
label_data = label_element.get_values()
```

Note:

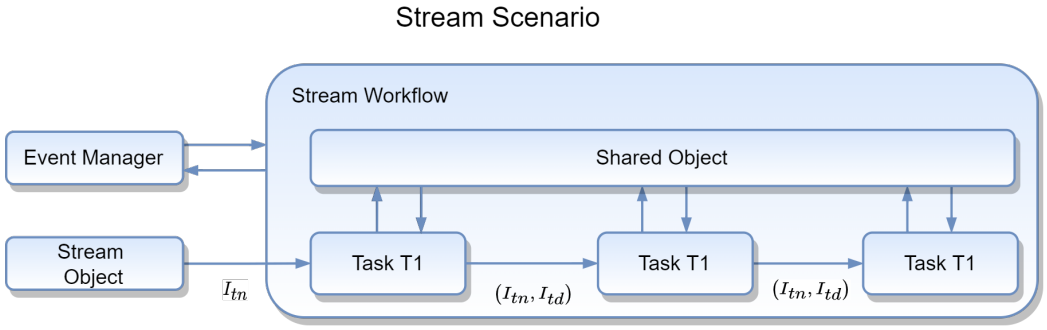
- The ids of the stream instances are managed internally by a Stream Workflow, and are also used for stream plotting functionalities. Changing instance ids might affect the performance of stream functionalities of MLPro.

Cross Reference

- *Howto BF-STREAMS-101: Basics of Streams*
- *API Reference: Streams*

Stream Processing

Handling streaming data sources and mining knowledge from them requires special types of processing tasks because of their live behaviour. Stream operations process new instances as they are available at every step. Along with a number of external and internal stream resources, MLPro's stream module provides processing functionalities like sliding window, rearranger, etc. specialized for streaming data.



In MLPro, streaming data is processed with a task and workflow architecture. A StreamTask is single operation performed on new stream instances and a StreamWorkflow is a list of tasks arranged sequentially with defined dependencies. **StreamTask** and **StreamWorkflow** are specialized classes inherited from MLPro's multiprocessing module. As

shown in the above figure the scenario fetches new I_{tn} instances from the stream object and each task then processes a list of new instances I_{tn} and deleted/obsolete instances I_{td} as shown in the figure. The processed instances are stored in the shared object for further accessibility.

Learn more

Stream Task

A StreamTask is a special stream processing task that takes a new instance as an input and delivers the processed instances as an output. A StreamTask also processes the obsolete/deleted instances from the workflow for following tasks.

StreamTask class in MLPro also provide provide plotting functionalities in 2D, 3D and nD, that plot the streaming instances by default. (A link to know more). Inherit from this class and implement the `_run(p_inst_new, p_inst_del)` method to implement custom stream tasks with inbuilt default plotting functionalities. This can be imported and used by including following:

```
#import stream models
from mlpro.bf.streams.models import StreamTask

#create a stream object
myStreamTask = Task(p_name = 'Task 1',
                    p_visualize = True,
                    p_logging = Log.C_LOG_ALL)
```

Currently MLPro provides following stream task implementations:

1. *Window*
2. *Rearranger*
3. *Deriver*

More StreamTask implementations will be available with future updates.

Stream Workflow

A StreamWorkflow in MLPro is a list of StreamTasks arranged hierarchically with user-defined dependencies on prior tasks in the workflow. A stream workflow receives new instance of the stream from the surrounding StreamScenario object at every step.

Note: A stream workflow carries a list of new instances and deleted/obsolete instances at every run. Both new and deleted instances are forwarded to subsequent tasks to be processed.

A stream workflow takes care of following functionalities:

1. Executing the tasks inside the workflow
2. Storing task specific results in the StreamShared Object
3. Fetching and delivering new and deleted instances among different tasks as per the defined dependency

StreamWorkflow can be imported and used as following:

```
#import stream models
from mlpro.bf.streams.models import *

#create a stream workflow object
myStreamWorkflow = StreamWorkflow( p_name='My Workflow',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=True,
                                   p_logging=Log.C_LOG_ALL))
```

A stream workflow consists a list of tasks within in a defined order and instance dependency. The instances processed by a task are forwarded to it's following task. The code block below shows how to add a task to an existing stream workflow:

```
# add task myStreamTask to the workflow myStreamWorkflow
myStreamWorkflow.add_task(p_task = 'Task 1')

#create another task
myStreamTask2 = StreamTask(p_name = 'Task 1',
                           p_visualize = True,
                           p_logging = Log.C_LOG_ALL)

# add the task to the workflow with task 1 as its predecessor
myStreamWorkflow.add_task(p_task = 'Task 2', p_predecessor = 'Task 1')
```

Each workflow has a shared object that stores instances and results of the stream task that can be accessed from other tasks in the workflow. StreamWorkflow also provides default plotting functionalities in 2D, 3D and nD, that plot all the instances in the workflow. Know more about MLPro's plotting functionalities.

Stream Scenario

A stream scenario in MLPro inherits from MLPro's scenario base class. The idea of a scenario in MLPro is to have all the elements together, required for a specific application, whether it is a training application or just a sample run. A scenario set's up the process parameters and runs the process for a given number of cycles as defined in the specific scenario implementation.

A stream scenario consists of two main elements:

- A stream object
- A streamtask workflow

Note: To plug these elements into the StreamScenario class, please implement the `_setup(p_mode, p_visualize, p_logging)` method of the same

---

A StreamScenario class takes care of the following tasks in a Stream processing application:

1. Fetching new instance at every step
2. Running the plugged in StreamWorkflow
3. Managing and updating the visualization windows
4. Storing the results of the workflow

Cross Reference

- Stream
- How To to be included
- API References

Stream Plotting

MLPro's streams module also provide plotting functionalities by default. The stream workflow and stream tasks can plot instances within the workflow and the task respectively. The default plotting functionality is available in 2 dimensional, 3 dimensional and N dimensional views. The plot view and specific plot properties can be set using a PlotSetting object. Below images show an example of the default plotting functionality in ND, 2D, 3D, respectively, in MLPro's streams module.

Cross References

- *Howto BF-STREAMS-102: Tasks Workflows And Stream Scenarios*
- *BF-MT - Multitasking*
- *API Reference: Streams*

Pool Objects

Native Streams

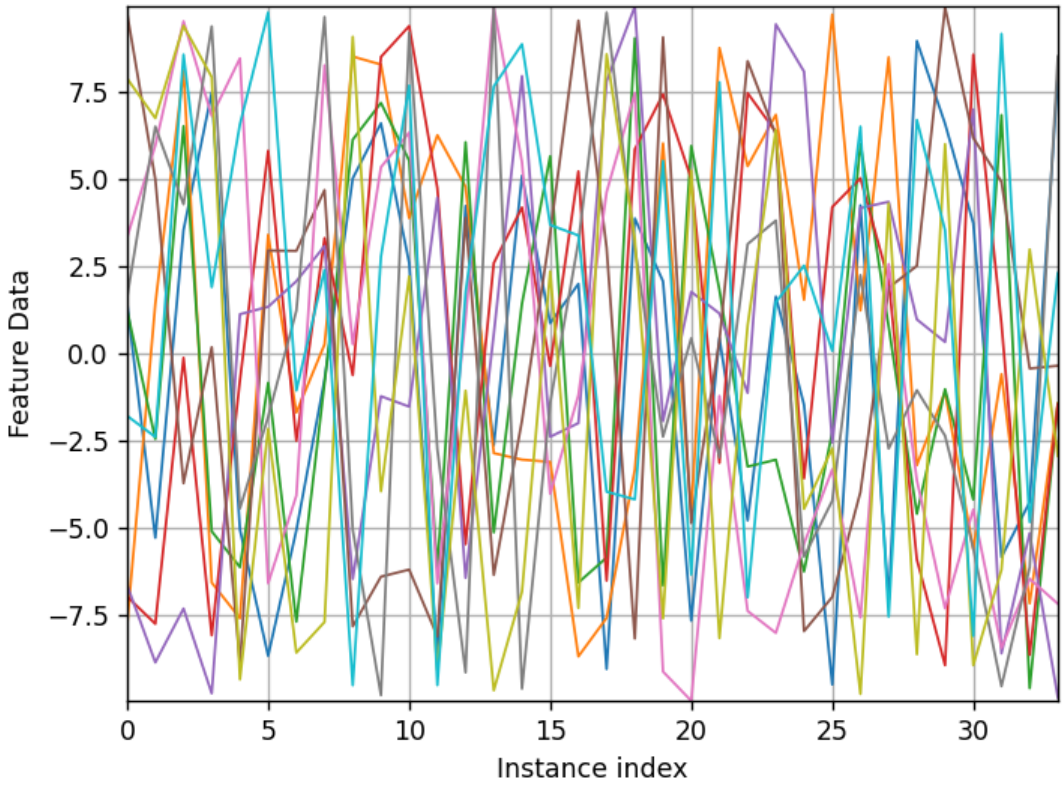
Along with third party stream support, MLPro also provides pool of native stream objects. The native streams in MLPro include the following implementations.

Please click here for Tutorial: *Howto BF-STREAMS-001: Accessing Native Data From MLPro*

Random 10 Dimensional

Ver. 1.0.0 (2022-12-13)

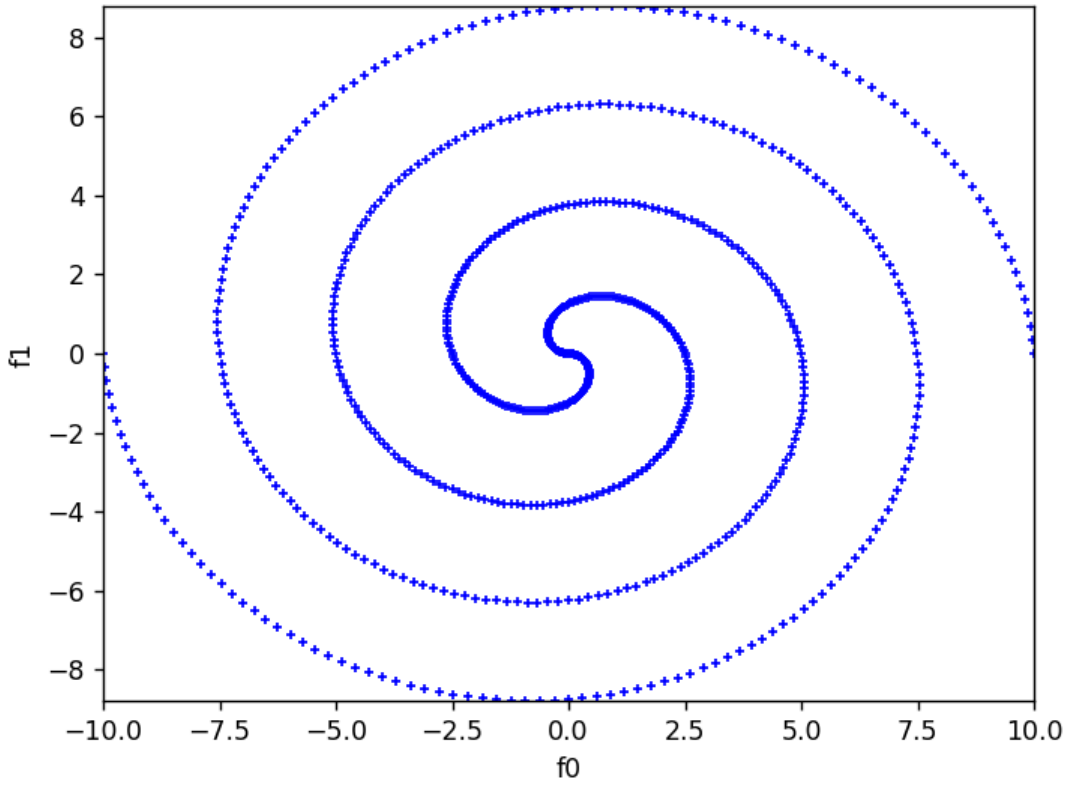
This module provides the native stream class StreamMLProRnd10D. This stream provides 1000 instances with 10-dimensional random feature data and 2-dimensional random label data.



Double Spiral 2D

Ver. 1.0.0 (2022-12-14)

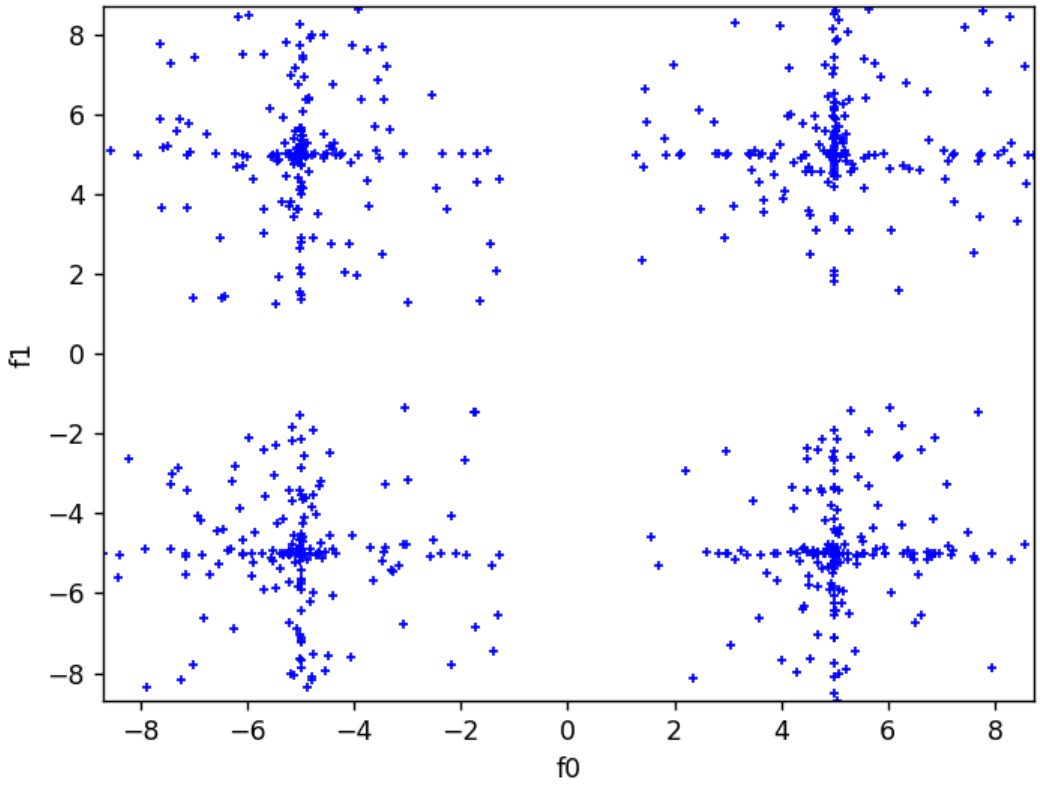
This module provides the native stream class DoubleSpiral2D. It provides 721 instances with 2-dimensional feature data that follow a double spiral pattern.



Static Clouds 2D

Ver. 1.0.0 (2022-12-15)

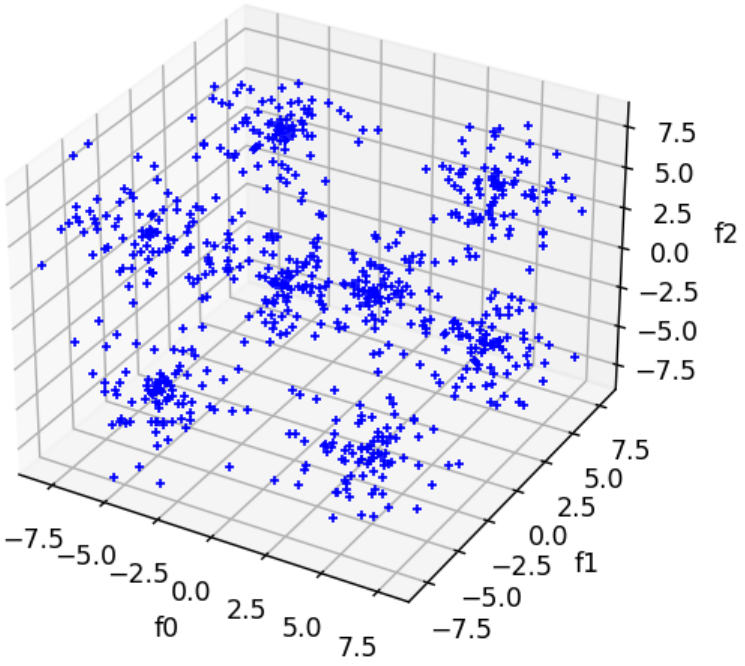
This module provides the native stream class StreamMLProStaticClouds2D. This stream provides 1000 instances with 2-dimensional random feature data placed around four fixed center points.



Static Clouds 3D

Ver. 1.0.0 (2022-12-15)

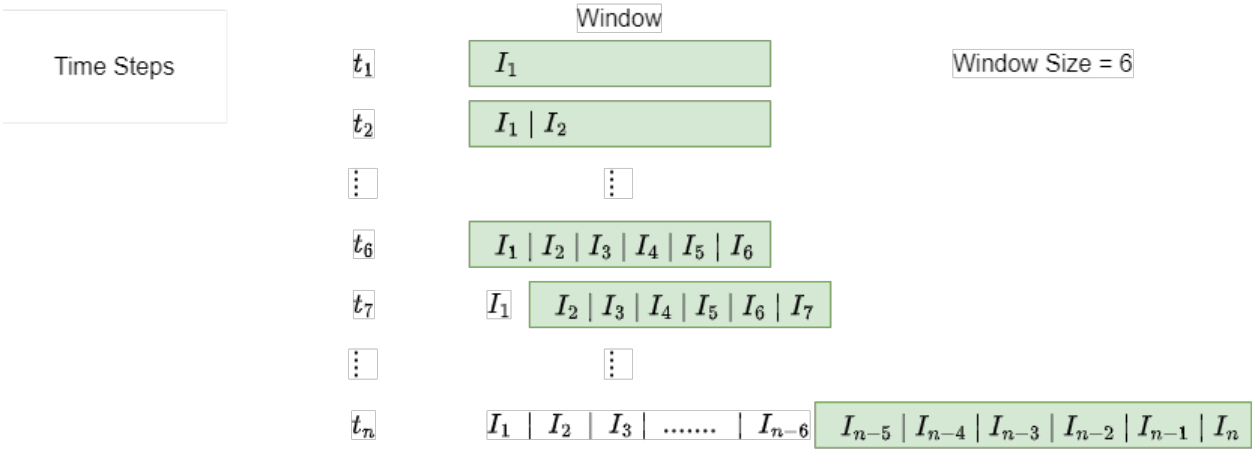
This module provides the native stream class StreamMLProStaticClouds3D. This stream provides 2000 instances with 3-dimensional random feature data placed around eight fixed center points.



Stream Tasks

Window

In streaming scenarios, data is available sequentially, and the amount of data received is directly proportional to the time for which the stream is active. In practice, this data accumulates in tremendous amounts as the application becomes complex. Processing data with minimum use of storage is important. A window task stores a small amount of data from the incoming stream, that can be used to process subsequent tasks based on a smaller amount of data that represents the stream behaviour.



The window task in MLPro, stores the most recent instances received from the stream in the buffer. The buffer size of the window is fixed and defined by the user. As soon as the buffer is full, the oldest instance is deleted form the buffer to add the latest instance to the buffer. The subsequent tasks in the workflow with dependency on window, have access to the data in the buffer.

Note: The availability of the buffered instances to the subsequent tasks can be delayed by setting the `p_delay` parameter to `True`. In this case, the buffered instances are only available once the buffer is completely full.

The window task of MLPro, also provides functionality to get statistical information about the buffered instances, such as Boundaries, Mean, Variance and Standard Deviation of the features of the instances in the buffer. Additionally, MLPro also provides visualization functionality for window, as shown below.

Cross Reference

- *Howto BF-STREAMS-110: Window*
- *API Reference: Streams*

Rearranger

A stream object consists of a Feature Space with a number of features defining each instance at a given timestep. A Rearranger task in MLPro maps the features of an input stream instances to a user defined feature space as an output. In other words, a rearranger task can be used to filter out un-interested features of a stream object for a particular task.

Cross Reference

- *Howto BF-STREAMS-111: Rearranger (2D)*
- *Howto BF-STREAMS-111: Rearranger (3D)*
- *Howto BF-STREAMS-111: Rearranger (nD)*
- *API Reference: Streams*

Deriver

The deriver task in MLPro provides a functionality to derive a selected feature and extend the feature with its derivative. In mathematics, the derivative is a concept that measures the rate of change of a function at a particular point. It is represented as the slope of the tangent line to a function at that point. The derivative is a fundamental tool in calculus and is used to study the properties of functions, such as their maxima, minima, and inflection points. It is also used in various scientific and engineering fields to study the behavior of systems that change over time, such as in the modeling of physical processes and in the analysis of dynamic systems. The derivative can be calculated using limits or through a number of differentiation rules for common functions, such as power, exponential, and logarithmic functions.

In the current implementation, we set up the basic formula of the derivation, as follows:

$f(x) = limx\ß0(f(x+x)f(x))/x$

Note: The order of derivative can be selected through `p_order_derivative`. If you would like to have two orders of derivative, then you have to add two separate tasks to the workflow.

Cross Reference

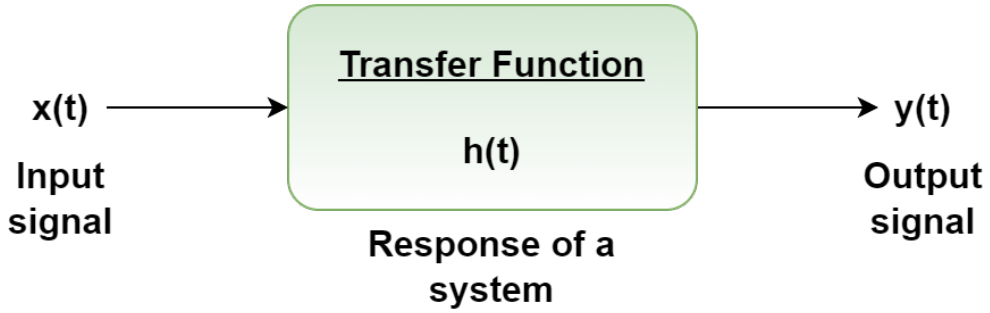
- *Howto BF-STREAMS-114: Deriver*
- *API Reference: Streams*

3.5.2 Physics

BF-PHYSICS is part of the third layer of MLPro-BF for application support, which provides elementary functionalities for the following topics:

Transfer Function

A transfer function is a mathematical representation of the relationship between the input and output of a time-invariant system. It is commonly used in control theory and electrical engineering to analyze and design systems with inputs and outputs, but it is not restricted only to those aspects. The transfer function provides valuable functionality to process inputs to outputs within a period of time. It can be used to design controllers that regulate the behaviour of the system, predict its response to inputs, and analyze the performance of the system in the frequency domain. Transfer functions play an important role in the design and analysis of control systems, communication systems, and signal processing systems.



In MLPro, there are three possibilities for transfer functions, which are:

1. Linear function
2. Custom function
3. Function approximation (future work)

Transfer Function class can be accessed as follows:

```
from mlpro.bf.physics import TransferFunction
```

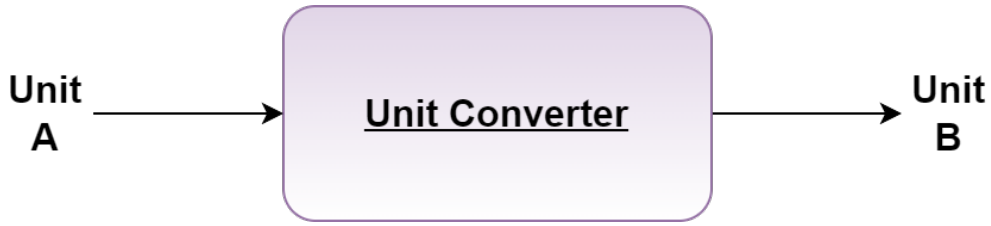
Cross Reference

- *Howto BF-PHYSICS-001: Transfer Functions*
- *Howto BF-PHYSICS-002: Unit Converter*
- *API Reference*

Unit Converter

MLPro's unit converter is a functionality to convert common units of measurements, such as:

- Length
- Pressure
- Electric Current
- Time
- Temperature
- Mass
- Power
- Force



Unit Converter class can be accessed as follows:

```
from mlpro.bf.physics.unitconverter import UnitConverter
```

Cross Reference

- *Howto BF PHYSICS 002*
- *API Reference*

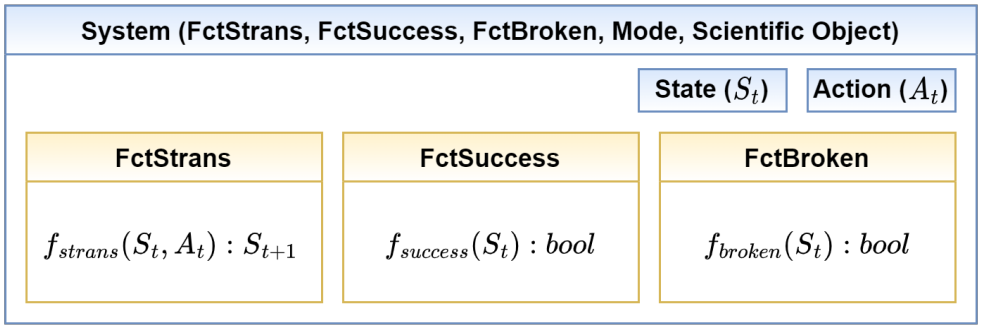
Physics-related functionality in a framework typically includes modules for dealing with simulating physical systems, performing numerical calculations and analysis, and many more.

3.5.3 State-based Systems

MLPro aims to standardize machine learning processes to accommodate complex applications in simplified reusable APIs. MLPro's Systems module standardizes state-based systems and their operation in a modular design. The keyword **state-based** implies the possibility to characteristically represent a system's unique state as a vector corresponding to a given timestep.

A state-based system has a definite condition at any given point in time, defined by a fixed number of variables that completely defines the condition. A system transits from a state to next state at each timestep based on the inherent state transition dynamics. However, this state transition is triggered by an external source of action, for example an Actuator.

In real application of state-based systems, such as controlled systems, it is highly interesting to maintain a desired system state, reach a system state or maximize system output, through optimum state transitions. Additionally, it is also an important concern to verify if the system is performing within the objective of the application or if the system has failed.



As shown in the figure above, MLPro's Systems module encapsulates the aforementioned functionalities into a standard template. The System object of MLPro can be reused to define any custom system with default methods to handle surrounding standard operations.

The system's module provides following objects and templates:

1. **System**:

The System class standardizes and provides the base template for any State-based System along with standard MLPro functionalities such as Logging, Timer, Cycle Management, Persistence, Real/Simulated mode and Reset. The System class additionally provides room for custom functionalities such as Reaction Simulation, Terminal State Monitoring such as Success and Broken. These custom functionalities can be incorporated by implementing the `_simulate_reaction()`, `_compute_success()`, `_compute_broken()` methods on Systems class or corresponding function classes (described below), which are then passed as a parameter to the system.

Note: The System class also supports operation in modes: **Real** and **Simulated**, based on which, it enables working with a real hardware or a simulated system respectively.

---

2. **FctStrans**:

The FctStrans (State Transition Function) standardizes the process of simulating the primary State Transition process of a System. The `simulate_reaction(p_state, p_action)` method of this class takes the current state of the environment and the action from the corresponding actuator as a parameter, and maps it to the next state of the system, based on the inherent dynamics.

Note: Please implement the `_simulate_reaction()` method of FctStrans, in order to re-use in a custom implementation.

3. **FctSuccess**:

A System state can be monitored through FctSuccess (Success Function) to determine if the system has reached the expected objective state/output. It maps the current state of the system to a boolean value indicating the success of a system.

Note: Please implement `_compute_success()` method of FctSuccess, in order to re-use it in a custom implementation.

4. **FctBroken**:

Similar to FctSuccess class, the FctBroken class standardizes the process of monitoring whether the system has reached a broken terminal state, by mapping the current state to a boolean value indicating the broken state.

Note: Please implement `_compute_broken()` method of FctBroken, in order to re-use it in custom implementation.

5. **State**:

The state object represents the current state of the system with respect to time. A state object inherits from the Element class of MLPro, which represents an element in a Multi-dimensional Set object, a State-Space in this case. The state consists information about the System for corresponding dimension of the related State-Space.

6. **Action**:

The Action object standardizes external input to the system. For example, input from a controller, input from an actuator or an agent in case of Reinforcement Learning. The standard Action object consists of an ActionElement or a list of ActionElements, in case of more than one action sources. The action element is similar to a state object, consisting corresponding values for all the dimension in the related action-space.

MLPro also provides the possibility to integrate real world hardware, such as controllers and hardware to the System object. Furthermore, Systems module integrates optional visualization and simulation functionalities from MuJoCo into MLPro for re-usability.

Learn more

MuJoCo Integration

MuJoCo is a well-known physics engine for its fast and accurate simulation. The aim is to facilitate research and development in robotics, biomechanics, graphics and animation, and other areas. More explanation about MuJoCo can be found in here.

In order to use the MuJoCo integration in MLPro, the following steps need to be done:

- **Create a MuJoCo Model**

  Create a MuJoCo model file accordingly to your design. Some example model are published by MuJoCo and can be accessed here. Below is an example of MuJoCo model file.

```
<mujoco>
  <option timestep="0.05" gravity="0 0 -9.81" integrator="RK4">
    <flag sensornoise="enable" energy="enable"/>
  </option>
  <worldbody>
        <light diffuse=".5 .5 .5" pos="0 0 3" dir="0 0 -1"/>
    <geom type="plane" size="1 1 0.1" rgba=".9 0 0 1"/>

    <body name="link1" pos="0 0 2" euler="0 0 0">
      <joint name="pin" type="hinge" axis = "0 -1 0" pos="0 0 0.5"/>
      <inertial pos="0 0 0" mass="1" diaginertia="1 1 1" />
      <geom type="cylinder" size="0.05 0.5" rgba="0 .9 0 1"/>
    </body>

  </worldbody>

  <actuator>
    <motor joint="pin" name="torque1" gear="1" ctrllimited="true" ctrlrange=
↪"-50 50"/>
  </actuator>
</mujoco>
```

The above model simulates one body called `link1` which has a cylindrical shape. It is attached to the world body with a joint called `pin` with the type of hinge. This joint is controlled by an actuator called `torque1` boundaries between -50 and 50.

- **Create a System**

  When you instantiate the System, put the MuJoCo model file path on `p_mujoco_file`. If the model is correct and the path is correct, then the wrapper will automatically wrap the state and action space based on the MuJoCo model.

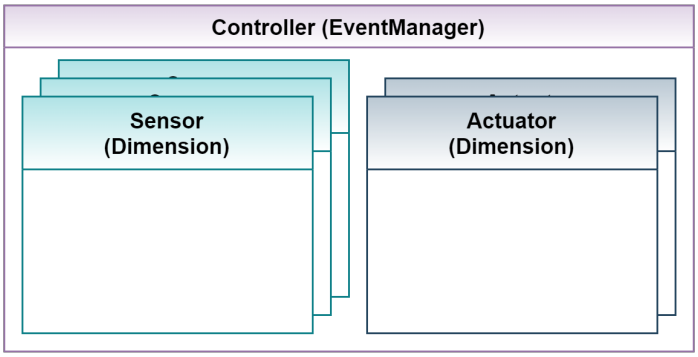
If you want to view your model only before including it in MLPro, you can use the MuJoCo tool by dragging and dropping the model file into it. The tool can be downloaded here.

Cross Reference

- MuJoCo Tool
- MuJoCo XML Reference
- MuJoCo Model Samples
- Unity Plug-in for MuJoCo
- *MuJoCo Wrapper*
- *Howto BF-SYSTEMS-002: Double Pendulum Systems wrapped with MuJoCo*
- *Howto BF-SYSTEMS-003: Cartpole Continuous Systems wrapped with MuJoCo*
- *Howto RL-AGENT-021: Train and Reload Single Agent Cartpole Discrete (MuJoCo)*
- *Howto RL-AGENT-022: Train and Reload Single Agent Cartpole Continuous (MuJoCo)*
- *Howto RL-ATT-002: Train and Reload Single Agent using Stagnation Detection Cartpole Discrete (MuJoCo)*
- *Howto RL-ATT-003: Train and Reload Single Agent using Stagnation Detection Cartpole Continuous (MuJoCo)*
- *Howto RL-ENV-005: Run Agent with random policy on double pendulum mujoco environment*

Hardware Access

MLPro also provides possibility to use the standardized Systems API with real world systems. It manages the interaction between real world hardware including sensors, actuators and controllers with the standard processes in MLPro framework.



As shown in the above figure, the controller class in MLPro registers a number of sensors and actuators for a system.

1. **Sensor**:

   A sensor observes a system to deliver characteristic information about the system at a given time. The Sensor class in MLPro inherits from the Dimension class.

2. **Actuator**:

   An actuator is responsible to generate an Action, which is executed in the real world system. Similar to Sensor class, the Actuator class is also inherited from the Dimension class of MLPro.

3. **Controller**:

   The controller is responsible to gather sensor data, compute the error signal and generate a corresponding action in order to maintain/reach the desired state of the system. The Controller class in MLPro manages the mapping details to map actions and states, to and from Actuators and Sensors, respectively.

Cross Reference

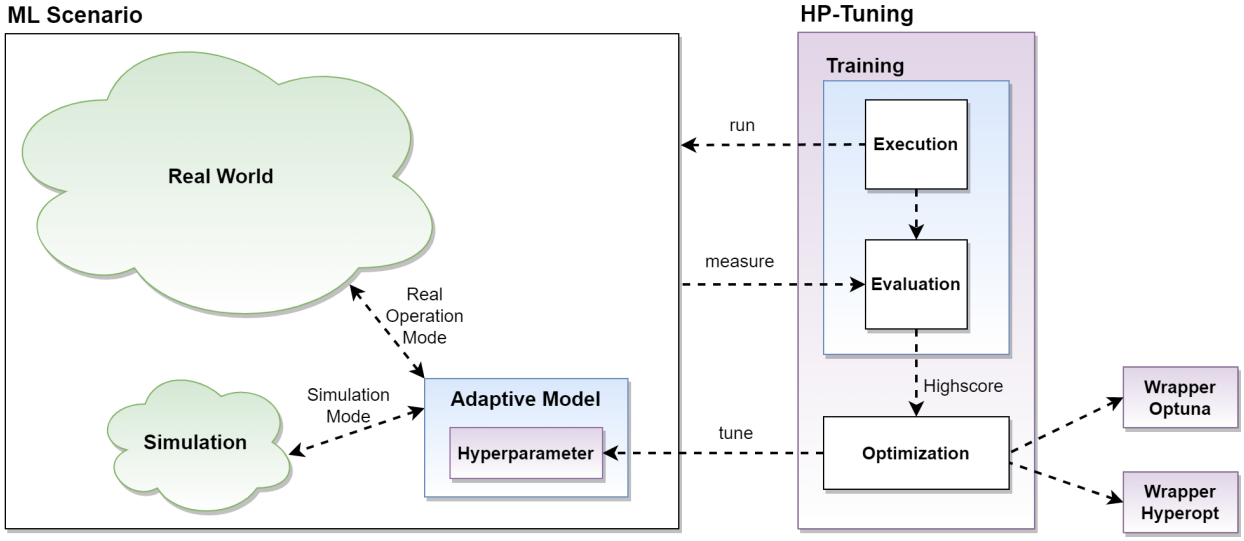
- *Howto BF-SYSTEMS-001: System, Controller, Actuator, Sensor*
- *API Reference BF-Systems*

Cross Reference

- *Howto BF-SYSTEMS-001: System, Controller, Actuator, Sensor*
- *Howto BF-SYSTEMS-002: Systems wrapped with MuJoCo*
- *API Reference BF-Systems*
- *API Reference BF-Systems Sample Pool*

3.6 Layer 4 - Machine Learning

One of the fundamental concepts in MLPro is to anchor universal standards for machine learning already in the basic functions. This shall facilitate the creation of higher ML functionalities and ensure their recombinability. The challenge here is to capture the nature of machine learning on an abstract and general level while establishing concrete templates and processes and solving elementary subtasks. The highest layer 4 of the basic functions of MLPro is dedicated to this topic.



The focus of the consideration is the *adaptive model* with its elementary properties

- Adaptivity
- Executability
- Parameterizability
- Persistence

Of course, this model does not exist just on its own. Rather, it interacts with a simulated or real object. For example, in the case of offline supervised learning, this can be a data set, in the case of online unsupervised learning, a data stream, or in the case of reinforcement learning, a state-based system. Topics like this are covered in MLPro in higher-level ML frameworks. On an abstract level, however, we introduce the *ML scenario* for this because although we do not yet know anything about the concrete ML application, we know that an adaptive model is involved. Furthermore, we propagate that the application is executable in "simulation" or "real operation" mode.

In MLPro, a model's *training and the tuning* of its hyperparameters are based on such an ML scenario. For tuning, powerful packages from third parties are already integrated at this low level using wrapper technology.

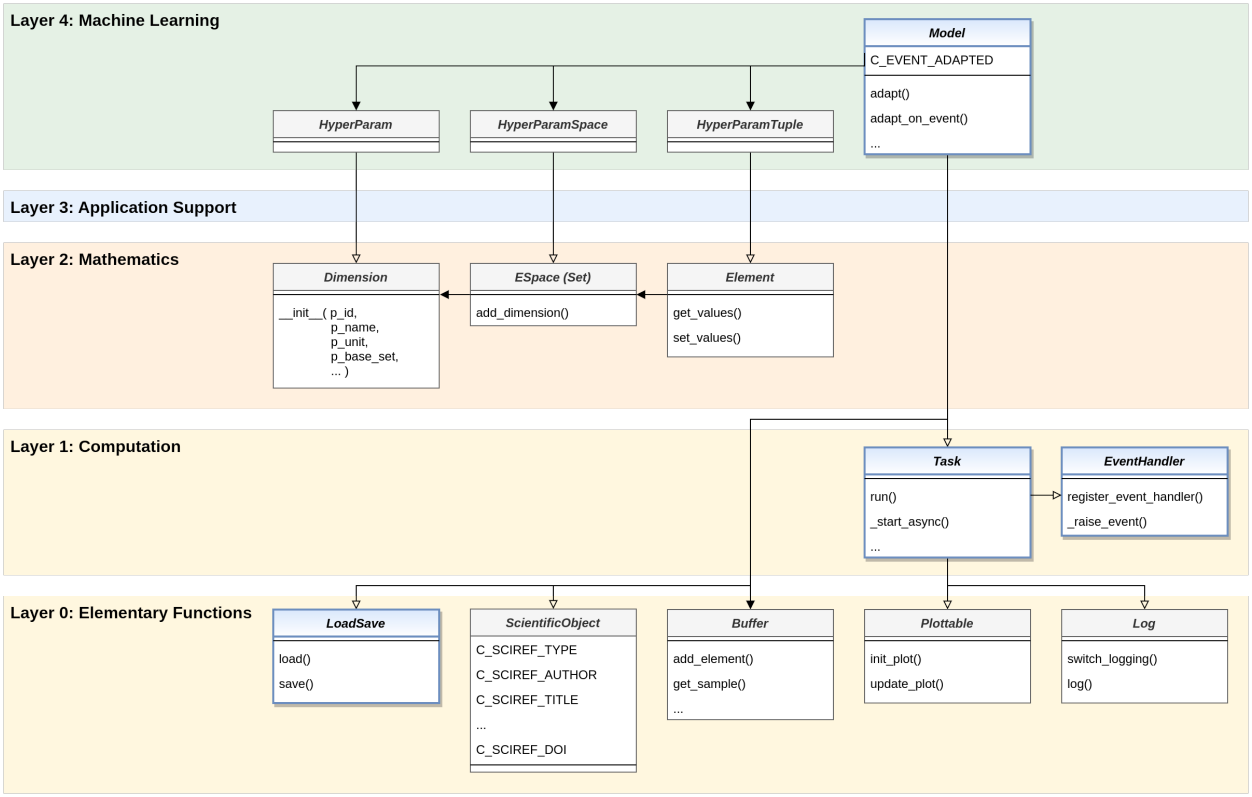
In more complex applications, it can be helpful to group multiple models and allow communication between them. It would be desirable here to optimally utilize the system resources through parallel processing. For this purpose, MLPro provides *adaptive workflows*.

In the field of systems engineering, the creation of suitable simulations or digital twins is an essential aspect. However, suppose a system cannot be described mathematically precisely enough due to its complexity or unknown influencing factors. In that case, machine learning methods can also be used to imitate the system behavior based on historical data and/or online monitoring. For this purpose, MLPro provides standards for adaptive systems.

Learn more

3.6.1 The Adaptive Model

MLPro provides the central template class **Model** for adaptive models. This bundles all properties important for machine learning on an abstract level. It represents the basis for all higher adaptive classes of the entire MLPro ecosystem and inherits its essential properties and possibilities for application-specific adjustments to them.



Performant Execution

As shown in the simplified class diagram above, the Model class is made up of numerous base classes of the lower levels through inheritance. So, from *Layer 1 - Computation*, it inherits the executability and asynchronous processing capabilities of class **Task**. In this way, it can also be combined in workflows to (parallelly/asynchronously) executable groups of models. From class **EventHandler** of the same level, it inherits the ability to raise events and forward them to registered event handlers.

Persistence

In particular, from *Layer 0 - Elementary Functions*, it inherits the ability of the **LoadSave** class to be able to be saved and reloaded. Other elementary capabilities such as logging, visualization, buffering of sample data, and referencing a scientific source are also fed in from this lower level.

Adaptivity

The Model class itself adds the ability to adapt. To this end, two mechanisms are introduced that support **explicit adaptation** based on external data and **event-oriented adaptation**. In both cases, the event **C_EVENT_ADAPTED** is raised, which can also be optionally handled as part of event handling. In this way, adaptation cascades can be triggered in a group of cooperating models.

Hyperparameters

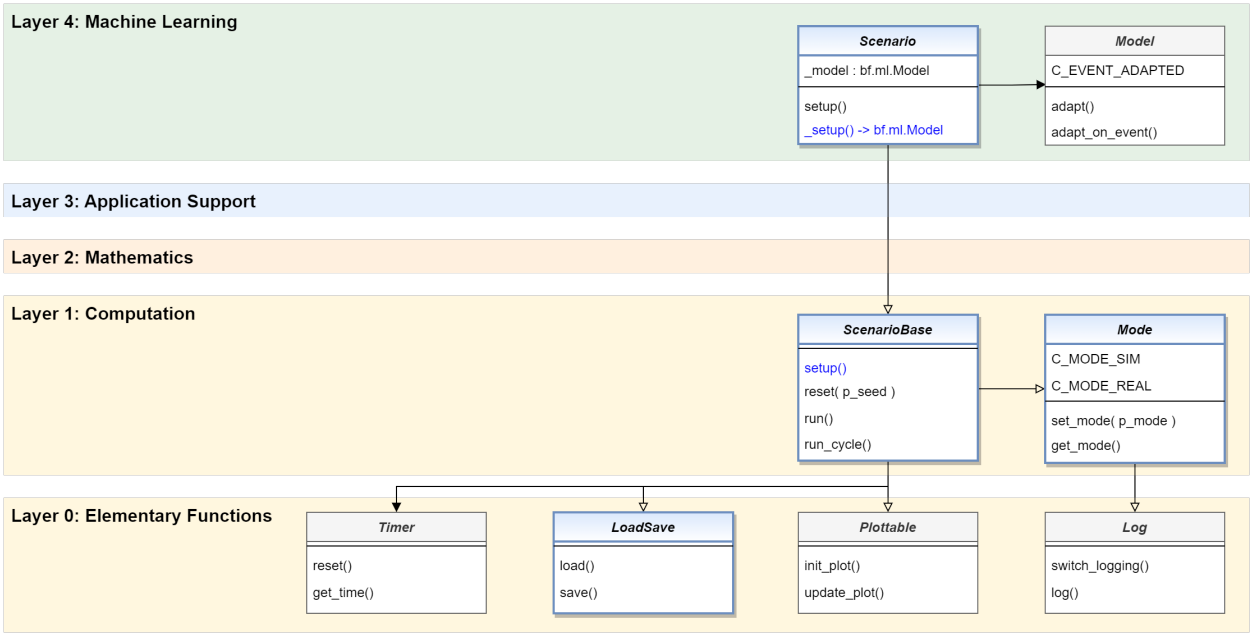
Also, at the top layer 4 for machine learning, a system for hyperparameters is introduced and added to the Model class. These, in turn, take up the concepts of **Dimension**, **Set/Space**, and **Element** from *Layer 2 - Mathematics*.

Cross Reference

- *Howto BF-ML-001: Adaptive Model*
- *Howto BF-ML-010: Hyperparameters*
- *Models in Reinforcement Learning: Policy, Agent, MultiAgent*
- *Models in Game Theory: Player, MultiPlayer*
- *BF-Events - Event Handling*
- *API Reference BF-ML*

3.6.2 ML Scenarios

As already mentioned, adaptive models in MLPro are combined with their concrete context to form an ML scenario. MLPro provides the abstract template class **bf.ml.Scenario** for this. At this level, this is not yet intended for use in your own customer applications, but is only used here to standardize the basic properties of an ML scenario.



From the root class of all scenarios in MLPro *bf.ops.ScenarioBase* it inherits the following properties

- Operation mode (simulation or real operation)
- Execution of cycles
- Persistence
- Visualization

and adds at this level the management of an internal adaptive model.

Cross Reference

- *Class bf.ops.ScenarioBase*
- *API Reference BF-ML*

3.6.3 Training and Tuning

A template for training models in their defined context is also introduced at this level. In a broader sense, this also includes finding an optimal value assignment for their hyperparameters. In MLPro, the **Training** class defines standards for this. Although abstract at this level, it fully implements the hyperparameter tuning here. The basic concept pursued here envisages executing an ML scenario under defined conditions and allowing the model contained therein to learn.

Persistence of Training Results

At the end of the training, the training results are saved in the file system. In particular, the entire scenario is saved here for later operational use. This includes both the trained model and the context in the last state.

Scoring

One of the training results is the **highscore**. Its determination is of course heavily dependent on the type of learning and can therefore only be specified in higher layers of MLPro. In any case, however, it is basically a real number that allows a qualitative statement about the learning performance of the model in its scenario.

Hyperparameter Tuning

Hyperparameter tuning is an optional training function performed by its own **HyperParamTuner** class. In particular, it defines the **maximize** method, which maximizes the highscore of a designated training by varying the hyperparameters of the model it contains. The optimization itself is not performed natively by MLPro, but by third-party packages. To this purpose, MLPro provides wrappers for *Optuna* and *Hyperopt*.

Cross Reference

- *API Reference BF-ML*
- *Wrapper for Optuna*
- *Wrapper for Hyperopt*

3.6.4 Adaptive Workflows

The **Model** class inherits from the **bf.mt.Task** class, among others. In combination with another class **AWorkflow**, all possibilities of MLPro's multitasking abilities are unlocked for machine learning:

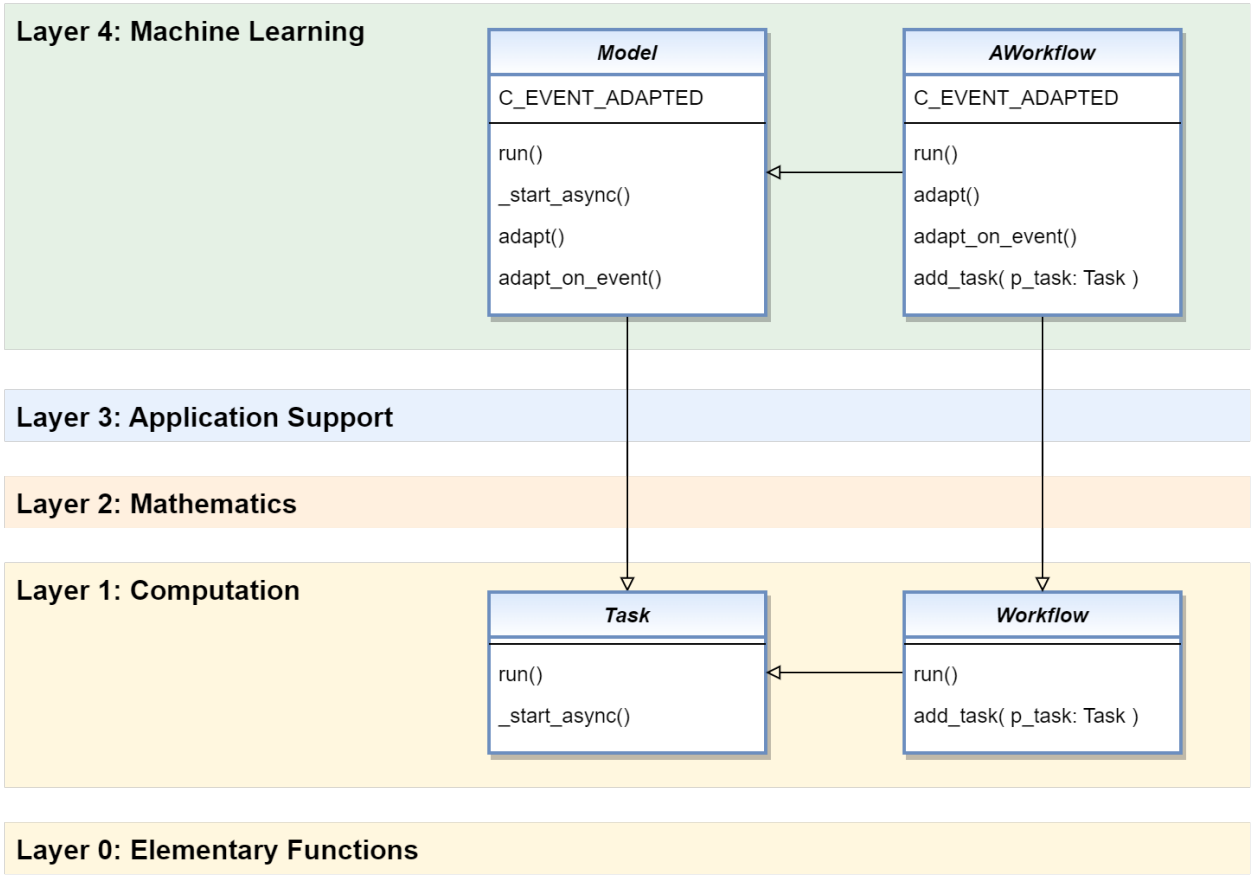
1. Models can execute internal methods asynchronously

2. Models can run as separate threads or processes

3. Models can be grouped into macro models in workflows

4. Models can share/exchange data using a shared object

This enables the efficient execution and adaptation of models and model groups using all available runtime resources.

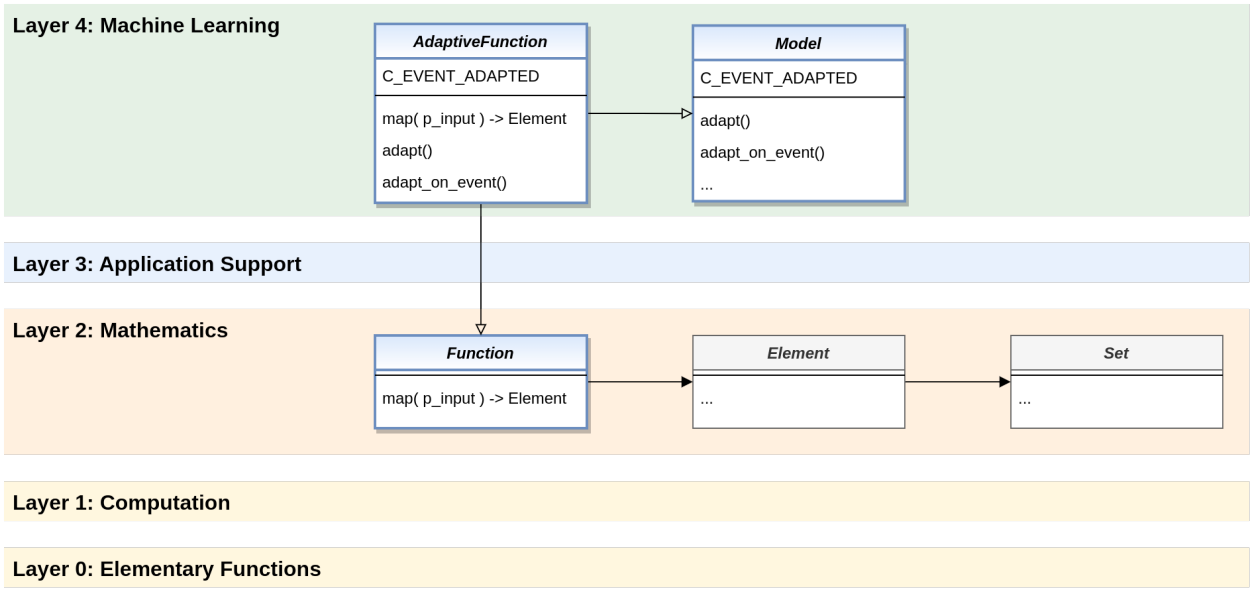


Cross Reference

- *BF-MT: Multitasking*

3.6.5 Adaptive Functions

A special kind of adaptive models are the **adaptive functions**. They combine the properties of a mathematical function with those of an adaptive model. The **class AdaptiveFunction** manifests and standardizes these functions without implementing concrete learning paradigms. The latter happens in higher MLPro frameworks, for example in MLPro-SL for offline/online supervised learning.



Adaptive functions have a high practical relevance. They are reused within MLPro, e.g. in connection with model-based agents. But they can also be used in general for predictions.

Cross Reference

- *BF-Math: Mathematics*
- *SL: Adaptive Functions for Supervised Learning*
- *RL: Model-based Agents*

3.6.6 Adaptive Systems (coming soon)

This functionality is in preparation. Further explanation coming soon…

Cross Reference

- *BF-Systems: State-based Systems*

Cross Reference

- *Related Howtos*
- *API Reference BF-ML - Machine Learning*
- *API Reference BF-ML-Systems - Adaptive Systems*

CHAPTER

FOUR

MLPRO-SL - SUPERVISED LEARNING

4.1 Overview

MLPro provides a subtopic package for supervised learning, namely MLPro-SL. At the moment, the implementation is still limited but we are working on it and improving it to bring you full supervised learning functionalities in the near future. MLPro-SL is designed to handle online and offline supervised learning, which means that the model can be used for different purposes, e.g. model-based reinforcement learning, online adaptivity, and more.

The current implementation covers:

- A base class of an adaptive function for supervised learning
- A base class of an adaptive function for feedforward neural networks, including MLP
- Ready-to-use PyTorch-based MLP networks in the pool of objects

Learn more

- *Getting started with MLPro-SL*

Cross Reference

- *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*
- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *Howto RL-MB-003: MBRL on RobotHTM Environment*
- *API Reference: MLPro-SL*
- *API Reference: MLPro-SL Pool of Objects*

4.2 Getting Started

As mentioned in the introductory section, MLPro-SL's functionalities are still limited and not ready to be labelled as the first version. However, we are working on it to enhance MLPro-SL and bring you full supervised learning functionalities soon.

At the moment, we provide a basic template class for *supervised learning adaptive function*, which has been extended to the feedforward neural network. We introduce MLP as a sample of the MLPro-SL model and provide a ready-to-use PyTorch-based multilayer perceptron network.

After following the below step-by-step guideline, we expect the user understands the MLPro-SL in practice and starts using MLPro-SL.

1. **What is MLPro?**
   If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially start with understanding MLPro by checking out the following steps:
   (a) *MLPro: An Introduction*
   (b) introduction video of MLPro
   (c) *installing and getting started with MLPro*
   (d) MLPro paper in Software Impact journal
2. **What is Supervised Learning?**
   If you have not dealt with supervised learning, we recommend starting to understand at least the basic concept of supervised learning. There are plenty of references, articles, papers, books, or videos on the internet that explains supervised learning. As an overview, supervised learning is a type of machine learning in which a model is trained on a labelled dataset to predict the output for new/unseen inputs. Supervised learning can be used to build a predictive model that can make predictions based on available data.
3. **What is MLPro-SL?**
   We expect that you have a basic knowledge of MLPro and supervised learning. Therefore, you need to understand the overview of MLPro-SL by following the steps below:
   (a) *MLPro-SL introduction page*
4. **Understanding Adapative Function in MLPro-SL**
   First of all, it is important to understand the adaptive function in MLPro-SL, which can be found on *this page*.
   Then, you can start following some of our howto files related to the adaptive function in MLPro-SL, which is used for model-based RL, as follows:
   (a) *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*
   (b) *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
   For more advanced supervised learning technique in model-based RL, e.g. applying a native model-based RL network, here is an example that can be used as a reference:
   (c) *Howto RL-MB-003: MBRL on RobotHTM Environment*
5. **Additional Guidance**
   After following the previous steps, we hope that you could practice MLPro-SL and start using this subpackage for your SL-related activities. For more advanced features, we highly recommend you to check out the following files:
   (a) *API Reference: MLPro-SL*
   (b) *API Reference: MLPro-SL Pool of Objects*

4.3 Adaptive Functions

In supervised learning, the adaptive function refers to the ability of the model to adjust its parameters in response to new input data. Specifically, it refers to the ability of the model to learn from the labelled training data and improve its performance on new/unseen data. During the training phase, the model is presented with a set of input features and the corresponding output labels, and it adjusts its parameters (e.g. weights and biases) to minimize the error between its predicted outputs and the true labels. This process of updating the model's parameters is often referred to as adaptation. The goal of this learning process is for the model to be able to accurately predict the output for input data. This is known as the model's generalization performance, and it is a key measure of its adaptive function.

We provide ready-to-use adaptive function models in MLPro-SL's pool of objects, which can be found, as follows:

4.3.1 Adaptive Function Pool

Adaptive Function Pytorch

First of all, it is important to understand the adaptive function in MLPro-SL, which can be found on *this page*.

In this functionality, we integrate PyTorch into MLPro-SL, thus PyTorch functionalities can be reused in MLPro-SL.

At the moment, we provide only PyTorch-based multilayer perceptron in the pool, but recurrent neural networks and transformers are planned to be included in the next version shortly.

Cross Reference

- *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*
- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *API Reference: MLPro-SL Pool of Objects*

Cross Reference

- *BF-ML: Adaptive Functions*
- *API Reference: MLPro-SL*

CHAPTER

FIVE

MLPRO-RL - REINFORCEMENT LEARNING

5.1 Overview

MLPro-RL is the first ready-to-use subpackage in MLPro that is intended for reinforcement learning (RL)-related activities. MLPro-RL provides complete base classes of the main RL components, e.g. agent, environment, policy, multiagent, and training. The training loop is developed based on the Markov Decision Process (MDP) model, as shown in the following diagram.

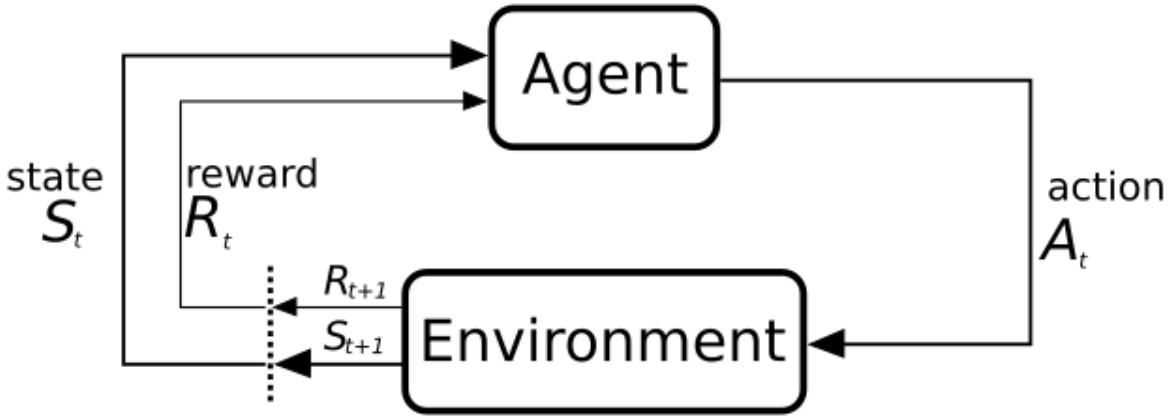


Fig. 1: This figure is taken from Sutton and Barto, licensed by CC BY-NC-ND 2.0.

An MDP model contains two major components, such as the environment and the agent. The agent can be considered as the decision maker, who chooses actions based on its policy by taking into account the current state of the environment. The environment is the surrounding where the agent lives and interacts. The actual condition in the environment is represented by states. MDP formulates the interaction between the agent and the environment, where the agent selects an action and sends the action to the environment. The environment reacts to the given action that makes the condition in the environment change. Then, the environment sends back the information in the form of states and reward to indicate the actual condition in the environment and the impact of the taken action on the environment respectively. Afterwards, the agent can adapt its policy and repeat the interactions until reaching optimality.

MLPro-RL can handle a broad scope of RL training, including model-free RL or model-based RL, single-agent or multi-agent, and simulation or real hardware mode. Hence, this subpackage can be a one-stop solution for students, educators, RL engineers or RL researchers to support their RL-related tasks. The structure of MLPro-RL can be found in the following figure.

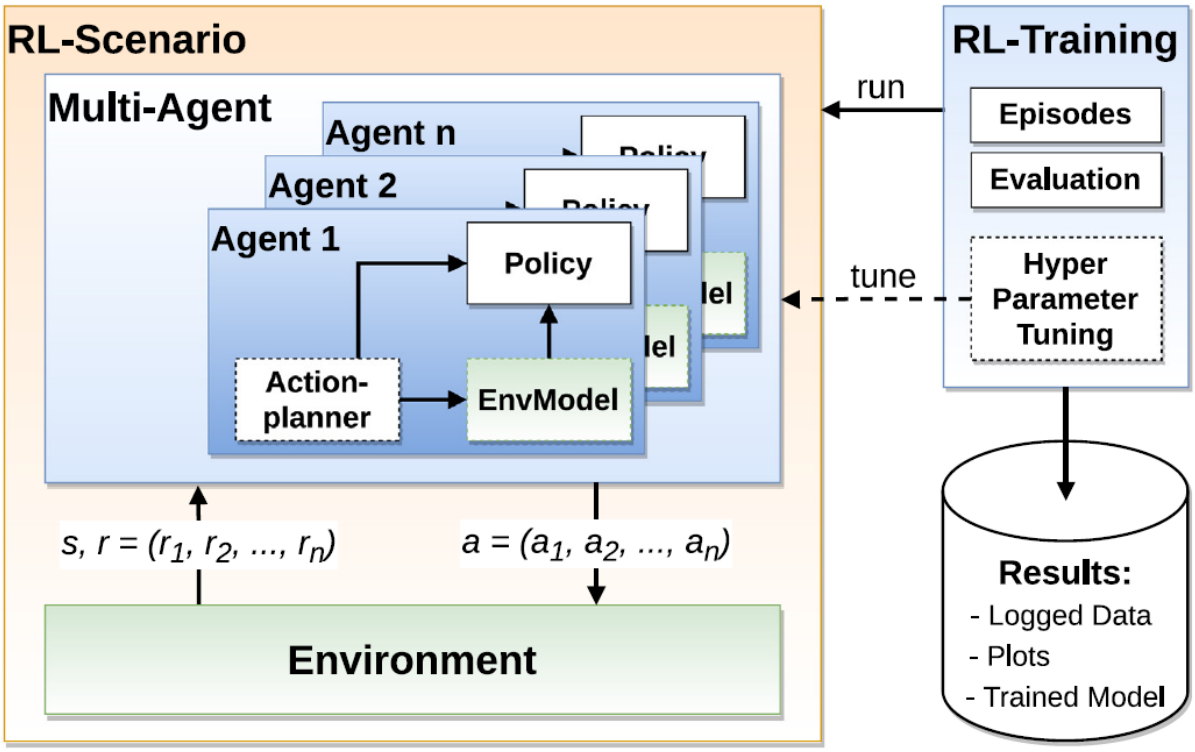


Fig. 2: This figure is taken from MLPro 1.0 paper.

If you are interested to utilize MLPro-RL, you can easily access the RL modules, as follows:

```
from mlpro.rl import *
```

Additionally, you can find the more comprehensive explanations of MLPro-RL including a sample application on controlling a UR5 Robot in this paper: MLPro 1.0 - Standardized Reinforcement Learning and Game Theory in Python.

Learn more

- *Getting started with MLPro-RL*

Cross Reference

- *Related Howtos*
- *API Reference: MLPro-RL*
- *API Reference: MLPro-RL Pool of Objects*
- MLPro 1.0 Paper
- MLPro GitHub

5.2 Getting Started

Here is a concise series to introduce all users to the MLPro-RL in a practical way, whether you are a first-timer or an experienced MLPro user.

If you are a first-timer, then you can begin with **Section (1) What is MLPro?**.

If you have understood MLPro but not reinforcement learning, then you can jump to **Section (2) What is Reinforcement Learning?**.

If you have experience in both MLPro and reinforcement learning, then you can directly start with **Section (3) What is MLPro-RL?**.

After following the below step-by-step guideline, we expect the user understands the MLPro-RL in practice and starts using MLPro-RL.

1. What is MLPro?

If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially start with understanding MLPro by checking out the following steps:

(a) *MLPro: An Introduction*

(b) introduction video of MLPro

(c) *installing and getting started with MLPro*

(d) MLPro paper in Software Impact journal

2. What is Reinforcement Learning?

If you have not dealt with reinforcement learning, we recommend starting to understand at least the basic concept of reinforcement learning. There are plenty of references, articles, papers, books, or videos on the internet that explains reinforcement learning. But, for deep understanding, we recommend you to read the book from Sutton and Barto, which is Reinforcement Learning: An Introduction.

3. What is MLPro-RL?

We expect that you have a basic knowledge of MLPro and reinforcement learning. Therefore, you need to understand the overview of MLPro-RL by following the steps below:

(a) *MLPro-RL introduction page*

(b) Section 4 of MLPro 1.0 paper

4. Understanding Environment in MLPro-RL

First of all, it is important to understand the structure of an environment in MLPro, which can be found on *this page*.

Then, you can start following some of our howto files related to the environment in MLPro-RL, as follows:

(a) *Howto RL-001: Reward*

(b) *Howto RL-AGENT-001: Run an Agent with Own Policy*

5. Understanding Agent in MLPro-RL

In reinforcement learning, we have two types of agents, such as a single-agent RL or a multi-agent RL. Both of the types are covered by MLPro-RL. To understand the different possibilities of an agent in MLPro, you can visit *this page*.

Then, you need to understand how to set up a single-agent and a multi-agent RL in MLPro-RL by following these examples:

(a) *Howto RL-AGENT-001: Run an Agent with Own Policy*

(b) *Howto RL-AGENT-003: Run Multi-Agent with Own Policy*

6. Selecting between Model-Free and Model-Based RL

In this section, you need to select your direction of the RL training, whether it is a model-free RL or a model-based RL. However, firstly, you can pay attention to these two pages, which are *RL scenario* and *training*, before selecting either of the paths below.

- Model-Free Reinforcement Learning

  To practice model-free RL in the MLPro-RL package, here are a video and some ready-to-use howto files that can be followed:

  (a) A sample application video of MLPro-RL on a UR5 robot

  (b) *Howto RL-AGENT-002: Train an Agent with Own Policy*

  (c) *Howto RL-AGENT-004: Train Multi-Agent with Own Policy*

- Model-Based Reinforcement Learning

  Model-based RL contains two learning paradigms, such as learning the environment (model-based learning) and utilizing the model (e.g. as an action planner). To practice model-based RL in the MLPro-RL package, here are a howto file that can be followed:

  (a) *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*

  (b) *Howto RL-MB-002: MBRL with MPC on Grid World Environment*

  For more advanced MBRL technique, e.g. applying a native MBRL network, here is an example that can be used as a reference:

  (c) *Howto RL-MB-003: MBRL on RobotHTM Environment*

7. Additional Guidance

After following the previous steps, we hope that you could practice MLPro-RL and start using this subpackage for your RL-related activities. For more advanced features, we highly recommend you to check out the following howto files:

(a) *Howto RL-AGENT-011: Train and Reload Single Agent (Gym)*

(b) *Howto RL-AGENT-021: Train and Reload Single Agent (MuJoCo)*

(c) *Howto RL-HT-001: Hyperopt*

(d) *Howto RL-HT-002: Optuna*

(e) *Howto RL-ATT-001: Stagnation Detection*

(f) *Howto RL-ATT-002: SB3 Policy with Stagnation Detection*

5.3 Environments

In RL, the environment refers to the physical, virtual, or abstract system in which the agent interacts and learns. The environment is the source of stimuli that the agent perceives and the arena in which it takes actions.

The environment is defined by a set of states, actions, and transition dynamics. The state space is the set of all possible states that the agent can observe, and the action space is the set of all possible actions that the agent can take. The transition dynamics describe how the environment changes in response to the agent's actions.

The environment also provides the agent with a reward signal that indicates how well it is doing in terms of achieving its goals. The reward function is a mapping from states and actions to real-valued scalars that quantifies the desirability of each state-action pair.

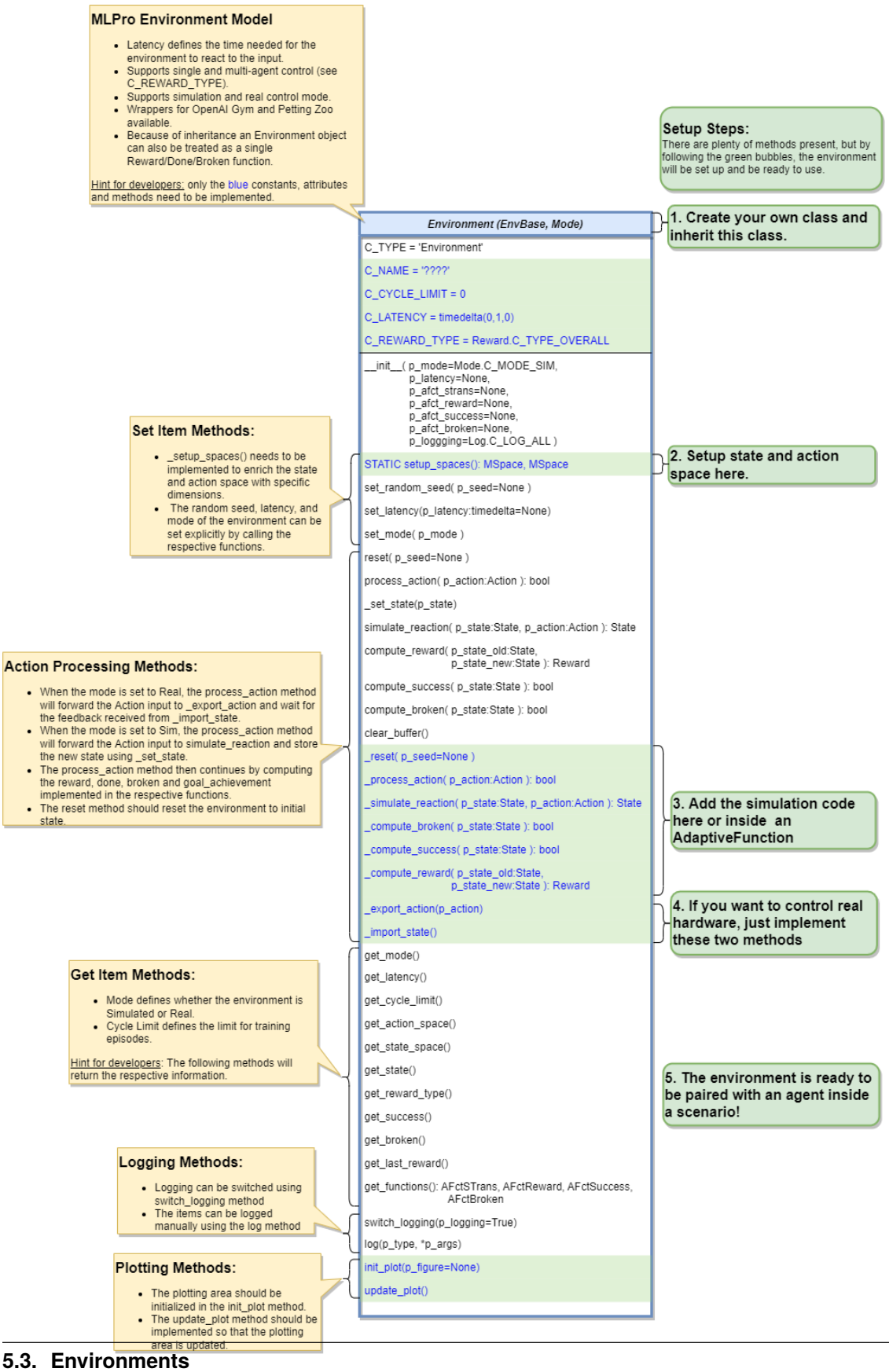
The agent interacts with the environment over a sequence of time steps. At each time step, the agent observes the current state of the environment and selects an action. The environment then transitions to a new state and returns a reward signal to the agent.

Overall, the environment in RL provides the agent with the necessary information to learn a policy that maps states to actions and maximizes the cumulative reward signal. The environment can be real-world or simulated, and can be described by a mathematical model or a black box.

MLPro-RL supplies two main classes for an environment to support model-free and model-based RL. The first base class is Environment, which has a role as a template for designing environments for both approaches. The second base class is EnvModel, which is adaptive and utilized in model-based RL. Both Environment and EnvModel classes inherit a common base class EnvBase and its fundamental properties, e.g. state and action space definition, reset the corresponding environment method, state transition method, etc.

There are two main possibilities to set up an environment in MLPro, such as,

5.3.1 Developing Custom Environments



- **Environment Creation for Simulation Mode**

  To create an environment that satisfies MLPro interface is immensly simple and straigtforward. Basically a MLPro environment is a class with 5 main functions. Each environment must apply the following mlpro functions:

```
from mlpro.rl.models import *

class MyEnvironment(Environment):
    """
    Custom Environment that satisfies mlpro interface.
    """
    C_NAME          = 'MyEnvironment'
    C_LATENCY       = timedelta(0,1,0)         # Default latency 1s
    C_REWARD_TYPE   = Reward.C_TYPE_OVERALL    # Default reward type

    def __init__(self, p_mode=C_MODE_SIM, p_latency:timedelta=None, p_
↪logging=True):
        """
        Parameters:
            p_mode              Mode of environment (simulation/real)
            p_latency           Optional: latency of environment. If not␣
↪provided
                                internal value C_LATENCY will be used by␣
↪default
            p_logging           Boolean switch for logging
        """

        super().__init__(p_latency=p_latency, p_logging=p_logging)
        self._setup_spaces()
        self.set_mode(p_mode)

    def _setup_spaces(self):
        """
        Implement this method to enrich the state and action space with␣
↪specific
        dimensions.
        """

        # Setup state space example
        # self.state_space.add_dim(Dimension(0, 'Pos', 'Position', '', 'm', 'm',␣
↪[-50,50]))
        # self.state_space.add_dim(Dimension(1, 'Vel', 'Velocity', '', 'm/sec', '\
↪frac{m}{sec}', [-50,50]))

        # Setup action space example
        # self.action_space.add_dim(Dimension(0, 'Rot', 'Rotation', '', '1/sec',
↪'\frac{1}{sec}', [-50,50]))
        ....

    def _simulate_reaction(self, p_action:Action) -> None:
        """
        Simulates a state transition of the environment based on a new␣
↪action.
```

(continues on next page)

(continued from previous page)

```
        Please use method set_state() for internal update.

        Parameters:
            p_action      Action to be processed
        """
        ....

    def reset(self) -> None:
        """
        Resets environment to initial state.
        """
        ....

    def compute_reward(self) -> Reward:
        """
        Computes a reward.

        Returns:
          Reward object
        """
        ....

    def _evaluate_state(self) -> None:
        """
        Updates the goal achievement value in [0,1] and the flags done and␣
↪broken
        based on the current state.
        """

        # state evaluations example
        # if self.done:
        #     self.goal_achievement = 1.0
        # else:
        #     self.goal_achievement = 0.0
        ....
```

One of the benefits for MLPro users is the variety of reward structures, which is useful for Multi-Agent RL and Game Theoretical approach. Three types of reward structures are supported in this framework, such as:

1. **C_TYPE_OVERALL** as the default type and is a scalar overall value
2. **C_TYPE_EVERY_AGENT** is a scalar for every agent
3. **C_TYPE_EVERY_ACTION** is a scalar for every agent and action.

- **Environment Creation for Real Hardware Mode**

  In MLPro, we can choose simulation mode or real hardward mode. For real hardware mode, the creation of an environment is very similar to simulation mode. You do not need to define **_simulate_reaction**, but you need to replace it with **_export_action** and **_import_state** as it is shown in the following:

```
from mlpro.rl.models import *
```

(continues on next page)

(continued from previous page)

```
class MyEnvironment(Environment):
    """
    Custom Environment that satisfies mlpro interface.
    """
    C_NAME          = 'MyEnvironment'
    C_LATENCY       = timedelta(0,1,0)         # Default latency 1s
    C_REWARD_TYPE   = Reward.C_TYPE_OVERALL    # Default reward type

    def __init__(self, p_mode=C_MODE_REAL, p_latency:timedelta=None, p_
↪logging=True):
        """
        Parameters:
            p_mode              Mode of environment (simulation/real)
            p_latency           Optional: latency of environment. If not␣
↪provided
                                internal value C_LATENCY will be used by␣
↪default
            p_logging           Boolean switch for logging
        """

        super().__init__(p_latency=p_latency, p_logging=p_logging)
        self._setup_spaces()
        self.set_mode(p_mode)

    def _setup_spaces(self):
        """
        Implement this method to enrich the state and action space with␣
↪specific
        dimensions.
        """

        # Setup state space example
        # self.state_space.add_dim(Dimension(0, 'Pos', 'Position', '', 'm', 'm',␣
↪[-50,50]))
        # self.state_space.add_dim(Dimension(1, 'Vel', 'Velocity', '', 'm/sec', '\
↪frac{m}{sec}', [-50,50]))

        # Setup action space example
        # self.action_space.add_dim(Dimension(0, 'Rot', 'Rotation', '', '1/sec',
↪'\frac{1}{sec}', [-50,50]))
        ....

    def _export_action(self, p_action:Action) -> bool:
        """
        Exports given action to be processed externally (for instance by a␣
↪real hardware).

        Parameters:
            p_action      Action to be exported

        Returns:
```

(continues on next page)

(continued from previous page)

```
            True, if action export was successful. False otherwise.
        """
        ....

    def _import_state(self) -> bool:
        """
        Imports state from an external system (for instance a real␣
↪hardware).
        Please use method set_state() for internal update.

        Returns:
          True, if state import was successful. False otherwise.
        """
        ....

    def reset(self) -> None:
        """
        Resets environment to initial state.
        """
        ....

    def compute_reward(self) -> Reward:
        """
        Computes a reward.

        Returns:
          Reward object
        """
        ....

    def _evaluate_state(self) -> None:
        """
        Updates the goal achievement value in [0,1] and the flags done and␣
↪broken
        based on the current state.
        """

        # state evaluations example
        # if self.done:
        #     self.goal_achievement = 1.0
        # else:
        #     self.goal_achievement = 0.0
        ....
```

- **Environment from Third Party Packages**

  Alternatively, if your environment follows Gym or PettingZoo interface, you can apply our relevant useful wrappers for the integration between third party packages and MLPro. For more information, please click *here*.

- **Environment Checker**

  To check whether your developed environment is compatible to MLPro interface, we provide a test script using unittest. At the moment, you can find the source code here. We will prepare a built-in

testing module in MLPro, show you how to excecute the testing soon and provides an example as well.

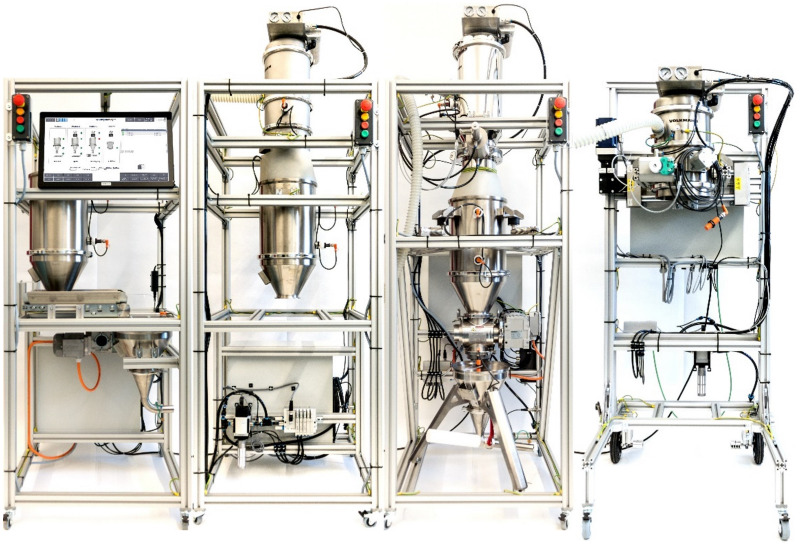
5.3.2 Reusing Environment from the Pool

Bulk Good Laboratory Plant (BGLP)

Ver. 2.3.0 (2023-02-22)

This module provides an RL environment of Bulk Good Laboratory Plant (BGLP).

The BGLP illustrates a smart production system with high flexibility and distributed control to transport bulk raw materials. One of the advantages of this laboratory test belt is the modularity in design, as depicted schematically below:



The BGLP consists of four modules, which are loading, storing, weighing, and filling stations respectively, and has conveying and dosing units as integral parts of the system. The interface between the modules is assembled via a mini hopper placed in the prior module. Then, the next module is fed by a vacuum pump, which operates in a discontinuous manner, before the goods are temporarily stored in a silo of the next module. The filling station has no silo because the main purpose of the station is to occupy the transport containers.

We utilize dissimilar actuators in modules 1-3 to transport the goods from the silo to the mini hopper. Module 1 utilizes a belt conveyor, that operates between 0 and 1800 rpm. Module 2 uses a vibratory conveyor, which can be completely switched on and off. Lastly, Module 3 utilizes a rotary feeder, that operates between 0 and 1450 rpm.

In the RL context, we consider the BGLP as a multi-agent system, where each actuator of the system is pointed as an agent or a player. The states information for each agent is the fill level of the prior reservoir and the fill level of the next reservoir.

Note: In this simulation, we assume that the actuator in Module D has a constant flow, which automatically matches the production demand in L/s. This parameter can be defined while setting up the BGLP environment. Therefore, 5 actuators are involved in this simulation instead of 6 actuators.

The BGLP environment can be imported via:

```
import mlpro.rl.pool.envs.bglp
```

Prerequisites Please install below package to use the MLPro's BGLP environment

- NumPy

General Information

Parameter	Value
Agents	5
Native Source	MLPro
Action Space Dimension	[5,]
Action Space Base Set	Real numbers, except Agent 3 uses Integer
Action Space Boundaries	[0,1]
State Space Dimension	[6,]
State Space Base Set	Real numbers
State Space Boundaries	[0,1]
Reward Structure	Individual reward for each agent

Note: You can change the configurations of the BGLP simulation, for instance, production demand (L/s), production target for batch operation (L), learning rates for reward calculation, and production scenario (batch or continuous). Batch production scenario refers to a process to satisfy a specific order in a sequence, thus the production target in L must be set. Meanwhile, continuous production scenario refers to a process to control a constant flow within a horizon, thus the production target (L) is not necessary and the target is fulfilled the production demand (L/s). The detailed explanations are available in the API reference section, see here.

Action Space

In this environment, we consider 5 actuators to be controlled. Thus, there are 5 agents and 5 joint actions because each agent requires an action. Every action is normalized within a range between 0 and 1, except for Agent 3. 0 means the minimum possible action and 1 means the maximum possible action. For Agent 3, the vibratory conveyor has a different character than other actuators, which mostly perform in a continuous manner. The vibratory conveyor can only be either fully switched-on or switched-off. Therefore the base set of action for Agent 3 is an integer (0/1). 0 means off and 1 means on.

Agent	Actuator	Station	Parameter	Boundaries
1	Conveyor Belt	A	rpm	450 … 1800
2	Vacuum Pump	B	on-duration (sec)	0 … 4.575
3	Vibratory Conveyor	B	on/off	0/1
4	Vacuum Pump	C	on-duration (sec)	0 … 9.5
5	Rotary Feeder	C	rpm	450 … 1450

State Space

The state information in the BGLP is the fill levels of the reservoirs. Each agent is always placed in between two reservoirs, e.g. between a silo and a hopper or vice versa. Therefore, each agent has two state information, which is shared with their neighbours. Every state is normalized within a range between 0 and 1. 0 means the minimum fill-level and 1 means the maximum fill-level.

Agent	State No.	Element	Station	Boundaries
1	1	Silo	A	0 … 17.42 L
	2	Hopper	A	0 … 9.1 L
2	1			
	2	Silo	B	0 … 17.42 L
3	1			
	2	Hopper	B	0 … 9.1 L
4	1			
	2	Silo	C	0 … 17.42 L
5	1			
	2	Hopper	C	0 … 9.1 L

Reward Structure

The reward structure is implemented according to this paper. You can also find the source code of the reward structure, here. The given reward is an individual scalar reward for each agent. To be noted, this reward function is more suitable for a continuous production scenario.

If you would like to implement a customized reward function, you can follow these lines of codes:

```
class MyBGLP(BGLP):

    def calc_reward(self):

        # Each agent has an individual reward
        if self.reward_type == Reward.C_TYPE_EVERY_AGENT:
            for actnum in range(len(self.acts)):
                acts = self.acts[actnum]
                self.reward[actnum] = 0
            return self.reward[:]

        # Overall reward
        elif self.reward_type == Reward.C_TYPE_OVERALL:
            self.overall_reward = 0
            return self.overall_reward
```

Cross Reference

- *API Reference*

Citation

If you apply this environment in your research or work, please *cite* us and the original paper.

Multi-Cartpole

Ver. 1.3.3 (2023-02-22)

This module provides an environment with multivariate state and action spaces based on the OpenAI Gym environment 'CartPole-v1'.

The multicartpole environment is an extension over the cartpole-v1 environment native to OpenAI Gym environments, where a cart is sliding over a flat surface and a pole is attached to the the middle of the cart at one end with frictionless turning joint. With multicartpole environment, we provide you the possibility to simulate multiple cartplole-v1 evriron-ments from gym. The goal of this environment is to maintain vertical position of the pole on the cart and stopping it

from falling over with the aid of pushing the cart to left or right. The multicartpole environment is visualized in the image below

This multicartpole environment can be imported via:

```
import mlpro.rl.pool.envs.multicartpole
```

The multicartpole environment can simulate 'n' number of cartpole-v1 environmments simultaneously, where the parameter 'n' can be set while instantiating the environment. The multicartpole environment can be instantiated as an mlpro environment class by including

```
env   = MultiCartPole(p_num_envs=3, p_logging=p_logging)
```

Screenshots

The multicartpole environment consists of 'n' number of internal cartpole-v1 gym environments running simultaneously. The environment starts with random state values and the agent computes actions based on the policy. As there are multiple sub-environments running simultaneously, MLPro offers agent object of type *multi-agent*, where a number of agents simultaneously simulate corresponding sub-environments. The agent computes an action value of 1 or 0 which refers to a left or right push respectively to the cart. These actions computed by the agents are processed in the corresponding gym sub environment through the MLPro to Gym wrapper functionality of MLPro. The output from the gym sub-environments is the set of new state values and the state flags including success, done, error. The new state of the multicartpole environment is a set of states of all internal sub-environments. The terminal state of multicartpole environment reaches when all the sub-environments are at a terminal state. The sub-environemnt which are terminal before the rest of the sub-environemnts the sub-environment is frozen until the rest of the sub-environemnts are frozen. For better understanding of the multi-cartpole environment and its implementation refer to this example implementation. Running this example implementation of multi-cartpole environment will produce visualisation as in the image below

Prerequisites

For the multicartpole environment to run properly please install the following python packages:

- NumPy
- Matplotlib
- OpenAI Gym

General Information

Parameter	Value per sub-environment
Agents	1
Native Source	MLPro
Action Space Dimension	[2,]
Action Space Base Set	Integer number
Action Space Boundaries	[0,1]
State Space Dimension	[4,]
State Space Base Set	Real number
Reward Structure	Overall reward

Action Space

Since the goal of the environment is to maintain the upright position of the cart, the cart is pushed to right or left for every run of the scenario. The action space for the multicartpole environment consists of push actions +1 and 0, denoting push towards right and left respectively. The size of the action space however is directly proportional to the number of child cartpole-v1 environments running within the multicartpole environment, for example a multicartpole environment for 3 sub environments has an action space of size 3.

Action	Value
Push Left	0
Push Right	1

Note: The action space for muticartpole environment consists of action spaces for all the sub-environments within the environment. Each of the action space actuates the assigned agent or muti-agent for the subenvironment. To know more about the the multi-agent class functionality native to MLPro refer to the appendix section.

State Space

The state space for the muticartpole environment returns state of every subenvironment within the environment including position of cart, velocity of cart, position of angel and the angular velocity of the pole. The states for a single cartpole environment running inside the multicartpole environment can be understood by the table below.

State	Boundaries
Cart Position	[-2.4,2.4]
Cart Velocity	
Angle of pole	[-0.209,0.209]
Angular Velocity of Pole	

The states of the muticartpole environment also return some flags giving additional information about the environment which includes

- Initial: The flag initial is set to true when an environment has been instantiated or has been reset after a successful or unsuccessful scenario run. The intital flag denotes that there are no adaptations made yet.
- Success: The success flag returns true whem a multicartpole environment has successfully run a scenario for a specified number of cycles. To run an environment sucessfully, the corresponding states of all the sub environments are within the boundaries as specified in the above table for the number of cycles specified. The scenario ends after the maximum number of cycles specified.
- Broken: The broken flag return true when the multicartpole environment is unsuccessful to run for the specified number of cycles. The broken state is set to true when the corresponding states of any sub-environments exceeds the state boundaries as mentionaed in the table above.
- Terminal: The flag terminal state defines end of an episode or end of a successful scenario of the multicartpole environment. The flag terminal is set to true when the either of the flags sucess or broken are true. The terminal flag is also set to true if the cycle extends the latency time or at the timeout. Once, the terminal flag is set to true, the environment terminates or resets based on the type of run and number of cycles.

More information about these state parameters related to the multi-cartpole environment can be found in the module descriptions.

Reward Structure

For multicartpole environment, an overall reward is awarded to the multi-agent. In a single sub-environment of cartpole-v1 a reward value of 1 is returned for every successful cycle run, keeping the states within boundaries. Subsequently, the reward awarded by the multi-cartpole environment is the weighted average of the rewards returned by every internal cartpole-v1 environment.

Cross Reference

- *Howto RL-AGENT-003: Run Multi-Agent with Own Policy*
- *Howto RL-AGENT-004: Train Multi-Agent with Own Policy*
- *API Reference*

Citation

If you apply this environment in your research or work, please *cite* us.

Grid World

Ver. 2.0.3 (2022-11-29)

This module provides an environment of customizable Gridworld.

Grid World is a very simple environment and suits to someone who just starts to understand Reinforcement Learning or Markov Decision Process.

In this Grid World environment, by default, the agent will be placed in a 2 dimensional grid world with the size of 8x8, tasked to reach the goal through position increment actions. The user can customize the dimension of the grid and decide the maximum number of steps. The agent is represented by number 1 and the goal is represented by number 2, where number 3 means that the agent is reaching the goal. In the latest version of Grid World, we provided the possibilities to set your own or random initial and/or goal positions. Moreover, there are two possible types of actions, such as continuous actions which can reached the goal in one-shot and discrete actions (only for 2-D grid world). The discrete actions consists of 'up', 'right', 'down', and 'left' (or 'north', 'east', 'south', and 'west') respectively. Here is the example of the grid world environment, by default and with random initial and goal states:

```
[[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 1],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0],
[0, 0, 0, 2, 0, 0, 0, 0],
[0, 0, 0, 0, 0, 0, 0, 0]]
```

At the moment, we have not incorporated any obstacles or walls, which will be added in the near future. The current implementation shows that if an action lead to a state outside the boundaries, then the state is back to the previous state.

This Grid World environment can be imported via:

```
import mlpro.rl.pool.envs.gridworld
```

Prerequisites

- NumPy

General Information

Parameter	Value
Number of agent	1
Native Source	MLPro
Action Space Dimension	Depends on the grid size, e.g. (8, 8), (8, 8, 8), etc.
Action Space Base Set	(Type 1) Real number
	(Type 2) Integer number
Action Space Boundaries	(Type 1) Depends on grid_size
	(Type 2) 0 to 3
State Space Dimension	Depends on the grid size
State Space Base Set	Integer number
State Space Boundaries	0 to 3
Reward Structure	Overall reward

Action Space

There are two types of actions that can be selected in the beginning of the training, such as continuous actions ('C_ACTION_TYPE_CONT') and discrete actions ('C_ACTION_TYPE_DISC_2D'). At the moment, the discrete action is limited to 2-dimensional grid world.

For continuous action, the action directly affects the location of the agent. The action is interpreted as increments towards the current location value. The dimension depends on the grid_size parameter. By default, there is a possibility to reach the target in one shot.

For discrete action, there are four possible actions that represented by number 0 to 3, as follows: Number '0' means 'up' or 'north'. Number '1' means 'right' or 'east'. Number '2' means 'down' or 'south'. Number '3' means 'left' or 'west'.

State Space

The state space is initialized from the grid_size parameter, which can be set up to however many dimension as needed. For example, the agent can be placed in a two dimensional world with a n x m size, three dimensional world with a n x m x p, or even more, for instance by setting `grid_size = (n,m)` or `grid_size = (n,m,p)`.

Additionally, the initial and goal position can be randomized or predefined.

Reward Structure

The default reward function is really simple and straight forward, where the reward is 1, if the agent reaches the goal. The reward is 1 minus the euclidean distance between goal states and current states, if the agent has not reached the goal yet.

```
reward = Reward(self.C_REWARD_TYPE)
rew = 1
euclidean_distance = np.linalg.norm(self.goal_pos-self.agent_pos)
if euclidean_distance !=0:
    rew = 1/euclidean_distance
if self.num_step >= self.max_step:
    rew -= self.max_step

reward.set_overall_reward(rew.item())
```

Cross Reference

- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *API Reference*

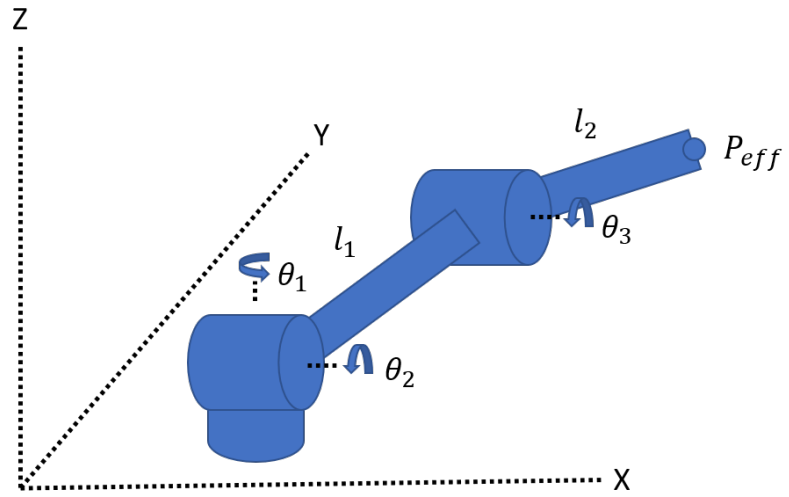
Citation

If you apply this environment in your research or work, please *cite* us.

Robot Manipulator on Homogeneous Matrix

Ver. 1.1.8 (2022-11-09)

This module provides an environment of a robot manipulator based on Homogeneous Matrix



This environment represents the robot manipulator in term of mathematical equations. The mathematical equations are based on rigid body transformation. In this case, the Homogeneous Transformation Matrix (HTM) is used for the structure. HTM is a matrix that contains both the translation rotation of a point with respect to some plane.

$$H = \begin{bmatrix} \mathbf{Rot} & \mathbf{Trans} \\ \mathbf{0} & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{Trans} \\ \mathbf{0} & 1 \end{bmatrix}}_{translation} \underbrace{\begin{bmatrix} \mathbf{Rot} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix}}_{rotation}$$

This robotinhtm environment can be imported via:

```
import mlpro.rl.pool.envs.robotinhtm
```

Prerequisites

- NumPy
- PyTorch

General Information

Parameter	Value
Agents	1
Native Source	MLPro
Action Space Dimension	[4,]
Action Space Base Set	Real number
Action Space Boundaries	[-pi,pi]
State Space Dimension	[6,]
State Space Base Set	Real number
State Space Boundaries	[-inf,inf]
Reward Structure	Overall reward

Action Space

By default, there are 4 action in this environment. The action space represents the angular velocity of each joint of the robot manipulator.

State Space

The state space consists of end-effector positions (x,y,z) of the robot manipulator and target positions (x,y,z).

Reward Structure

By default, the reward structures are shown in the following equation:

$$reward = -1 * \frac{distError}{initDist} - stepReward$$

Cross Reference

- *Howto RL-ENV-002: SB3 Policy on RobotHTM Environment*
- *Howto RL-MB-001: MBRL on RobotHTM Environment*
- *API Reference*

Citation

If you apply this environment in your research or work, please *cite* us and the original paper.

Double Pendulum

Ver. 2.3.2 (2023-03-09)

The Double Pendulum environment is an implementation of a classic control problem of Double Pendulum system. The dynamics of the system are based on the Double Pendulum implementation by Matplotlib. The double pendulum is a system of two poles, with the inner pole connected to a fixed point at one end and to outer pole at other end. The native implementation of Double Pendulum consists of an input motor providing the torque in either directions to actuate the system.

Note:

MLPro provides two implementations of Double Pendulum environment named DoublePendulumS4 and DoublePendulumS7.

- The DoublePendulumS4 environment is a basic implementation with four dimensional state space including angles and angular velocities of both the poles.

- The static 7 dimensional implementation of Double Pendulum environment in MLProis a seven dimensional state space with derived angular acceleration values and input torque. MLPro also provides a default reward strategy based on normalized state space and Euclidean Distances of the states.

The double pendulum environment can be imported via:

```
import mlpro.rl.pool.envs.doublependulum
```

The environment can be initialised with specifying the initial angles of both poles, masses of both poles, lenghts of poles, maximum torque value and scenario related parameters including step size and actuation step size. The initial positions of the poles refer to the position of the poles at the beginning of each RL episode, which can be set to 'up', 'down', 'random'. The default values for length and mass of each pole in the double pendulum are set to 1 and 1 respectively. The environment behaviour can be understood by running How To 20 in MLPro's sample implementation examples.

Note:

- The visualisation of the environment can be turned off by setting the visualize parameter in training/scenario initialisation to false

Screenshots

Prerequisites Please install below packages to use the MLPro's double pendulum environment

- NumPy
- Matplotlib
- SciPy

General Information

Parameter	Value
Agents	1
Native Source	MLPro
Action Space Dimension	1
Action Space Base Set	Real number
State Space Dimension	4 (for DoublePendulumS4), 7 (for DoublePendulumS7)
State Space Base Set	Real number
Reward Structure	Overall reward

Action Space The goal of the environment is to maintain the vertical position of both the poles. The inner pole is actuated by a motor, and thus the action space of Double Pendulum environment is a continuous variable ranging between the negative maximum torque and positive maximum torque, where positive torque refers to clockwise torque and vice versa. The max torque can be passed as a *parameter* in the initialisation of environment.

Parameter	Range
Torque	[-max_torque, max_torque]

State Space

The state space for the double pendulum environment returns state of poles in the system including angles of both poles, velocity of poles, angular acceleration of the poles. The states for double pendulum environment can be understood by the table below.

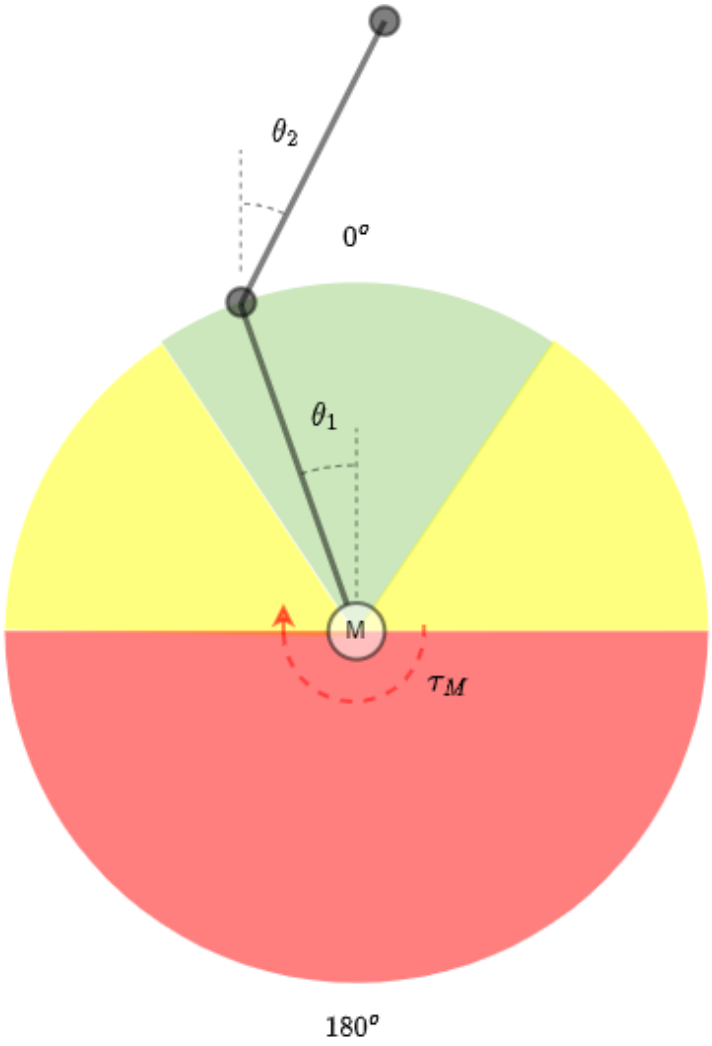
State	Description	Range	Unit	DoublePendulumS4	DoublePendulumS7
Theta 1	Angle of the inner pole	[-180, 180]	degrees	X	X
Omega 1	Angular velocity of inner pole	[-800, 800]	degrees per second	X	X
Alpha 1	Angular Acceleration of outer pole	[-6800, 6800]	degrees per second squared	-	X
Theta 2	Angle of the outer pole	[-180, 180]	degrees	X	X
Omega 2	Angular velocity of outer pole	[-950, 950]	degrees per second	X	X
Alpha 2	Angular acceleration of outer pole	[-9700, 9700]	degrees per second squared	-	X
Torque	Input torque to the inner pole	[-max torque, max torque]	Newton times meter	-	X

Note: The boundaries for the velocity and acceleration are highly influenced by the initital position of the arms and the current torque being actuated on the inner pole. These parameters are further dependent on the specific application, scenario or purpose of research.

Current implementation of DP environment in MLPro returns success when the current state of the environment is within a distance lesser than threshold distance from the goal state.

Reward Structure

The goal of the environment is to reach a complete vertical position for both the inner and outer pole, i.e. the goal state is given as vector $S_g = (0, 0, 0, 0, 0, 0)$. The environment delivers a continuous reward to the agent based on the new and old states of the environment. The environment is divided into three zones based on the position of the inner and outer pole.



As shown in the figure above, the three zones and the reward strategies corresponding to the zone are:

1. **Red Zone** : The swing up zone for angle of inner pole less than -90^o or more than $+90^o$. The reward signal in this zone maximizes the motion of the inner pole of the double pendulum.

$$r_a(t) = (|\theta_{1n(t-1))} - \theta_{1n(t)}|) + (|\theta'_{1n(t)} + \theta^n_{1n(t-1)}| - |\theta'_{1n(t-1)} + \theta^n_{1n(t-1)}|)$$

where,

$r_a(t)$ is reward at time step t,

θ_{1n} is normalized angle of inner pole

θ_{2n} is normalized angle of outer pole

2. **Yellow Zone** : Outer pole swing up zone for angle of inner pole more than -90^o or less than $+90^o$. The reward is based on the euclidean distance between new and old states, with 75% weight to the states of outer pole and 25% to that of inner pole.

$$r_b(t) = |s_{gb} - s_{b(t-1)}| - |s_{gb} - sb(t)|$$

where,

s_b is the state space in yellow zone as $(\theta_{1n}, \theta_{2n}, \theta'_{2n}, \theta''_{2n})$

s_{gb} is the goal state in Yellow zone, i.e. $(0, 0, 0, 0)$

3. **Green Zone** : Balancing zone for angle of either or both inner or outer pole more than -36^o or less than $+36^o$. The reward in this zone is proportional to the environments progress towards the goal state.

$$r_a(t) = |s_{gn} - s_{n(t-1)}| - |s_{gn} - s_{n(t)}|$$

where,

s_{gn} is the normalized goal state of the environment

s_n is the normalized state

Cross Reference

- *Howto RL-ENV-005: SB3 Policy on Double Pendulum Environment*
- *API Reference*

Citation

If you apply this environment in your research or work, please *cite* us.

Alternatively, you can also *reuse available environments from 3rd-party packages via wrapper classes* (currently available: OpenAI Gym or PettingZoo).

For reusing the 3rd packages, we develop a wrapper technology to transform the environment from the 3rd-party package to the MLPro-compatible environment. Additionally, we also provide the wrapper for the other way around, which is from MLPro Environment to the 3rd-party package. At the moment, there are two ready-to-use wrapper classes. The first wrapper class is intended for OpenAI Gym and the second wrapper is intended for PettingZoo. The guide to using the wrapper classes is step-by-step explained in our how-to files, as follows:

(1) *OpenAI Gym to MLPro*,

(2) *MLPro to OpenAI Gym*,

(3) *PettingZoo to MLPro*, and

(4) *MLPro to PettingZoo*.

5.4 Agents

In RL, an agent is an autonomous entity that interacts with an environment, receiving rewards for performing certain actions and updates its behavior based on that feedback. The agent's goal is to learn a policy that maximizes its cumulative reward over time.

From a scientific perspective, the agent is typically modeled as a decision-making system that maps states of the environment to actions through a policy. The policy can be deterministic or probabilistic and can be learned through various RL algorithms such as Q-Learning, SARSA, or Policy Gradient methods. The agent's performance is evaluated using metrics such as reward, cumulative reward, and value functions. Overall, an agent in RL provides an algorithm that makes decisions and learns from experience to optimize its performance in a given task.

MLPro-RL supplies a special agent model landscape, which covers different RL scenarios including a simple single-agent RL, a multi-agent RL, and model-based agents with an optional action planner. For the multi-agent RL, the structure is constructed by assigning multiple single agents in a group. The main component of each single-agent (either single-agent or multi-agent RL) is the policy. The basic class of the policy is inherited from the ML Model of basic MLPro functionality and extended by the RL-related function of the action calculation. The users can inherit the basic class of the policy to implement their *own custom algorithms* or simply use algorithms from third-party packages via *wrapper classes*. The other possibility would be *importing algorithms from the pool object*. For an overview, the simplified class diagram of agents in MLPro is described below.

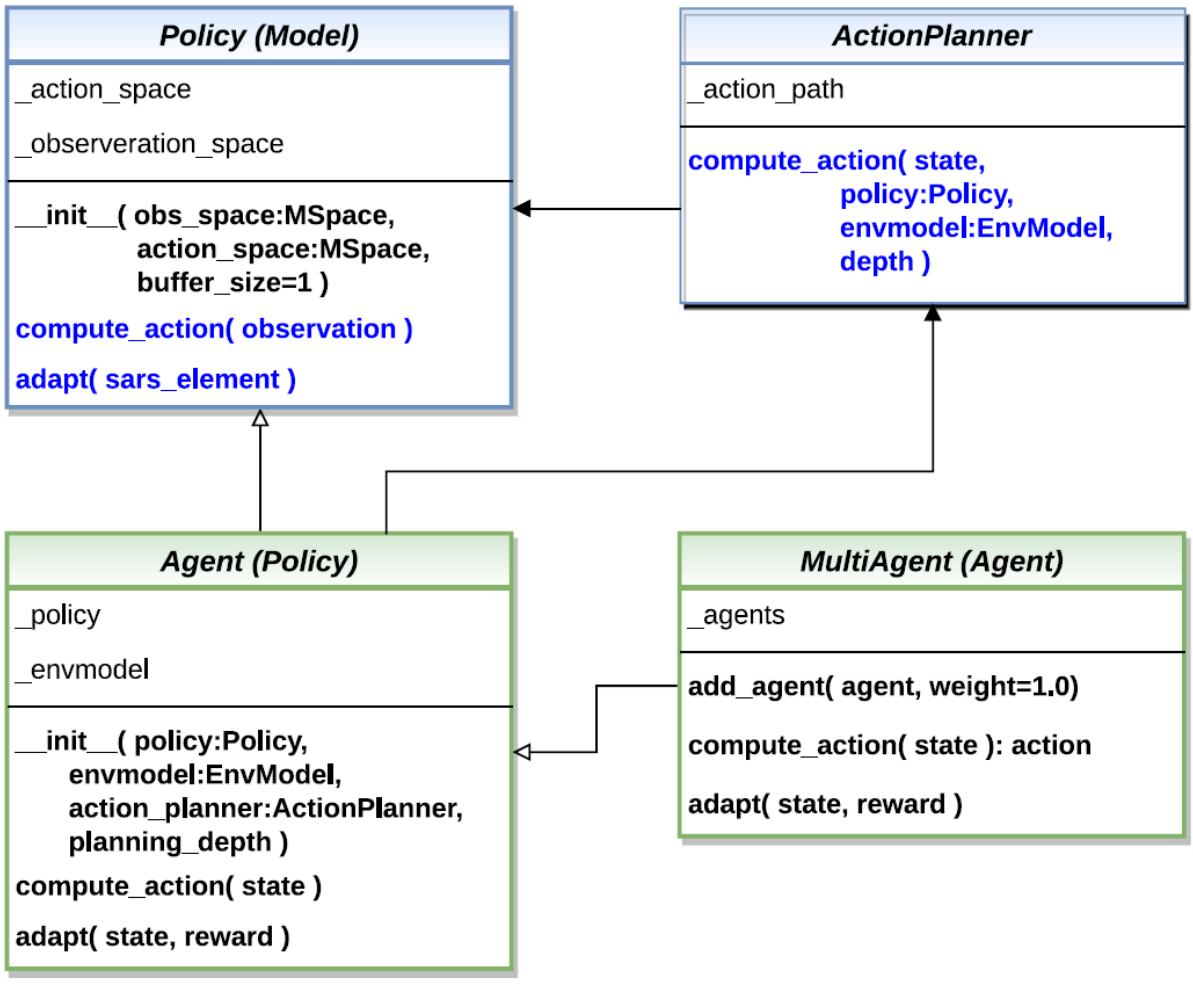


Fig. 3: This figure is taken from MLPro 1.0 paper.

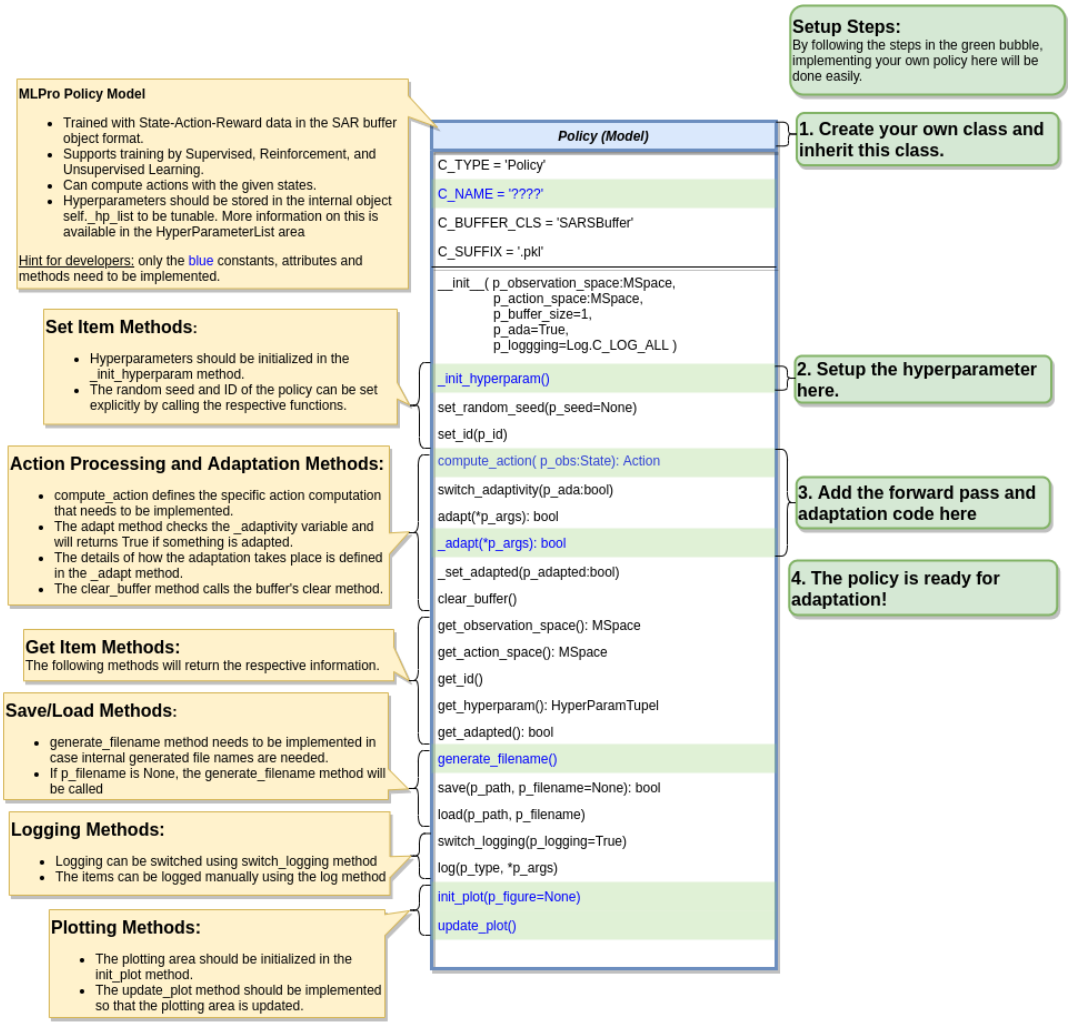
Moreover, an environment model (known as *EnvModel class*) can be supplemented to a single agent, i.e. for model-based RL cases. This class can be used for model-based learning, which learns the behaviour or dynamics of the environment. Another possible extension of the model-based agent is an action planner. Action planner uses the environment model (or EnvModel) to plan the next action by predicting the output on a certain horizon. An example of action planner algorithms is *Model Predictive Control (MPC)*, which is also provided in MLPro.

Additionally, you can find more comprehensive explanations of agents in MLPro-RL including a sample application on controlling a UR5 Robot in this paper: MLPro 1.0 - Standardized Reinforcement Learning and Game Theory in Python.

Here are some subsections of the agent model landscape of MLPro-RL, which might be interesting for the users:

5.4.1 Custom Policies

- **Policy Creation**



Creating a custom RL policy that satisfies the MLPro interface is straightforward. First of all, the users need to inherit a base Policy class. Then, the users can develop their custom policies by fulfilling at least 2 main functions, namely **compute_action** and **_adapt**, as shown in the following code. The compute action method (**compute_action**) is a function to calculate an action in the current state. Meanwhile, the adapt method (**_adapt**) is a function to optimize the policy according to past experience.

```
from mlpro.rl.models import *

class MyPolicy (Policy):
    """
    Creates a policy that satisfies mlpro interface.
    """

    C_NAME      = 'MyPolicy'

    def compute_action(self, p_state: State) -> Action:
        """
        Specific action computation method to be redefined.

        Parameters:
            p_state       State of environment

        Returns:
            Action object
        """
        ....


    def _adapt(self, *p_args) -> bool:
        """
        Adapts the policy based on State-Action-Reward (SAR) data that will␣
↪be expected as a SAR
        buffer object. Please call super-method at the beginning of your␣
↪own implementation and
        adapt only if it returns True.

        Parameters:
            p_arg[0]            SAR Buffer object
        """

        if not super().adapt(*p_args): return False

        ....
        return True
```

Hyperparameters of the policy should be stored in the internal object **self._hp_list**, so that they can be tuned from outside. The hyperparameter initialization method (**_init_hyperparam**) can be used in this case. To set up a hyperparameter space, please refer to our *how-to file*.

- **Policy from Third Party Packages**

  Alternatively, the user can also apply algorithms from Stable Baselines 3 by using the developed relevant wrapper for the integration between third-party packages and MLPro. For more information, please click *here*.

- **Algorithm Checker**

  A test script using a unit test to check the developed policies will be available soon!

5.4.2 Policy Pool

Random Action Generator

Ver. 1.0.4 (2022-11-02)

This module providew random genarator for multi purposes, e.g. testing environment, etc..

General Information

A random generator for a specific action space with defined boundaries is an algorithm that generates random values within a specific range or set of constraints. This type of random generator is often used for testing environments, generate sample data for model-based learning, and many more. The random generator can be attach to an RL agent and will detect the action space of the agent as well as the boundaries. The action space refers to the set of possible actions of the agent, and the boundaries define the range of values within which the actions must lie. Then, an action is randomly computed by the generator using uniform distribution.

For example, if the action space of an agent is defined as a joint velocity of a robot, the boundaries may be defined as the minimum and maximum velocity of the joint. The random generator would then generate random values within these boundaries that represent the possible velocity of the robot. The specific method used to generate random values within the action space boundaries depends on the type of data being generated and the desired properties of the generated data.

This Random Action Generator policy can be imported via:

```
import mlpro.rl.pool.policies.randomgenerator
```

Cross Reference

- API Reference

Citation

If you apply this environment in your research or work, please *cite* us.

Model Predictive Control (MPC)

Ver. 1.1.1 (2023-02-04)

This module provides a default implementation of model predictive control (MPC).

Prerequisites

- NumPy

General Information

We introduce an MPC method as action planner in the model-based RL territory. Monte Carlo MPC is a control algorithm that uses a Monte Carlo simulation-based approach to generate control actions for a dynamic system. It is a type of MPC, which is a well-established control algorithm that predicts the future behavior of a system based on a mathematical model and uses this information to generate optimal control actions.

In this Monte Carlo MPC, instead of using a single action prediction of the future system behavior, a large number of simulations are run, each with different random actions variations in the model parameters. Based on these trials, Monte Carlo MPC generates control actions that minimize a defined cost function, taking into account the control objectives. The control actions are updated at each time step based on new measurements of the system state.

Monte Carlo MPC is particularly useful in situations where the system is uncertain, unpredictable, or subject to significant external disturbances, as it allows for a probabilistic treatment of these uncertainties. It has found applications in a variety of fields, including autonomous systems, robotics, and process control.

This MPC policy can be imported via:

```
import mlpro.rl.pool.actionplanner.mpc
```

Multiprocessing has also been incorporated into MPC, which allows parallel computations. Depending on the number of planning horizon, but we believe that this reduces the training time massively.

Cross Reference

- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *API Reference*

Citation

If you apply this environment in your research or work, please *cite* us and the original paper.

5.4.3 Model-Based Agents

Model-Based Agents have a dissimilar learning target as Model-Free Agents, whereas learning the environment model is not required in the model-free RL. An environment model can be incorporated into a single agent, see EnvModel for an overview. Then, this model learns the behaviour and dynamics of the environment. After learning the environment, the model is optimized to be able to accurately predict the output states, rewards, or status of the environment with respect to the calculated actions. As a result, if the predictions of the subsequent state and reward diverge too far from the actual values of the environment, the environment model itself is incorporated into the agent's adaptation process and is always retrained. An adaptation in the environment model necessitates an adaptation in the policy. The foundation for this is an internal episodic training of the policy in interaction with the environment model.

After having a model that can accurately predict the behaviour of the environment, the single agent is optionally extended as an action planner. The action planner can be used by the environment to plan the next actions, e.g. using Model Predictive Control. In MLPro-RL, we have also provided a base class for ActionPlanner, where only the action planner method (**_plan_action**) and an optional custom setup method (**_setup**) are needed to be adjusted, as shown below:

```
from mlpro.rl.models import *

class MyActionPlanner (ActionPlanner):
    """
    Creates an action planner that satisfies mlpro interface.
    """

    C_NAME      = 'MyActionPlanner'

    def _setup(self):
        """
        Optional custom setup method.
        """

        pass


    def _plan_action(self, p_obs: State) -> SARSBuffer:
        """
        Custom planning algorithm to fill the internal action path (self._action_path).␣
↪Search width
```

(continues on next page)

(continued from previous page)

```
        and depth are restricted by the attributes self._width_limit and self._
↪prediction_horizon.
        Parameters
        ----------
        p_obs : State
            Observation data.
        Returns
        -------
        action_path : SARSBuffer
            Sequence of SARSElement objects with included actions that lead to the best␣
↪possible reward.
        """

        raise NotImplementedError
```

Environment Model (EnvModel)

To set up environment model, the adaptive function needs to created first. In this case, our adaptive function will predict the next state of the environment based on provided action. After that, we need to create another class that is inherited from the actual environment module and **EnvModel**, in this case RobotHTM. For now, we only use the state transition model. The reward, success and broken model are taken from the original environment module.

```
from mlpro.rl.model_env import EnvModel
from mlpro.rl.pool.envs.robotinhtm import RobotHTM

class OurEnvModel(RobotHTM, EnvModel):
    C_NAME = "Our Env Model"

    # Put necessary input argument in initialization
    def __init__(
        self,
        p_num_joints=4,
        p_target_mode="Random",
        p_ada=True,
        p_logging=False,
    ):

        # Initialize the actual environment to get all environment functionalities, such␣
↪as
        # _simulate_reaction, _reset, _compute_reward, _compute_broken and _compute_
↪success
        RobotHTM.__init__(self, p_num_joints=p_num_joints, p_target_mode=p_target_mode)

        # Setup Adaptive Function
        afct_strans = AFctSTrans(
            OurStatePredictor,
            p_state_space=self._state_space,
            p_action_space=self._action_space,
            p_threshold=1.8,
            p_buffer_size=20000,
            p_ada=p_ada,
            p_logging=p_logging,
```

(continues on next page)

(continued from previous page)

```
        )

        # In this case set only p_afct_strans, which tells the module to use
        # _simulate_reaction from the adaptive function instead of from the actual␣
↪environment
        # Set to None to use function such as compute_reward, compute_broken and compute_
↪success
        # from the actual environment
        EnvModel.__init__(
            self,
            p_observation_space=self._state_space,
            p_action_space=self._action_space,
            p_latency=timedelta(seconds=self.dt),
            p_afct_strans=afct_strans,
            p_afct_reward=None,
            p_afct_success=None,
            p_afct_broken=None,
            p_ada=p_ada,
            p_logging=p_logging,
        )

        self.reset()
```

Cross Reference

- *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*
- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *Howto RL-MB-003: MBRL on RobotHTM Environment*
- *MLPro-SL*

5.4.4 Multi-Agents

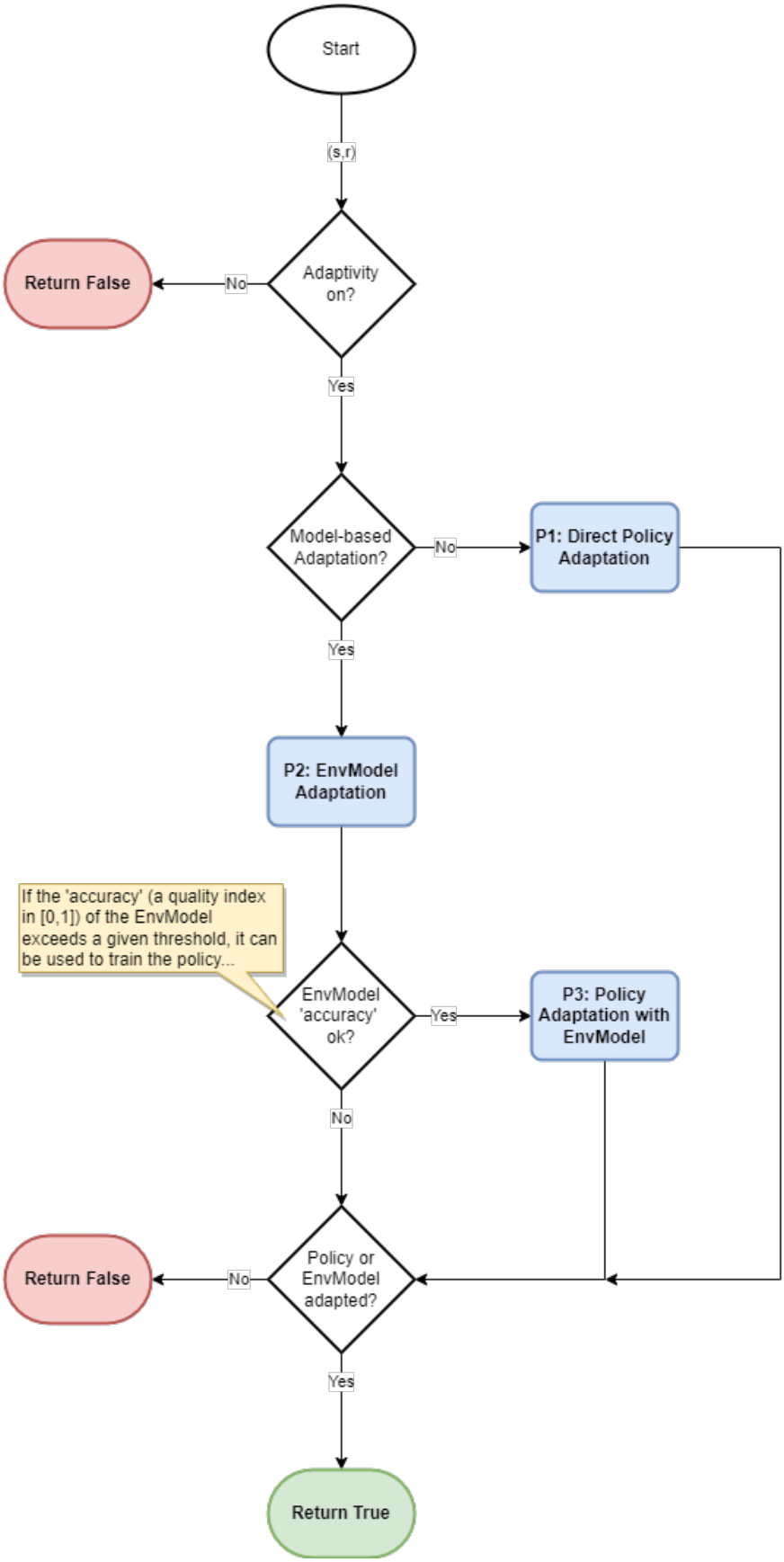
In Reinforcement Learning, multi-agent refers to a scenario where multiple independent agents interact with each other and with their environment in an attempt to accomplish a common goal or optimize their own reward. In this setting, the behavior of each agent not only depends on its own actions and observations, but also on the actions and observations of other agents, leading to a complex, dynamic and interactive decision-making process. The scientific study of multi-agent reinforcement learning involves modeling the interdependence and cooperation/competition among agents, and developing algorithms for agents to learn and adapt to the environment effectively.

MLPro-RL is not only compatible with single-agent RL but also multi-agent RL, where the extent of the agent landscape is completed by the multi-agent model. It is compatible with single-agent but does not have its own policy. Instead, it is utilized to combine and control any quantity of single agents that together control the action calculation. Every single agent in this situation interacts with a separate portion of the surrounding multi-observation agents and action space. Multi-agent interactions take place in appropriate contexts that support the scalar reward per agent reward type. These are native applications that incorporate the MLPro environment template or PettingZoo environments that may be incorporated using the corresponding *wrapper class* offered by MLPro.

Cross Reference

- *Howto RL-AGENT-004: Train Multi-Agent with Own Policy*
- *MLPro-RL: Training*

The following flowchart describes the adaptation procedure of an agent. In the beginning, the loop checks whether it is model-based RL or model-free RL. If it is a model-free RL, then the loop is jumped to a direct policy adaptation. Then, the current step ended after the policy adaptation. Meanwhile, in the model-based RL, the EnvModel is first adapted, and then the loop checks whether the accuracy of the EnvModel exceeds a given threshold. This activity is to make sure that the EnvModel is accurate enough for policy adaptation. If the accuracy is higher than the threshold, then the policy adaptation takes place with EnvModel. Otherwise, the current step is ended without any policy adaptations.


Start
(s,r)
Adaptivity on?
No
Return False
Yes
Model-based Adaptation?
No
P1: Direct Policy Adaptation
Yes
P2: EnvModel Adaptation
If the 'accuracy' (a quality index in [0,1]) of the EnvModel exceeds a given threshold, it can be used to train the policy...
EnvModel 'accuracy' ok?
Yes
P3: Policy Adaptation with EnvModel
No
Policy or EnvModel adapted?
No
Return False
Yes
Return True

5.5 Scenarios

A scenario in reinforcement learning refers to a specific problem or task that the agent is trying to learn how to solve. A scenario defines the environment in which the agent operates, including the state space, the action space, the reward function, and the transition dynamics.

In RL, the agent interacts with the environment over a sequence of time steps. At each time step, the agent receives an observation of the current state of the environment and selects an action. The environment then transitions to a new state and returns a reward signal to the agent.

The scenario provides the agent with a set of goals to be achieved, and the reward function quantifies how well the agent is doing in terms of achieving these goals. The reward function can be designed to encourage certain behaviors, such as reaching a specific target state, or penalize certain behaviors, such as taking actions that lead to a state of low reward.

The scenario also defines the state and action spaces, which are the sets of all possible states and actions that the agent can experience and take, respectively. The transition dynamics describe how the environment changes in response to the agent's actions.

Overall, a scenario in RL defines the problem that the agent is trying to solve, and provides the necessary information for the agent to learn a policy that maps states to actions and maximizes the cumulative reward signal.

In MLPro-RL, a class **RLScenario** inherits the functionality from class **Scenario** in the basic function level, where the **RLScenario** class combines RL agents and an environment into an executable unit.

One of the MLPro's features is enabling the user to apply a template class for an RL scenario consisting of an environment and agents. Moreover, the users can create either a single-agent scenario or a multi-agent scenario in a simple manner by inheriting **RLScenario** base class and redefining its **_setup** function.

Cross Reference

- *MLPro-RL: Training*
- *Howto RL-AGENT-001: Run an Agent with Own Policy*
- *Howto RL-AGENT-003: Run Multi-Agent with Own Policy*

5.6 Training and Tuning

In RL, the agent and the environment interact over a sequence of time steps. At each time step, the agent receives an observation of the current state of the environment and selects an action. The environment then transitions to a new state and returns a reward signal to the agent. This process continues until some terminal state is reached.

The agent uses the observed state-action-reward sequences to update its policy, either through model-based methods that estimate the underlying dynamics of the environment, or model-free methods that directly estimate the value or the policy. The policy is used to select actions in subsequent interactions with the environment, allowing the agent to learn from its mistakes and improve over time.

In MLPro-RL, a class **RLTraining** inherits the functionality from class **Training** in the basic function level, where the **RLTraining** class are used for training and hyperparameter tuning of RL agents. We implement episodic training algorithms and make the corresponding extended training data and results as well as the trained agents available in the file system. In this RL training, we always start with a defined random initial state of the environment and evaluate at each time step whether one of the following three categories is satisfied,

(1) **Event Success**: This means that the defined target state is reached and the actual episode is ended.

(2) **Event Broken**: This means that the defined target state is no longer reachable and the actual episode is ended.

(3) **Event Timeout**: This means that the maximum training cycles for an episode are reached and the actual episode is ended.

If none of the events is satisfied, then the training continues. The goal of the training is to maximize the score of the repetitive evaluations. In this case, a *stagnation detection functionality* can be incorporated to avoid a long training time without any more improvements. The training can be ended, once the stagnation is detected. For more information, you can read Section 4.3 of MLPro 1.0 paper.

In MLPro-RL, we simplify the process of setting up an RL scenario and training for both single-agent and multi-agent RL, as shown below:

- **Single-Agent Scenario Creation**

```
from mlpro.rl.models import *

class MyScenario(Scenario):

    C_NAME      = 'MyScenario'

    def _setup(self, p_mode, p_ada:bool, p_logging:bool):
        """
        Here's the place to explicitely setup the entire rl scenario. Please bind your env to
        self._env and your agent to self._agent.

        Parameters:
            p_mode              Operation mode of environment (see Environment.C_MODE_*)
            p_ada               Boolean switch for adaptivity of agent
            p_logging           Boolean switch for logging functionality
        """

        # Setup environment
        self._env    = MyEnvironment(....)

        # Setup an agent with selected policy
        self._agent = Agent(
            p_policy=MyPolicy(
             p_state_space=self._env.get_state_space(),
             p_action_space=self._env.get_action_space(),
             ....
             ),
             ....
         )

# Instantiate scenario
myscenario  = MyScenario(p_scenario=myscenario, ....)

# Train agent in scenario
training    = Training(....)
training.run()
```

- **Multi-Agent Scenario Creation**

```
from mlpro.rl.models import *

class MyScenario(Scenario):

    C_NAME      = 'MyScenario'

    def _setup(self, p_mode, p_ada:bool, p_logging:bool):
        """
        Here's the place to explicitely setup the entire rl scenario. Please bind your env to
        self._env and your agent to self._agent.

        Parameters:
            p_mode              Operation mode of environment (see Environment.C_MODE_*)
            p_ada               Boolean switch for adaptivity of agent
            p_logging           Boolean switch for logging functionality
        """

        # Setup environment
        self._env    = MyEnvironment(....)

        # Create an empty mult-agent
        self._agent     = MultiAgent(....)

        # Add Single-Agent #1 with own policy (controlling sub-environment #1)
        self._agent.add_agent = Agent(
            self._agent = Agent(
                p_policy=MyPolicy(
                 p_state_space=self._env.get_state_space().spawn[....],
                 p_action_space=self._env.get_action_space().spawn[....],
                 ....
                 ),
                 ....
             ),
             ....
         )

        # Add Single-Agent #2 with own policy (controlling sub-environment #2)
        self._agent.add_agent = Agent(....)

        ....

# Instantiate scenario
myscenario  = MyScenario(p_scenario=myscenario, ....)

# Train agent in scenario
training    = Training(....)
training.run()
```

Cross Reference

- A sample application video of MLPro-RL on a UR5 robot
- *Howto RL-AGENT-002: Train an Agent with Own Policy*
- *Howto RL-AGENT-004: Train Multi-Agent with Own Policy*
- *Howto RL-AGENT-011: Train and Reload Single Agent (Gym)*
- *Howto RL-AGENT-021: Train and Reload Single Agent (MuJoCo)*
- *Howto RL-ATT-001: Train and Reload Single Agent using Stagnation Detection (Gym)*
- *Howto RL-ATT-002: Train and Reload Single Agent using Stagnation Detection (MuJoCo)*
- *Howto RL-MB-001: Train and Reload Model Based Agent (Gym)*
- *Howto RL-MB-002: MBRL with MPC on Grid World Environment*
- *MLPro-BF-ML: Training and Tuning*

5.7 3rd Party Support

MLPro allows the user to reuse widely-used 3rd-party packages and integrate them to the MLPro interface and also the other way around via wrapper classes. Therefore, the user is free to select an environment and/or a policy from the 3rd-party packages and the native MLPro-RL. It is also possible to combine an environment and a policy from different packages.

At the moment, we have five ready-to-use wrapper classes related to RL from 3rd-party packages to MLPro and two wrapper classes from MLPro to 3rd-party packages, such as:

No	Wrapper Class	Origin	Target	Wrapped RL Components
1	WrEnvGYM2MLPro	OpenAI Gym	MLPro	RL Environments
2	WrEnvMLPro2GYM	MLPro	OpenAI Gym	RL Environments
3	WrEnvPZOO2MLPro	PettingZoo	MLPro	Multi-Agent RL Environments
4	WrEnvMLPro2PZoo	MLPro	PettingZoo	Multi-Agent RL Environments
5	WrPolicySB32MLPro	StableBaselines3	MLPro	Off-Policy and On-Policy RL Algorithms

Moreover, wrapper classes for hyperparameter tuning by *Hyperopt* and *Optuna* can also be incorporated to your RL training.

5.7.1 RL Environment: OpenAI Gym to MLPro

Here is the wrapper class to convert RL Environment from OpenAI Gym to MLPro. The implementation is pretty simple and straightforward. The user can call the wrapper class while setting up an environment, as follows:

```
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
import gym

p_gym_env = gym.make('CartPole-v1', new_step_api=True, render_mode=None)
self._env = WrEnvGYM2MLPro(p_gym_env, p_logging=True)
```

Cross Reference

- *Howto RL-AGENT-001: Run an Agent with Own Policy*
- *API Reference*

5.7.2 RL Environment: MLPro to OpenAI Gym

Here is the wrapper class to convert RL Environment from MLPro to OpenAI Gym. The implementation is pretty simple and straightforward. The user can call the wrapper class while setting up an environment, as follows:

```
from mlpro.wrappers.openai_gym import WrEnvMLPro2GYM
from mlpro.rl.pool.envs.gridworld import GridWorld

mlpro_env = GridWorld(p_logging=Log.C_LOG_ALL)
env = WrEnvMLPro2GYM(mlpro_env, p_state_space=None, p_action_space=None, p_new_step_
↪api=True)
```

Cross Reference

- *Howto RL-WP-001: MLPro to OpenAI Gym*
- *API Reference*

5.7.3 RL Environment: PettingZoo to MLPro

Here is the wrapper class to convert RL Environment from PettingZoo to MLPro. The implementation is pretty simple and straightforward. The user can call the wrapper class while setting up an environment, as follows:

```
from pettingzoo.butterfly import pistonball_v6
from mlpro.wrappers.pettingzoo import WrEnvPZOO2MLPro

p_zoo_env = pistonball_v6.env()
self._env = WrEnvPZOO2MLPro(p_zoo_env, p_logging=True)
```

Cross Reference

- *Howto RL-WP-003: Run Multi-Agent on PettingZoo Environment*
- *API Reference*

5.7.4 RL Environment: MLPro to PettingZoo

Here is the wrapper class to convert RL Environment from MLPro to PettingZoo. The implementation is pretty simple and straightforward. The user can call the wrapper class while setting up an environment, as follows:

```
from mlpro.wrappers.pettingzoo import WrEnvMLPro2PZoo
from mlpro.rl.pool.envs.bglp import BGLP

mlpro_env = BGLP(p_logging=Mode.C_LOG_ALL)
env = WrEnvMLPro2PZoo(mlpro_env, p_num_agents=5, p_state_space=None, p_action_
↪space=None).pzoo_env
```

Cross Reference

- *Howto RL-WP-002: MLPro to PettingZoo*
- *API Reference*

5.7.5 RL Policy: StableBaselines3 to MLPro

Here is the wrapper class to convert RL Environment from StableBaselines3 to MLPro. The wrapper provides both the On-Policy and Off-Policy from StableBaselines3. The implementation is pretty simple and straightforward. The user can call the wrapper class while setting up an environment, as follows:

```
from stable_baselines3 import PPO
from mlpro.rl.wrappers import WrPolicySB32MLPro

class MyScenario(Scenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada, p_logging):
        gym_env     = gym.make('CartPole-v1')
        self._env   = WrEnvGYM2MLPro(gym_env, p_logging=False)

        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=5,
            env=None,
            _init_setup_model=False,
            device="cpu")

        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_logging=p_logging)

        return Agent(
            p_policy=policy_wrapped,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_logging=p_logging
        )
```

Cross Reference

- *Howto RL-WP-004: Train an Agent with SB3*
- *Howto RL-WP-005: Validation SB3 Wrapper (On-Policy)*
- *Howto RL-WP-006: Validation SB3 Wrapper (Off-Policy)*
- *API Reference*

CHAPTER

SIX

MLPRO-GT - GAME THEORY

6.1 Overview

Game Theory (GT) is a well-known branch of economic studies as a theoretical approach to modelling the strategic interaction between multiple individuals or players in a specific circumstance. GT can also be adopted in the scientific and engineering area, for instance, to optimize decision-making processes in a strategic setting. Moreover, GT has successfully solved Multi-Agent Reinforcement Learning (MARL) problems. If you would like to know more about the corporation between GT and MARL, you can have a look at these papers:

(1) Self-optimization using a State-based Potential Game approach and

(2) Potential game-based distributed optimization of a production unit.

MLPro-GT is developed as a sub-framework for the cooperative GT approach to solving MARL problems by inheriting a handful of main functionalities of MLPro-RL, such as the environment model, the agent model, and the environment-agent interaction model. This sub-framework focuses on the cooperative GT approach to Markov games. A Markov game contains a group of independent players that make decisions simultaneously, see the figure below for an overview.

If you are interested to utilize MLPro-GT, you can easily access the GT modules, as follows:

```
from mlpro.gt import *
```

Additionally, you can find more comprehensive explanations of MLPro-GT including a sample application and difference with a native RL approach in this paper: MLPro 1.0 - Standardized Reinforcement Learning and Game Theory in Python. The simplified diagram below shows the architecture of MLPro-GT, where MLPro-GT serves as a child package and MLPro-RL is its parent package.

Learn more

- *Getting started with MLPro-GT*

Cross Reference

- *Related Howtos*
- *API Reference: MLPro-GT*
- *API Reference: MLPro-GT Pool of Objects*
- MLPro 1.0 Paper
- MLPro GitHub

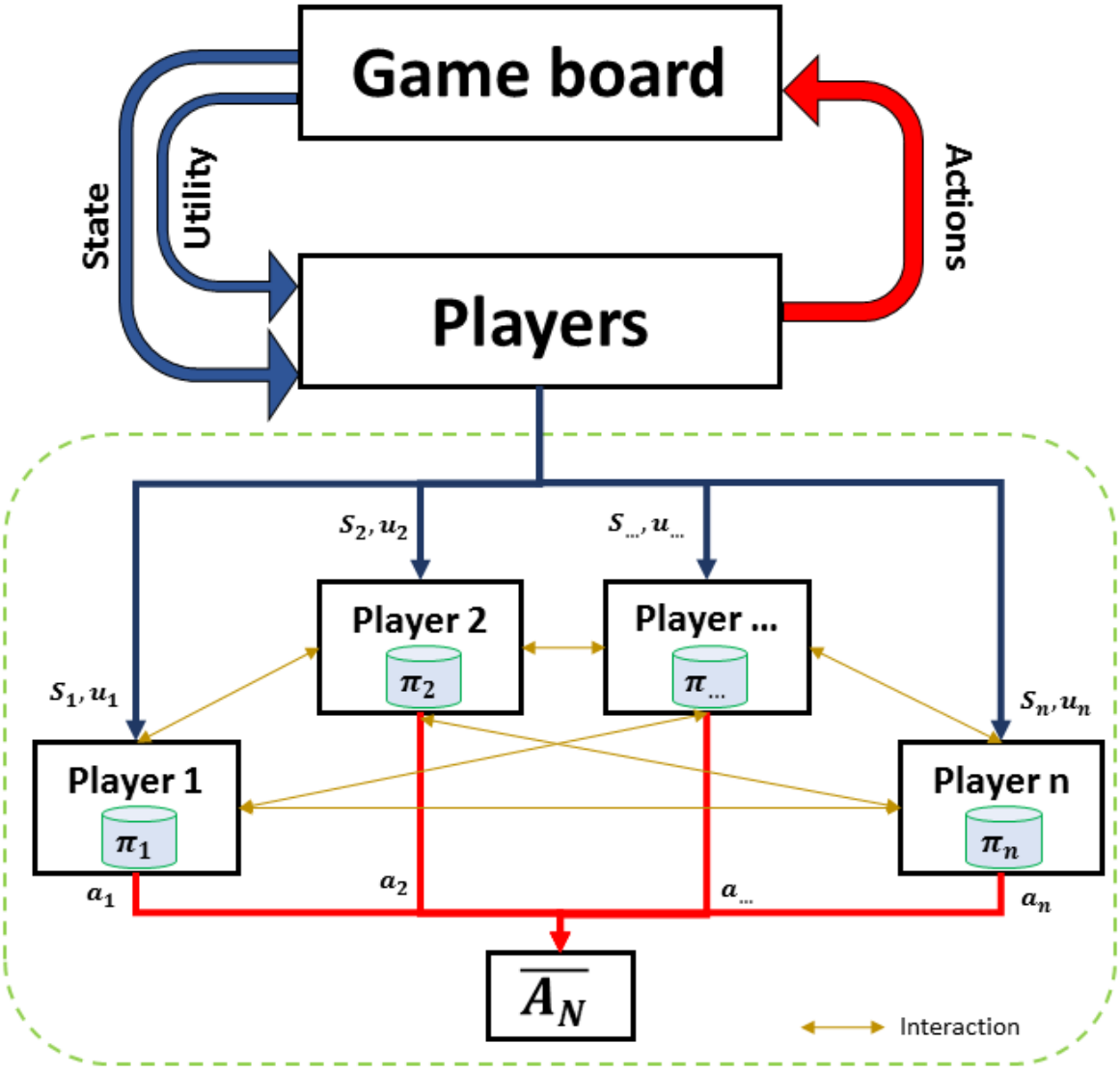
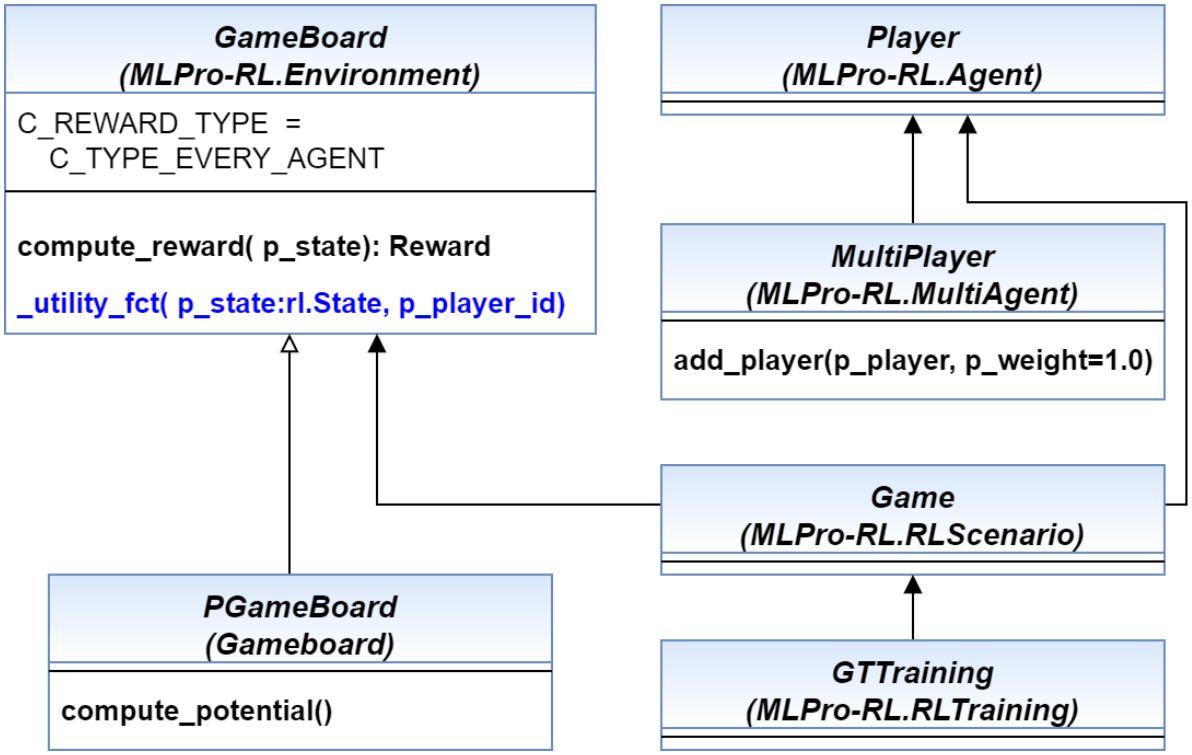


Fig. 1: This figure is taken from MLPro 1.0 paper.



6.2 Getting Started

Here is a concise series to introduce all users to the MLPro-GT in a practical way, whether you are a first-timer or an experienced MLPro user.

If you are a first-timer, then you can begin with **Section (1) What is MLPro?**.

If you have understood MLPro but not the game theoretical approach in the engineering field, then you can jump to **Section (2) What is Game Theory?**.

If you have experience in both MLPro and game theory, then you can directly start with **Section (3) What is MLPro-GT?**.

After following the below step-by-step guideline, we expect the user understands the MLPro-GT in practice and starts using MLPro-GT.

1. What is MLPro?

If you are a first-time user of MLPro, you might wonder what is MLPro. Therefore, we recommend initially starting with understanding MLPro by checking out the following steps:

(a) *MLPro: An Introduction*

(b) introduction video of MLPro

(c) *installing and getting started with MLPro*

(d) MLPro paper in Software Impact journal

2. What is Game Theory?

If you have not dealt with game theory for engineering applications, we recommend starting to understand at least the basic concept of game theory. There are plenty of references, articles, papers, books, or videos on the internet that explains the game theory. But, for deep understanding, we recommend you to read the book from Dario Bauso, which is Game Theory with Engineering Applications.

3. What is MLPro-GT?

We expect that you have a basic knowledge of MLPro and game theory. Therefore, you need to understand the overview of MLPro-GT by following the steps below:

(a) *MLPro-GT introduction page*

(b) Section 5 of MLPro 1.0 paper

4. Understanding Game Board and Player in MLPro-GT

First of all, it is important to understand the structure of a game board in MLPro-GT, which can be found on *this page*.

In reinforcement learning, we have two types of agents, such as a single-agent RL or a multi-agent RL. Both of the types are covered by MLPro-RL. Meanwhile, in MLPro-GT, we focus on a multi-player GT because there are no significant advantages of using game theory for single-player. To understand a player in MLPro-GT, you can visit *this page*.

Then, you can start following some of our howto files and a sample application that shows how to run and train multi-player with their own policy, as follows:

(a) *Howto GT-001: Run Multi-Player with Own Policy*

(b) *Howto GT-002: Train Multi-Player*

(c) Section 6.2 of MLPro 1.0 paper

5. Additional Guidance

After following the previous steps, we hope that you could practice MLPro-GT and start using this subpackage for your GT-related activities. For more advanced features, we highly recommend you to check out the following howto files:

(a) *Howto RL-HT-001: Hyperopt*

(b) *Howto RL-HT-002: Optuna*

(c) *Howto RL-ATT-001: Stagnation Detection*

6.3 Game Boards

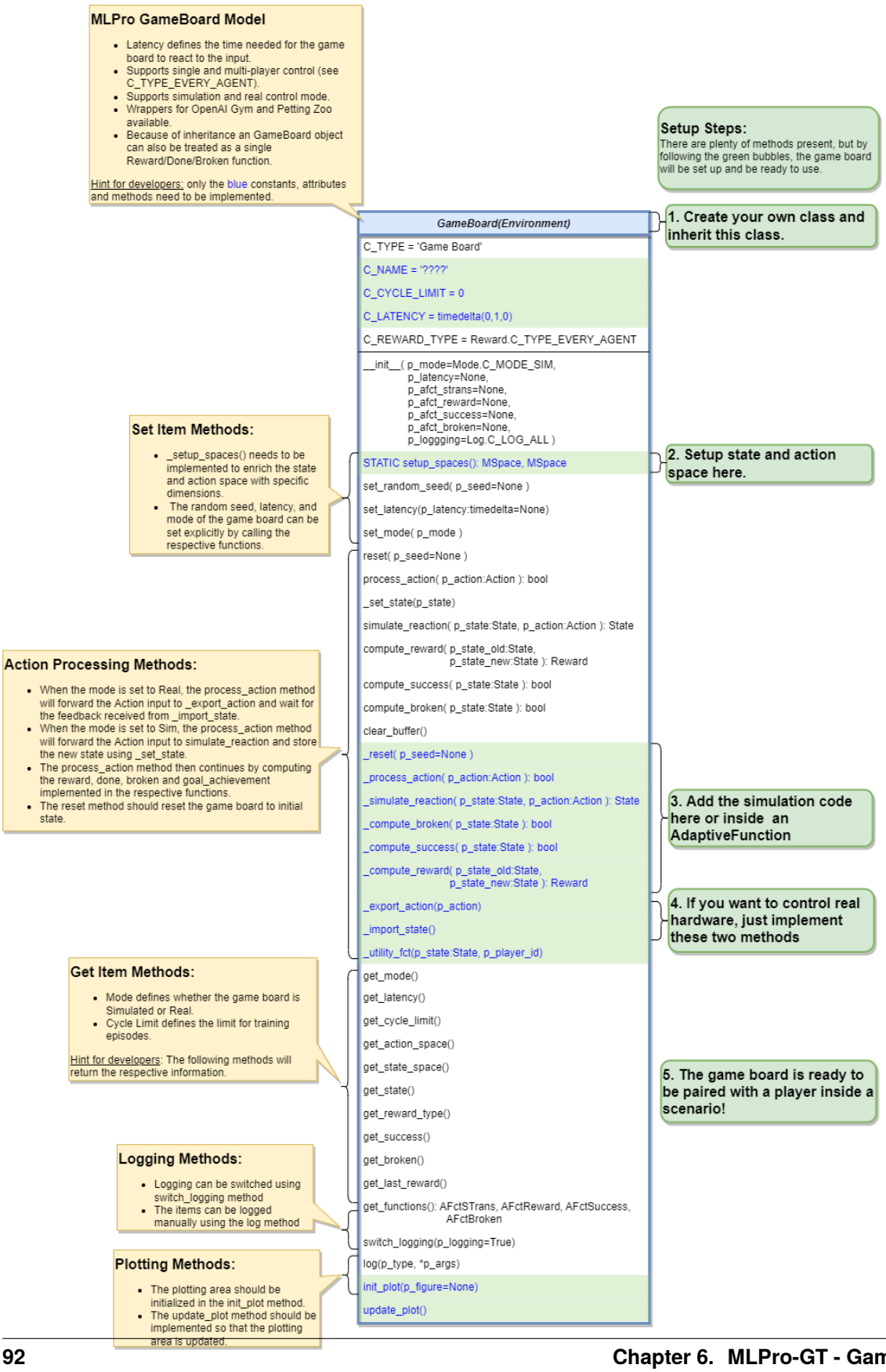
There are three main possibilities to set up a game board in MLPro, such as,

(1) the user can develop a custom game board,

(2) the user can reuse available RL Environments from MLPro, OpenAI Gym, or PettingZoo, and

(3) the user can reuse the provided game boards by accessing them from the pool of objects.

6.3.1 Custom Game boards



- **Game Board Creation for Simulation Mode**

  Creating a game board that satisfies the MLPro interface is immensely simple and straightforward. Basically, an MLPro game board is a class with several main functions. Each game board must apply the following MLPro functions:

```
from mlpro.gt.models import *

class MyGameBoard(GameBoard):
    """
    Custom game board that satisfies mlpro interface.
    """
    C_NAME          = 'MyGameboard'
    C_LATENCY       = timedelta(0,1,0)              # Default latency 1s
    C_REWARD_TYPE   = Reward.C_TYPE_EVERY_AGENT     # Default reward type

    def __init__(self, p_mode=C_MODE_SIM, p_latency:timedelta=None, p_
↪logging=True):
        """
        Parameters:
            p_mode              Mode of environment (simulation/real)
            p_latency           Optional: latency of environment. If not␣
↪provided
                                internal value C_LATENCY will be used by␣
↪default
            p_logging           Boolean switch for logging
        """

        super().__init__(p_latency=p_latency, p_logging=p_logging)
        self._setup_spaces()
        self.set_mode(p_mode)

    def _setup_spaces(self):
        """
        Static template method to set up and return state and action space␣
↪of environment.

        Returns
        -------
        state_space : MSpace
            State space object
        action_space : MSpace
            Action space object
        """

        # Setup state space example
        # self.state_space.add_dim(Dimension('Pos', 'Position', '', 'm', 'm', [-
↪50,50]))
        # self.state_space.add_dim(Dimension('Vel', 'Velocity', '', 'm/sec', '\
↪frac{m}{sec}', [-50,50]))

        # Setup action space example
        # self.action_space.add_dim(Dimension('Rot', 'Rotation', '', '1/sec', '\
↪frac{1}{sec}', [-50,50]))
```

(continues on next page)

(continued from previous page)

```
        ....

    def _process_action(self, p_action: Action) -> bool:
        """
        Custom method for state transition. To be implemented in a child␣
↪class. See method
        process_action() for further details.

        Parameters
        ----------
        p_action : Action
            Action to be processed

        Returns
        -------
        success : bool
            True, if action processing was successfull. False otherwise.
        """
        ....

    def _simulate_reaction(self, p_state: State, p_action: Action) -> State:
        """
        Custom implementation to simulate a state transition. See method␣
↪simulate_reaction() for
        further details.

        Parameters
        ----------
        p_state : State
            Current state.
        p_action : Action
            Action.

        Returns
        -------
        State
            Subsequent state after transition
        """
        ....

    def _reset(self, p_seed=None) -> None:
        """
        Custom method to reset the environment to an initial/defined state.

        Parameters
        ----------
        p_seed : int
            Seed parameter for an internal random generator

        """
        ....
```

(continues on next page)

(continued from previous page)

```
    def _compute_reward(self, p_state_old: State, p_state_new: State) ->␣
↪Reward:
        """
        Custom reward computation method. See method compute_reward() for␣
↪further details.

        Parameters
        ----------
        p_state_old : State
            Optional state before transition. If None the internal previous␣
↪state of the environment
            is used.
        p_state_new : State
            Optional tate after transition. If None the internal current␣
↪state of the environment
            is used.

        Returns
        -------
        Reward
            Reward object.
        """
        ....

    def _compute_success(self, p_state: State) -> bool:
        """
        Custom method for state evaluation 'success'. See method compute_
↪success() for further details.

        Parameters
        ----------
        p_state : State
            State to be assessed.

        Returns
        -------
        bool
            True, if the given state is a 'success' state. False otherwise.
        """
        ....

    def _compute_broken(self, p_state: State) -> bool:
        """
        Custom method for state evaluation 'broken'. See method compute_
↪broken() for further details.

        Parameters
        ----------
        p_state : State
            State to be assessed.

        Returns
```

(continues on next page)

(continued from previous page)

```
        -------
        bool
            True, if the given state is a 'broken' state. False otherwise.
        """
        ....

    def _utility_fct(self, p_state: State, p_player_id):
        """
        Computes utility of given player. To be redefined.
        """
        ....
```

One of the benefits for MLPro users is the variety of reward structures, which is useful for a multi-player GT approach. Three types of reward structures are supported in this framework, such as:

1. **C_TYPE_OVERALL** is the default type and is a scalar overall value,
2. **C_TYPE_EVERY_AGENT** is a scalar for every player, and
3. **C_TYPE_EVERY_ACTION** is a scalar for every player and action.

To set up state- and action spaces using our basic functionalities, please refer to *Howto GT-002: Train Multi-Player*.

- **Game Board Creation for Real Hardware Mode**

  In MLPro, we can choose simulation mode or real hardware mode. For real hardware mode, the creation of an environment is very similar to thesimulation mode. You do not need to define **_simulate_reaction**, but you need to replace it with **_export_action** and **_import_state** as it is shown in the following:

```
def _export_action(self, p_action: Action) -> bool:
    """
    Mode C_MODE_REAL only: exports given action to be processed externally
    (for instance by a real hardware). Please redefine.

    Parameters
    ----------
    p_action : Action
        Action to be exported

    Returns
    -------
    bool
        True, if action export was successful. False otherwise.

    """

    raise NotImplementedError

def _import_state(self) -> bool:
    """
    Mode C_MODE_REAL only: imports state from an external system (for␣
↪instance a real hardware).
    Please redefine. Please use method set_state() for internal update.
```

(continues on next page)

(continued from previous page)

```
    Returns
    -------
    bool
        True, if state import was successful. False otherwise.

    """

    raise NotImplementedError
```

- **Game Board Checker**

  To check whether your developed game board is compatible with the MLPro interface, we provide a test script using Unittest. At the moment, you can find the source code here. We will prepare a built-in testing module in MLPro, show you how to execute the testing soon and provides an example as well.

6.3.2 Reusing RL Environments

- **Transfering RL Environment to GT Game Board**

  In MLPro, we can simply transfer an RL environment to a GT game board by inheriting GameBoard functionality, as it is shown in the following:

```
from mlpro.gt.models import *
from mlpro.rl.pool.envs.dummy_environment import DummyEnv

class MyGameBoard_GT(DummyEnv, GameBoard):
    C_NAME          = 'MyGameBoard_GT'

    def __init__(self, p_logging=True):
        DummyEnv.__init__(self, p_reward_type=Reward.C_TYPE_EVERY_AGENT)
```

- **Game board from Third Party Packages**

  Alternatively, if your environment follows Gym or PettingZoo interface, you can apply our relevant useful wrappers for the integration between third-party packages and MLPro. For more information about the available third-party packages, please click *here*. Then, you need to transfer the wrapped RL environment to a GT Game Board.

Cross Reference

- *MLPro-RL*

6.3.3 Game board Pool

BGLP

This game board reuses a native implementation of RL environment, which you can find the details here.

This BGLP game board can be imported via:

```
import mlpro.gt.pool.boards.bglp
```

Multicartpole

This game board reuses a native implementation of RL environment, which you can find the details here.

This multi-cartpole game board can be imported via:

```
import mlpro.gt.pool.boards.multicartpole
```

6.4 Players

In the game theoretical approach, the player takes decisions based on the actual states in a game. A game contains two or more players that can be working cooperatively or compete with each other. Each player is supplied with a policy, where the policy can be used for the decision-making process and also optimized. The decision-making of the player is in the form of the next action.

The three main tasks of the player are as follows,

(1) Compute new action based on the current state.

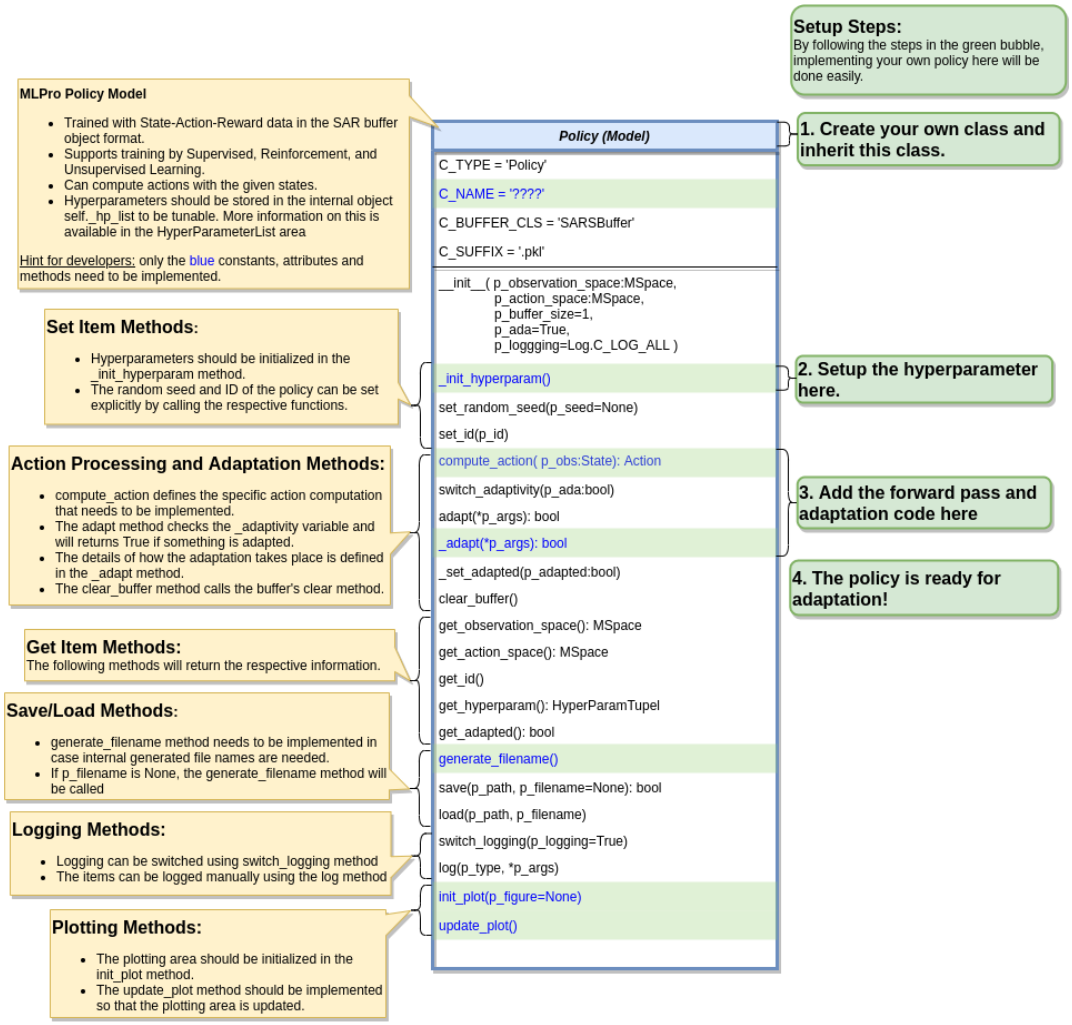
(2) Calculate local utility value based on the previous state and next state.

(3) Optimize their policy based on the state and the selected action.

In MLPro, you can customize your GT-based policy or import the provided policies in the pool of objects (unavailable at the moment).

6.4.1 Custom Policies

- **Policy Creation**



Creating a GT policy that satisfies the MLPro interface is pretty straightforward. You just require to assure that the GT policy consists of at least the following 3 main functions:

```
from mlpro.rl.models import *

class MyPolicy(Policy):
    """
    Creates a policy that satisfies the mlpro interface.
    """
    C_NAME          = 'MyPolicy'

    def _init_hyperparam(self, **p_par):
        """
        Implementation-specific hyperparameters can be added here. Please␣
↪follow these steps:
        a) Add each hyperparameter as an object of type HyperParam to the␣
↪internal hyperparameter
           space object self._hyperparam_space
        b) Create a hyperparameter tuple and bind it to self._hyperparam_
↪tuple
        c) Set the default value for each hyperparameter
```

(continues on next page)

(continued from previous page)

```
        Parameters
        ----------
        p_par : Dict
            Further model-specific parameters, that are passed through the␣
↪constructor.

        """
        ....

    def compute_action(self, p_obs: State) -> Action:
        """
        Specific action computation method to be redefined.

        Parameters
        ----------
        p_obs : State
            Observation data.

        Returns
        -------
        action : Action
            Action object.

        """
        ....

    def _adapt(self, *p_args) -> bool:
        """
        Adapts the policy based on State-Action-Reward-State (SARS) data.

        Parameters
        ----------
        p_arg[0] : SARSElement
            Object of type SARSElement.

        Returns
        -------
        adapted : bool
            True, if something has been adapted. False otherwise.

        """
        ....
```

To set up a hyperparameter space, please refer to *Howto BF-ML-010: Hyperparameters*.

- **Algorithm Checker**

  A test script using the unit test to check the developed policies will be available soon!

CHAPTER

SEVEN

MLPRO-OA - ONLINE ADAPTIVITY

7.1 Overview

Sub-framework for online machine learning. Coming soon…

7.2 Getting Started

Add text here!

CHAPTER

EIGHT

A1 - EXAMPLE POOL

8.1 MLPro-BF - Basic Functions

The following examples demonstrate various basic functionalities of MLPro-BF:

8.1.1 Layer 0 - Elementary Functions

Various Functions

Howto BF-001: Logging

Ver. 1.1.2 (2023-03-02)

This module demonstrates the Log class functionality.

You will learn:

1. How to use the log functionality of MLPro in custom implementations.
2. How to use different log levels to log information, warnings and error.
3. How to switch logging functionality.

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_001_logging.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-10-07  1.0.0     DA       Creation
## -- 2021-11-03  1.1.0     DA       Introduction of new log type C_LOG_TYPE_S for␣
↪success messages
## -- 2021-11-13  1.1.1     DA       Refactoring
## -- 2023-03-02  1.1.2     LSB      Refactoring
## -------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
"""
Ver. 1.1.2 (2023-03-02)

This module demonstrates the Log class functionality.

You will learn:

1. How to use the log functionality of MLPro in custom implementations.

2. How to use different log levels to log information, warnings and error.

3. How to switch logging functionality.
"""


from mlpro.bf.various import Log




# 1 Reuse logging property in your own class by inheriting from class Log
class MyClass(Log):

    # These constants are inherited fromm class Log and will be logged in every log line.
↪..
    C_TYPE  = 'Demo class'
    C_NAME  = 'MyClass'

    def __init__(self, p_logging=True):
        # The constructor of class Log initializes the internal logging and writes the␣
↪first line...
        super().__init__(p_logging=p_logging)


    def my_method(self):
        # The log types I/E/W are also inherited from class Log...
        self.log(self.C_LOG_TYPE_I, 'Let me tell you what\'s going on...')
        self.log(self.C_LOG_TYPE_W, 'Something is weird...')
        self.log(self.C_LOG_TYPE_E, 'And here something failed...')
        self.log(self.C_LOG_TYPE_I, 'But don\'t worry. Everything is fine. It\'s just a␣
↪demo:)')
        self.log(self.C_LOG_TYPE_S, 'This method terminated successfully!\n')




if __name__ == "__main__":

    # 2 Log everything inside your class...
    print('\n--\n-- Example 1: By default everything is logged...\n--\n')
    mc = MyClass(p_logging=Log.C_LOG_ALL)
    mc.my_method()
```

(continues on next page)

(continued from previous page)

```
    # 3 Log nothing inside your class
    print('\n--\n-- Example 2: Now logging is switched off...\n--\n')
    mc.switch_logging(Log.C_LOG_NOTHING)
    mc.my_method()


    # 4 Log warnings and errors only
    print('\n--\n-- Example 3: Only warnings and errors are logged...\n--\n')
    mc.switch_logging(Log.C_LOG_WE)
    mc.my_method()

    # 5 Log errors only
    print('\n--\n-- Example 4: Only errors are logged...\n--\n')
    mc.switch_logging(Log.C_LOG_E)
    mc.my_method()

    # 6 Log everything again
    print('\n--\n-- Example 5: Everything is logged again...\n--\n')
    mc.switch_logging(Log.C_LOG_ALL)
    mc.my_method()
```

Results

```
2022-09-02  14:19:26.962434  I  Demo class MyClass: Let me tell you what's going on...
2022-09-02  14:19:26.962437  W  Demo class MyClass: Something is weird...
2022-09-02  14:19:26.962500  E  Demo class MyClass: And here something failed...
2022-09-02  14:19:26.962505  I  Demo class MyClass: But don't worry. Everything is fine. It's just a demo:)
2022-09-02  14:19:26.962508  S  Demo class MyClass: This method terminated successfully!
```

Cross Reference

- *API Reference: Various*

Howto BF-002: Timer

Ver. 1.0.2 (2023-03-02)

This module demonstrates the Timer class functionality.

You will learn:

1. How to use the timer functionality of MLPro in native and custom implementations.
2. How to use the lap functionality of MLPro's Timer.

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_002_timer.py
## -------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-05-19  1.0.0     DA       Creation
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library␣
↪name
## -- 2021-11-13  1.0.1     DA       Minor fix
## -- 2023-03-02  1.0.2     LSB      Refactoring
## ------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.2 (2023-03-02)

This module demonstrates the Timer class functionality.

You will learn:

1. How to use the timer functionality of MLPro in native and custom implementations.

2. How to use the lap functionality of MLPro's Timer.
"""


import time
import random
from datetime import timedelta
from mlpro.bf.various import Timer, Log




# Demo class
class TimerDemo (Log):

    C_TYPE  = 'Demo class'
    C_NAME  = 'Timer'

    def __init__(self, p_timer:Timer):
        self.timer = p_timer
        super().__init__()


    def log(self, p_type, *p_args):
        super().log(p_type, 'Process time', self.timer.get_time(), ', Cycle', self.timer.
↪get_lap_id(), 'Lap time', self.timer.get_lap_time(), '--', *p_args)


    def run_step(self, p_step_id):
        self.log(self.C_LOG_TYPE_I, 'Process step', p_step_id, 'started')
        duration = 0.6 * random.random()
        time.sleep(duration)
        self.log(self.C_LOG_TYPE_I, 'Process step', p_step_id, 'ended after', duration,
```

(continues on next page)

(continued from previous page)

```
↪'seconds')


    def run_cycle(self):
        self.run_step(1)
        self.run_step(2)
        self.run_step(3)
        if not self.timer.finish_lap():
            self.log(self.C_LOG_TYPE_W, 'Last process cycle timed out!!')


    def run(self):
        for i in range(10):
            self.run_cycle()




if __name__ == "__main__":

    # Example 1
    print('\n\n\nExample 1: Timer in virtual time mode with lap duration 1 day and 15␣
↪seconds...\n\n')
    t = Timer(Timer.C_MODE_VIRTUAL, timedelta(1,15,0))
    d = TimerDemo(t)
    d.run()


    # Example 2
    print('\n\n\nExample 2: Timer in real time mode with lap duration 1 second and␣
↪forced timeout situations...\n\n')
    t = Timer(Timer.C_MODE_REAL, timedelta(0,1,0))
    d = TimerDemo(t)
    d.run()
```

Results

```
Example 1: Timer in virtual time mode with lap duration 1 day and 15 seconds...


2023-02-10  15:10:52.396617  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Instantiated
2023-02-10  15:10:52.396617  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 1 started
2023-02-10  15:10:52.981296  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 1 ended after 0.5727354329679715 seconds
2023-02-10  15:10:52.981296  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 2 started
2023-02-10  15:10:53.250889  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 2 ended after 0.2671762029006218 seconds
2023-02-10  15:10:53.250889  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 3 started
2023-02-10  15:10:53.723793  I  Demo class "Timer": Process time 0:00:00 , Cycle 0 Lap time 0:00:00 -- Process step 3 ended after 0.46618259516486654 seconds
2023-02-10  15:10:53.723793  I  Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 1 started
2023-02-10  15:10:54.310007  I  Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 1 ended after 0.5742077882721884 seconds
2023-02-10  15:10:54.310007  I  Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 2 started
2023-02-10  15:10:54.533070  I  Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 2 ended after 0.21479102210586493 seconds
2023-02-10  15:10:54.533070  I  Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 3 started
2023-02-10  15:10:54.612608  I  Demo class "Timer": Process time 1 day, 0:00:15 , Cycle 1 Lap time 0:00:00 -- Process step 3 ended after 0.06589391986663488 seconds
2023-02-10  15:10:54.612608  I  Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 1 started
2023-02-10  15:10:54.961244  I  Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 1 ended after 0.33888097514695076 seconds
2023-02-10  15:10:54.961244  I  Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 2 started
2023-02-10  15:10:55.278839  I  Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 2 ended after 0.3068213153874194 seconds
2023-02-10  15:10:55.278839  I  Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 3 started
2023-02-10  15:10:55.706568  I  Demo class "Timer": Process time 2 days, 0:00:30 , Cycle 2 Lap time 0:00:00 -- Process step 3 ended after 0.4120912986840059 seconds
```

Cross Reference

- *API Reference: Various*

Howto BF-003: Store and plot data

Ver. 1.2.4 (2023-03-02)

This module demonstrates how to store, plot, save and load variables.

You will learn:

1. How to use the DataStoring class of MLPro and its functionalities.
2. How to add frames and data names in the data storing object.
3. How to memorize data using the DataStoring class.
4. How to plot the memorized frame from DataStoring object.

Prerequisites

PLease install following packages to run this howto

- Matplotlib

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_003_store_plot_and_save_variables.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    DeŚscription
## -- 2021-06-16  1.0.0     SY       Creation/Release
## -- 2021-06-21  1.1.0     SY       Adjustment to updated DataPlotting class
## -- 2021-07-01  1.2.0     SY       Adjustment due to extension in save and load data
## -- 2021-09-11  1.2.1     MRD      Change Header information to match our new library␣
↪name
## -- 2021-10-25  1.2.2     SY       Adjustment due to improvement in DataPlotting
## -- 2021-10-26  1.2.3     SY       Rename module
## -- 2023-03-02  1.2.4     LSB      Refactoring
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.2.4 (2023-03-02)

This module demonstrates how to store, plot, save and load variables.

You will learn:

1. How to use the DataStoring class of MLPro and its functionalities.

2. How to add frames and data names in the data storing object.
```

(continues on next page)

(continued from previous page)

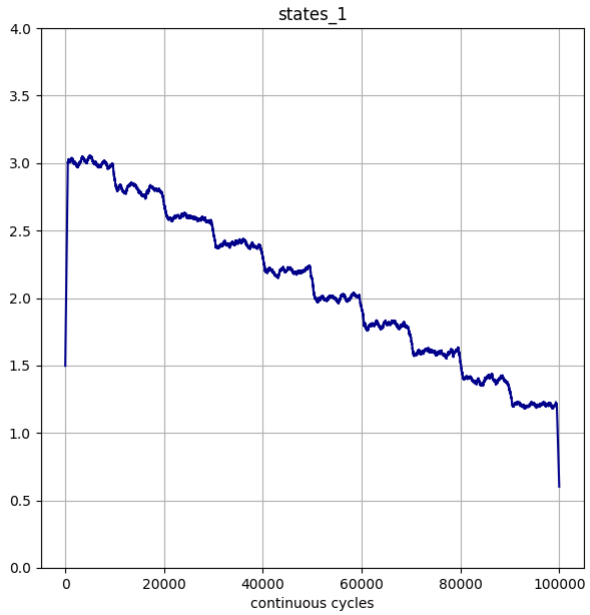
```
3. How to memorize data using the DataStoring class.

4. How to plot the memorized frame from DataStoring object.
"""


from mlpro.bf.various import *
from mlpro.bf.data import *
from mlpro.bf.plot import *
import random


if __name__ == "__main__":
    num_eps          = 10
    num_cycles       = 10000
    data_names       = ["reward","states_1","states_2","model_loss"]
    data_printing    = {"reward":         [True,0,10],
                        "states_1":       [True,0,4],
                        "states_2":       [True,0,4],
                        "model_loss":     [True,0,-1]}

    ## 1. How to store data ##
    mem = DataStoring(data_names)
    for ep in range(num_eps):
        ep_id = ("ep. %s"%str(ep+1))
        mem.add_frame(ep_id)
        for i in range(num_cycles):
            mem.memorize("reward",ep_id,random.uniform(0+(ep*0.5),5+(ep*0.5)))
            mem.memorize("states_1",ep_id,random.uniform(2-(ep*0.2),4-(ep*0.2)))
            mem.memorize("states_2",ep_id,random.uniform(0+(ep*0.2),2+(ep*0.2)))
            mem.memorize("model_loss",ep_id,random.uniform(0.25-(ep*0.02),1-(ep*0.07)))

    ## 2. How to plot stored data ##
    # 2.1. Plotting data per cycle
    # mem_plot    = DataPlotting(mem, p_type=DataPlotting.C_PLOT_TYPE_CY, p_window=100,
    #                            p_showing=True, p_printing=data_printing, p_figsize=(7,
↪7),
    #                            p_color="darkblue")
    # 2.2. Plotting data with continuous cycle
    mem_plot    = DataPlotting(mem, p_type=DataPlotting.C_PLOT_TYPE_EP, p_window=1000,
                               p_showing=True, p_printing=data_printing, p_figsize=(7,
↪7),
                               p_color="darkblue")
    # 2.3. Plotting data per epsiode according to its mean value
    # mem_plot    = DataPlotting(mem, p_type=DataPlotting.C_PLOT_TYPE_EP_M, p_window=1,
    #                            p_showing=True, p_printing=data_printing, p_figsize=(7,
↪7),
    #                            p_color="darkblue")
    mem_plot.get_plots()

    ## 3. How to save plots and data in binary file (variables, classes, etc.) ##
    path_save   = input("Input path_save : ")
```

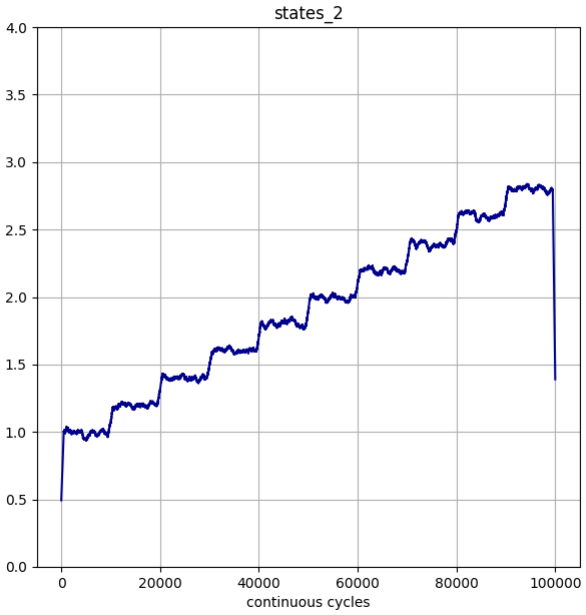
(continues on next page)

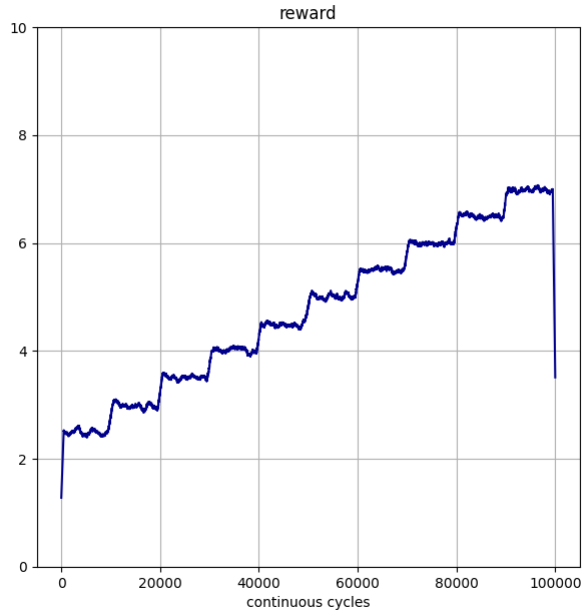
(continued from previous page)

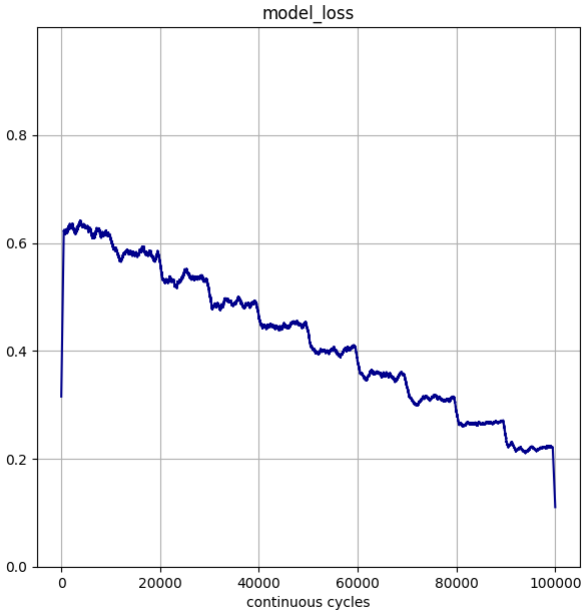
```
    #Do not include quote-unquote ("" or " ) into target path name
    mem_plot.save_plots(path_save, "pdf")
    mem_plot.save(path_save, "plot_memory")
    mem.save(path_save, "data_memory")
    mem.save_data(path_save, "data_storage", "\t")

    ## 4. How to load data from binary file ##
    path_load   = path_save
    mem_load    = DataStoring.load(path_load, "data_memory")
    print("Comparison :")
    print("Original data                : %.5f"%mem.memory_dict["reward"]["ep. 1"][0])
    print("Loaded data from binary file : %.5f"%mem_load.memory_dict["reward"]["ep. 1
↪"][0])

    ## 5. How to load data from csv file ##
    data_names = []
    mem_from_csv = DataStoring(data_names)
    mem_from_csv.load_data(path_load, "data_storage.csv", "\t")
    print("Comparison :")
    print("Original data             : %.5f"%mem.memory_dict["reward"]["ep. 1"][0])
    print("Loaded data from csv file : %.5f"%mem_from_csv.memory_dict["reward"]["ep. 1
↪"][0])
```

Results




states_2
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
20000
40000
60000
80000
100000
continuous cycles


reward
10
8
6
4
2
0
0
20000
40000
60000
80000
100000
continuous cycles



Cross Reference

- *API Reference: Various*

Howto BF-004: Buffers

Ver. 1.0.1 (2023-03-02)

This module demonstrates how to use classes Buffer and BufferElement.

You will learn:

1. How to use the buffer, buffer element class of MLPro in native and custom implementations.
2. How to add values to a buffer element object.
2. How to add buffer elements with data to the buffer object.
3. How to get the buffer elements and data from a buffer object and clear a buffer object.

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_004_buffers.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-10-26  1.0.0     SY       Creation/Release
## -- 2023-03-02  1.0.1     LSB      Refactoring
## -------------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-----------

"""
Ver. 1.0.1 (2023-03-02)

This module demonstrates how to use classes Buffer and BufferElement.

You will learn:

1. How to use the buffer, buffer element class of MLPro in native and custom␣
↪implementations.

2. How to add values to a buffer element object.

2. How to add buffer elements with data to the buffer object.

3. How to get the buffer elements and data from a buffer object and clear a buffer␣
↪object.
"""


from mlpro.bf.data import *
import random



if __name__ == "__main__":

    # 1. Instantiate a buffer with random sampling
    buffer     = BufferRnd(p_size=100)
    num_cycles = 150


    # 2 Generate random values and store them to the Buffer
    for i in range(num_cycles):

        # 2.1 Store the values and their names in a BufferElement
        buffer_element  =  BufferElement({"reward":random.uniform(-10,10),
                                          "actions":[random.uniform(0,1),random.uniform(0,
↪1)]})

        # 2.2 Example: add value element in the developed BufferElement
        buffer_element.add_value_element(dict(accuracy=random.uniform(0,1)))

        # 2.3 Add the BufferElement into the Buffer
        buffer.add_element(buffer_element)
        print('Cycle : %.i'%int(i+1))

        # 2.4 Checking whether buffer is full or not
        if not buffer.is_full():
            print('Buffer is not full yet, keep collecting data!\n')
        else:
```

(continues on next page)

(continued from previous page)

```
            print('Buffer is full, ready to use!')

            # 2.5 Get all data from the Buffer
            all_data = buffer.get_all()
            _actions            = all_data["actions"]
            _reward             = all_data["reward"]
            _accuracy           = all_data["accuracy"]

            # 2.6 Get sample data from the Buffer, you define your sampling strategy by
            # redifining method _gen_sample_ind(self, p_num:int)
            sample_data = buffer.get_sample(p_num=10)
            print('Get sample!\n')
            _actions_sample    = sample_data["actions"]
            _reward_sample     = sample_data["reward"]
            _accuracy_sample   = sample_data["accuracy"]

    # 3 To clear your buffer
    if buffer is not None:
        buffer.clear()
        print('Buffer is cleared!')
```

Results

```
Cycle : 96
Buffer is not full yet, keep collecting data!

Cycle : 97
Buffer is not full yet, keep collecting data!

Cycle : 98
Buffer is not full yet, keep collecting data!

Cycle : 99
Buffer is not full yet, keep collecting data!

Cycle : 100
Buffer is full, ready to use!
Get sample!

Cycle : 101
Buffer is full, ready to use!
Get sample!

Cycle : 102
Buffer is full, ready to use!
Get sample!
```

Cross Reference

- *API Reference: Various*

User Interaction

Howto BF-UI-001: SciUI - Reuse of interactive 2D/3D Input Space

Ver. 1.0.3 (2022-10-08)

Demo scenarios for SciUI framework that shows the reuse of the interactive 2D/3D input space class. Can be executed directly...

Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_ui_001_reuse_of_interactive_2d_3d_input_space.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-06-20  0.0.0     DA       Creation
## -- 2021-07-03  1.0.0     DA       Release of first version
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library name
## -- 2022-01-06  1.0.1     DA       Corrections
## -- 2022-03-21  1.0.2     SY       Refactoring following class Dimensions update
## -- 2022-10-08  1.0.3     DA       Refactoring following class Dimensions update
## -------------------------------------------------------------------------------------------

"""
Ver. 1.0.3 (2022-10-08)

Demo scenarios for SciUI framework that shows the reuse of the interactive 2D/3D input space class.
Can be executed directly...
"""



from mlpro.bf.ui.sciui.framework import *
from mlpro.bf.ui.sciui.pool.iis import InteractiveInputSpace
from mlpro.bf.math import *
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------
class DemoIIS2D(SciUIScenario):

    C_NAME          = 'Demo for interactive 2D Input Space'
    C_VERSION       = '1.0.0'
    C_RELEASED      = True
    C_VISIBLE       = True

## -------------------------------------------------------------------------------------
    def init_component(self):
        super().init_component()

        # 1 Add scenario-specific variables to shared db
        InteractiveInputSpace.enrich_shared_db(self.shared_db)
        self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x1',
                                                      p_description='',
                                                      p_name_latex='x_1',
                                                      p_unit='m',
                                                      p_unit_latex='m',
                                                      p_boundaries=[-5,5]) )
        self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x2',
                                                      p_description='',
                                                      p_name_latex='x_2',
                                                      p_unit='m/s',
                                                      p_unit_latex='\\frac{m}{s}',
                                                      p_boundaries=[-25,25]) )

        # 2 Build scenario structure
        self.add_component(InteractiveInputSpace(self.shared_db, p_row=0, p_col=0, p_
↪padx=5, p_logging=self._level))




## -------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------
class DemoIIS3D(SciUIScenario):

    C_NAME          = 'Demo for interactive 3D Input Space'
    C_VERSION       = '1.0.0'
    C_RELEASED      = True
    C_VISIBLE       = True
```

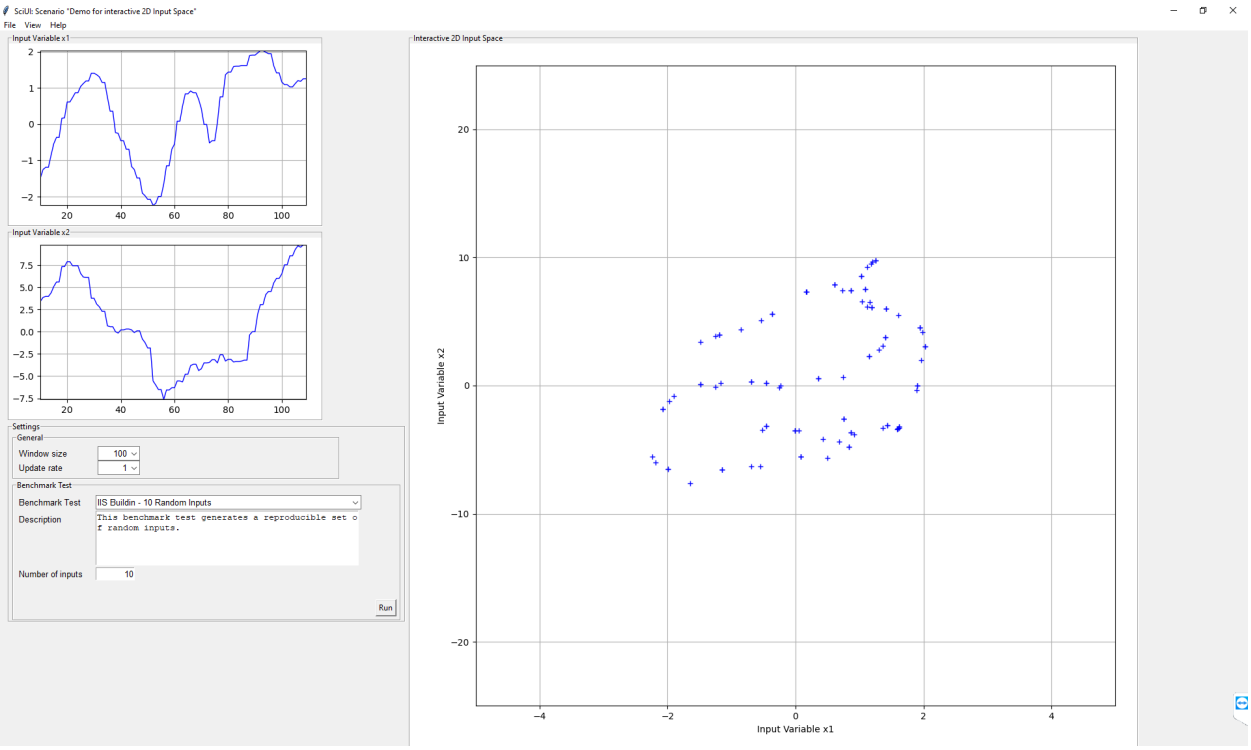
(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
    def init_component(self):
        super().init_component()

        # 1 Add scenario-specific variables to shared db
        InteractiveInputSpace.enrich_shared_db(self.shared_db)
        self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x1',
                                                      p_description='',
                                                      p_name_latex='x_1',
                                                      p_unit='m',
                                                      p_unit_latex='m',
                                                      p_boundaries=[-5,5]) )
        self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x2',
                                                      p_description='',
                                                      p_name_latex='x_2',
                                                      p_unit='m/s',
                                                      p_unit_latex='\\frac{m}{s}',
                                                      p_boundaries=[-25,25]) )
        self.shared_db.iis_ispace.add_dim( Dimension( p_name_short='x3',
                                                      p_description='',
                                                      p_name_latex='x_3',
                                                      p_unit='m/s^2',
                                                      p_unit_latex='\\frac{m}{s^2}',
                                                      p_boundaries=[-15,15] ))

        # 2 Build scenario structure
        self.add_component(InteractiveInputSpace(self.shared_db, p_row=0, p_col=0, p_
↪padx=5, p_logging=self._level))




if (__name__ == '__main__'):
    from mlpro.bf.ui.sciui.main import SciUI
    SciUI()
```

Results

The SciUI application should start and show an interactive demo of a 2D/3D input space as follows:



Cross Reference

- *API Reference: SciUi*

8.1.2 Layer 1 - Computation

Event Handling

Howto BF-EH-001: Event Handling

Ver. 1.0.1 (2022-10-12)

This module demonstrates the use of MLPro's event handling as a property in own classes. To this regard, a demo class MyMainClass is set up that inherits event functionalities from MLPro's class EventManager. Furthermore an own sample event handler class MyHandlerClass is implemented.

You will learn:

1) how to implement an own class with event management functionality
2) how to implement an own event handler class
3) how to register event handlers
4) how to fire events

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
```

(continues on next page)

(continued from previous page)

```
## -- Module  : howto_bf_eh_001_event_handling.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-08-21  1.0.0     DA       Creation/release
## -- 2022-10-12  1.0.1     DA       Refactoring/Renaming
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.1 (2022-10-12)

This module demonstrates the use of MLPro's event handling as a property in own classes.␣
↪To this
regard, a demo class MyMainClass is set up that inherits event functionalities from MLPro
↪'s class
EventManager. Furthermore an own sample event handler class MyHandlerClass is␣
↪implemented.


You will learn:

1) how to implement an own class with event management functionality

2) how to implement an own event handler class

3) how to register event handlers

4) how to fire events

"""



from mlpro.bf.various import Log
from mlpro.bf.events import *



# 1 Definition of custom class that inherits event management functionalities from MLPro
↪'s class EventManager
class MyMainClass (EventManager):

    C_NAME          = 'My main class'

    C_EVENT_OWN     = 'MYEVENT'

    def __init__(self, p_logging=Log.C_LOG_ALL):
        super().__init__(p_logging)
```

(continues on next page)

(continued from previous page)

```
    def do_something(self):
        eventobj = Event(p_raising_object=self, p_par1='Hello', p_par2='World!')
        self._raise_event(self.C_EVENT_OWN, eventobj)



# 2 Definition of custom event handler class
class MyHandlerClass (Log):

    C_TYPE          = 'Event handler'
    C_NAME          = 'My handler'

    def myhandler(self, p_event_id, p_event_object:Event):
        self.log(Log.C_LOG_TYPE_I, 'Received event id', p_event_id)
        self.log(Log.C_LOG_TYPE_I, 'Event data:', p_event_object.get_data())



# 3 Instantiation of own event handler and main class as event manager
if __name__ == "__main__":
    # 3.1 Interactive/Demo mode
    myhandlerobj    = MyHandlerClass()
    mymainobj       = MyMainClass()

else:
    # 3.2 Unit test mode
    myhandlerobj    = MyHandlerClass(p_logging=Log.C_LOG_NOTHING)
    mymainobj       = MyMainClass(p_logging=Log.C_LOG_NOTHING)


# 4 Own event handler is registered on main class
mymainobj.register_event_handler(MyMainClass.C_EVENT_OWN, myhandlerobj.myhandler)


# 5 Event is fired
mymainobj.do_something()


# 6 Own event handler is removed from main class
mymainobj.remove_event_handler(MyMainClass.C_EVENT_OWN, myhandlerobj.myhandler)


# 7 Same event is fired again
mymainobj.do_something()
```

Results

```
2023-02-10  15:50:45.050902  I  Event handler "My handler": Instantiated
2023-02-10  15:50:45.050902  I  EventManager "My main class": Instantiated
2023-02-10  15:50:45.050902  I  EventManager "My main class": Event "MYEVENT" fired
2023-02-10  15:50:45.050902  I  EventManager "My main class": Calling handler 0
2023-02-10  15:50:45.050902  I  Event handler "My handler": Received event id MYEVENT
2023-02-10  15:50:45.051902  I  Event handler "My handler": Event data: {'p_par1': 'Hello', 'p_par2': 'World!'}
2023-02-10  15:50:45.051902  I  EventManager "My main class": Event "MYEVENT" fired
2023-02-10  15:50:45.051902  I  EventManager "My main class": No handlers registered for event "MYEVENT"
```

Cross References

- *API Reference: Event Handling*

Multitasking

Howto BF-MT-001: Multitasking - Parallel Algorithms

Ver. 1.3.0 (2022-10-13)

This module demonstrates the use of classes ASync and Shared as part of MLPro's multitasking concept. Both classes are used to implement a simple parallel algorithm class MyParallelAlgorithm with a method _async_subtask() for asynchronous execution collecting results in a shared object.

In three runs method _async_subtask() is executed several times a) synchronously, b) as threads and c) as processes. Depending on the number of cores per cpu, the operating system, and further factors multiprocessing outperforms multithreading more ore less drastically. Method MyParallelAlgoritm.execute() determines and logs the speed factor of multithreading and multiprocessing in comparison to serial/synchronous computation. Open the perfmeter of your system and play with number of tasks and their duration to observe the behavior.

All sub-tasks store dummy results in a shared object. It is no surprise that the order of result entries in multithreading and multiprocessing mode does not 100% match the order of sub-task starts.

You will learn:

1) The meaning and basic properties of the classes Async and Shared.

2) How to set up an own class with parallel running sub-tasks inside.

3) How to collect results of the parallel sub-functions in a shared object.

Prerequisites

Please install following packages to run this howto

- Multiprocess

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_mt_001_parallel_algorithms.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-03  1.0.0     DA       Creation/release
## -- 2022-10-09  1.1.0     DA       Simplification
```

(continues on next page)

(continued from previous page)

```
## -- 2022-10-12  1.2.0     DA       Restructuring of demo steps
## -- 2022-10-13  1.3.0     DA       Restructuring of demo steps
## -------------------------------------------------------------------------------
----------

"""
Ver. 1.3.0 (2022-10-13)

This module demonstrates the use of classes ASync and Shared as part of MLPro's
multitasking concept.
Both classes are used to implement a simple parallel algorithm class MyParallelAlgorithm
with a
method _async_subtask() for asynchronous execution collecting results in a shared object.

In three runs method _async_subtask() is executed several times a) synchronously, b) as
threads and
c) as processes. Depending on the number of cores per cpu, the operating system, and
further factors
multiprocessing outperforms multithreading more ore less drastically. Method
MyParallelAlgoritm.execute()
determines and logs the speed factor of multithreading and multiprocessing in comparison
to serial/synchronous
computation. Open the perfmeter of your system and play with number of tasks and their
duration to observe
the behavior.

All sub-tasks store dummy results in a shared object. It is no surprise that the order
of result entries
in multithreading and multiprocessing mode does not 100% match the order of sub-task
starts.

You will learn:

1) The meaning and basic properties of the classes Async and Shared.

2) How to set up an own class with parallel running sub-tasks inside.

3) How to collect results of the parallel sub-functions in a shared object.

"""


from time import sleep
from mlpro.bf.various import Log
import multiprocess as mp
import mlpro.bf.mt as mt
from datetime import datetime, timedelta
from cmath import pi, sin, cos, tan
import random
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
## ------------------------------------------------------------------------------------------
class MyParallelAlgorithm (mt.Async):
    """
    This class demonstrates how to run methods asynchronously and to collect results in a shared
    object.
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_TYPE      = 'Demo'
    C_NAME      = 'Parallel Algorithm'

## ------------------------------------------------------------------------------------------
    def __init__( self,
                  p_num_tasks:int,
                  p_duration:timedelta,
                  p_range_max=mt.Async.C_RANGE_PROCESS,
                  p_class_shared=mt.Shared,
                  p_logging=Log.C_LOG_ALL ):

        super().__init__( p_range_max=p_range_max,
                          p_class_shared=p_class_shared,
                          p_logging=p_logging )

        self._num_tasks    = p_num_tasks
        self._duration     = p_duration
        self._duration_sec = self._duration.seconds + self._duration.microseconds / 1000000


## ------------------------------------------------------------------------------------------
    def execute(self):
        # Log at the beginning of a run
        if self._range == self.C_RANGE_NONE:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'synchronous tasks started')
        elif self._range == self.C_RANGE_THREAD:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous tasks as threads started')
        else:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous tasks as processes started')

        # Start number of tasks (a)synchronously
        time_start = datetime.now()
        for t in range(self._num_tasks):
            self._start_async( p_target=self._async_subtask, p_tid=t)
```

(continues on next page)

(continued from previous page)

```
        self.wait_async_tasks()
        time_end   = datetime.now()

        # Determination of speed factor (no parallelism = 1)
        duration_real       = time_end - time_start
        duration_real_sec   = duration_real.seconds + duration_real.microseconds /␣
↪1000000
        speed_factor        = round( self._num_tasks * self._duration_sec / duration_
↪real_sec, 2)

        # Log of speed factor
        if self._range == self.C_RANGE_NONE:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'synchronous␣
↪tasks ended (speed factor =', speed_factor, ')')
        elif self._range == self.C_RANGE_THREAD:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous␣
↪tasks as threads ended (speed factor =', speed_factor, ')')
        else:
            self.log(Log.C_LOG_TYPE_S, 'Execution of', self._num_tasks, 'asynchronous␣
↪tasks as processes ended (speed factor =', speed_factor, ')')

        # Log of results collected in the shared object
        self.log(Log.C_LOG_TYPE_I, 'Results in shared object are:\n',self._so.get_
↪results())


## ------------------------------------------------------------------------------------
↪-----------
    def _async_subtask(self, p_tid):

        self.log(Log.C_LOG_TYPE_I, 'Task', p_tid, 'started')

        # 1 Sub-task needs to check in on shared object
        self._so.checkin( p_tid=p_tid )

        # 2 Dummy implementation to simulate a busy sub-task
        time_start = datetime.now()
        result     = 0

        while True:
            # do something meaningful
            for i in range(300):
                result += sin(random.random()*pi) * cos(random.random()*pi) * tan(random.
↪random()*pi)

            time_current = datetime.now()
            time_diff    = time_current - time_start
            if time_diff >= self._duration: break

        # 3 Sub-task can optionally store resuls in the shared object.
        self._so.add_result(p_tid=p_tid, p_result=result)
```

(continues on next page)

(continued from previous page)

```
        # 4 Sub-task needs to check out from shared object
        self._so.checkout( p_tid=p_tid )

        self.log(Log.C_LOG_TYPE_I, 'Task', p_tid, 'stopped')




if __name__ == "__main__":

    # 1 Preparation of execution

    # https://docs.python.org/3/library/multiprocessing.html?highlight=freeze_support
↪#multiprocessing.freeze_support
    mp.freeze_support()

    num_tasks   = 20
    duration    = timedelta(0,1,0)
    pause_sec   = 5
    logging     = Log.C_LOG_ALL



    # 2 Execution of demo class (synchronously)
    a = MyParallelAlgorithm( p_num_tasks = num_tasks,
                             p_duration = duration,
                             p_range_max = mt.Async.C_RANGE_NONE,
                             p_logging = logging )

    a.execute()



    # 3 Execution of demo class (asynchonously, multithreading)
    a.log(Log.C_LOG_TYPE_W, 'Short break for better observation of CPU load in perfmeter
↪')
    sleep(pause_sec)

    a = MyParallelAlgorithm( p_num_tasks = num_tasks,
                             p_duration = duration,
                             p_range_max = mt.Async.C_RANGE_THREAD,
                             p_logging = logging )

    a.execute()



    # 4 Execution of demo class (asynchronously, multiprocessing)
    a.log(Log.C_LOG_TYPE_W, 'Short break for better observation of CPU load in perfmeter
↪')
```

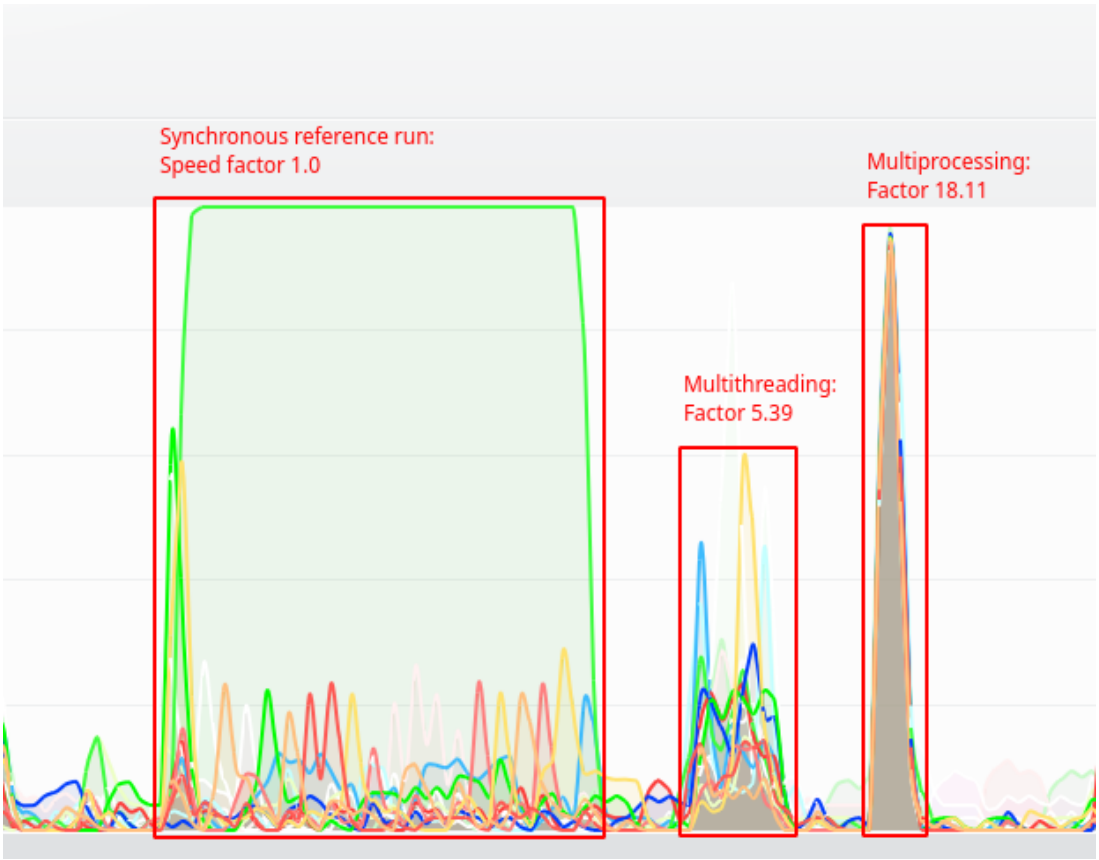
(continues on next page)

(continued from previous page)

```
    sleep(pause_sec)

    a = MyParallelAlgorithm( p_num_tasks = num_tasks,
                             p_duration = duration,
                             p_range_max = mt.Async.C_RANGE_PROCESS,
                             p_logging = logging )

    a.execute()
```

Results

The howto example logs details of the three runs and in particular the speed factors of multithreading and multiprocessing in comparison to the serial/synchronous execution. On a PC with an AMD Ryzen 7 CPU (8/16 cores) running Linux, the system monitor shows an approx. 5x speedup with multithreading and an approx. 18x speedup with multiprocessing.



Cross Reference

- *API Reference: Multitasking*

Howto BF-MT-002: Multitasking - Tasks and Workflows

Ver. 1.3.1 (2022-11-07)

This module demonstrates the use of tasks and workflows as part of MLPro's multitasking concept. To this regard, a demo custom task class is implemented. At first the task class is instantiated 9 times, added to a workflow, and chained by predecessor relations. In two experiments the workflow is executed synchronously and in multithreading mode. In the latter case, the tasks are partly executed parallel which increases the computation performance.

In both experiments pseudo results are stored in a shared object.

You will learn:

1) How to implement an own custom task
2) How to store results in a shared object
3) How to add tasks to a workflow
4) How to run tasks and workflows in various ranges of asynchronicity

Prerequisites

To run this howto please install the following packages

- Multiprocess

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_mt_002_tasks_and_workflows.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-04  1.0.0     DA       Creation/release
## -- 2022-10-09  1.1.0     DA       Simplification
## -- 2022-10-12  1.2.0     DA       Restructuring of demo steps
## -- 2022-10-13  1.3.0     DA       Simplification and reduction to multithreading
## -- 2022-11-07  1.3.1     DA       Minor correction
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.3.1 (2022-11-07)

This module demonstrates the use of tasks and workflows as part of MLPro's multitasking␣
↪concept.
To this regard, a demo custom task class is implemented. At first the task class is␣
↪instantiated 9
times, added to a workflow, and chained by predecessor relations. In two experiments the␣
↪workflow is
executed synchronously and in multithreading mode. In the latter case, the tasks are␣
↪partly executed
parallel which increases the computation performance.
```

(continues on next page)

(continued from previous page)

```
In both experiments pseudo results are stored in a shared object.

You will learn:

1) How to implement an own custom task

2) How to store results in a shared object

3) How to add tasks to a workflow

4) How to run tasks and workflows in various ranges of asynchronicity

"""



from time import sleep
from mlpro.bf.various import Log
import mlpro.bf.mt as mt
from datetime import datetime, timedelta
from cmath import pi, sin, cos, tan
import random



## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------
class MyTask (mt.Task):
    """
    Demo implementation of a task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'My task'

## -------------------------------------------------------------------------------------------
    def __init__( self,
                  p_duration:timedelta,
                  p_name:str=None,
                  p_range_max: int = mt.Task.C_RANGE_THREAD,
                  p_autorun=mt.Task.C_AUTORUN_NONE,
                  p_class_shared=None,
                  p_visualize:bool=False,
                  p_logging=Log.C_LOG_ALL ):

        super().__init__( p_name=p_name,
                          p_range_max=p_range_max,
                          p_autorun=p_autorun,
```

(continues on next page)

(continued from previous page)

```
                          p_class_shared=p_class_shared,
                          p_visualize=p_visualize,
                          p_logging=p_logging )

        self._duration = p_duration


## ------------------------------------------------------------------------------------
↪-----------
    def _run(self, **p_kwargs):

        tid = self.get_tid()

        # 1 Dummy implementation to simulate a busy sub-task
        time_start = datetime.now()
        result     = 0

        while True:
            # do something meaningful
            for i in range(500):
                result += sin(random.random()*pi) * cos(random.random()*pi) * tan(random.
↪random()*pi)

            time_current = datetime.now()
            time_diff    = time_current - time_start
            if time_diff >= self._duration: break

        # 3 Sub-task can optionally store resuls in the shared object.
        self._so.add_result(p_tid=self.get_name(), p_result=result)




# 1 Preparation of execution
if __name__ == '__main__':
    # 1.1 Preparation for demo mode
    duration    = timedelta(0,1,0)
    pause_sec   = 5
    logging     = Log.C_LOG_ALL

else:
    # 1.2 Preparation for unit test mode
    num_tasks   = 2
    duration    = timedelta(0,0,10000)
    pause_sec   = 0
    logging     = Log.C_LOG_NOTHING



# 2 Creation of a workflow with 9 tasks within
```

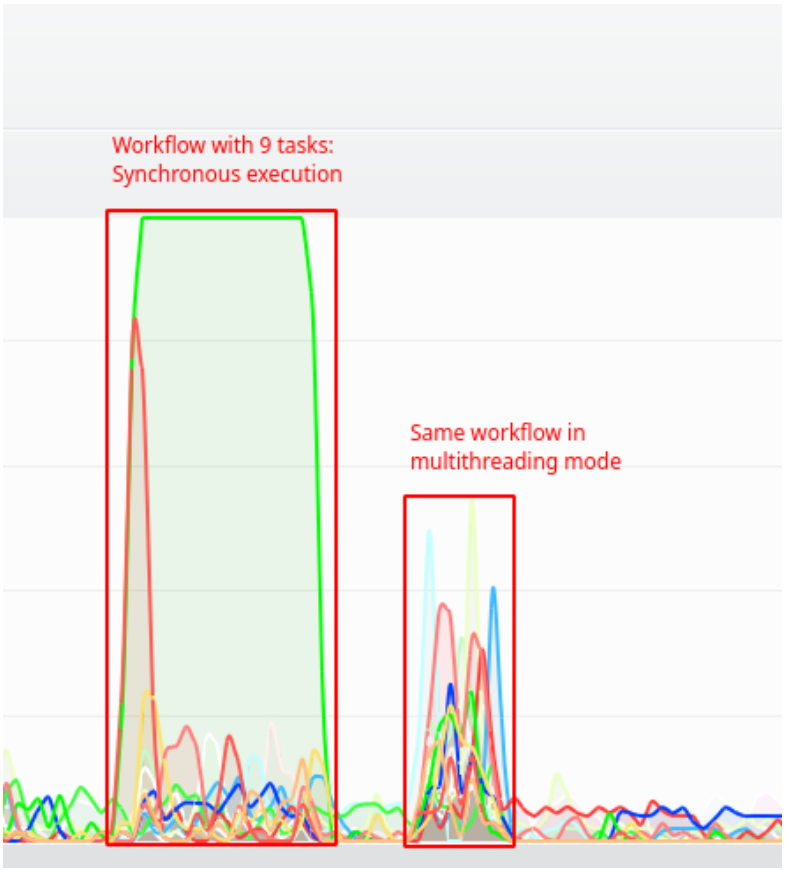
(continues on next page)

(continued from previous page)

```
# 2.1 Creation of 9 tasks
t1a = MyTask( p_duration=duration, p_name='t1a', p_logging=logging )
t1b = MyTask( p_duration=duration, p_name='t1b', p_logging=logging )
t1c = MyTask( p_duration=duration, p_name='t1c', p_logging=logging )

t2a = MyTask( p_duration=duration, p_name='t2a', p_logging=logging )
t2b = MyTask( p_duration=duration, p_name='t2b', p_logging=logging )
t2c = MyTask( p_duration=duration, p_name='t2c', p_logging=logging )

t3a = MyTask( p_duration=duration, p_name='t3a', p_logging=logging )
t3b = MyTask( p_duration=duration, p_name='t3b', p_logging=logging )
t3c = MyTask( p_duration=duration, p_name='t3c', p_logging=logging )

# 2.2 Create a workflow and add the tasks
wf = mt.Workflow( p_name='wf1',
                  p_range_max=mt.Workflow.C_RANGE_THREAD,
                  p_class_shared=mt.Shared,
                  p_logging=logging )

# 2.2.1 At first we add three tasks that build the starting points of our workflow
wf.add_task( p_task=t1a )
wf.add_task( p_task=t1b )
wf.add_task( p_task=t1c )

# 2.2.2 Then, we add three further tasks that shall start when their predecessor tasks␣
↪have finished
wf.add_task( p_task=t2a, p_pred_tasks=[t1a] )
wf.add_task( p_task=t2b, p_pred_tasks=[t1b] )
wf.add_task( p_task=t2c, p_pred_tasks=[t1c] )

# 2.2.3 Finally, we add three further tasks that build the end of our task chains
wf.add_task( p_task=t3a, p_pred_tasks=[t2a, t2b, t2c] )
wf.add_task( p_task=t3b, p_pred_tasks=[t2a, t2b, t2c] )
wf.add_task( p_task=t3c, p_pred_tasks=[t2a, t2b, t2c] )



# 3 Run the workflow synchronously
wf.run( p_range=mt.Workflow.C_RANGE_NONE, p_wait=True )
wf.log(Log.C_LOG_TYPE_I, 'Result in shared object:\n', wf.get_so().get_results())


# 4 Clear result list in shared object and wait for next run (for better observation)
wf.get_so().clear_results()
wf.log(Log.C_LOG_TYPE_W, 'Short break for better observation of CPU load in perfmeter')
sleep(pause_sec)



# 5 Run the same workflow asynchronously in multithreading mode
wf.run( p_range=mt.Workflow.C_RANGE_THREAD, p_wait=True)
wf.log(Log.C_LOG_TYPE_I, 'Result in shared object:\n', wf.get_so().get_results())
```

Results

The howto example logs details of the two runs (workflow synchronously/multithreading). A short break between the workflow runs allows a better observation of CPU load in the system monitor.



Cross References

- *API Reference: Multiprocessing*

8.1.3 Layer 2 - Mathematics

Howto BF-MATH-001: Dimensions, Spaces and Elements

Ver. 1.1.1 (2023-03-02)

This module demonstrates how to create a space and subspaces and to spawn elements.

You will learn:

1. How to use the space, subspace, dimension and elements classes of MLPro in native and custom implementations.
2. How to create a Space object and how to add dimensions to the Space.
3. How to create an Element in a defined space, i.e. element with values for number of dimensions for the related space.
4. How to change values of an Element object.
5. How to calculate the euclidean distance between two elements of given Space.

Prerequisites

Please install following packages to run this howto

- Numpy

Executable code

```
## -------------------------------------------------------------------------------------------
-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_math_001_spaces_and_elements.py
## -------------------------------------------------------------------------------------------
-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-05-28  1.0.0     DA       Creation/Release
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library␣
↪name
## -- 2021-09-23  1.0.1     DA       Adaption to changes in class Element
## -- 2021-12-03  1.0.2     DA       New method copy_append_spaces()
## -- 2022-02-25  1.0.3     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-12-09  1.1.0     DA       Refactoring due to new restrictions in class Set
## -- 2023-03-02  1.1.1     LSB      Refactoring
## -------------------------------------------------------------------------------------------
-----------

"""
Ver. 1.1.1 (2023-03-02)

This module demonstrates how to create a space and subspaces and to spawn elements.

You will learn:

1. How to use the space, subspace, dimension and elements classes of MLPro in native and␣
↪custom implementations.

2. How to create a Space object and how to add dimensions to the Space.

3. How to create an Element in a defined space, i.e. element with values for number of␣
↪dimensions for the related space.

4. How to change values of an Element object.

5. How to calculate the euclidean distance between two elements of given Space.
"""


from mlpro.bf.various import Log
from mlpro.bf.math import *
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------
class MathDemo(Log):

    C_TYPE      = 'Demo'
    C_NAME      = 'Spaces & Elements'

    # Some constants for dimension indices to make it more understandable
    C_POS       = 0
    C_VEL       = 1
    C_ACC       = 2
    C_ANG       = 3
    C_AVEL      = 4
    C_AACC      = 5

## ------------------------------------------------------------------------------------------
↪-----------
    def __init__(self, p_logging=True):
        super().__init__(p_logging=p_logging)
        self.create_euclidian_space()
        self.create_subspace1()
        self.create_subspace2()
        self.create_subspace3()
        self.create_element()
        self.change_elem_values()
        self.calculate_distance()


## ------------------------------------------------------------------------------------------
↪-----------
    def create_euclidian_space(self):
        self.espace = ESpace()
        self.espace.add_dim(Dimension( p_name_short='Pos', p_name_long='Position', p_
↪unit='m', p_unit_latex='m', p_boundaries=[0,100]))
        self.espace.add_dim(Dimension( p_name_short='Vel', p_name_long='Velocity', p_
↪unit='m/s', p_unit_latex='\frac{m}{s}', p_boundaries=[-100,100]))
        self.espace.add_dim(Dimension( p_name_short='Acc', p_name_long='Acceleration', p_
↪unit='m/qs', p_unit_latex='\frac{m}{s^2}', p_boundaries=[-100,100]))
        self.espace.add_dim(Dimension( p_name_short='Ang', p_name_long='Angle', p_unit=
↪'deg', p_unit_latex='deg', p_boundaries=[-45,45]))
        self.espace.add_dim(Dimension( p_name_short='AVel', p_name_long='Angle Velocity',
↪ p_unit='deg/s', p_unit_latex='\frac{deg}{s}', p_boundaries=[-100,100]))
        self.espace.add_dim(Dimension( p_name_short='AAcc', p_name_long='Angle␣
↪Acceleration', p_unit='deg/qs', p_unit_latex='\frac{deg}{s^2}', p_boundaries=[-100,
↪100]))

        _ids = self.espace.get_dim_ids()
        self.C_POS       = _ids[0]
        self.C_VEL       = _ids[1]
        self.C_ACC       = _ids[2]
```

(continues on next page)

(continued from previous page)

```
        self.C_ANG       = _ids[3]
        self.C_AVEL      = _ids[4]
        self.C_AACC      = _ids[5]

        self.log(self.C_LOG_TYPE_I, '6-dimensional Euclidian space created')


## -------------------------------------------------------------------------------------
↪-----------
    def create_subspace1(self):
        self.subspace1 = self.espace.spawn([self.C_POS, self.C_VEL, self.C_ACC])
        self.log(self.C_LOG_TYPE_I, 'Subspace 1 - Number of dimensions and short name of␣
↪second dimension:', self.subspace1.get_num_dim(), self.subspace1.get_dim(self.C_VEL).
↪get_name_short())


## -------------------------------------------------------------------------------------
↪-----------
    def create_subspace2(self):
        self.subspace2 = self.espace.spawn([self.C_ANG, self.C_AVEL, self.C_AACC])
        self.log(self.C_LOG_TYPE_I, 'Subspace 2 - Number of dimensions and short name of␣
↪third dimension:', self.subspace2.get_num_dim(), self.subspace2.get_dim(self.C_AACC).
↪get_name_short())


## -------------------------------------------------------------------------------------
↪-----------
    def create_subspace3(self):
        self.subspace3 = self.espace.spawn([self.C_POS, self.C_ANG])
        self.log(self.C_LOG_TYPE_I, 'Subspace 3 - Number of dimensions and short name of␣
↪second dimension:', self.subspace3.get_num_dim(), self.subspace3.get_dim(self.C_ANG).
↪get_name_short())


## -------------------------------------------------------------------------------------
↪-----------
    def create_element(self):
        self.elem = Element(self.espace)
        self.log(self.C_LOG_TYPE_I, 'New element created with dim ids / values:', self.
↪elem.get_dim_ids(), ' / ', self.elem.get_values())


## -------------------------------------------------------------------------------------
↪-----------
    def change_elem_values(self):
        # Changing a value indexed by a unique dimension index...
        self.elem.set_value(self.C_POS, 4.77)
        self.elem.set_value(self.C_VEL, -8.22)
        self.log(self.C_LOG_TYPE_I, 'Element changed to ', self.elem.get_values())


## -------------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-----------
    def calculate_distance(self):
        e1 = Element(self.espace)
        e2 = Element(self.espace)
        e2.set_value(self.C_AACC,1)
        self.log(self.C_LOG_TYPE_I, 'New element e1 =', e1.get_values())
        self.log(self.C_LOG_TYPE_I, 'New element e2 =', e2.get_values())
        self.log(self.C_LOG_TYPE_I, 'Euclidian distance between e1 and e2 =', self.
↪espace.distance(e1,e2))



if __name__ == "__main__":
    demo = MathDemo()
```

Results

```
2023-02-10  16:00:33.045206  I  Demo "Spaces & Elements": Instantiated
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": 6-dimensional Euclidian space created
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": Subspace 1 - Number of dimensions and short name of second dimension: 3 Vel
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": Subspace 2 - Number of dimensions and short name of third dimension: 3 AAcc
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": Subspace 3 - Number of dimensions and short name of second dimension: 2 Ang
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": New element created with dim ids / values: ['f593bb88-1dc4-48a5-b5a2-dc7a55ef7c20', '891247bf-1539-4b10-b91c-ac2832b15a86'
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": Element changed to  [ 4.77 -8.22  0.    0.    0.    0.  ]
2023-02-10  16:00:33.046206  I  Demo "Spaces & Elements": New element e1 = [0. 0. 0. 0. 0. 0.]
2023-02-10  16:00:33.047204  I  Demo "Spaces & Elements": New element e2 = [0. 0. 0. 0. 0. 1.]
2023-02-10  16:00:33.047204  I  Demo "Spaces & Elements": Euclidian distance between e1 and e2 = 1.0
```

Cross Reference

- *API Reference: Math*

Howto BF-MATH-010: Normalizers

Ver. 1.0.4 (2022-11-03) Example file for demonstrating the use of MLPro's normalizer for normalizing and denormalizing data.

You will learn:

1. How to update parameters for Normalization
2. How to update parameters based on single element/data
3. How to normalize a data element (ndarray/mlpro element) by MinMax or ZTransofrm
4. How to denormalize a data element (ndarray/mlpro element) by MinMax or ZTransofrm
5. How to renormalize the data element (ndarray/mlpro element) with respect to the changed parameters

Prerequisites

Please install following packages to run this howto

- Numpy

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_math_010_normalizers.py
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-09-16  0.0.0     LSB      Creation
## -- 2022-09-25  1.0.0     LSB      Release of first version
## -- 2022-10-01  1.0.1     LSB      Renormalization
## -- 2022-10-06  1.0.2     LSB      Refactoring
## -- 2022-10-16  1.0.3     LSB      Updating z-transform parameters based on a new data/
↪element(np.ndarray)
## -- 2022-11-03  1.0.4     LSB      refacoring for update with replaced data (a-
↪transformer)
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.4 (2022-11-03)
Example file for demonstrating the use of MLPro's normalizer for normalizing and de-
↪normalizing data.


You will learn:

1. How to update parameters for Normalization

2. How to update parameters based on single element/data

3. How to normalize a data element (ndarray/mlpro element) by MinMax or ZTransofrm

4. How to denormalize a data element (ndarray/mlpro element) by MinMax or ZTransofrm

5. How to renormalize the data element (ndarray/mlpro element) with respect to the␣
↪changed parameters
"""
import numpy as np

from mlpro.bf.math.normalizers import *



# checking for internal unit tests
if __name__ == '__main__':
    p_printing = True
else:
    p_printing = False




# Creating Numpy dummy Dataset
my_dataset = np.asarray(([45,-7,65,-87],[21.3,47.1,-41.02,89],[0.12,98.11,11,-56.01],[7.
↪12,55.01,4.78,5.3],
```

(continues on next page)

(continued from previous page)

```
                        [-44.371,-0.521,14.12,8.5],[77.13,-23.14,-7.54,12.32],[8.1,27.61,-
↪31.01,17.8],
                        [4.22,-84.21,47.12,82.11],[-53.22,1.024,5.044,71.23],[9.4,-4.39,
↪12.51,83.01]))


# Creating a dummy set with dummy dimensions
my_set = Set()
my_set.add_dim(Dimension(p_name_short='1', p_boundaries=[10,19]))
my_set.add_dim(Dimension(p_name_short='2', p_boundaries=[-9,10]))


# Creating a dummy element to normalize
my_state = Element(my_set)
my_state.set_values([19,8])


# Creating Normalizer object
my_normalizer_minmax = NormalizerMinMax()
my_normalizer_ztrans = NormalizerZTrans()


# 1. Setting parameters for NormalizationZTrans
my_normalizer_ztrans.update_parameters(p_dataset = my_dataset)
if p_printing:
    print('01. Parameters updated for the Z transformer\n\n')


# 2. Normalizing a numpy array/ a dataset (as an array) in Z transformation
normalized_data = my_normalizer_ztrans.normalize(p_data=my_dataset)
if p_printing:
    print('02. Normalized value(Z transformer):\n', normalized_data,'\n\n')


# 3. De-normalizing a numpy array/ a dataset (as an array) in Z transformation
denormalized_data = my_normalizer_ztrans.denormalize(p_data=normalized_data)
if p_printing:
    print('03. Denormalized value (Z transformer):\n', denormalized_data,'\n\n')


# 4. Updating the parameters using a new element to the dataset
new_data = np.asarray([12,-71,74,-12], dtype=np.float64).reshape(1,4)
my_normalizer_ztrans.update_parameters(p_data_new = new_data)
if p_printing:
    print('04. Parameters updated for the Z transformer\n\n')


# 5. Normalizing the new element with new parameters
normalize_new = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n05. Normalized Data (Z transformer):', normalize_new,'\n\n')
```

(continues on next page)

(continued from previous page)

```
# 6. Validating the changed parameters
#    6.1 Adding the new element in the dataset
my_dataset = np.append(my_dataset, new_data, axis=0)
#    6.2 Setting up the parameters based on the new dataset
my_normalizer_ztrans.update_parameters(p_dataset=my_dataset)
#    6.3 Normalizing the element for validation
normalized_val = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n06. Normalized Data (validation Z transformer): ', normalized_val, '\n\n')


# 7. Updating parameters with replaced data

p_data_old = my_dataset[1].copy()
my_dataset[1] = np.asarray([4.1,7.8,-41, 15.3], dtype=np.float64).reshape(1,4)
my_normalizer_ztrans.update_parameters(p_data_new=np.asarray([4.1,7.8,-41, 15.3],␣
↪dtype=np.float64).reshape(1,4),
                                       p_data_del=p_data_old)
if p_printing:
    print('\n07. Normalization parameters updated for z-transformer based on replaced␣
↪data')


# 8. Normalizing the new element with new parameters
normalize_new = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n08. Normalized Data (Z transformer):', normalize_new,'\n\n')



# 9. Validating the updated parameters
#   9.1. Updating parameters based on the new dataset
my_normalizer_ztrans.update_parameters(p_dataset=my_dataset)
normalized_val = my_normalizer_ztrans.normalize(new_data)
if p_printing:
    print('\n09. Normalized Data (validation Z transformer): ', normalized_val, '\n\n')



# 10. Setting parameters for Normalization
my_normalizer_minmax.update_parameters(p_set = my_set)
if p_printing:
    print('10. Parameters updated for the MinMax Normalizer\n\n')


# 11. Normalizing using MinMax
normalized_state = my_normalizer_minmax.normalize(my_state)
if p_printing:
    print('11. Normalized value (MinMax Normalizer):\n', normalized_state.get_values(),'\
↪n\n')
```

(continues on next page)

(continued from previous page)

```
# 12. De-normalizing using MinMAx
denormalized_state = my_normalizer_minmax.denormalize(normalized_state)
if p_printing:
    print('12. Denormalized value (MinMax Normalizer):\n', denormalized_state.get_
↪values(),'\n\n')


# 13. Updating the boundaries of the dimension
my_set.get_dim(p_id=my_set.get_dim_ids()[0]).set_boundaries([-10,51])
my_set.get_dim(p_id=my_set.get_dim_ids()[1]).set_boundaries([-5,10])
if p_printing:
    print('13. Boundaries updated (MinMax Normalizer)\n\n')


# 14. Updating tbe normalization parameters for the new set
my_normalizer_minmax.update_parameters(my_set)
if p_printing:
    print('14. Parameters updated for MinMax normalizer\n\n')


# 15. Renormalizing the previously normalized data with the new parameters
re_normalized_state = my_normalizer_minmax.renormalize(normalized_state)
if p_printing:
    print('15. Renormalized value (MinMax Normalizer):\n', re_normalized_state.get_
↪values(),'\n\n')


# 16. Validating the renormalization
normalized_state = my_normalizer_minmax.normalize(my_state)
if p_printing:
    print('16. Normalized value (Validation renormalization):\n', normalized_state.get_
↪values(),'\n\n')
```

Results

The results will be available as follows

```
01. Parameters updated for the Z transformer


02. Normalized value(Z transformer):
 [[ 1.04497494 -0.38178705  1.89666962 -1.91464488]
 [ 0.38490458  0.7682956  -1.63116063  1.15923813]
 [-0.20498109  1.85268962  0.09981211 -1.37339696]
 [-0.0100236   0.93645002 -0.10715926 -0.30260282]
 [-1.44410306 -0.24405349  0.20363054 -0.24671404]
 [ 1.93982982 -0.72489859 -0.51710897 -0.17999681]
 [ 0.01727045  0.35396823 -1.29807649 -0.08428727]
 [-0.0907917  -2.0231527   1.30171013  1.0389026 ]
 [-1.69055718 -0.21120917 -0.09837462  0.84888074]
 [ 0.05347684 -0.32630247  0.15005757  1.05462132]]
```

(continues on next page)

(continued from previous page)

```
03. Denormalized value (Z transformer):
 [[ 45.     -7.     65.    -87.   ]
 [ 21.3    47.1   -41.02   89.   ]
 [  0.12   98.11   11.    -56.01 ]
 [  7.12   55.01    4.78    5.3  ]
 [-44.371  -0.521  14.12    8.5  ]
 [ 77.13  -23.14   -7.54   12.32 ]
 [  8.1    27.61  -31.01   17.8  ]
 [  4.22  -84.21   47.12   82.11 ]
 [-53.22    1.024   5.044  71.23 ]
 [  9.4    -4.39   12.51   83.01 ]]


04. Parameters updated for the Z transformer



05. Normalized Data (Z transformer): [[ 0.11994467 -1.47066196  1.74588644 -0.56725513]]



06. Normalized Data (validation Z transformer):  [[ 0.11994467 -1.47066196  1.74588644 -
↪0.56725513]]



07. Normalization parameters updated for z-transformer based on replaced data

08. Normalized Data (Z transformer): [[ 0.16683367 -1.45314853  1.7459814  -0.48625054]]



09. Normalized Data (validation Z transformer):  [[ 0.16683367 -1.45314853  1.7459814  -
↪0.48625054]]


10. Parameters updated for the MinMax Normalizer


11. Normalized value (MinMax Normalizer):
 [1.         0.78947368]


12. Denormalized value (MinMax Normalizer):
 [19.  8.]


13. Boundaries updated (MinMax Normalizer)
```

(continues on next page)

(continued from previous page)

```
14. Parameters updated for MinMax normalizer


15. Renormalized value (MinMax Normalizer):
 [2.60655738 9.86666667]


16. Normalized value (Validation renormalization):
 [-0.58667025  0.98222222]
```

Cross References

- *API Reference: Normalizers*

8.1.4 Layer 3 - Application Support

Stream Processing

Howto BF-STREAMS-001: Accessing Native Data From MLPro

Ver. 1.0.2 (2023-02-02)

This module demonstrates the use of native generic data streams provided by MLPro. To this regard, all data streams of the related provider class will be determined and iterated.

You will learn:

1) How to access MLPro's native data streams.
2) How to iterate the instances of a native stream.
3) How to access feature data of a native stream.

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_001_accessing_native_data_from_mlpro.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-11-08  1.0.0     DA       Creation
## -- 2022-12-14  1.0.1     DA       Corrections
## -- 2023-02-02  1.0.2     DA       Correction of time measurement
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.2 (2023-02-02)

This module demonstrates the use of native generic data streams provided by MLPro. To␣
```

(continues on next page)

(continued from previous page)

```
↪this regard,
all data streams of the related provider class will be determined and iterated.

You will learn:

1) How to access MLPro's native data streams.

2) How to iterate the instances of a native stream.

3) How to access feature data of a native stream.

"""


from datetime import datetime
from mlpro.bf.streams.streams import *
from mlpro.bf.various import Log



# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging     = Log.C_LOG_ALL
else:
    logging     = Log.C_LOG_NOTHING


# 1 Create a Wrapper for OpenML stream provider
mlpro = StreamProviderMLPro(p_logging=logging)


# 2 Determine native data streams provided by MLPro
for stream in mlpro.get_stream_list( p_logging=logging ):
    stream.switch_logging( p_logging=logging )
    try:
        labels = stream.get_label_space().get_num_dim()
    except:
        labels = 0

    stream.log(Log.C_LOG_TYPE_W, 'Features:', stream.get_feature_space().get_num_dim(),
↪', Labels:', labels, ', Instances:', stream.get_num_instances() )

if __name__ == '__main__':
    input('\nPress ENTER to iterate all streams dark...\n')


# 3 Performance test: iterate all data streams dark and measure the time
for stream in mlpro.get_stream_list( p_logging=logging ):
    stream.switch_logging( p_logging=logging )
    stream.log(Log.C_LOG_TYPE_W, 'Number of instances:', stream.get_num_instances() )
    stream.switch_logging( p_logging=Log.C_LOG_NOTHING )
```

(continues on next page)

(continued from previous page)

```
    # 3.1 Iterate all instances of the stream
    tp_start = datetime.now()
    myiterator = iter(stream)
    for i, curr_instance in enumerate(myiterator):
        curr_data = curr_instance.get_feature_data().get_values()

    tp_end       = datetime.now()
    duration     = tp_end - tp_start
    duration_sec = ( duration.seconds * 1000000 + duration.microseconds + 1 ) / 1000000
    rate         = myiterator.get_num_instances() / duration_sec

    myiterator.switch_logging( p_logging=logging )
    myiterator.log(Log.C_LOG_TYPE_W, 'Done in', round(duration_sec,3), ' seconds␣
↪(throughput =', round(rate), 'instances/sec)')
```

Results

```
2023-02-11  22:40:18.898725  I  Stream Provider "MLPro": Instantiated
2023-02-11  22:40:18.898725  I  Stream "Random 10D x 1000": Instantiated
2023-02-11  22:40:18.898725  I  Stream "Double Spiral 2D x 721": Instantiated
2023-02-11  22:40:18.898725  I  Stream "Static Clouds 2D": Instantiated
2023-02-11  22:40:18.898725  I  Stream "Static Clouds 3D": Instantiated
2023-02-11  22:40:18.898725  I  Stream Provider "MLPro": Getting list of streams...
2023-02-11  22:40:18.898725  I  Stream Provider "MLPro": Number of streams found: 4
2023-02-11  22:40:18.898725  W  Stream "Random 10D x 1000": Features: 10 , Labels: 2 ,␣
↪Instances: 1000
2023-02-11  22:40:18.898725  W  Stream "Double Spiral 2D x 721": Features: 2 , Labels: 0␣
↪, Instances: 721
2023-02-11  22:40:18.898725  W  Stream "Static Clouds 2D": Features: 2 , Labels: 0 ,␣
↪Instances: 1000
2023-02-11  22:40:18.898725  W  Stream "Static Clouds 3D": Features: 3 , Labels: 0 ,␣
↪Instances: 2000

Press ENTER to iterate all streams dark...

2023-02-11  22:53:21.191034  I  Stream Provider "MLPro": Getting list of streams...
2023-02-11  22:53:21.191034  I  Stream Provider "MLPro": Number of streams found: 4
2023-02-11  22:53:21.191034  W  Stream "Random 10D x 1000": Number of instances: 1000
2023-02-11  22:53:21.222301  W  Stream "Random 10D x 1000": Done in 0.031  seconds␣
↪(throughput = 31982 instances/sec)
2023-02-11  22:53:21.222301  W  Stream "Double Spiral 2D x 721": Number of instances: 721
2023-02-11  22:53:21.222301  W  Stream "Double Spiral 2D x 721": Done in 0.0  seconds␣
↪(throughput = 721000000 instances/sec)
2023-02-11  22:53:21.222301  W  Stream "Static Clouds 2D": Number of instances: 1000
2023-02-11  22:53:21.237922  W  Stream "Static Clouds 2D": Done in 0.016  seconds␣
↪(throughput = 64012 instances/sec)
2023-02-11  22:53:21.237922  W  Stream "Static Clouds 3D": Number of instances: 2000
2023-02-11  22:53:21.275686  W  Stream "Static Clouds 3D": Done in 0.038  seconds␣
↪(throughput = 52959 instances/sec)
```

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-051: Accessing Data from OpenML

Prerequisites

Please install the following packages to run this examples properly:

- OpenML
- Numpy

Executable code

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-052: Accessing Data from Scikit-Learn

Ver. 1.1.1 (2023-02-02)

This module demonstrates the use of Scikit-learn datasets as streams in MLPro. To this regard, MLPro provides wrapper classes to standardize stream access in own ML applications.

You will learn:

1) How to access datasets of the Scikit-learn project.
2) How to iterate the instances of an Scikit-learn stream.
3) How to access feature and label data of a data stream.

Prerequisites

Please install the following packages to run this examples properly:

- Scikit-Learn
- Numpy

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_052_accessing_data_from_scikitlearn.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-06-16  0.0.0     LSB      Creation
## -- 2022-06-16  1.0.0     LSB      Release of first version
## -- 2022-06-18  1.0.1     LSB      Restructured logging output
## -- 2022-06-25  1.0.2     LSB      Refactoring for new label and instance class
## -- 2022-10-12  1.0.3     DA       Renaming
## -- 2022-11-08  1.1.0     DA       Refactoring after changes on class Stream
## -- 2023-02-02  1.1.1     DA       Correction of time measurement
## -------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
"""
Ver. 1.1.1 (2023-02-02)

This module demonstrates the use of Scikit-learn datasets as streams in MLPro. To this␣
↪regard, MLPro
provides wrapper classes to standardize stream access in own ML applications.

You will learn:

1) How to access datasets of the Scikit-learn project.

2) How to iterate the instances of an Scikit-learn stream.

3) How to access feature and label data of a data stream.

"""


from mlpro.wrappers.sklearn import *
from mlpro.bf.various import Log




## 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    num_inst    = 10
    logging     = Log.C_LOG_ALL
else:
    print('\n', datetime.now(), __file__)
    num_inst    = 2
    logging     = Log.C_LOG_NOTHING


# 1 Create a Wrapper for OpenML stream provider
sk_learn = WrStreamProviderSklearn(p_logging=logging)


# 2 Get a list of streams available at the stream provider
stream_list = sk_learn.get_stream_list(p_logging=logging)


# 3 Get a specific stream from the stream provider
mystream = sk_learn.get_stream( p_name='iris', p_logging=logging)


# 4 Get the feature space of the stream
feature_space = mystream.get_feature_space()
sk_learn.log(mystream.C_LOG_TYPE_I,"Number of features in the stream:",feature_space.get_
↪num_dim(),'\n\n')
```

(continues on next page)

(continued from previous page)

```
# 5 Set up an iterator for the stream
myiterator = iter(mystream)


# 6 Fetching some stream instances
myiterator.log(mystream.C_LOG_TYPE_W,'Fetching first', str(num_inst), 'stream instances..
↪.')
for i in range(num_inst):
    curr_instance   = next(myiterator)
    curr_data       = curr_instance.get_feature_data().get_values()
    curr_label      = curr_instance.get_label_data().get_values()
    myiterator.log(mystream.C_LOG_TYPE_I, 'Instance', str(i) + ': \n   Data:', curr_
↪data[0:14], '...\n   Label:', curr_label)


# 7 Resetting the iterator
myiterator = iter(mystream)


# 8 Fetching all 150 instances
myiterator.log(mystream.C_LOG_TYPE_W,'Fetching all 150 instances...')
for i, curr_instance in enumerate(myiterator):
    if i == num_inst:
        myiterator.log(Log.C_LOG_TYPE_W, 'Rest of the 150 instances dark...')
        myiterator.switch_logging(p_logging=Log.C_LOG_NOTHING)
        tp_start = datetime.now()

    curr_data       = curr_instance.get_feature_data().get_values()
    curr_label      = curr_instance.get_label_data().get_values()
    myiterator.log(mystream.C_LOG_TYPE_I, 'Instance', str(i) + ': \n   Data:', curr_
↪data[0:14], '...\n   Label:', curr_label)

# 8.1 Some statistics...
tp_end = datetime.now()
duration = tp_end - tp_start
duration_sec = ( duration.seconds * 1000000 + duration.microseconds + 1 ) / 1000000
rate = ( mystream.get_num_instances() - num_inst ) / duration_sec

myiterator.switch_logging(p_logging=logging)
myiterator.log(Log.C_LOG_TYPE_W, 'Done in', round(duration_sec,3), ' seconds (throughput␣
↪=', round(rate), 'instances/sec)')
```

Results

```
2023-02-11  22:51:43.599295  I  Wrapper "Stream Provider Scikit-learn": Instantiated
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Wrapped package␣
↪scikit-learn installed in version 1.0.2
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Getting list of␣
↪streams...
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Number of␣
↪streams found: 10
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Name of␣
```

(continues on next page)

(continued from previous page)

```
↪requested stream: iris
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Getting list of␣
↪streams...
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Number of␣
↪streams found: 20
2023-02-11  22:51:43.897836  I  Stream "Sklearn stream "iris"": Ready to access in mode 0
2023-02-11  22:51:43.897836  I  Wrapper "Stream Provider Scikit-learn": Number of␣
↪features in the stream: 4


2023-02-11  22:51:43.897836  I  Stream "Sklearn stream "iris"": Reset
2023-02-11  22:51:43.897836  W  Stream "Sklearn stream "iris"": Fetching first 10 stream␣
↪instances...
2023-02-11  22:51:43.897836  I  Stream "Sklearn stream "iris"": Instance 0:
   Data: [5.1 3.5 1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 1:
   Data: [4.9 3.  1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 2:
   Data: [4.7 3.2 1.3 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 3:
   Data: [4.6 3.1 1.5 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 4:
   Data: [5.  3.6 1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 5:
   Data: [5.4 3.9 1.7 0.4] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 6:
   Data: [4.6 3.4 1.4 0.3] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 7:
   Data: [5.  3.4 1.5 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 8:
   Data: [4.4 2.9 1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 9:
   Data: [4.9 3.1 1.5 0.1] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Reset
2023-02-11  22:51:43.913475  W  Stream "Sklearn stream "iris"": Fetching all 150␣
↪instances...
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Reset
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 0:
   Data: [5.1 3.5 1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 1:
   Data: [4.9 3.  1.4 0.2] ...
```

(continues on next page)

(continued from previous page)

```
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 2:
   Data: [4.7 3.2 1.3 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 3:
   Data: [4.6 3.1 1.5 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 4:
   Data: [5.  3.6 1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 5:
   Data: [5.4 3.9 1.7 0.4] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 6:
   Data: [4.6 3.4 1.4 0.3] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 7:
   Data: [5.  3.4 1.5 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 8:
   Data: [4.4 2.9 1.4 0.2] ...
   Label: [0]
2023-02-11  22:51:43.913475  I  Stream "Sklearn stream "iris"": Instance 9:
   Data: [4.9 3.1 1.5 0.1] ...
   Label: [0]
2023-02-11  22:51:43.913475  W  Stream "Sklearn stream "iris"": Rest of the 150␣
↪instances dark...
2023-02-11  22:51:43.913475  W  Stream "Sklearn stream "iris"": Done in 0.0  seconds␣
↪(throughput = 140000000 instances/sec)
```

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-053: Accessing Data from River

Ver. 1.1.4 (2023-02-02)

This module demonstrates the use of River datasets as streams in MLPro. To this regard, MLPro provides wrapper classes to standardize stream access in own ML applications.

You will learn:

1) How to access datasets of the River project.
2) How to iterate the instances of an River stream.
3) How to access feature and label data of a data stream.

Prerequisites

Please install the following packages to run this examples properly:

- river
- Numpy

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_053_accessing_data_from_river.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-06-14  0.0.0     LSB      Creation
## -- 2022-06-14  1.0.0     LSB      Release of first version
## -- 2022-06-25  1.0.1     LSB      Refactoring for new label and instance class
## -- 2022-10-12  1.0.2     DA       Renaming
## -- 2022-11-07  1.1.0     DA       Refactoring after changes on class Stream
## -- 2022-11-08  1.1.1     DA       Minor improvements
## -- 2022-11-19  1.1.2     DA       Get string by name
## -- 2022-11-21  1.1.3     DA       Correction on logging
## -- 2023-02-02  1.1.4     DA       Correction of time measurement
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.1.4 (2023-02-02)

This module demonstrates the use of River datasets as streams in MLPro. To this regard,␣
↪MLPro
provides wrapper classes to standardize stream access in own ML applications.

You will learn:

1) How to access datasets of the River project.

2) How to iterate the instances of an River stream.

3) How to access feature and label data of a data stream.

"""


from datetime import datetime
from mlpro.wrappers.river import *
from mlpro.bf.various import Log




# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    num_inst    = 10
    logging     = Log.C_LOG_ALL
else:
```

(continues on next page)

(continued from previous page)

```
    print('\n', datetime.now(), __file__)
    num_inst    = 2
    logging     = Log.C_LOG_NOTHING


# 1 Create a Wrapper for River stream provider
river  = WrStreamProviderRiver(p_logging=logging)


# 2 Get a list of streams available at the stream provider
stream_list = river.get_stream_list(p_logging = logging)


# 3 Get stream "Bikes" from the stream provider
mystream = river.get_stream(p_name='Bikes', p_logging=logging)


# 4 Get the feature space of the stream
feature_space = mystream.get_feature_space()
river.log(mystream.C_LOG_TYPE_I,"Number of features in the stream:",feature_space.get_
↪num_dim())


# 5 Set up an iterator for the stream
myiterator = iter(mystream)


# 6 Fetching some stream instances
myiterator.log(mystream.C_LOG_TYPE_W,'Fetching first', str(num_inst), 'stream instances..
↪.')
for i in range(num_inst):
    curr_instance   = next(myiterator)
    curr_data       = curr_instance.get_feature_data().get_values()
    curr_label      = curr_instance.get_label_data().get_values()
    myiterator.log(mystream.C_LOG_TYPE_I, 'Instance', str(i) + ': \n   Data:', curr_
↪data[0:14], '...\n   Label:', curr_label)


# 7 Resetting the iterator
myiterator = iter(mystream)


# 8 Fetching all 182,470 instances
myiterator.log(mystream.C_LOG_TYPE_W,'Fetching all 182,470 instances...')
for i, curr_instance in enumerate(myiterator):
    if i == num_inst:
        myiterator.log(Log.C_LOG_TYPE_W, 'Rest of the 182,470 instances dark...')
        myiterator.switch_logging(p_logging=Log.C_LOG_NOTHING)
        tp_start = datetime.now()

    curr_data       = curr_instance.get_feature_data().get_values()
    curr_label      = curr_instance.get_label_data().get_values()
```

(continues on next page)

(continued from previous page)

```
    myiterator.log(mystream.C_LOG_TYPE_I, 'Instance', str(i) + ': \n   Data:', curr_
↪data[0:14], '...\n   Label:', curr_label)

# 8.1 Some statistics...
tp_end = datetime.now()
duration = tp_end - tp_start
duration_sec = ( duration.seconds * 1000000 + duration.microseconds + 1 ) / 1000000
rate = ( myiterator.get_num_instances() - num_inst ) / duration_sec

myiterator.switch_logging(p_logging=logging)
myiterator.log(Log.C_LOG_TYPE_W, 'Done in', round(duration_sec,3), ' seconds (throughput␣
↪=', round(rate), 'instances/sec)')
```

Results

```
2023-02-11  22:53:16.342309  I  Wrapper "Stream Provider Scikit-learn": Instantiated
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Wrapped package␣
↪scikit-learn installed in version 1.0.2
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Getting list of␣
↪streams...
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Number of␣
↪streams found: 10
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Name of␣
↪requested stream: iris
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Getting list of␣
↪streams...
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Number of␣
↪streams found: 20
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Ready to access in mode 0
2023-02-11  22:53:16.593570  I  Wrapper "Stream Provider Scikit-learn": Number of␣
↪features in the stream: 4


2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Reset
2023-02-11  22:53:16.593570  W  Stream "Sklearn stream "iris"": Fetching first 10 stream␣
↪instances...
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 0:
   Data: [5.1 3.5 1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 1:
   Data: [4.9 3.  1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 2:
   Data: [4.7 3.2 1.3 0.2] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 3:
   Data: [4.6 3.1 1.5 0.2] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 4:
   Data: [5.  3.6 1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 5:
```

(continues on next page)

(continued from previous page)

```
   Data: [5.4 3.9 1.7 0.4] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 6:
   Data: [4.6 3.4 1.4 0.3] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 7:
   Data: [5.  3.4 1.5 0.2] ...
   Label: [0]
2023-02-11  22:53:16.593570  I  Stream "Sklearn stream "iris"": Instance 8:
   Data: [4.4 2.9 1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 9:
   Data: [4.9 3.1 1.5 0.1] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Reset
2023-02-11  22:53:16.609203  W  Stream "Sklearn stream "iris"": Fetching all 150␣
↪instances...
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Reset
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 0:
   Data: [5.1 3.5 1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 1:
   Data: [4.9 3.  1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 2:
   Data: [4.7 3.2 1.3 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 3:
   Data: [4.6 3.1 1.5 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 4:
   Data: [5.  3.6 1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 5:
   Data: [5.4 3.9 1.7 0.4] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 6:
   Data: [4.6 3.4 1.4 0.3] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 7:
   Data: [5.  3.4 1.5 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 8:
   Data: [4.4 2.9 1.4 0.2] ...
   Label: [0]
2023-02-11  22:53:16.609203  I  Stream "Sklearn stream "iris"": Instance 9:
   Data: [4.9 3.1 1.5 0.1] ...
   Label: [0]
2023-02-11  22:53:16.609203  W  Stream "Sklearn stream "iris"": Rest of the 150␣
↪instances dark...
2023-02-11  22:53:16.609203  W  Stream "Sklearn stream "iris"": Done in 0.0  seconds␣
↪(throughput = 140000000 instances/sec)
```

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-101: Basics of Streams

Ver. 1.0.0 (2022-12-14)

This module demonstrates the principles of stream processing with MLPro. To this regard, a stream of a stream provider is combined with a stream workflow to a stream scenario. The workflow consists of a custom task only. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_101_basics.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-27  0.0.0     DA       Creation
## -- 2022-12-14  1.0.0     DA       First implementation
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.0 (2022-12-14)

This module demonstrates the principles of stream processing with MLPro. To this regard,␣
↪a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow␣
↪consists of
a custom task only. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.
```

(continues on next page)

(continued from previous page)

```
"""


from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *




## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -------------------------------------------------------------------------------------------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass




## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo'

## -------------------------------------------------------------------------------------------
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a stream from OpenML
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('StaticClouds3D', p_logging=p_logging)

        # 2 Set up a stream workflow
```

(continues on next page)

(continued from previous page)

```
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 2.1 Set up and add an own custom task
        task = MyTask( p_name='t1', p_visualize=p_visualize, p_logging=logging )
        workflow.add_task( p_task=task )

        # 3 Return stream and workflow
        return stream, workflow



# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 721
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                         p_cycle_limit=cycle_limit,
                         p_visualize=visualize,
                         p_logging=logging )


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-102: Tasks Workflows And Stream Scenarios

Ver. 1.0.0 (2022-11-22)

This module demonstrates the principles of stream processing with MLPro. To this regard, stream tasks are added to a stream workflow. This in turn is combined with a stream of a stream provider to a a stream scenario. The latter one can be executed.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_001_tasks_workflows_and_stream_scenarios.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-27  0.0.0     DA       Creation
## -- 2022-11-22  1.0.0     DA       First implementation
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.0 (2022-11-22)

This module demonstrates the principles of stream processing with MLPro. To this regard,␣
↪stream tasks
are added to a stream workflow. This in turn is combined with a stream of a stream␣
↪provider to a
a stream scenario. The latter one can be executed.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.

"""
```

(continues on next page)

(continued from previous page)

```
from mlpro.bf.streams import *
from mlpro.wrappers.openml import WrStreamProviderOpenML



## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'My stream task'


## -------------------------------------------------------------------------------------------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass



## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Nine tasks'

## -------------------------------------------------------------------------------------------
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a stream from OpenML
        openml  = WrStreamProviderOpenML(p_logging=p_logging)
        stream  = openml.get_stream(p_id=75, p_mode=p_mode, p_logging=p_logging)


        # 2 Set up a stream workflow based on a custom stream task

        # 2.1 Creation of 9 tasks
```

(continues on next page)

(continued from previous page)

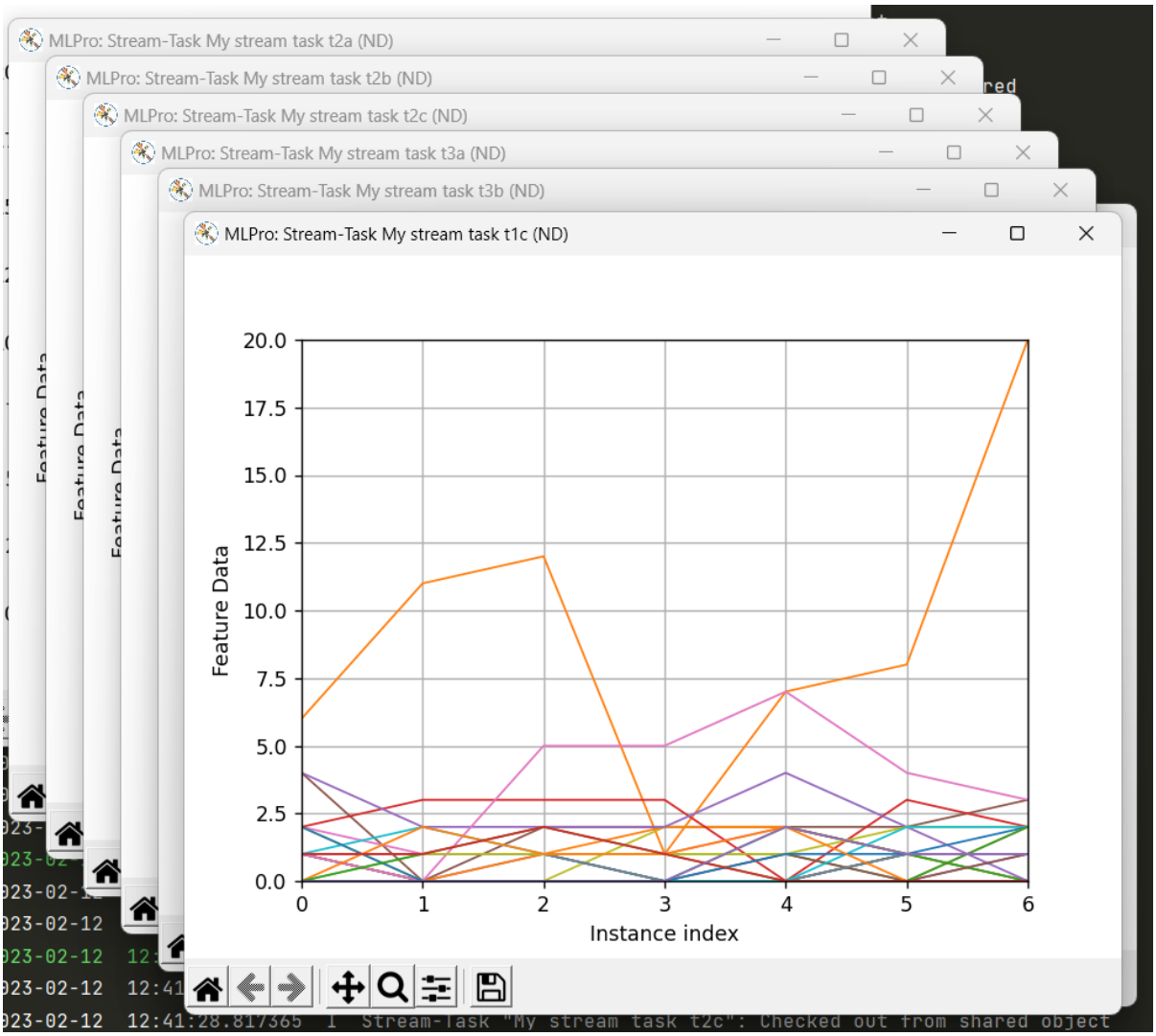
```
        t1a = MyTask( p_name='t1a', p_visualize=p_visualize, p_logging=logging )
        t1b = MyTask( p_name='t1b', p_visualize=p_visualize, p_logging=logging )
        t1c = MyTask( p_name='t1c', p_visualize=p_visualize, p_logging=logging )

        t2a = MyTask( p_name='t2a', p_visualize=p_visualize, p_logging=logging )
        t2b = MyTask( p_name='t2b', p_visualize=p_visualize, p_logging=logging )
        t2c = MyTask( p_name='t2c', p_visualize=p_visualize, p_logging=logging )

        t3a = MyTask( p_name='t3a', p_visualize=p_visualize, p_logging=logging )
        t3b = MyTask( p_name='t3b', p_visualize=p_visualize, p_logging=logging )
        t3c = MyTask( p_name='t3c', p_visualize=p_visualize, p_logging=logging )

        # 2.2 Create a workflow and add the tasks
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
↪#StreamWorkflow.C_RANGE_THREAD,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 2.2.1 At first we add three tasks that build the starting points of our␣
↪workflow
        workflow.add_task( p_task=t1a )
        workflow.add_task( p_task=t1b )
        workflow.add_task( p_task=t1c )

        # 2.2.2 Then, we add three further tasks that shall start when their predecessor␣
↪tasks have finished
        workflow.add_task( p_task=t2a, p_pred_tasks=[t1a] )
        workflow.add_task( p_task=t2b, p_pred_tasks=[t1b] )
        workflow.add_task( p_task=t2c, p_pred_tasks=[t1c] )

        # 2.2.3 Finally, we add three further tasks that build the end of our task chains
        workflow.add_task( p_task=t3a, p_pred_tasks=[t2a, t2b, t2c] )
        workflow.add_task( p_task=t3b, p_pred_tasks=[t2a, t2b, t2c] )
        workflow.add_task( p_task=t3c, p_pred_tasks=[t2a, t2b, t2c] )


        # 3 Return stream and workflow
        return stream, workflow




# 1 Preparation of demo/unit test mode
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    cycle_limit = 10
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 1.2 Parameters for internal unit test
```

(continues on next page)

(continued from previous page)

```
    cycle_limit = 2
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                         p_cycle_limit=cycle_limit,
                         p_visualize=visualize,
                         p_logging=logging )


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results



Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-110: Window

Ver. 1.0.1 (2022-12-14)

This module demonstrates the functionality of stream window task in MLPro.

You will learn:

1) How to implement an own custom stream task.
2) How to set up a stream workflow based on stream tasks.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to run a stream scenario dark or with default visualization.

Prerequisites

Please install the following packages to run this example properly:

- Numpy
- Matplotlib

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_110_stream_task_window.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-11-27  1.0.0     LSB      Creation
## -- 2022-12-14  1.1.0     DA       - Changed the stream provider from OpenML to MLPro
## --                                  - Added a custom task behind the window task
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.1 (2022-12-14)

This module demonstrates the functionality of stream window task in MLPro.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with default visualization.
"""


from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Window



## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------------
↪-----------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass



## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class MyStreamScenario(StreamScenario):

    C_NAME      = 'Demo Window'

## -------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_visualize:bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪logging)


        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf-window',
                                   p_range_max=StreamWorkflow.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging)

        # 2.1 Set up and add a window task
        task_window = Window( p_buffer_size=30,
                              p_name = 't1',
                              p_delay = True,
                              p_visualize = p_visualize,
                              p_enable_statistics = True )
        workflow.add_task(task_window)

        # 2.2 Set up and add an own custom task
        task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=logging )
        workflow.add_task( p_task=task_custom, p_pred_tasks=[task_window] )


        # 3 Return stream and workflow
        return stream, workflow



if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    # 1.1 Parameters for demo mode
    cycle_limit = 100
    logging = Log.C_LOG_ALL
    visualize = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging = Log.C_LOG_NOTHING
    visualize = False


# 2 Instantiate the stream scenario
myscenario = MyStreamScenario(p_mode=Mode.C_MODE_REAL,
    p_cycle_limit=cycle_limit,
    p_visualize=visualize,
    p_logging=logging)


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot()
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-111: Rearranger (2D)

Ver. 1.0.0 (2022-12-14)

This module demonstrates the principles of stream processing with MLPro. To this regard, a stream of a stream provider is combined with a stream workflow to a stream scenario. The workflow consists of a standard task Rearranger and a custom task. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.
2) How to set up a stream workflow based on stream tasks.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to run a stream scenario dark or with visualization.

Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_111_stream_task_rearranger_2d.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-27  0.0.0     DA       Creation
## -- 2022-12-14  1.0.0     DA       First implementation
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.0 (2022-12-14)

This module demonstrates the principles of stream processing with MLPro. To this regard,␣
↪a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow␣
↪consists of
a standard task Rearranger and a custom task. The stream scenario is used to process␣
↪some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

"""


from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger



## -------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------------
↪-----------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -------------------------------------------------------------------------------------------
↪-----------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass



## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See␣
↪class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Rearranger'

## -------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 2.1 Set up and add a rearranger task to reduce the feature and label space
        features     = stream.get_feature_space().get_dims()
        features_new = [ ( 'F', features[1:3] ) ]

        task_rearranger = Rearranger( p_name='t1',
                                      p_range_max=Task.C_RANGE_THREAD,
                                      p_visualize=p_visualize,
```

(continues on next page)

(continued from previous page)

```
                                        p_logging=p_logging,
                                        p_features_new=features_new )

        workflow.add_task( p_task=task_rearranger )

        # 2.2 Set up and add an own custom task
        task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=logging )
        workflow.add_task( p_task=task_custom, p_pred_tasks=[task_rearranger] )


        # 3 Return stream and workflow
        return stream, workflow




# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                         p_cycle_limit=cycle_limit,
                         p_visualize=visualize,
                         p_logging=logging )


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                        p_step_rate = 2 ) )
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-112: Rearranger (3D)

Ver. 1.0.1 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard, a stream of a stream provider is combined with a stream workflow to a stream scenario. The workflow consists of a standard task Rearranger and a custom task. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.
2) How to set up a stream workflow based on stream tasks.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to run a stream scenario dark or with visualization.

Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_112_stream_task_rearranger_3d.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-27  0.0.0     DA       Creation
## -- 2022-12-14  1.0.0     DA       First implementation
## -- 2023-02-07  1.0.1     SY       Refactoring module name
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.1 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard,␣
↪a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow␣
↪consists of
a standard task Rearranger and a custom task. The stream scenario is used to process␣
↪some instances.

You will learn:
```

(continues on next page)

(continued from previous page)

```
1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

"""


from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger



## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -------------------------------------------------------------------------------------------
↪-----------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass



## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See␣
↪class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Rearranger'

## -------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=p_logging )

        # 2.1 Set up and add a rearranger task to reduce the feature and label space
        features     = stream.get_feature_space().get_dims()
        features_new = [ ( 'F', features[1:4] ) ]

        task_rearranger = Rearranger( p_name='t1',
                                      p_range_max=Task.C_RANGE_THREAD,
                                      p_visualize=p_visualize,
                                      p_logging=p_logging,
                                      p_features_new=features_new )

        workflow.add_task( p_task=task_rearranger )

        # 2.2 Set up and add an own custom task
        task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=p_logging )
        workflow.add_task( p_task=task_custom, p_pred_tasks=[task_rearranger] )


        # 3 Return stream and workflow
        return stream, workflow




# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 2 Instantiate the stream scenario
```

(continues on next page)

(continued from previous page)

```
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                         p_cycle_limit=cycle_limit,
                         p_visualize=visualize,
                         p_logging=logging )


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                        p_step_rate = 2 ) )
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-113: Rearranger (nD)

Ver. 1.0.1 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard, a stream of a stream provider is combined with a stream workflow to a stream scenario. The workflow consists of a standard task Rearranger and a custom task. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.
2) How to set up a stream workflow based on stream tasks.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to run a stream scenario dark or with visualization.

Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
```

(continues on next page)

(continued from previous page)

```
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_113_stream_task_rearranger_nd.py
## -------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-27  0.0.0     DA       Creation
## -- 2022-12-14  1.0.0     DA       First implementation
## -- 2023-02-07  1.0.1     SY       Refactoring module name
## -------------------------------------------------------------------------------------

"""
Ver. 1.0.1 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard, a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow consists of
a standard task Rearranger and a custom task. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to run a stream scenario dark or with visualization.

"""


from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger



## -------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------------
↪-----------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass



## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See␣
↪class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Rearranger'

## -------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('Rnd10Dx1000', p_mode=p_mode, p_logging=p_
↪logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 2.1 Set up and add a rearranger task to reduce the feature and label space
        features     = stream.get_feature_space().get_dims()
        labels       = stream.get_label_space().get_dims()

        features_new = [ ( 'F', [ features[1] ] ),
                         ( 'L', [ labels[1] ] ),
                         ( 'F', features[5:8] ) ]
        labels_new   = [ ( 'L', [ labels[0] ] ),
                         ( 'F', features[4:6] ) ]

        task_rearranger = Rearranger( p_name='t1',
                                      p_range_max=Task.C_RANGE_THREAD,
                                      p_visualize=p_visualize,
                                      p_logging=p_logging,
                                      p_features_new=features_new,
                                      p_labels_new=labels_new )
```

(continues on next page)

(continued from previous page)

```
        workflow.add_task( p_task=task_rearranger )

        # 2.2 Set up and add an own custom task
        task_custom = MyTask( p_name='t2', p_visualize=p_visualize, p_logging=logging )
        workflow.add_task( p_task=task_custom, p_pred_tasks=[task_rearranger] )


        # 3 Return stream and workflow
        return stream, workflow




# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                         p_cycle_limit=cycle_limit,
                         p_visualize=visualize,
                         p_logging=logging )


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                        p_step_rate = 2 ) )
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results

Cross Reference

- *API Reference: Streams*

Howto BF-STREAMS-114: Deriver

Ver. 1.2.0 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard, a stream of a stream provider is combined with a stream workflow to a stream scenario. The workflow consists of a standard task Deriver and a custom task. The stream scenario is used to process some instances.

You will learn:

1) How to implement an own custom stream task.
2) How to set up a stream workflow based on stream tasks.
3) How to set up a stream scenario based on a stream and a processing stream workflow.
4) How to add a task Deriver and how to extend the features.
5) How to run a stream scenario dark or with visualization.

Prerequisites

Please install the following packages to run this example properly:

- Matplotlib
- Tkinter

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_streams_114_stream_task_deriver.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-02-02  0.0.0     SY       Creation
## -- 2023-02-05  1.0.0     SY       First version release
## -- 2023-02-07  1.1.0     SY       Change the dataset to doublespiral2d
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.2.0 (2023-02-07)

This module demonstrates the principles of stream processing with MLPro. To this regard,␣
↪a stream of
a stream provider is combined with a stream workflow to a stream scenario. The workflow␣
↪consists of
a standard task Deriver and a custom task. The stream scenario is used to process some␣
↪instances.

You will learn:
```

(continues on next page)

(continued from previous page)

```
1) How to implement an own custom stream task.

2) How to set up a stream workflow based on stream tasks.

3) How to set up a stream scenario based on a stream and a processing stream workflow.

4) How to add a task Deriver and how to extend the features.

5) How to run a stream scenario dark or with visualization.

"""


from mlpro.bf.streams import *
from mlpro.bf.streams.streams import *
from mlpro.bf.streams.tasks import Rearranger, Deriver



## -------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------
↪-----------
class MyTask (StreamTask):
    """
    Demo implementation of a stream task with custom method _run().
    """

    # needed for proper logging (see class mlpro.bf.various.Log)
    C_NAME      = 'Custom'

## -------------------------------------------------------------------------------------
↪-----------
    def _run(self, p_inst_new: list, p_inst_del: list):
        pass



## -------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------
↪-----------
class MyScenario (StreamScenario):
    """
    Example of a custom stream scenario including a stream and a stream workflow. See␣
↪class
    mlpro.bf.streams.models.StreamScenario for further details and explanations.
    """

    C_NAME      = 'Demo Deriver'
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_visualize: bool, p_logging):

        # 1 Import a native stream from MLPro
        provider_mlpro = StreamProviderMLPro(p_logging=p_logging)
        stream = provider_mlpro.get_stream('DoubleSpiral2D', p_mode=p_mode, p_logging=p_
↪logging)

        # 2 Set up a stream workflow
        workflow = StreamWorkflow( p_name='wf1',
                                   p_range_max=Task.C_RANGE_NONE,
                                   p_visualize=p_visualize,
                                   p_logging=logging )

        # 2.1 Set up and add a rearranger task to reduce the feature and label space
        features = stream.get_feature_space().get_dims()
        features_new = [ ( 'F', features[0:1] ) ]

        task_rearranger = Rearranger( p_name='t1',
                                      p_range_max=Task.C_RANGE_THREAD,
                                      p_visualize=p_visualize,
                                      p_logging=p_logging,
                                      p_features_new=features_new )

        workflow.add_task( p_task=task_rearranger )

        # 2.2 Set up and add a deriver task to extend the feature and label space (1st␣
↪derivative)
        features = task_rearranger._feature_space.get_dims()
        derived_feature = features[0]

        task_deriver_1 = Deriver( p_name='t2',
                                  p_range_max=Task.C_RANGE_THREAD,
                                  p_visualize=p_visualize,
                                  p_logging=p_logging,
                                  p_features=features,
                                  p_label=None,
                                  p_derived_feature=derived_feature,
                                  p_derived_label=None,
                                  p_order_derivative=1 )

        workflow.add_task( p_task=task_deriver_1, p_pred_tasks=[task_rearranger] )

        # 2.3 Set up and add a deriver task to extend the feature and label space (2nd␣
↪derivative)
        features = task_deriver_1._feature_space.get_dims()
        derived_feature = features[0]

        task_deriver_2 = Deriver( p_name='t3',
                                  p_range_max=Task.C_RANGE_THREAD,
                                  p_visualize=p_visualize,
```

(continues on next page)

(continued from previous page)

```
                                    p_logging=p_logging,
                                    p_features=features,
                                    p_label=None,
                                    p_derived_feature=derived_feature,
                                    p_derived_label=None,
                                    p_order_derivative=2 )

        workflow.add_task( p_task=task_deriver_2, p_pred_tasks=[task_rearranger, task_
↪deriver_1] )

        # 3 Return stream and workflow
        return stream, workflow




# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    cycle_limit = 200
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 1.2 Parameters for internal unit test
    cycle_limit = 2
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 2 Instantiate the stream scenario
myscenario = MyScenario( p_mode=Mode.C_MODE_SIM,
                         p_cycle_limit=cycle_limit,
                         p_visualize=visualize,
                         p_logging=logging )


# 3 Reset and run own stream scenario
myscenario.reset()

if __name__ == '__main__':
    myscenario.init_plot( p_plot_settings=PlotSettings( p_view = PlotSettings.C_VIEW_ND,
                                                        p_step_rate = 2 ) )
    input('Press ENTER to start stream processing...')

myscenario.run()

if __name__ == '__main__':
    input('Press ENTER to exit...')
```

Results

Cross Reference

- *API Reference: Deriver*

Physics

Howto BF-PHYSICS-001: Transfer Functions

Ver. 1.0.5 (2023-02-04)

This module provides an example of using the transfer function method in MLPro for both default and custom implementation.

You will learn:

1) How to use the default type of transfer function (linear function)

2) How to set up your own function

Prereauistes

Please install following packages to run this how to

- Matplotlib

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_physics_001_set_up_transfer_functions.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-08-24  0.0.0     SY/ML    Creation
## -- 2022-11-22  0.0.1     SY       Shift from mlpro repo to mlpro-mpps repo
## -- 2022-11-22  1.0.0     SY/ML    Release of first version
## -- 2023-01-14  1.0.1     SY       Shift from mlpro-at_basis.bf to mlpro.bf
## -- 2023-01-15  1.0.2     SY       Package renaming
## -- 2023-01-16  1.0.3     SY       Update due to __call__
## -- 2023-01-24  1.0.4     SY       Quality Assurance on TransferFunction
## -- 2023-02-04  1.0.5     SY       Shift UnitConverter from bf.systems to bf.physics
## -------------------------------------------------------------------------------------
↪-----------


"""
Ver. 1.0.5 (2023-02-04)

This module provides an example of using the transfer function method in MLPro for both␣
↪default and
custom implementation.

You will learn:
```

(continues on next page)

(continued from previous page)

```
1) How to use the default type of transfer function (linear function)

2) How to set up your own function

"""


from mlpro.bf.math import *
from mlpro.bf.physics import *
import math



if __name__ == "__main__":
    p_print = True
    p_visualize = True
else:
    p_print = False
    p_visualize = False


# 1. Using default type

# 1.1. Initialize a given default transfer function
myTF_linear = TransferFunction(p_name='Linear_TF',
                               p_type=TransferFunction.C_TRF_FUNC_LINEAR,
                               p_dt=0.01,
                               m=5,
                               b=2)


# 1.2. Call the defined transfer function
# 1.2.1. For a specific point
p_input = 10
output = myTF_linear(p_input)
if p_print:
    print(output)

# 1.2.2. Within a specific range
p_range = 5
output = myTF_linear(p_input, p_range)
if p_print:
    print(output)

# 1.3. Plot the graph
if p_visualize:
    myTF_linear.plot(p_input, p_input+p_range)


# 2. Using own function

# 2.1. Set up your custom function class
```

(continues on next page)

(continued from previous page)

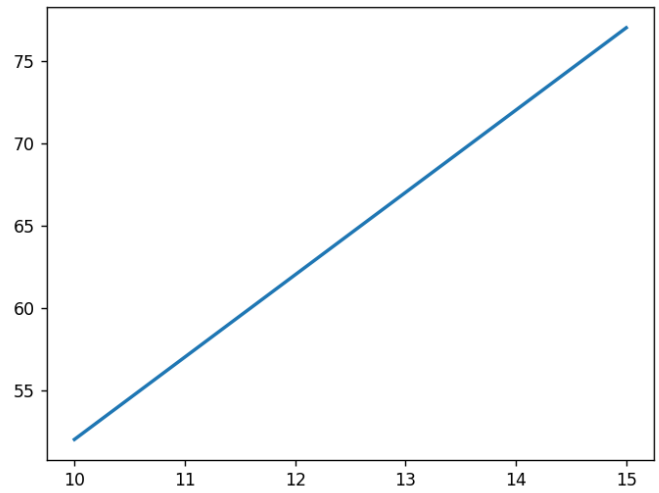
```
class MyTransferFunction(TransferFunction):

    # 2.1.1. Set up which parameters required for your transfer function
    def _set_function_parameters(self, p_args) -> bool:
        """
        y(t) = A cos(w * t - phi)
        """
        if self.get_type() == self.C_TRF_FUNC_CUSTOM:
            try:
                self.A = p_args['A']
                self.w = p_args['w']
                self.phi = p_args['phi']
            except:
                raise NotImplementedError('One/More parameters for this function is␣
↪missing.')
        return True

    # 2.1.2. Set up the mathematical calculation of your transfer function
    def _custom_function(self, p_input, p_range=None):
        """
        y(t) = A cos(w * t - phi)
        """
        if p_range is None:
            return self.A * math.cos(self.w * p_input - self.phi)
        else:
            points = int(p_range/self.dt)
            output = 0
            for x in range(points+1):
                current_input = p_input + x * self.dt
                output += self.A * math.cos(self.w * current_input - self.phi)
            return output

# 2.2. Initialize the transfer function
myFunction = MyTransferFunction(p_name='DGL_solution',
                                p_type=TransferFunction.C_TRF_FUNC_CUSTOM,
                                p_dt=0.05,
                                A = 3.5,          # Current
                                w = 314.15,       # angular velocity
                                phi = -120)       # angle offset


# 2.3. Call the defined transfer function
# 2.3.1. For a specific point
p_input = 0
output = myFunction(p_input)
if p_print:
    print(output)

# 2.3.2. Within a specific range
p_range = 10
output = myFunction(p_input, p_range)
if p_print:
```

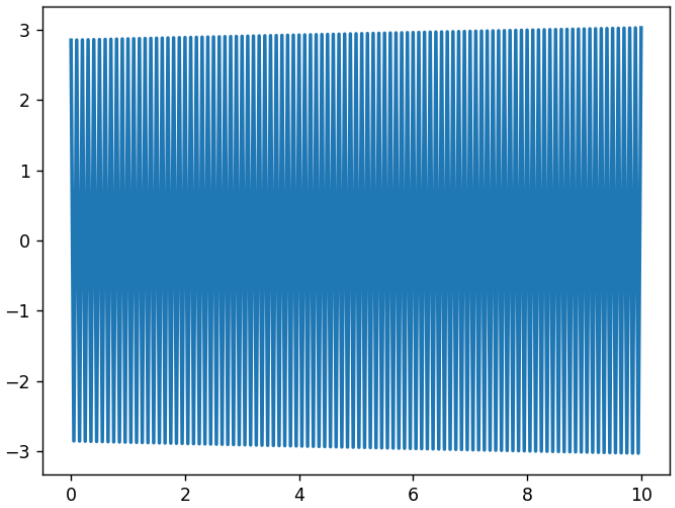
(continues on next page)

(continued from previous page)

```
    print(output)

# 2.4. Plot the graph
if p_visualize:
    myFunction.plot(p_input, p_input+p_range)
```

Results





Cross Reference

- API Reference: Physics

Howto BF-PHYSICS-002: Unit Converter

Ver. 1.0.2 (2023-02-04)

This module provides an example of using the unit converter in MLPro.

You will learn:

1) How to use the the unit converter

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_physics_002_unit_converter.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-01-15  0.0.0     SY       Creation
## -- 2023-01-15  1.0.0     SY       Release of first version
## -- 2023-01-16  1.0.1     SY       Renaming and debugging
## -- 2023-02-04  1.0.2     SY       Shift UnitConverter from bf.math to bf.physics
## -------------------------------------------------------------------------------------------
↪-----------


"""
Ver. 1.0.2 (2023-02-04)

This module provides an example of using the unit converter in MLPro.

You will learn:

1) How to use the the unit converter

"""


from mlpro.bf.physics.unitconverter import UnitConverter
from mlpro.bf.various import Log



if __name__ == "__main__":
    p_print = True
else:
    p_print = False

# 1 Initialize unit converters
conv_length = UnitConverter(p_name='conv_length',
                            p_type=UnitConverter.C_UNIT_CONV_LENGTH,
                            p_unit_in='m',
```

(continues on next page)

(continued from previous page)

```
                              p_unit_out='km')

conv_pressure = UnitConverter(p_name='conv_pressure',
                              p_type=UnitConverter.C_UNIT_CONV_PRESSURE,
                              p_unit_in='bar',
                              p_unit_out='Pa')

conv_current = UnitConverter(p_name='conv_current',
                             p_type=UnitConverter.C_UNIT_CONV_CURRENT,
                             p_unit_in='mA',
                             p_unit_out='A')

conv_force = UnitConverter(p_name='conv_force',
                           p_type=UnitConverter.C_UNIT_CONV_FORCE,
                           p_unit_in='N',
                           p_unit_out='J/cm')

conv_power = UnitConverter(p_name='conv_power',
                           p_type=UnitConverter.C_UNIT_CONV_POWER,
                           p_unit_in='W',
                           p_unit_out='kW')

conv_mass = UnitConverter(p_name='conv_mass',
                          p_type=UnitConverter.C_UNIT_CONV_MASS,
                          p_unit_in='kg',
                          p_unit_out='lb')

conv_time = UnitConverter(p_name='conv_time',
                          p_type=UnitConverter.C_UNIT_CONV_TIME,
                          p_unit_in='hr',
                          p_unit_out='s')

conv_temperature = UnitConverter(p_name='conv_temperature',
                                 p_type=UnitConverter.C_UNIT_CONV_TEMPERATURE,
                                 p_unit_in='K',
                                 p_unit_out='F')


# 2. Call the defined unit converters
conv_set = [conv_length,
            conv_pressure,
            conv_current,
            conv_force,
            conv_power,
            conv_mass,
            conv_time,
            conv_temperature]
p_input = 10

for conv in conv_set:
    output = conv(p_input)
    if p_print:
```

(continues on next page)

(continued from previous page)

```
        print('We convert %.1f%s to %.2f%s'%(p_input, conv._unit_in, output, conv._unit_
↪out))
```

Results

```
2023-02-10  17:54:42.123738  I  TransferFunction "Linear_TF": Instantiated
52
32314.5
2023-02-10  17:55:14.475198  I  TransferFunction "DGL_solution": Instantiated
2.8496333968429663
2.937498276638952
```

Cross Reference

- API Reference: Physics

State-based Systems

Howto BF-SYSTEMS-001: System, Controller, Actuator, Sensor

Ver. 1.1.0 (2022-12-09)

This module demonstrates the principles of using classes System, Controller, Actuator and Sensor. To this regard we assume a custom system with two state and action components and a custom controller that represents the hardware pendant with two sensors and actuators.

You will learn:

1) How to set up an own state based system.
2) How to implement an own controller.
3) How to assign states to sensors and actions to actuators.
4) How to communicate with sensors and actuators using a controller.

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_systems_001_systems_controllers_actuators_sensors.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-12-05  1.0.0     DA       Creation
## -- 2022-12-09  1.1.0     DA       Simplification
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.1.0 (2022-12-09)
```

(continues on next page)

(continued from previous page)

```
This module demonstrates the principles of using classes System, Controller, Actuator␣
↪and Sensor. To
this regard we assume a custom system with two state and action components and a custom␣
↪controller
that represents the hardware pendant with two sensors and actuators.

You will learn:

1) How to set up an own state based system.

2) How to implement an own controller.

3) How to assign states to sensors and actions to actuators.

4) How to communicate with sensors and actuators using a controller.

"""


from mlpro.bf.various import Log
from mlpro.bf.systems import *
import random



class MyController (Controller):

    C_NAME      = 'Dummy'

    def _reset(self) -> bool:
        self.log(Log.C_LOG_TYPE_S, 'Pseudo-reset of the controller')
        return True


    def _get_sensor_value(self, p_id):
        """
        Pseudo-implementation for getting a sensor value.
        """
        self.log(Log.C_LOG_TYPE_S, 'Pseudo-import of a sensor value...')
        return random.random()


    def _set_actuator_value(self, p_id, p_value) -> bool:
        self.log(Log.C_LOG_TYPE_S, 'Pseudo-export of an actuator value:', str(p_value))
        return True



class MySystem (System):
```

(continues on next page)

(continued from previous page)

```
    C_NAME      = 'Dummy'

    @staticmethod
    def setup_spaces():

        # 1 State space
        state_space = ESpace()
        state_space.add_dim( p_dim = Dimension( p_name_short='State 1') )
        state_space.add_dim( p_dim = Dimension( p_name_short='State 2') )

        # 2 Action space
        action_space = ESpace()
        action_space.add_dim( p_dim = Dimension( p_name_short='Action 1') )
        action_space.add_dim( p_dim = Dimension( p_name_short='Action 2') )

        return state_space, action_space


    def _reset(self, p_seed=None) -> None:
        pass




# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging     = Log.C_LOG_ALL
    latency     = timedelta(0,1,0)
else:
    logging     = Log.C_LOG_NOTHING
    latency     = timedelta(0,0,100000)


# 1 Instantiate own system in simulation mode
sys = MySystem( p_latency=latency, p_logging=logging )


# 2 Instantiate and configure own controller
con = MyController( p_id=0, p_name='2x2', p_logging=logging )

s1 = Sensor(p_name_short='Sensor 1')
s2 = Sensor(p_name_short='Sensor 2')
a1 = Actuator(p_name_short='Actuator 1')
a2 = Actuator(p_name_short='Actuator 2')

con.add_sensor( p_sensor=s1 )
con.add_sensor( p_sensor=s2 )
con.add_actuator( p_actuator=a1 )
con.add_actuator( p_actuator=a2 )
```

(continues on next page)

(continued from previous page)

```
# 3 Add controller to system and assign sensors to states and actuators to actions
sys.add_controller( p_controller=con,
                    p_mapping=[ ( 'S', 'State 1', 'Sensor 1' ),
                                ( 'S', 'State 2', 'Sensor 2' ),
                                ( 'A', 'Action 1', 'Actuator 1' ),
                                ( 'A', 'Action 2', 'Actuator 2') ] )


# 4 Switch system to real mode
sys.set_mode( p_mode=Mode.C_MODE_REAL )


# 5 Reset system
sys.reset()


# 6 Process an action
sys.process_action( p_action= Action( p_agent_id=0,
                                      p_action_space=sys.get_action_space(),
                                      p_values=np.array([1,2]) ) )
```

Results

```
2023-02-10  18:02:11.139147  I  System "Dummy": Instantiated
2023-02-10  18:02:11.140156  I  Controller "Dummy 2x2": Instantiated
2023-02-10  18:02:11.140156  I  System "Dummy": Adding controller "Dummy 2x2"...
2023-02-10  18:02:11.140156  I  System "Dummy": State component "State 1" assigned to␣
↪sensor "Sensor 1"
2023-02-10  18:02:11.140156  I  System "Dummy": State component "State 2" assigned to␣
↪sensor "Sensor 2"
2023-02-10  18:02:11.140156  I  System "Dummy": Action component "Action 1" assigned to␣
↪actuator "Actuator 1"
2023-02-10  18:02:11.140156  I  System "Dummy": Action component "Action 2" assigned to␣
↪actuator "Actuator 2"
2023-02-10  18:02:11.140156  I  System "Dummy": Controller "Dummy 2x2" added successfully
2023-02-10  18:02:11.140156  I  System "Dummy": Operation mode set to 1
2023-02-10  18:02:11.140156  I  System "Dummy": Reset
2023-02-10  18:02:11.140156  I  Controller "Dummy 2x2": Reset started
2023-02-10  18:02:11.140156  S  Controller "Dummy 2x2": Pseudo-reset of the controller
2023-02-10  18:02:11.140156  I  Controller "Dummy 2x2": Reset finished successfully
2023-02-10  18:02:11.141156  I  System "Dummy": Start importing state...
2023-02-10  18:02:11.141156  I  Controller "Dummy 2x2": Getting value of sensor "Sensor 1
↪"...
2023-02-10  18:02:11.141156  S  Controller "Dummy 2x2": Pseudo-import of a sensor value..
↪.
2023-02-10  18:02:11.141156  I  Controller "Dummy 2x2": Value of sensor "Sensor 1" = 0.
↪8862529182725326
2023-02-10  18:02:11.141156  I  Controller "Dummy 2x2": Getting value of sensor "Sensor 2
↪"...
2023-02-10  18:02:11.141156  S  Controller "Dummy 2x2": Pseudo-import of a sensor value..
```

(continues on next page)

(continued from previous page)

```
↪.
2023-02-10  18:02:11.141156  I  Controller "Dummy 2x2": Value of sensor "Sensor 2" = 0.
↪6398967672160553
2023-02-10  18:02:11.141156  I  System "Dummy": Assessment for success...
2023-02-10  18:02:11.141156  I  System "Dummy": Assessment for breakdown...
2023-02-10  18:02:11.141156  I  System "Dummy": Start processing action
2023-02-10  18:02:11.141156  I  System "Dummy": Actions of agent 0 = [1 2]
2023-02-10  18:02:11.142153  I  System "Dummy": Start exporting action...
2023-02-10  18:02:11.142153  I  Controller "Dummy 2x2": Setting new value of actuator
↪"Actuator 1"...
2023-02-10  18:02:11.142153  S  Controller "Dummy 2x2": Pseudo-export of an actuator␣
↪value: 1
2023-02-10  18:02:11.142153  I  Controller "Dummy 2x2": Value of actuator "Actuator 1"␣
↪set to 1
2023-02-10  18:02:11.142153  I  Controller "Dummy 2x2": Setting new value of actuator
↪"Actuator 2"...
2023-02-10  18:02:11.142153  S  Controller "Dummy 2x2": Pseudo-export of an actuator␣
↪value: 2
2023-02-10  18:02:11.142153  I  Controller "Dummy 2x2": Value of actuator "Actuator 2"␣
↪set to 2
2023-02-10  18:02:11.142153  I  System "Dummy": Waiting the system latency time of 1.0␣
↪seconds...
2023-02-10  18:02:12.158016  I  System "Dummy": Start importing state...
2023-02-10  18:02:12.158016  I  Controller "Dummy 2x2": Getting value of sensor "Sensor 1
↪"...
2023-02-10  18:02:12.158016  S  Controller "Dummy 2x2": Pseudo-import of a sensor value..
↪.
2023-02-10  18:02:12.158016  I  Controller "Dummy 2x2": Value of sensor "Sensor 1" = 0.
↪897198578560347
2023-02-10  18:02:12.158016  I  Controller "Dummy 2x2": Getting value of sensor "Sensor 2
↪"...
2023-02-10  18:02:12.158016  S  Controller "Dummy 2x2": Pseudo-import of a sensor value..
↪.
2023-02-10  18:02:12.158016  I  Controller "Dummy 2x2": Value of sensor "Sensor 2" = 0.
↪8543962410135211
2023-02-10  18:02:12.158016  I  System "Dummy": Assessment for success...
2023-02-10  18:02:12.158016  I  System "Dummy": Assessment for breakdown...
2023-02-10  18:02:12.158016  I  System "Dummy": Assessment for success...
2023-02-10  18:02:12.158016  I  System "Dummy": Assessment for breakdown...
2023-02-10  18:02:12.158016  I  System "Dummy": Action processing finished successfully
```

Cross Reference

- *API Reference: Systems*

Howto BF-SYSTEMS-002: Double Pendulum Systems wrapped with MuJoCo

Prerequisites

Please install the following packages to run this examples properly:

- MuJoCo
- lxml
- glfw

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_systems_002_doublependulum_systems_wrapped_with_mujoco.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-01-06  0.0.0     MRD       Creation
## -- 2023-01-06  1.0.0     MRD       Release
## -- 2023-02-04  1.0.1     SY        Renaming
## -- 2023-02-13  1.0.2     MRD       Refactor
## -- 2023-02-23  1.0.3     MRD       Rename file
## -- 2023-03-08  1.0.4     MRD       Remove Custom class
## -------------------------------------------------------------------------------------------
↪-----------


"""
Ver. 1.0.3 (2023-03-08)

This module demonstrates the principles of using classes System and uses MuJoCo wrapper␣
↪to simulate
the pre-defined model.

You will learn:

1) How to set up a Pendulum System wrapped with MuJoCo

2) How to run the system

"""


import mlpro
from mlpro.bf.various import Log
from mlpro.bf.systems import *
```

(continues on next page)

(continued from previous page)

```
# 0 Prepare Demo/Unit test mode
if __name__ == "__main__":
    logging     = Log.C_LOG_ALL
    visualize   = True
    loop_cycle  = 1000
else:
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    loop_cycle  = 100


# 1 Instantiate own system in simulation mode
model_file = os.path.join(os.path.dirname(mlpro.__file__), "bf/systems/pool/mujoco",
↪"doublependulum.xml")
sys = System(p_logging=logging, p_mujoco_file=model_file, p_visualize=visualize)

# 2 Reset system
sys.reset()

# 3 Process an action
for x in range(loop_cycle):
    # Random Action
    action = np.random.uniform(-1, 1, size=(1,))
    sys.process_action( p_action= Action( p_agent_id=0,
                                          p_action_space=sys.get_action_space(),
                                          p_values=action))
```

Results

The MuJoCo windows appears and shows the simulation of a pendulum system.

Cross Reference

- *API Reference: Systems*

Howto BF-SYSTEMS-003: Cartpole Continuous Systems wrapped with MuJoCo

Prerequisites

Please install the following packages to run this examples properly:

- MuJoCo
- lxml
- glfw

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
```

(continues on next page)

(continued from previous page)

```
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_systems_003_cartpole_continuous_systems_wrapped_with_mujoco.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-02-23  0.0.0     MRD       Creation
## -- 2023-02-23  1.0.0     MRD       Release
## -- 2023-03-07  1.0.1     MRD       Remove CartPoleSystem Class
## -------------------------------------------------------------------------------------
↪-----------


"""
Ver. 1.0.1 (2023-03-07)

This module demonstrates the principles of using classes System and uses MuJoCo wrapper␣
↪to simulate
the pre-defined model.

You will learn:

1) How to set up a Cartpole Continuous System wrapped with MuJoCo

2) How to run the system

"""


import mlpro
from mlpro.bf.various import Log
from mlpro.bf.systems import *




# 0 Prepare Demo/Unit test mode
if __name__ == '__main__':
    logging     = Log.C_LOG_ALL
    visualize   = True
    loop_cycle  = 1000
else:
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    loop_cycle  = 100


# 1 Instantiate own system in simulation mode
model_file = os.path.join(os.path.dirname(mlpro.__file__), "bf/systems/pool/mujoco",
↪"cartpole.xml")
sys = System(p_logging=logging, p_mujoco_file=model_file, p_visualize=visualize)
```

(continues on next page)

(continued from previous page)

```
# 2 Reset system
sys.reset()

# 3 Process an action
for x in range(loop_cycle):
    # Random Action
    action = np.random.uniform(-50, 50, size=(1,))
    sys.process_action( p_action= Action( p_agent_id=0,
                                          p_action_space=sys.get_action_space(),
                                          p_values=action ) )
```

Results

The MuJoCo windows appears and shows the simulation of a pendulum system.

Cross Reference

- *API Reference: Systems*

8.1.5 Layer 4 - Machine Learning

Basics

Howto BF-ML-001: Adaptive Model

Ver. 1.0.0 (2023-02-15)

This module demonstrates the basic properties of an adaptive model in MLPro. It also gives an overview of all custom methods that can be used for own purposes.

You will learn:

1) How to implement a custom model
2) How to let your model adapt on a dataset
3) How to run your model as a task

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_ml_001_adaptive_model.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-02-15  1.0.0     DA       Creation
## -------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
"""
Ver. 1.0.0 (2023-02-15)

This module demonstrates the basic properties of an adaptive model in MLPro. It also␣
↪gives an overview
of all custom methods that can be used for own purposes.

You will learn:

1) How to implement a custom model

2) How to let your model adapt on a dataset

3) How to run your model as a task

"""


from mlpro.bf.ml import *



## -------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------
↪-----------
class MyModel (Model):

    # Please give your implementation a suitable name...
    C_NAME              = 'Demo'

    # If you want to use the visualization features you need to activate it explicitely..
↪.
    C_PLOT_ACTIVE       = True

    # Any related scientific literature? Please add the bibliographic parameters. See␣
↪class
    # mlpro.bf.various.ScientificObject for further information...
    C_SCIREF_TYPE       = ScientificObject.C_SCIREF_TYPE_ARTICLE
    C_SCIREF_AUTHOR     = 'Detlef Arend, Mochammad Rizky Diprasetya, Steve Yuwono,␣
↪Andreas Schwung'
    C_SCIREF_TITLE      = 'MLPro - An integrative middleware framework for standardized␣
↪machine learning tasks in Python'
    C_SCIREF_JOURNAL    = 'Software Impacts'
    C_SCIREF_PUBLISHER  = 'Elsevier'
    C_SCIREF_YEAR       = '2022'
    C_SCIREF_VOLUME     = '14'
    C_SCIREF_DOI        = '10.1016/j.simpa.2022.100421'


## -------------------------------------------------------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪-----------
    def _init_hyperparam(self, p_hp_layers : int, p_hp_neurons_per_layer : int ):
        """
        Define and initialize the specific hyperparameters of your model here. See␣
↪classes Model,
        HyperParamSpace, HyperParam, HyerParamTuple of sub-package mlpro.bf.ml for␣
↪further information.

        Parameters
        ----------
        p_hp_layers : int
            Number of layers.
        p_hp_neurons_per_layer : int
            Number of neurons per layer.
        """

        # 1 Define the hyperparameter space of your model...
        self._hyperparam_space.add_dim( HyperParam( p_name_short = 'number of layers',
                                                    p_base_set = Dimension.C_BASE_SET_N,
                                                    p_boundaries=[1,5]) )

        self._hyperparam_space.add_dim( HyperParam( p_name_short = 'neurons per layer',
                                                    p_base_set = Dimension.C_BASE_SET_N,
                                                    p_boundaries=[1,20]) )

        # 2 Initialize the hyperparameters on external values
        self._hyperparam_tuple = HyperParamTuple( p_set=self._hyperparam_space )
        self._hyperparam_tuple.set_values( [p_hp_layers, p_hp_neurons_per_layer] )


## ----------------------------------------------------------------------------------------
↪-----------
    def _adapt(self, p_dataset : list) -> bool:
        """
        This custom method is intended for explicit adaptation. Here you can specify␣
↪concrete
        adaptation data parameters and implement a suitable algorithm.

        Parameters
        ----------
        p_dataset : list
            A dataset

        Returns
        -------
        adapted : bool
            True, if something has been adapted. False otherwise.
        """

        self.log(Log.C_LOG_TYPE_I, 'Incoming dataset:\n', p_dataset)
        self.log(Log.C_LOG_TYPE_I, 'My current hyperparameters are', self.get_
↪hyperparam().get_values() )
```

(continues on next page)

(continued from previous page)

```
        self.log(Log.C_LOG_TYPE_W, 'By the way: I do not really adapt something;)')
        return False


## ------------------------------------------------------------------------------------------
    def _adapt_on_event(self, p_event_id: str, p_event_object: Event) -> bool:
        """
        This custom method can be used to adapt something based on an event. The methods␣
↪needs to be
        registered as an event handler on a separate object that in turn raises the␣
↪event. See class
        mlpro.bf.events.EventHandler for further information. Context informations can␣
↪be extracted
        from the related event object in parameter p_event_object.

        Returns
        -------
        adapted : bool
            True, if something has been adapted. False otherwise.
        """

        self.log(Log.C_LOG_TYPE_I, 'Custom adaptation based on event', p_event_id)
        self.log(Log.C_LOG_TYPE_W, 'By the way: I do not really adapt something;)')
        return False


## ------------------------------------------------------------------------------------------
    def _run(self, p_input : float):
        """
        This custom method implements the operational activites of your model. Here you␣
↪can specify
        concrete runtime parameters to be processed.
        """

        self.log(Log.C_LOG_TYPE_I, 'Incoming data to be processed:', str(p_input))
        self.log(Log.C_LOG_TYPE_W, 'By the way: I do not really process something;)')


## ------------------------------------------------------------------------------------------
    def get_accuracy(self) -> float:
        """
        This custom method returns the current accuracy of your model...

        Returns
        -------
        accuracy : float
            Accuracy of the model as a scalar value in interval [0,1]
        """
```

(continues on next page)

(continued from previous page)

```
        return 0.0


## ------------------------------------------------------------------------------------------
    def _init_plot_2d(self, p_figure: Figure, p_settings: PlotSettings):
        """
        This custom method is intended to prepare a 2-dimensional plot. See classes␣
↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further␣
↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot initialization...')
        return super()._init_plot_2d(p_figure, p_settings)


## ------------------------------------------------------------------------------------------
    def _init_plot_3d(self, p_figure: Figure, p_settings: PlotSettings):
        """
        This custom method is intended to prepare a 3-dimensional plot. See classes␣
↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further␣
↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot initialization...')
        return super()._init_plot_3d(p_figure, p_settings)


## ------------------------------------------------------------------------------------------
    def _init_plot_nd(self, p_figure: Figure, p_settings: PlotSettings):
        """
        This custom method is intended to prepare a n-dimensional plot. See classes␣
↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further␣
↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot initialization...')
        return super()._init_plot_nd(p_figure, p_settings)


## ------------------------------------------------------------------------------------------
    def _update_plot_2d(self, p_settings: PlotSettings, **p_kwargs):
        """
        This custom method is intended to update your 2-dimensional plot. See classes␣
↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further␣
```

(continues on next page)

(continued from previous page)

```
↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot updates...')
        return super()._update_plot_2d(p_settings, **p_kwargs)


## ------------------------------------------------------------------------------------
↪-----------
    def _update_plot_3d(self, p_settings: PlotSettings, **p_kwargs):
        """
        This custom method is intended to update your 3-dimensional plot. See classes␣
↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further␣
↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot updates...')
        return super()._update_plot_3d(p_settings, **p_kwargs)


## ------------------------------------------------------------------------------------
↪-----------
    def _update_plot_nd(self, p_settings: PlotSettings, **p_kwargs):
        """
        This custom method is intended to update your n-dimensional plot. See classes␣
↪PlotSettings and
        Plottable of sub-package mlpro.bf.plot or the online documentation for further␣
↪information.
        """

        self.log(Log.C_LOG_TYPE_W, 'My specific plot updates...')
        return super()._update_plot_nd(p_settings, **p_kwargs)




# 1 Preparation of demo/unit test mode
if __name__ == '__main__':
    # 1.1 Parameters for demo mode
    visualize   = True
    logging     = Log.C_LOG_ALL

else:
    # 1.2 Parameters for internal unit test
    visualize   = False
    logging     = Log.C_LOG_NOTHING


# 2 Instantiation of your custom model (multitasking is disabled)
mymodel = MyModel( p_range_max = Range.C_RANGE_NONE,
                   p_visualize = visualize,
```

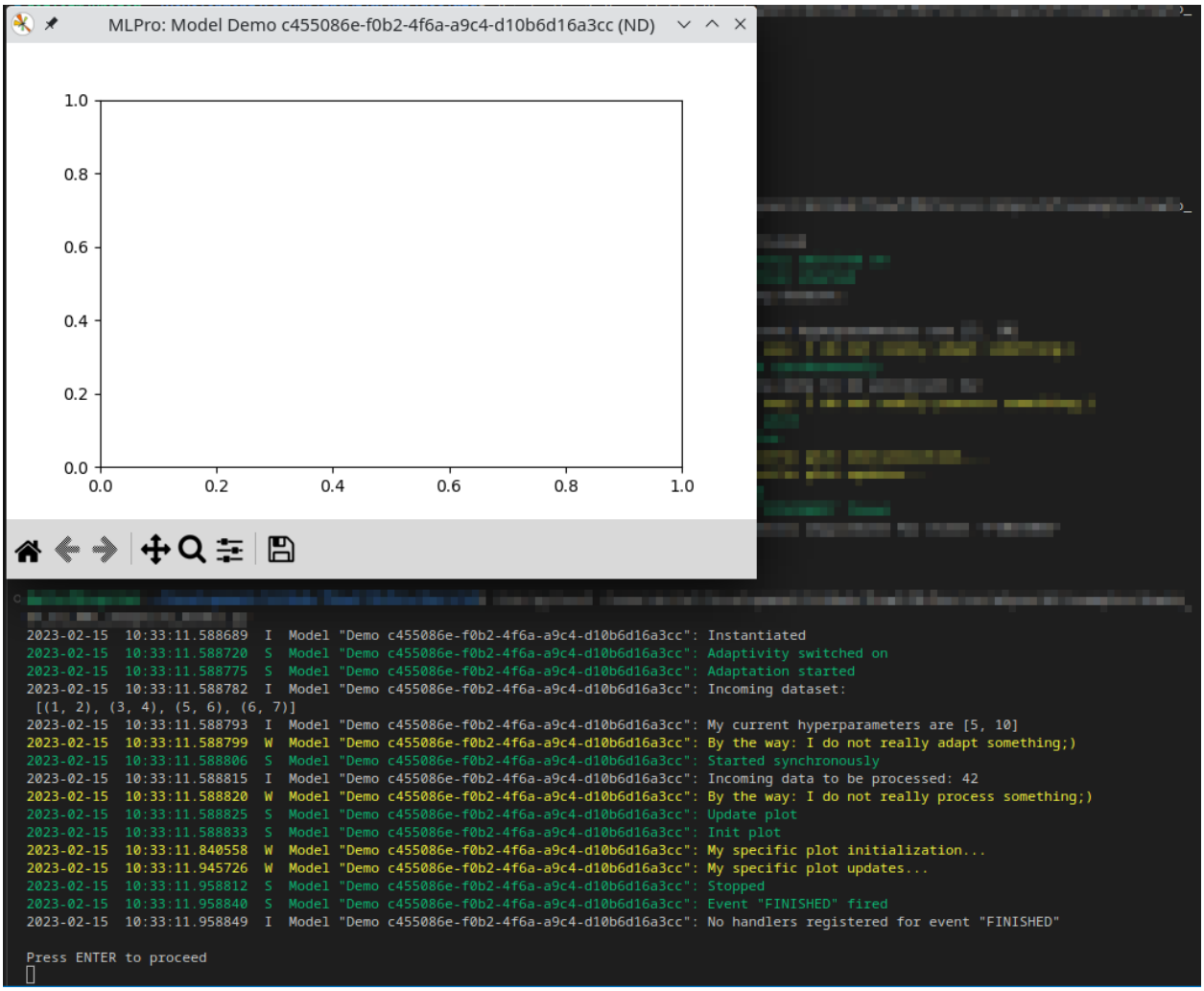
(continues on next page)

(continued from previous page)

```
                    p_logging = logging,
                    p_hp_layers = 5,
                    p_hp_neurons_per_layer = 10 )


# 3 Pseudo-adaptation based on a dataset
ds = [ (1,2), (3,4), (5,6), (6,7) ]
mymodel.adapt( p_dataset = ds )


# 4 Now we let your model do it's job
mymodel.run( p_input = 42 )

if __name__ == '__main__':
    input( '\nPress ENTER to proceed\n')
```

Results

As shown below, the howto logs all steps and a demo window for visualization appears…



Cross Reference

- *API Reference: Machine Learning*

Howto BF-ML-010: Hyperparameters

Ver. 1.1.1 (2023-03-02)

This module demonstrates how to set-up hyperparameters using available HyperParamTuple, HyperParamSpace, and HyperParam classes.

You will learn:

1. How to use the Hyperparameter class of MLPro and its functionalities in Native and custom implementations.
2. How to create hyperparameter space and add dimensions to the space.
3. How to create and set values for a hyperparameter tuple.

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.bf.examples
## -- Module  : howto_bf_ml_010_hyperparameters.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-08-31  0.0.0     SY       Creation
## -- 2021-09-01  1.0.0     SY       Release of first version
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library name
## -- 2021-12-10  1.0.1     DA       Refactoring, little beautifying
## -- 2022-02-25  1.0.2     SY       Refactoring due to auto generated ID in class Dimension
## -- 2022-10-12  1.0.3     DA       Renaming/refactoring
## -- 2023-02-15  1.1.0     DA       Renaming
## -- 2023-03-02  1.1.1     LSB      Refactoring
## -------------------------------------------------------------------------------------------

"""
Ver. 1.1.1 (2023-03-02)

This module demonstrates how to set-up hyperparameters using available HyperParamTuple,
HyperParamSpace, and HyperParam classes.

You will learn:

1. How to use the Hyperparameter class of MLPro and its functionalities in Native and custom implementations.

2. How to create hyperparameter space and add dimensions to the space.
```

(continues on next page)

(continued from previous page)

```
3. How to create and set values for a hyperparameter tuple.
"""



from mlpro.bf.ml import *


# 1 Setup a class that requires a tuple of hyperparameters
class MyHyperparameter:

    def __init__(self):
        # 1.1 Construct a hyperparameter space using HyperParamSpace() and an empty␣
↪tuple
        self._hyperparam_space  = HyperParamSpace()
        self._hyperparam_tuple  = None
        self._init_hyperparam()


    def _init_hyperparam(self):
        # 1.2 Declare hyperparameters with unique id, names, and data type
        self._hyperparam_space.add_dim(HyperParam('num_states','Z'))
        self._hyperparam_space.add_dim(HyperParam('smoothing','R'))
        self._hyperparam_space.add_dim(HyperParam('lr_rate','R'))
        self._hyperparam_space.add_dim(HyperParam('buffer_size','Z'))
        self._hyperparam_space.add_dim(HyperParam('update_rate','Z'))
        self._hyperparam_space.add_dim(HyperParam('sampling_size','Z'))
        self._hyperparam_tuple = HyperParamTuple(self._hyperparam_space)

        # 1.3 Set the hyperparameter with a default value
        ids_ = self._hyperparam_tuple.get_dim_ids()
        self._hyperparam_tuple.set_value(ids_[0], 100)
        self._hyperparam_tuple.set_value(ids_[1], 0.035)
        self._hyperparam_tuple.set_value(ids_[2], 0.0001)
        self._hyperparam_tuple.set_value(ids_[3], 100000)
        self._hyperparam_tuple.set_value(ids_[4], 100)
        self._hyperparam_tuple.set_value(ids_[5], 256)


    def get_hyperparam(self) -> HyperParamTuple:
        return self._hyperparam_tuple




if __name__ == "__main__":
    printing = True
else:
    printing = False


# 2 Get value from the hyperparameter tuple
```

(continues on next page)

(continued from previous page)

```
myParameter         = MyHyperparameter()
for idx in myParameter.get_hyperparam().get_dim_ids():
    par_name = myParameter.get_hyperparam().get_related_set().get_dim(idx).get_name_
↪short()
    par_val  = myParameter.get_hyperparam().get_value(idx)
    if printing: print('Variable with ID %s = %.2f'%(par_name, par_val))


# 3 Overwrite current value with new desired value
ids_ = myParameter.get_hyperparam().get_dim_ids()
myParameter.get_hyperparam().set_value(ids_[0], 50)

if printing:
    print('\nA new value for variable ID ids_[0]')
    print('Variable with ID ids_[0] = %.2f'%(myParameter.get_hyperparam().get_value(ids_
↪[0])))
```

Results

```
Variable with ID num_states = 100.00
Variable with ID smoothing = 0.04
Variable with ID lr_rate = 0.00
Variable with ID buffer_size = 100000.00
Variable with ID update_rate = 100.00
Variable with ID sampling_size = 256.00

A new value for variable ID ids_[0]
Variable with ID ids_[0] = 50.00
```

Cross Reference

- *API Reference: Machine Learning*

Adaptive State-based Systems

8.2 MLPro-SL - Supervised Learning

Coming soon…

8.3 MLPro-RL - Reinforcement Learning

The following examples demonstrate various functionalities of MLPro-RL:

8.3.1 Elementary or Uncategorized Topics

Howto RL-001: Reward

Ver. 1.0.3 (2023-03-02)

This module shows how to create and interprete reward objects in own projects.

You will learn:

1. How to use the reward class of MLPro.
2. How to use reward class for different reward types supported in MLPro.

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_001_reward.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-05-30  0.0.0     DA       Creation
## -- 2021-05-31  1.0.0     DA       Release of first version
## -- 2021-09-11  1.0.1     MRD      Change Header information to match our new library␣
↪name
## -- 2022-10-13  1.0.2     SY       Refactoring
## -- 2023-03-02  1.0.3     LSB      Refactoring
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.3 (2023-03-02)

This module shows how to create and interprete reward objects in own projects.

You will learn:

1. How to use the reward class of MLPro.

2. How to use reward class for different reward types supported in MLPro.
"""


from mlpro.bf.various import Log
from mlpro.rl import Reward



class MyLog(Log):
    C_TYPE      = 'Reward Demo'
    C_NAME      = ''
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    # 1 Some initial stuff
    my_log = MyLog()

    # 1.1 Unique agent ids
    C_AGENT_1       = 1
    C_AGENT_2       = 2
    C_AGENT_3       = 3

    # 1.2 Unique action ids
    C_AGENT_1_ACT_1 = 1
    C_AGENT_1_ACT_2 = 2
    C_AGENT_1_ACT_3 = 3
    C_AGENT_2_ACT_1 = 4


    # 2 Rewards as single overall scalar values (independent from agents and actions)
    my_log.log(Log.C_LOG_TYPE_I, 'Example for reward type C_TYPE_OVERALL:')
    reward = Reward(p_type=Reward.C_TYPE_OVERALL)
    reward.set_overall_reward(4.77)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward is just a scalar...', reward.get_agent_
↪reward(0), '\n')


    # 3 Rewards as scalar values for every agent
    my_log.log(Log.C_LOG_TYPE_I, 'Example for reward type C_TYPE_EVERY_AGENT')
    reward = Reward(p_type=Reward.C_TYPE_EVERY_AGENT)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward is a list with entries for each agent...')
    reward.add_agent_reward(C_AGENT_1, 4.77)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1 added:', reward.get_agent_reward(C_
↪AGENT_1))
    reward.add_agent_reward(C_AGENT_2, 5.19)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 2 added:', reward.get_agent_reward(C_
↪AGENT_2))
    reward.add_agent_reward(C_AGENT_3, 0.23)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 3 added:', reward.get_agent_reward(C_
↪AGENT_3), '\n')

    # 4 Rewards as scalar values for every agent and it's actions
    my_log.log(Log.C_LOG_TYPE_I, 'Example for reward type C_TYPE_EVERY_ACTION')
    reward = Reward(p_type=Reward.C_TYPE_EVERY_ACTION)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward is a list with entries for each agent and its␣
↪action components...')
    reward.add_action_reward(C_AGENT_1, C_AGENT_1_ACT_1, 1.23)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1, action 1 added:', reward.get_
↪action_reward(C_AGENT_1, C_AGENT_1_ACT_1))
    reward.add_action_reward(C_AGENT_1, C_AGENT_1_ACT_2, 0.47)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1, action 2 added:', reward.get_
↪action_reward(C_AGENT_1, C_AGENT_1_ACT_2))
    reward.add_action_reward(C_AGENT_1, C_AGENT_1_ACT_3, 1.63)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 1, action 3 added:', reward.get_
↪action_reward(C_AGENT_1, C_AGENT_1_ACT_3))
```

(continues on next page)

(continued from previous page)

```
    reward.add_action_reward(C_AGENT_2, C_AGENT_2_ACT_1, 4.23)
    my_log.log(Log.C_LOG_TYPE_I, 'Reward for agent 2, action 4 added:', reward.get_
↪action_reward(C_AGENT_2, C_AGENT_2_ACT_1))
```

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*

8.3.2 Agents

Howto RL-AGENT-001: Run an Agent with Own Policy

Ver. 1.3.1 (2023-01-14)

This module shows how to run an own policy inside the standard agent model with an OpenAI Gym environment using MLPro framework.

You will learn:

1) How to set up a native policy for an agent
2) How to set up an agent
3) How to set up a scenario
4) How to wrap Gym environment to MLPro environment
5) How to run the scenario

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_001_run_agent_with_own_policy_on_gym_environment.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-05-09  0.0.0     DA       Creation
## -- 2021-06-06  1.0.0     DA       Released first version
## -- 2021-08-28  1.1.0     DA       Introduced Policy
## -- 2021-09-11  1.1.0     MRD      Change Header information to match our new library␣
↪name
## -- 2021-09-29  1.1.1     SY       Change name: WrEnvGym to WrEnvGYM2MLPro
## -- 2021-10-06  1.1.2     DA       Refactoring
## -- 2021-10-18  1.1.3     DA       Refactoring
## -- 2021-11-15  1.2.0     DA       Refactoring
```

(continues on next page)

(continued from previous page)

```
## -- 2021-11-16  1.2.1     DA       Added explicit scenario reset with constant seeding
## -- 2021-12-03  1.2.2     DA       Refactoring
## -- 2022-07-20  1.2.3     SY       Update due to the latest introduction of Gym 0.25
## -- 2022-10-13  1.2.4     SY       Refactoring
## -- 2022-11-01  1.2.5     DA       Refactoring
## -- 2022-11-02  1.2.6     DA       Refactoring
## -- 2022-11-07  1.3.0     DA       Refactoring
## -- 2023-01-14  1.3.1     MRD      Removing default parameter new_step_api and render_
↪mode for gym
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.3.1 (2023-01-14)

This module shows how to run an own policy inside the standard agent model with an␣
↪OpenAI Gym environment using
MLPro framework.

You will learn:

1) How to set up a native policy for an agent

2) How to set up an agent

3) How to set up a scenario

4) How to wrap Gym environment to MLPro environment

5) How to run the scenario

"""


from mlpro.bf.math import *
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
import gym
import random




# 1 Implement your own agent policy
class MyPolicy (Policy):

    C_NAME      = 'MyPolicy'

    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)
```

(continues on next page)

(continued from previous page)

```
    def compute_action(self, p_state: State) -> Action:
        # 1.1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 1.2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = random.random()

        # 1.3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)


    def _adapt(self, p_sars_elem: SARSElement) -> bool:
        # 1.4 Adapting the internal policy is up to you...
        self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')

        # 1.5 Only return True if something has been adapted...
        return False




# 2 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize:bool, p_logging) -> Model:
        # 2.1 Setup environment
        gym_env     = gym.make('CartPole-v1')
        self._env   = WrEnvGYM2MLPro( p_gym_env=gym_env, p_visualize=p_visualize, p_
↪logging=p_logging)

        # 2.2 Setup standard single-agent with own policy
        return Agent( p_policy=MyPolicy( p_observation_space=self._env.get_state_space(),
                                         p_action_space=self._env.get_action_space(),
                                         p_buffer_size=1,
                                         p_ada=p_ada,
                                         p_visualize=p_visualize,
                                         p_logging=p_logging),
                      p_envmodel=None,
                      p_name='Smith',
                      p_ada=p_ada,
                      p_visualize=p_visualize,
                      p_logging=p_logging)




# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # 3.1 Parameters for demo mode
```

(continues on next page)

(continued from previous page)

```
    cycle_limit = 100
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 3.2 Parameters for internal unit test
    cycle_limit = 10
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 3.3 Create your scenario and run some cycles
myscenario  = MyScenario(
        p_mode=Mode.C_MODE_SIM,
        p_ada=True,
        p_cycle_limit=cycle_limit,
        p_visualize=visualize,
        p_logging=logging
)

myscenario.reset(p_seed=3)
myscenario.run()
```

Results

The OpenAI Gym cartpole window appears and shows a randowm behavior while the demo run is logged on the console. Demo should terminate after 61 cycles because of the fixed random seed.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-AGENT-002: Train an Agent with Own Policy

Ver. 1.4.1 (2023-01-14)

This module shows how to train an agent with a custom policy inside on an OpenAI Gym environment using MLPro framework.

You will learn:

1) How to set up a native policy for an agent
2) How to set up an agent
3) How to set up a scenario
4) How to wrap Gym environment to MLPro environment
5) How to run the scenario and train the agent

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_002_train_agent_with_own_policy_on_gym_environment.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-06-03  0.0.0     DA       Creation
## -- 2021-06-06  1.0.0     DA       Released first version
## -- 2021-06-25  1.1.0     DA       Extended by data logging/storing (user home␣
↪directory)
## -- 2021-07-06  1.1.1     SY       Bugfix due to method Training.save_data() update
## -- 2021-08-28  1.2.0     DA       Introduced Policy
## -- 2021-09-11  1.2.0     MRD      Change Header information to match our new library␣
↪name
## -- 2021-09-28  1.2.1     SY       Updated due to implementation of method get_cycle_
↪limits()
## -- 2021-09-29  1.2.2     SY       Change name: WrEnvGym to WrEnvGYM2MLPro
## -- 2021-10-06  1.2.3     DA       Refactoring
## -- 2021-10-18  1.2.4     DA       Refactoring
## -- 2021-11-15  1.3.0     DA       Refactoring
## -- 2021-12-03  1.3.1     DA       Refactoring
## -- 2021-12-07  1.3.2     DA       Refactoring
```

(continues on next page)

(continued from previous page)

```
## -- 2022-07-20  1.3.3     SY       Update due to the latest introduction of Gym 0.25
## -- 2022-10-13  1.3.4     SY       Refactoring
## -- 2022-11-01  1.3.5     DA       Refactoring
## -- 2022-11-02  1.3.6     DA       Refactoring
## -- 2022-11-07  1.4.0     DA       Refactoring
## -- 2023-01-14  1.4.1     MRD      Removing default parameter new_step_api and render_
↪mode for gym
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.4.1 (2023-01-14)

This module shows how to train an agent with a custom policy inside on an OpenAI Gym␣
↪environment using
MLPro framework.

You will learn:

1) How to set up a native policy for an agent

2) How to set up an agent

3) How to set up a scenario

4) How to wrap Gym environment to MLPro environment

5) How to run the scenario and train the agent

"""


from mlpro.bf.math import *
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
import gym
import random
from pathlib import Path




# 1 Implement your own agent policy
class MyPolicy (Policy):

    C_NAME      = 'MyPolicy'

    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)


    def compute_action(self, p_state: State) -> Action:
```

(continues on next page)

(continued from previous page)

```
        # 1.1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 1.2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = random.random()

        # 1.3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)


    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        # 1.4 Adapting the internal policy is up to you...
        self.log(self.C_LOG_TYPE_I, 'Sorry, I am a stupid agent...')

        # 1.5 Only return True if something has been adapted...
        return False




# 2 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 2.1 Setup environment
        gym_env     = gym.make('CartPole-v1')
        self._env   = WrEnvGYM2MLPro(gym_env, p_visualize=p_visualize, p_logging=p_
↪logging)

        # 2.2 Setup and return standard single-agent with own policy
        return Agent(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space(),
                    p_action_space=self._env.get_action_space(),
                    p_buffer_size=10,
                    p_ada=p_ada,
                    p_visualize=p_visualize,
                    p_logging=p_logging
                ),
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )
```

(continues on next page)

(continued from previous page)

```
# 3 Create scenario and start training

if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    cycle_limit = 500
    logging     = Log.C_LOG_WE
    visualize   = True
    path        = str(Path.home())

else:
    # 3.2 Parameters for internal unit test
    cycle_limit = 50
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None


# 3.3 Create and run training object
training = RLTraining(
        p_scenario_cls=MyScenario,
        p_cycle_limit=cycle_limit,
        p_path=path,
        p_visualize=visualize,
        p_logging=logging )

training.run()
```

Results

The Gym Cartpole environment window should appear. Afterwards, the training should run for a few episodes before terminating and printing the result. The training log is also stored in the location specified.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training Results of run 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Scenario          : RL-Scenario Matrix
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Model             : Agent Smith
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Start time stamp  : YYYY-MM-DD HH:MM:SS.
↪SSSSSS
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- End time stamp    : YYYY-MM-DD HH:MM:SS.
↪SSSSSS
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Duration          : 0:00:09.209252
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Start cycle id    : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- End cycle id      : 499
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training cycles   : 500
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Evaluation cycles : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Adaptations       : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- High score        : None
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Results stored in : "C:\Users\%username%\
↪YYYY-MM-DD  HH:MM:SS Training RL"
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training Episodes : 23
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Evaluations       : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
```

(continues on next page)

(continued from previous page)

```
↪----------------------------------
```

The local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-AGENT-003: Run Multi-Agent with Own Policy

Ver. 1.4.0 (2023-02-22)

This module shows how to run an own multi-agent with the enhanced multi-action environment MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up a native policy for an agent
2) How to set up a multiagent
3) How to set up a scenario
4) How to run the scenario

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_003_run_multiagent_with_own_policy_on_multicartpole_
↪environment.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-05-20  0.0.0     DA       Creation
## -- 2021-06-06  1.0.0     DA       Release of first version
```

(continues on next page)

(continued from previous page)

```
## -- 2021-08-28  1.1.0     DA       Introduced Policy
## -- 2021-09-11  1.1.0     MRD      Change Header information to match our new library␣
↪name
## -- 2021 09-26  1.1.1     MRD      Change the import module due to the change of the␣
↪pool
## --                                 folder structer
## -- 2021-10-06  1.1.2     DA       Refactoring
## -- 2021-10-18  1.1.3     DA       Refactoring
## -- 2021-11-15  1.2.0     DA       Refactoring
## -- 2022-02-25  1.2.1     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-10-13  1.2.2     SY       Refactoring
## -- 2022-11-01  1.2.3     DA       Refactoring
## -- 2022-11-02  1.2.4     DA       Refactoring
## -- 2022-11-07  1.3.0     DA       Refactoring
## -- 2023-02-22  1.4.0     DA       User can now arrange the three gym windows before␣
↪the demo run
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.4.0 (2023-02-22)

This module shows how to run an own multi-agent with the enhanced multi-action␣
↪environment
MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up a native policy for an agent

2) How to set up a multiagent

3) How to set up a scenario

4) How to run the scenario

"""


from mlpro.rl import *
from mlpro.rl.pool.envs.multicartpole import MultiCartPole
import random




# 1 Implement your own agent policy
class MyPolicy (Policy):

    C_NAME      = 'MyPolicy'
```

(continues on next page)

(continued from previous page)

```
    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)


    def compute_action(self, p_state: State) -> Action:
        # 1.1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 1.2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = random.random()

        # 1.3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)


    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        # 1.4 Adapting the internal policy is up to you...
        self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')

        # 1.5 Only return True if something has been adapted...
        return False




# 2 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 2.1 Setup Multi-Agent Environment (consisting of 3 OpenAI Gym Cartpole envs)
        self._env   = MultiCartPole(p_num_envs=3, p_visualize=p_visualize, p_logging=p_
↪logging)


        # 2.2 Setup Multi-Agent

        # 2.2.1 Create empty Multi-Agent
        multi_agent = MultiAgent(
            p_name='Smith',
            p_ada=True,
            p_visualize=p_visualize,
            p_logging=p_logging
        )

        # 2.2.2 Add Single-Agent #1 with own policy (controlling sub-environment #1)
        ss_ids = self._env.get_state_space().get_dim_ids()
        as_ids = self._env.get_action_space().get_dim_ids()
        multi_agent.add_agent(
            p_agent=Agent(
```

(continues on next page)

(continued from previous page)

```
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[0],ss_
↪ids[1],ss_ids[2],ss_ids[3]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[0]]),
                    p_buffer_size=1,
                    p_ada=True,
                    p_logging=p_logging
                ),
                p_envmodel=None,
                p_name='Smith-1',
                p_id=0,
                p_ada=True,
                p_logging=p_logging
            ),
            p_weight=0.3
        )

        # 2.2.3 Add Single-Agent #2 with own policy (controlling sub-environments #2,#3)
        multi_agent.add_agent(
            p_agent=Agent(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[4],ss_
↪ids[5],ss_ids[6],ss_ids[7],ss_ids[8],ss_ids[9],ss_ids[10],ss_ids[11]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[1],as_
↪ids[2]]),
                    p_buffer_size=1,
                    p_ada=True,
                    p_logging=p_logging
                ),
                p_envmodel=None,
                p_name='Smith-2',
                p_id=1,
                p_ada=True,
                p_logging=p_logging
            ),
            p_weight=0.7
        )

        # 2.3 Adaptive ML model (here: our multi-agent) is returned
        return multi_agent



# 3 Create scenario and run some cycles
if __name__ == '__main__':
    # 3.1 Parameters for demo mode
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 3.2 Parameters for internal unit test
```

(continues on next page)

(continued from previous page)

```
    logging     = Log.C_LOG_NOTHING
    visualize   = False


myscenario  = MyScenario(
        p_mode=Mode.C_MODE_SIM,
        p_ada=True,
        p_cycle_limit=200,
        p_visualize=visualize,
        p_logging=logging
)

myscenario.reset()

if __name__ == '__main__':
    input('\nPlease arrange the three cartpole windows and press ENTER...\n')

myscenario.run()
```

Results

Similar output as in *Howto RL-AGENT-001* is displayed. However, there will be multiple windows of the environment open.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-AGENT-004: Train Multi-Agent with Own Policy

Ver. 1.4.0 (2022-11-07)

This module shows how to train an own multi-agent with the enhanced multi-action environment MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up a native policy for an agent
2) How to set up a multiagent
3) How to set up a scenario
4) How to run the scenario and train the agents

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_004_train_multiagent_with_own_policy_on_multicartpole_
↪environment.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-06-06  0.0.0     DA       Creation
## -- 2021-06-06  1.0.0     DA       Release of first version
## -- 2021-07-01  1.1.0     DA       Extended by data logging/storing (user home␣
↪directory)
## -- 2021-07-06  1.1.1     SY       Bugfix due to method Training.save_data() update
## -- 2021-08-28  1.2.0     DA       Introduced Policy
## -- 2021-09-11  1.2.0     MRD      Change Header information to match our new library␣
↪name
## -- 2021 09-26  1.2.1     MRD      Change the import module due to the change of the␣
↪pool
## --                                 folder structer
## -- 2021-10-06  1.2.2     DA       Refactoring
## -- 2021-11-15  1.3.0     DA       Refactoring
## -- 2021-12-07  1.3.1     DA       Refactoring
## -- 2022-02-25  1.3.2     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-10-13  1.3.3     SY       Refactoring
## -- 2022-11-01  1.3.4     DA       Refactoring
## -- 2022-11-02  1.3.5     DA       Refactoring
## -- 2022-11-07  1.4.0     DA       Refactoring
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.4.0 (2022-11-07)

This module shows how to train an own multi-agent with the enhanced multi-action␣
↪environment
MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up a native policy for an agent

2) How to set up a multiagent

3) How to set up a scenario

4) How to run the scenario and train the agents

"""
```

(continues on next page)

(continued from previous page)

```
from mlpro.rl import *
from mlpro.rl.pool.envs.multicartpole import MultiCartPole
import random
from pathlib import Path
import os
from datetime import datetime



# 1 Implement your own agent policy
class MyPolicy(Policy):

    C_NAME      = 'MyPolicy'

    def compute_action(self, p_state: State) -> Action:
        # 1.1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 1.2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = random.random()

        # 1.3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)


    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        # 1.4 Adapting the internal policy is up to you...
        self.log(self.C_LOG_TYPE_I, 'Sorry, I am a stupid agent...')

        # 1.5 Only return True if something has been adapted...
        return False




# 2 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:

        # 2.1 Setup Multi-Agent Environment (consisting of 3 OpenAI Gym Cartpole envs)
        self._env   = MultiCartPole(p_num_envs=3, p_visualize=p_visualize, p_logging=p_
↪logging)


        # 2.2 Setup Multi-Agent
```

(continues on next page)

(continued from previous page)

```
        # 2.2.1 Create empty Multi-Agent
        multi_agent     = MultiAgent(
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )

        # 2.2.2 Add Single-Agent #1 with own policy (controlling sub-environment #1)
        ss_ids = self._env.get_state_space().get_dim_ids()
        as_ids = self._env.get_action_space().get_dim_ids()
        multi_agent.add_agent(
            p_agent=Agent(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[0],ss_
↪ids[1],ss_ids[2],ss_ids[3]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[0]]),
                    p_buffer_size=1,
                    p_ada=p_ada,
                    p_logging=p_logging
                ),
                p_envmodel=None,
                p_name='Smith-1',
                p_id=0,
                p_ada=p_ada,
                p_logging=p_logging
            ),
            p_weight=0.3
        )

        # 2.2.3 Add Single-Agent #2 with own policy (controlling sub-environments #2,#3)
        multi_agent.add_agent(
            p_agent=Agent(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[4],ss_
↪ids[5],ss_ids[6],ss_ids[7],ss_ids[8],ss_ids[9],ss_ids[10],ss_ids[11]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[1],as_
↪ids[2]]),
                    p_buffer_size=1,
                    p_ada=p_ada,
                    p_visualize=p_visualize,
                    p_logging=p_logging
                ),
                p_envmodel=None,
                p_name='Smith-2',
                p_id=1,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging
            ),
            p_weight=0.7
```

(continues on next page)

(continued from previous page)

```
        )

        # 2.3 Adaptive ML model (here: our multi-agent) is returned
        return multi_agent



# 3 Create scenario and start training
if __name__ == '__main__':
    # 3.1 Parameters for demo mode
    cycle_limit = 200
    logging     = Log.C_LOG_WE
    visualize   = True
    path        = str(Path.home())

else:
    # 3.2 Parameters for internal unit test
    cycle_limit = 10
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None


# 3.3 Create and run training object
training = RLTraining(
        p_scenario_cls=MyScenario,
        p_cycle_limit=cycle_limit,
        p_path=path,
        p_visualize=visualize,
        p_logging=logging )

training.run()
```

Results

Similar output as in *Howto RL-AGENT-002* is displayed. However, three cartpole windows will be opened while the training log runs through. In addition, the training result folder will contain one more pkl file for the second agent.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-AGENT-011: Train and Reload Single Agent (Gym)

Ver. 1.3.0 (2023-03-04)

This module shows how to train a single agent and load it again to do some extra cycles.

You will learn:

1. How to use the RLScenario class of MLPro.
2. How to save a scenario after some run.
3. How to reload the saved scenario and re-run for additional cycles.

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_011_train_and_reload_single_agent_gym.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-01-28  0.0.0     MRD      Creation
## -- 2022-01-28  1.0.0     MRD      Released first version
## -- 2022-05-19  1.0.1     MRD      Re-use the agent not for the re-training process
## --                                 Remove commenting and numbering
## -- 2022-05-19  1.0.2     MRD      Re-add the commenting and reformat the numbering in␣
↪comment
## -- 2022-07-20  1.0.3     SY       Update due to the latest introduction of Gym 0.25
## -- 2022-10-13  1.0.4     SY       Refactoring
## -- 2022-10-17  1.0.5     SY       Debugging
## -- 2022-11-01  1.0.6     DA       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-01-14  1.1.1     MRD      Removing default parameter new_step_api and render_
↪mode for gym
## -- 2023-02-12  1.1.2     MRD      Save to MLPro folder path for CI test
## -- 2023-02-15  1.1.3     MRD      Adjust parameter
## -- 2023-02-20  1.2.0     DA       Simplification after changes on class bf.ml.Training
## -- 2023-03-02  1.2.1     LSB      Refactoring
## -- 2023-03-04  1.3.0     DA       Renamed
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.3.0 (2023-03-04)

This module shows how to train a single agent and load it again to do some extra cycles.
```

(continues on next page)

(continued from previous page)

```
You will learn:

1. How to use the RLScenario class of MLPro.

2. How to save a scenario after some run.

3. How to reload the saved scenario and re-run for additional cycles.

"""


import gym
from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from pathlib import Path



# 1 Implement your own RL scenario
class MyScenario (RLScenario):
    C_NAME = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        gym_env = gym.make('CartPole-v1')
        self._env = WrEnvGYM2MLPro(gym_env, p_visualize=p_visualize, p_logging=p_logging)

        # 1.2 Setup Policy From SB3
        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=10,
            env=None,
            _init_setup_model=False,
            device="cpu",
            seed=1)

        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)

        # 1.4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped,
```

(continues on next page)

(continued from previous page)

```
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )



if __name__ == '__main__':
    # Parameters for demo mode
    cycle_limit = 10000
    adaptation_limit = 0
    stagnation_limit = 0
    eval_frequency = 0
    eval_grp_size = 0
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging )



# 3 Training
training.run()



# 4 Reload the scenario
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    input( '\nTraining finished. Press ENTER to reload and run the scenario...\n')

scenario = MyScenario.load( p_path = training.get_training_path() + os.sep + 'scenario' )


# 5 Reset Scenario
scenario.reset()


# 6 Run Scenario
scenario.run()

if __name__ != '__main__':
    from shutil import rmtree
    rmtree(training.get_training_path())
else:
    input( '\nPress ENTER to finish...')
```

Results

The Gym Cartpole environment window appears. Afterwards, the training runs for a few episodes before terminating and printing the result.

After termination the local result folders contain the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

Both training results are from the same agent.

Cross Reference

- *MLPro-RL: Training*
- *API Reference*

Howto RL-AGENT-021: Train and Reload Single Agent Cartpole Discrete (MuJoCo)

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_021_train_and_reload_single_agent_mujoco_cartpole_discrete.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-02-23  0.0.0     MRD      Creation
## -- 2023-02-23  1.0.0     MRD      Released first version
## -- 2023-02-23  1.0.1     MRD      Refactor
## -- 2023-03-02  1.0.2     LSB      Refactoring
## -- 2023-03-04  1.1.0     DA       Renamed
## -- 2023-03-07  1.1.1     MRD      Renamed
## -------------------------------------------------------------------------------------------

"""
Ver. 1.1.1 (2023-03-07)

This module shows how to train a single agent with SB3 Policy on Cartpole Discrete MuJoCo Environment.

You will learn:

1. How to use MLPro's RLScenario class.

2. How to create sb3 policy object.

3. How to create SB3 policy in MLPro.

4. How to setup and run RLTraining in MLPro.

"""


from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from mlpro.rl.pool.envs.cartpole import CartpoleMujocoDiscrete
from pathlib import Path


class MyScenario(RLScenario):
    C_NAME = "Matrix"

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = CartpoleMujocoDiscrete(p_logging=logging, p_visualize=visualize)
```

(continues on next page)

(continued from previous page)

```
        # 1.2 Setup Policy From SB3
        policy_sb3 = PPO(policy="MlpPolicy", n_steps=10, env=None, _init_setup_
↪model=False, device="cpu", seed=1)

        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
        )

        # 1.4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped, p_envmodel=None, p_name="Smith", p_ada=p_ada, p_
↪visualize=p_visualize, p_logging=p_logging
        )


# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # Parameters for demo mode
    cycle_limit = 10000
    adaptation_limit = 0
    stagnation_limit = 0
    eval_frequency = 0
    eval_grp_size = 0
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
```

(continues on next page)

(continued from previous page)

```
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging,
)


# 3 Training
training.run()


# 4 Reload the scenario
if __name__ == '__main__':
    input( '\nTraining finished. Press ENTER to reload and run the scenario...\n')

scenario = MyScenario.load( p_path = training.get_training_path() + os.sep + 'scenario' )


# 5 Reset Scenario
scenario.reset()


# 6 Run Scenario
scenario.run()

if __name__ != '__main__':
    from shutil import rmtree
    rmtree(training.get_training_path())
else:
    input( '\nPress ENTER to finish...')
```

Results

The MuJoCo Cartpole environment window appears. Afterwards, the training runs for a few episodes before terminating and printing the result.

After termination the local result folders contain the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

Both training results are from the same agent.

Cross Reference

- *MLPro-RL: Training*
- *API Reference*

Howto RL-AGENT-022: Train and Reload Single Agent Cartpole Continuous (MuJoCo)

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_agent_022_train_and_reload_single_agent_mujoco_cartpole_
↪continuous.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-03-07  0.0.0     MRD      Creation
## -- 2023-03-07  1.0.0     MRD      Released first version
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.0 (2023-03-07)

This module shows how to train a single agent with SB3 Policy on Cartpole Continuous␣
↪MuJoCo Environment.

You will learn:

1. How to use MLPro's RLScenario class.

2. How to create sb3 policy object.

3. How to create SB3 policy in MLPro.

4. How to setup and run RLTraining in MLPro.

"""


from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from mlpro.rl.pool.envs.cartpole import CartpoleMujocoContinuous
from pathlib import Path


class MyScenario(RLScenario):
    C_NAME = "Matrix"
```

(continues on next page)

(continued from previous page)

```
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = CartpoleMujocoContinuous(p_logging=logging, p_visualize=visualize)

        # 1.2 Setup Policy From SB3
        policy_sb3 = PPO(policy="MlpPolicy", n_steps=10, env=None, _init_setup_
↪model=False, device="cpu", seed=1)

        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
        )

        # 1.4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped, p_envmodel=None, p_name="Smith", p_ada=p_ada, p_
↪visualize=p_visualize, p_logging=p_logging
        )


# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # Parameters for demo mode
    cycle_limit = 10000
    adaptation_limit = 0
    stagnation_limit = 0
    eval_frequency = 0
    eval_grp_size = 0
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
```

(continues on next page)

(continued from previous page)

```
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging,
)


# 3 Training
training.run()
```

Results

The MuJoCo Cartpole environment window appears. Afterwards, the training runs for a few episodes before terminating and printing the result.

After termination the local result folders contain the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

Both training results are from the same agent.

Cross Reference

- *MLPro-RL: Training*
- *API Reference*

8.3.3 Environments

Howto RL-ENV-001: SB3 Policy on RobotHTM Environment

Ver. 1.3.0 (2023-02-23)

This module shows how to train a wrapped SB3 policy on MLPro's native Robothtm environment.

You will learn:

1) How to set up a scenario for Robothtm and also with SB3 wrapper
2) How to run the scenario and train the agent
3) How to plot from the generated results

Prerequisites

Please install the following packages to run this examples properly:

- Pytorch
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_env_001_train_agent_with_SB3_policy_on_robothtm_environment.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-12-01  0.0.0     MRD      Creation
## -- 2021-12-01  1.0.0     MRD      First Release
## -- 2021-12-07  1.0.1     DA       Refactoring
## -- 2021-12-08  1.0.2     MRD      Add parameter to change the hidden layer of the␣
↪policy
## -- 2022-05.30  1.0.3     DA       Refactoring
## -- 2022-10-13  1.0.4     SY       Refactoring
## -- 2022-11-01  1.0.5     DA       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-02-02  1.2.0     DA       Refactoring
## -- 2023-02-23  1.3.0     DA       Renamed
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.3.0 (2023-02-23)

This module shows how to train a wrapped SB3 policy on MLPro's native Robothtm␣
↪environment.

You will learn:

1) How to set up a scenario for Robothtm and also with SB3 wrapper

2) How to run the scenario and train the agent

3) How to plot from the generated results

"""


import torch
from mlpro.bf.plot import DataPlotting
from mlpro.rl import *
from mlpro.rl.pool.envs.robotinhtm import RobotHTM
from stable_baselines3 import PPO
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from pathlib import Path
```

(continues on next page)

(continued from previous page)

```
# 1 Implement your own RL scenario
class ScenarioRobotHTM (RLScenario):
    C_NAME = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = RobotHTM(p_target_mode="fix", p_visualize=p_visualize, p_logging=p_
↪logging)

        policy_kwargs = dict(activation_fn=torch.nn.Tanh,
                             net_arch=[dict(pi=[128, 128], vf=[128, 128])])

        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=100,
            env=None,
            _init_setup_model=False,
            policy_kwargs=policy_kwargs,
            device="cpu",
            seed=2)

        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)

        # 1.2 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )



# 2 Create scenario and start training
if __name__ == "__main__":
    # 2.1 Parameters for demo mode
    cycle_limit = 100000
    adaptation_limit = 150
    stagnation_limit = 5
    eval_frequency = 5
```

(continues on next page)

(continued from previous page)

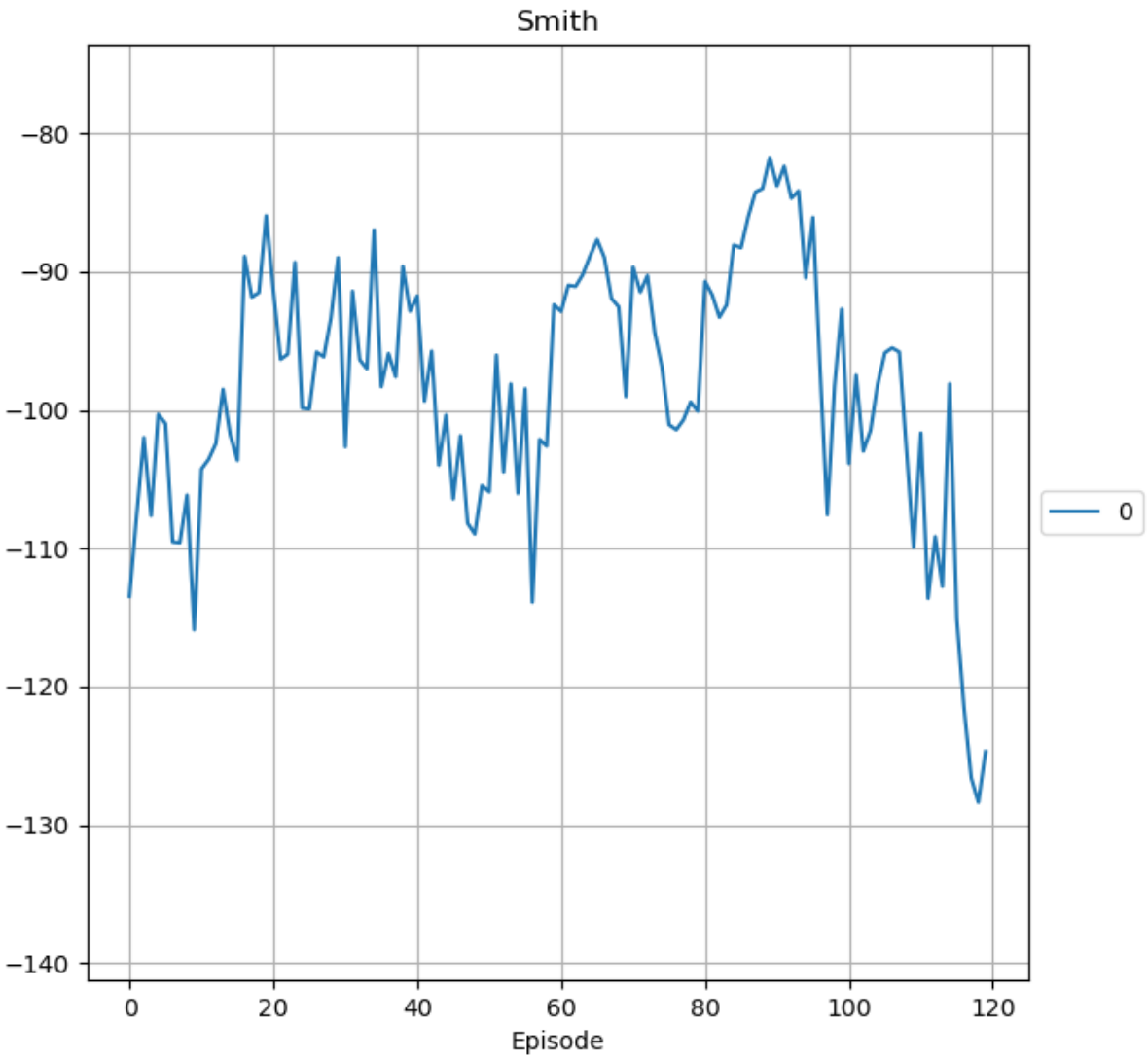
```
    eval_grp_size = 5
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())
    plotting = True

else:
    # 2.2 Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = None
    plotting = False


# 3 Train agent in scenario
training = RLTraining(
    p_scenario_cls=ScenarioRobotHTM,
    p_cycle_limit=cycle_limit,
    p_cycles_per_epi_limit=100,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_score_ma_horizon=7,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging
)

training.run()


# 4 Create Plotting Class
class MyDataPlotting(DataPlotting):
    def get_plots(self):
        """
        A function to plot data
        """
        for name in self.data.names:
            maxval = 0
            minval = 0
            if self.printing[name][0]:
                fig = plt.figure(figsize=(7, 7))
                raw = []
                label = []
                ax = fig.subplots(1, 1)
                ax.set_title(name)
```

(continues on next page)

(continued from previous page)

```
                ax.grid(True, which="both", axis="both")
                for fr_id in self.data.frame_id[name]:
                    raw.append(np.sum(self.data.get_values(name, fr_id)))
                    if self.printing[name][1] == -1:
                        maxval = max(raw)
                        minval = min(raw)
                    else:
                        maxval = self.printing[name][2]
                        minval = self.printing[name][1]

                    label.append("%s" % fr_id)
                ax.plot(raw)
                ax.set_ylim(minval - (abs(minval) * 0.1), maxval + (abs(maxval) * 0.1))
                ax.set_xlabel("Episode")
                ax.legend(label, bbox_to_anchor=(1, 0.5), loc="center left")
                self.plots[0].append(name)
                self.plots[1].append(ax)
                if self.showing:
                    plt.show()
                else:
                    plt.close(fig)


# 5 Plotting with MLpro
if __name__ == "__main__":
    data_printing = {"Cycle": [False],
                     "Day": [False],
                     "Second": [False],
                     "Microsecond": [False],
                     "Smith": [True, -1]}

    mem = training.get_results().ds_rewards
    mem_plot = MyDataPlotting(mem, p_showing=plotting, p_printing=data_printing)
    mem_plot.get_plots()
```

Results



After the environment is initiated, the training will run for the specified amount of limits. However, it is noted that there is no visualization available for this environment. The training log is stored in the location specified and a figure plot similar to the figure above will be produced.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training Results of run 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Scenario          : RL-Scenario Matrix
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Model             : Agent Smith
```

(continues on next page)

(continued from previous page)

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Start time stamp  : YYYY-MM-DD HH:MM:SS.
↪SSSSSS
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- End time stamp    : YYYY-MM-DD HH:MM:SS.
↪SSSSSS
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Duration          : 0:00:59.209252
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Start cycle id    : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- End cycle id      : 11999
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training cycles   : 12000
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Evaluation cycles : 12500
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Adaptations       : 120
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- High score        : -83.8038799825578
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Results stored in : "C:\Users\%username%\
↪YYYY-MM-DD  HH:MM:SS Training RL"
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training Episodes : 120
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Evaluations       : 25
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ------------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ------------------------------------------
↪-----------------------------------
```

The local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-ENV-002: Manual Validation of Double Pendulum

Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- Matplotlib

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
```

(continues on next page)

(continued from previous page)

```
## -- Module  : howto_rl_env_002_manual_validation_of_double_pendulum.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-10-10  1.0.0     LSB      Creation/Release
## -- 2023-02-02  1.1.0     DA       Renamed and refactored
## -- 2023-02-23  1.2.0     DA       Renamed
## -------------------------------------------------------------------------------------------
↪-----------


"""
Ver. 1.2.0 (2023-02-23)

This module is used to validate the dp environment. This howto enables:

1. Setting up a double pendulum environment.

2. Validating the double pendulum environment for user defined actions.

"""




from mlpro.bf.math import *
from mlpro.rl.models import *
from mlpro.rl.pool.envs.doublependulum import *
import numpy as np




if __name__ == '__main__':
    p_input = True
else:
    p_input = False




## -------------------------------------------------------------------------------------------
↪---------
## -------------------------------------------------------------------------------------------
↪---------
class ActionGenerator(Policy):
    """
    Action Generation based on user input
    """

    def set_user_action(self, p_action):
```

(continues on next page)

(continued from previous page)

```
        if p_input:
            self.user_action = np.asarray([p_action])
        else:
           self.user_action = np.zeros(1)

    def compute_action(self, p_state:State):

        return Action(self._id, self._action_space, self.user_action)

    def _adapt(self, **p_kwargs) -> bool:
        self.log(self.C_LOG_TYPE_W, 'Sorry I am not adapting anything')
        return False



# 1 Implement the random RL scenario
class ScenarioDoublePendulum(RLScenario):

    C_NAME      = 'Double Pendulum with Random Actions'

    def _setup(self, p_mode, p_ada, p_visualize,  p_logging):
        self.user_action_cycles = 0

        # 1.1 Setup environment
        self._env   = DoublePendulumS7( p_init_angles='up',
                                        p_max_torque=10,
                                        p_visualize=p_visualize,
                                        p_logging=p_logging )


        # 1.2 Setup and return random action agent
        policy_user = ActionGenerator(p_observation_space=self._env.get_state_space(),
                                        p_action_space=self._env.get_action_space(),
                                        p_buffer_size=1,
                                        p_ada=1,
                                        p_logging=p_logging)

        self.user_action_cycles = 0

        return Agent(
            p_policy=policy_user,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_logging=p_logging
        )


    def _run_cycle(self):

        if p_input and self.get_cycle_id() == self.user_action_cycles:
            p_torque = int(input('Enter the amount of torque in Nm:'))
```

(continues on next page)

(continued from previous page)

```
            self.get_agent()._policy.set_user_action(p_torque)
            p_cycles = int(input('Enter the amount of cycles to be executed:'))

        elif not p_input:
            self.get_agent()._policy.set_user_action(0)
            p_cycles = 0

        else:
            p_cycles = 0

        self.user_action_cycles += p_cycles
        success, error, adapted, end_of_data = super()._run_cycle()
        return success, error, adapted, end_of_data


# 2 Create scenario and run the scenario
if __name__ == "__main__":
    # 2.1 Parameters for demo mode
    cycle_limit         = 20000
    logging             = Log.C_LOG_ALL
    visualize           = True
    plotting            = True
else:
    # 2.2 This demo is not suitable for unit test
    exit(0)



# 3 Create your scenario and run some cycles
myscenario  = ScenarioDoublePendulum(
    p_mode=Mode.C_MODE_SIM,
    p_ada=True,
    p_cycle_limit=cycle_limit,
    p_visualize=visualize,
    p_logging=logging
)

myscenario.reset(p_seed=3)
myscenario.run()
```

Results

Executing the above script will ask the user to input the agent's action and the number cycles for the which the action shall be simulated.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-ENV-003: Run Agent with random action in Double Pendulum Environment

Ver. 1.2.0 (2023-02-23)

This module shows how to run the double pendulum environment using random actions agent.

You will learn:

1) How to set up an own agent using MLPro's builtin random actions policy
2) How to set up an own RL scenario including your agent and MLPro's double pendulum environment
3) How to reset and run your own scenario

Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- Matplotlib

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_env_003_run_agent_with_random_actions_on_double_pendulum_environment.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-04-23  0.0.0     YI       Creation
## -- 2022-04-28  0.0.0     YI       Changing the Scenario and Debugging
## -- 2022-05-16  1.0.0     SY       Code cleaning, remove unnecessary, release the first version
## -- 2022-06-21  1.0.1     SY       Adjust the name of the module, utilize RandomGenerator class
## -- 2022-08-02  1.0.2     LSB      Parameters for internal unit testing
## -- 2022-08-05  1.0.3     SY       Refactoring
## -- 2022-08-23  1.0.4     DA       Refactoring
## -- 2022-09-06  1.0.5     LSB/DA   Refactoring
## -- 2022-10-13  1.0.6     SY       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-02-23  1.2.0     DA       Renamed
## -------------------------------------------------------------------------------------------


"""
Ver. 1.2.0 (2023-02-23)

This module shows how to run the double pendulum environment using random actions agent.

You will learn:
```

(continues on next page)

(continued from previous page)

```
1) How to set up an own agent using MLPro's builtin random actions policy

2) How to set up an own RL scenario including your agent and MLPro's double pendulum␣
↪environment

3) How to reset and run your own scenario

"""


from mlpro.bf.math import *
from mlpro.rl.models import *
from mlpro.rl.pool.envs.doublependulum import *
from mlpro.rl.pool.policies.randomgenerator import RandomGenerator
from pathlib import Path




# 1 Implement the random RL scenario
class ScenarioDoublePendulum(RLScenario):

    C_NAME      = 'Double Pendulum with Random Actions'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env   = DoublePendulumS7(p_init_angles='random', p_max_torque=10, p_
↪visualize=p_visualize, p_logging=p_logging)


        # 1.2 Setup and return random action agent
        policy_random = RandomGenerator(p_observation_space=self._env.get_state_space(),
                                        p_action_space=self._env.get_action_space(),
                                        p_buffer_size=1,
                                        p_ada=1,
                                        p_visualize=p_visualize,
                                        p_logging=p_logging)

        return Agent(
            p_policy=policy_random,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )



# 2 Create scenario and run the scenario
if __name__ == "__main__":
    # 2.1 Parameters for demo mode
```

(continues on next page)

(continued from previous page)

```
    cycle_limit         = 300
    logging             = Log.C_LOG_ALL
    visualize           = True
    plotting            = True
else:
    # 2.2 Parameters for unittest
    cycle_limit         = 2
    logging             = Log.C_LOG_NOTHING
    visualize           = False
    plotting            = False



# 3 Create your scenario and run some cycles
myscenario  = ScenarioDoublePendulum(
    p_mode=Mode.C_MODE_SIM,
    p_ada=True,
    p_cycle_limit=cycle_limit,
    p_visualize=visualize,
    p_logging=logging
)

myscenario.reset(p_seed=3)
myscenario.run()
```

Results

Running this howto shall generate visualizations for double endulum environment and the reward, resectively.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

Howto RL-ENV-005: Run Agent with random policy on double pendulum mujoco environment

Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- MuJoCo
- lxml

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
```

(continues on next page)

(continued from previous page)

```
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_env_005_run_agent_with_random_policy_on_double_pendulum_mujoco_
↪environment.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-09-17  0.0.0     MRD       Creation
## -- 2022-12-11  0.0.1     MRD       Refactor due to new bf.Systems
## -- 2022-12-11  1.0.0     MRD       First Release
## -- 2023-01-07  1.0.1     MRD       Add State Mapping between MuJoCo model and␣
↪Environment State Space
## -- 2023-01-27  1.1.0     MRD       Implement Pendulum Environment, refactor due to␣
↪different MuJoCo
## --                                  mechanism
## -- 2023-02-13  1.1.1     MRD       Refactor
## -- 2023-02-23  1.2.0     DA        Renamed
## -------------------------------------------------------------------------------------
↪-----------


"""
Ver. 1.2.0 (2023-02-23)

This module shows how to run a random policy on Double Pendulum with MuJoCo Simulation.

You will learn:

1) How to set up an own agent using MLPro's builtin random actions policy

2) How to set up an own RL scenario including your agent and MLPro's double pendulum␣
↪environment

3) How to integrate MuJoCo as the Simulation

4) How to reset and run your own scenario

"""



import random
import numpy as np
import os

import mlpro
from mlpro.bf.ml import Model
from mlpro.bf.ops import Mode
from mlpro.bf.various import Log
from mlpro.rl.models_agents import Policy, Agent
from mlpro.rl.models_train import RLScenario
from mlpro.bf.systems import State, Action
from mlpro.rl.models_env_ada import SARSElement
```

(continues on next page)

(continued from previous page)

```
from mlpro.rl.models_env import Environment
from mlpro.rl.models_agents import Reward
from mlpro.bf.systems import *


# 1 Implement the Environment
class PendulumEnvironment (Environment):

    C_NAME          = 'PendulumEnvironment'
    C_REWARD_TYPE   = Reward.C_TYPE_OVERALL

    def __init__(self,
                 p_mode=Mode.C_MODE_SIM,
                 p_mujoco_file=None,
                 p_frame_skip: int = 1,
                 p_state_mapping=None,
                 p_action_mapping=None,
                 p_camera_conf: tuple = (None, None, None),
                 p_visualize: bool = False,
                 p_logging=Log.C_LOG_ALL):

        super().__init__(p_mode=p_mode,
                         p_mujoco_file=p_mujoco_file,
                         p_frame_skip=p_frame_skip,
                         p_state_mapping=p_state_mapping,
                         p_action_mapping=p_action_mapping,
                         p_camera_conf=p_camera_conf,
                         p_visualize=p_visualize,
                         p_logging=p_logging)


        self._state = State(self._state_space)
        self.reset()


    def _compute_reward(self, p_state_old: State = None, p_state_new: State = None) ->␣
↪Reward:
        reward = Reward(self.C_REWARD_TYPE)
        reward.set_overall_reward(1)
        return reward


    def _reset(self, p_seed=None) -> None:
        pass

# 1 Implement your own agent policy
class MyPolicy (Policy):

    C_NAME      = 'MyPolicy'

    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)
```

(continues on next page)

(continued from previous page)

```
    def compute_action(self, p_state: State) -> Action:
        # 1.1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 1.2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = np.random.uniform(-50, 50)

        # 1.3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)


    def _adapt(self, p_sars_elem: SARSElement) -> bool:
        # 1.4 Adapting the internal policy is up to you...
        self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')

        # 1.5 Only return True if something has been adapted...
        return False


# 2 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize:bool, p_logging) -> Model:
        # 2.1 Setup environment
        model_file = os.path.join(os.path.dirname(mlpro.__file__), "bf/systems/pool/
↪mujoco", "doublependulum.xml")
        self._env = PendulumEnvironment(p_logging=logging, p_mujoco_file=model_file, p_
↪visualize=visualize)

        # 2.2 Setup standard single-agent with own policy
        return Agent( p_policy=MyPolicy( p_observation_space=self._env.get_state_space(),
                                         p_action_space=self._env.get_action_space(),
                                         p_buffer_size=1,
                                         p_ada=p_ada,
                                         p_visualize=p_visualize,
                                         p_logging=p_logging),
                      p_envmodel=None,
                      p_name='Smith',
                      p_ada=p_ada,
                      p_visualize=p_visualize,
                      p_logging=p_logging)

# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    cycle_limit = 2000
    logging     = Log.C_LOG_ALL
```

(continues on next page)

(continued from previous page)

```
    visualize   = True

else:
    # 3.2 Parameters for internal unit test
    cycle_limit = 10
    logging     = Log.C_LOG_NOTHING
    visualize   = False


# 3.3 Create your scenario and run some cycles
myscenario  = MyScenario(
        p_mode=Mode.C_MODE_SIM,
        p_ada=True,
        p_cycle_limit=cycle_limit,
        p_visualize=visualize,
        p_logging=logging
)

myscenario.reset(p_seed=3)

myscenario.run()
```

Results

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*

8.3.4 Adaptive Environments

Howto RL-AE-001: Available Soon

8.3.5 Model-based Reinforcement Learning

Howto RL-MB-001: Train and Reload Model Based Agent (Gym)

Ver. 1.0.3 (2023-03-10)

This module shows how to train a single agent in MBRL and load it again to do some extra cycles.

At the moment, this module has not provided the success story, but we focus more on demonstrating how to execute MBRL in MLPro. However, in the near future, we are going to optimize the settings and assure that we bring a success story of this approach.

You will learn:

1) How to use the RLScenario class of MLPro.
2) How to save a scenario after some run.
3) How to reload the saved scenario and re-run for additional cycles.

Prerequisites

Please install the following packages to run this examples properly:

- PyTorch
- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_mb_001_train_and_reload_model_based_agent_gym.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-03-07  1.0.0     DA       Adapted from howto_rl_agent_011
## -- 2023-03-08  1.0.1     SY       Refactoring and quality assurance
## -- 2023-03-10  1.0.2     SY       Renumbering module
## -- 2023-03-10  1.0.3     SY       Refactoring
## -------------------------------------------------------------------------------------------

"""
Ver. 1.0.3 (2023-03-10)

This module shows how to train a single agent in MBRL and load it again to do some extra␣
↪cycles.

At the moment, this module has not provided the success story, but we focus more on␣
↪demonstrating
how to execute MBRL in MLPro. However, in the near future, we are going to optimize the␣
↪settings and
assure that we bring a success story of this approach.

You will learn:

1) How to use the RLScenario class of MLPro.

2) How to save a scenario after some run.

3) How to reload the saved scenario and re-run for additional cycles.

"""


import gym
import torch
from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
```

(continues on next page)

(continued from previous page)

```
from mlpro.sl.pool.afct.fnn.pytorch.mlp import PyTorchMLP
from pathlib import Path




## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------

# 1 Implement your own RL scenario
class MyScenario (RLScenario):

    C_NAME = 'Matrix'


## -------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:

        # 1.1 Setup environment
        gym_env = gym.make('CartPole-v1')
        self._env = WrEnvGYM2MLPro(gym_env, p_visualize=p_visualize, p_logging=p_logging)
        self._env.reset()

        # 1.2 Setup Policy from SB3
        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=10,
            env=None,
            _init_setup_model=False,
            device="cpu",
            seed=1)

        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)


        # 1.4 Setup a model-based agent

        # 1.4.1 Setup adaptive function for state transition and reward function
        afct_strans = AFctSTrans( p_afct_cls=PyTorchMLP,
                                  p_state_space=self._env.get_state_space(),
```

(continues on next page)

(continued from previous page)

```
                                p_action_space=self._env.get_action_space(),
                                p_threshold=0.1,
                                p_buffer_size=5000,
                                p_ada=p_ada,
                                p_visualize=p_visualize,
                                p_logging=p_logging,
                                p_update_rate=1,
                                p_num_hidden_layers=3,
                                p_hidden_size=128,
                                p_activation_fct=torch.nn.ReLU,
                                p_output_activation_fct=torch.nn.ReLU,
                                p_optimizer=torch.optim.Adam,
                                p_loss_fct=torch.nn.MSELoss,
                                p_learning_rate=3e-4 )


        afct_rewards = AFctReward( p_afct_cls=PyTorchMLP,
                                p_state_space=self._env.get_state_space(),
                                p_action_space=self._env.get_action_space(),
                                p_threshold=0.1,
                                p_buffer_size=5000,
                                p_ada=p_ada,
                                p_visualize=p_visualize,
                                p_logging=p_logging,
                                p_update_rate=1,
                                p_num_hidden_layers=3,
                                p_hidden_size=128,
                                p_activation_fct=torch.nn.ReLU,
                                p_output_activation_fct=torch.nn.ReLU,
                                p_optimizer=torch.optim.Adam,
                                p_loss_fct=torch.nn.MSELoss,
                                p_learning_rate=3e-4 )


        # 1.4.2 Setup environment model
        envmodel = EnvModel( p_observation_space=self._env.get_state_space(),
                             p_action_space=self._env.get_action_space(),
                             p_latency=self._env.get_latency(),
                             p_afct_strans=afct_strans,
                             p_afct_reward=afct_rewards,
                             p_afct_success=self._env,
                             p_afct_broken=self._env,
                             p_ada=p_ada,
                             p_init_states=self._env.get_state(),
                             p_visualize=p_visualize,
                             p_logging=p_logging )


        mb_training_param = dict( p_cycle_limit=100,
                                p_cycles_per_epi_limit=100,
                                p_max_stagnations=0,
                                p_collect_states=False,
```

(continues on next page)

(continued from previous page)

```
                                    p_collect_actions=False,
                                    p_collect_rewards=False,
                                    p_collect_training=False )

        # 1.4.3 Setup standard single-agent with sb3 policy and environment model
        return Agent(
            p_policy=policy_wrapped,
            p_envmodel=envmodel,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
            **mb_training_param
        )




## ------------------------------------------------------------------------------------------
↪----------
## ------------------------------------------------------------------------------------------
↪----------

if __name__ == '__main__':
    # Parameters for demo mode
    cycle_limit = 10000
    adaptation_limit = 0
    stagnation_limit = 0
    eval_frequency = 0
    eval_grp_size = 0
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
```

(continues on next page)

(continued from previous page)

```
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging )



# 3 Training
training.run()



# 4 Reload the scenario
if __name__ == '__main__':
    input( '\nTraining finished. Press ENTER to reload and run the scenario...\n')

scenario = MyScenario.load( p_path = training.get_training_path() + os.sep + 'scenario' )


# 5 Reset Scenario
scenario.reset()


# 6 Run Scenario
scenario.run()

if __name__ != '__main__':
    from shutil import rmtree
    rmtree(training.get_training_path())
else:
    input( '\nPress ENTER to finish...')
```

Results

After the environment is initiated, the training will run for the specified amount of limits. The expected initial console output can be seen below.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": -- Training episode 0 started...
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------

YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent "": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  S  Agent "": Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": -- Training run 0 started...
```

(continues on next page)

(continued from previous page)

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------

YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": -- Training episode 0 started...
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training "RL": ----------------------------------------
↪-----------------------------------

YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent "": Action computation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent "": Action computation finished
YYYY-MM-DD  HH:MM:SS.SSSSSS  S  Agent "": Adaptation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent "": Action computation started
...
```

After termination the local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Environment Model*
- *API Reference - RL Scenario and Training*

Howto RL-MB-002: MBRL with MPC on Grid World Environment

Ver. 2.0.2 (2023-03-10)

This module shows how to incorporate MPC in Model-Based RL on Grid World problem as well as using PyTorch-based MLP network from MLPro-SL's pool of objects.

You will learn:

1) How to set up an own agent on Grid World problem
2) How to set up model-based RL (MBRL) training using network from MLPro-SL's pool
3) How to incorporate MPC into MBRL training
4) How to use multiprocessing on MPC

Prerequisites

Please install the following packages to run this examples properly:

- PyTorch
- NumPy

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_mb_002_grid_world_environment.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-09-19  0.0.0     SY       Creation
## -- 2022-10-06  1.0.0     SY       Release first version
## -- 2022-10-07  1.0.1     SY       Add plotting
## -- 2022-10-13  1.0.2     SY       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-02-02  1.2.0     DA       Refactoring
## -- 2023-02-04  1.2.1     SY       Add multiprocessing functionality and refactoring
## -- 2023-02-10  1.2.2     SY       Switch multiprocessing to threading
## -- 2023-03-07  2.0.0     SY       Update due to MLPro-SL
## -- 2023-03-08  2.0.1     SY       Refactoring
## -- 2023-03-10  2.0.2     SY       Refactoring
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 2.0.2 (2023-03-10)

This module shows how to incorporate MPC in Model-Based RL on Grid World problem as well␣
↪as using
PyTorch-based MLP network from MLPro-SL's pool of objects.

You will learn:

1) How to set up an own agent on Grid World problem

2) How to set up model-based RL (MBRL) training using network from MLPro-SL's pool

3) How to incorporate MPC into MBRL training

4) How to use multiprocessing on MPC

"""


import torch
import numpy as np
from mlpro.bf.plot import DataPlotting
from mlpro.bf.math import *
from mlpro.rl import *
```

(continues on next page)

(continued from previous page)

```
from mlpro.rl.models import *
from mlpro.rl.pool.envs.gridworld import *
from mlpro.rl.pool.policies.randomgenerator import RandomGenerator
from mlpro.sl.pool.afct.fnn.pytorch.mlp import PyTorchMLP
from pathlib import Path
from mlpro.rl.pool.actionplanner.mpc import MPC
from mlpro.rl.pool.envs.gridworld import *
import mlpro.bf.mt as mt




## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------

# 1 Implement the random RL scenario
class ScenarioGridWorld(RLScenario):

    C_NAME      = 'Grid World with Random Actions'


## -------------------------------------------------------------------------------------------
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env   = GridWorld(p_logging=p_logging,
                                p_action_type=GridWorld.C_ACTION_TYPE_DISC_2D,
                                p_max_step=100)


        # 1.2 Setup and return random action agent
        policy_random = RandomGenerator(p_observation_space=self._env.get_state_space(),
                                        p_action_space=self._env.get_action_space(),
                                        p_buffer_size=1,
                                        p_ada=1,
                                        p_logging=p_logging)

        # Setup Adaptive Function
        afct_strans = AFctSTrans(
            PyTorchMLP,
            p_state_space=self._env._state_space,
            p_action_space=self._env._action_space,
            p_threshold=0.1,
            p_buffer_size=100,
            p_ada=p_ada,
            p_logging=p_logging,
            p_update_rate=1,
            p_num_hidden_layers=3,
            p_hidden_size=128,
```

(continues on next page)

(continued from previous page)

```
            p_activation_fct=torch.nn.ReLU,
            p_output_activation_fct=torch.nn.ReLU,
            p_optimizer=torch.optim.Adam,
            p_loss_fct=torch.nn.MSELoss,
            p_learning_rate=3e-4
        )

        envmodel = EnvModel(
            p_observation_space=self._env._state_space,
            p_action_space=self._env._action_space,
            p_latency=self._env.get_latency(),
            p_afct_strans=afct_strans,
            p_afct_reward=self._env,
            p_afct_success=self._env,
            p_afct_broken=self._env,
            p_ada=p_ada,
            p_init_states=self._env.get_state(),
            p_logging=p_logging
        )

        mb_training_param = dict(p_cycle_limit=100,
                                 p_cycles_per_epi_limit=100,
                                 p_max_stagnations=0,
                                 p_collect_states=False,
                                 p_collect_actions=False,
                                 p_collect_rewards=False,
                                 p_collect_training=False)

        return Agent(
            p_policy=policy_random,
            p_envmodel=envmodel,
            p_em_acc_thsld=0.8,
            p_action_planner=MPC(p_range_max=mt.Async.C_RANGE_THREAD,
                                 p_logging=p_logging),
            p_predicting_horizon=5,
            p_controlling_horizon=1,
            p_planning_width=50,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
            **mb_training_param
        )




## ------------------------------------------------------------------------------------------
↪----------
## ------------------------------------------------------------------------------------------
↪----------
```

(continues on next page)

(continued from previous page)

```
# 2 Train agent in scenario
if __name__ == "__main__":
    # 2.1 Parameters for demo mode
    cycle_limit = 5000
    logging     = Log.C_LOG_ALL
    visualize   = True
    path        = str(Path.home())
    plotting    = True
else:
    # 2.2 Parameters for internal unit test
    cycle_limit = 10
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None
    plotting    = False

training = RLTraining(
    p_scenario_cls=ScenarioGridWorld,
    p_cycle_limit=cycle_limit,
    p_cycles_per_epi_limit=100,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_collect_training=True,
    p_path=path,
    p_logging=logging,
)

training.run()




## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------

# 3 Plotting with MLpro
class MyDataPlotting(DataPlotting):


## ------------------------------------------------------------------------------------------
↪-----------
    def get_plots(self):
        """
        A function to plot data
        """
        for name in self.data.names:
            maxval = 0
```

(continues on next page)

(continued from previous page)

```
            minval = 0
            if self.printing[name][0]:
                fig = plt.figure(figsize=(7, 7))
                raw = []
                label = []
                ax = fig.subplots(1, 1)
                ax.set_title(name)
                ax.grid(True, which="both", axis="both")
                for fr_id in self.data.frame_id[name]:
                    raw.extend(self.data.get_values(name, fr_id))
                    if self.printing[name][1] == -1:
                        maxval = max(raw)
                        minval = min(raw)
                    else:
                        maxval = self.printing[name][2]
                        minval = self.printing[name][1]

                    label.append("%s" % fr_id)
                ax.plot(raw)
                ax.set_ylim(minval - (abs(minval) * 0.1), maxval + (abs(maxval) * 0.1))
                plt.xlabel("continuous cycles")
                self.plots[0].append(name)
                self.plots[1].append(ax)
                if self.showing:
                    plt.show()
                else:
                    plt.close(fig)




## ------------------------------------------------------------------------------------------
↪----------
## ------------------------------------------------------------------------------------------
↪----------

if __name__ == "__main__":
    data_printing = {
        "Cycle": [False],
        "Day": [False],
        "Second": [False],
        "Microsecond": [False],
        "Smith": [True, -1],
    }

    mem = training.get_results().ds_rewards
    mem_plot = MyDataPlotting(mem, p_showing=plotting, p_printing=data_printing)
    mem_plot.get_plots()
```

Results

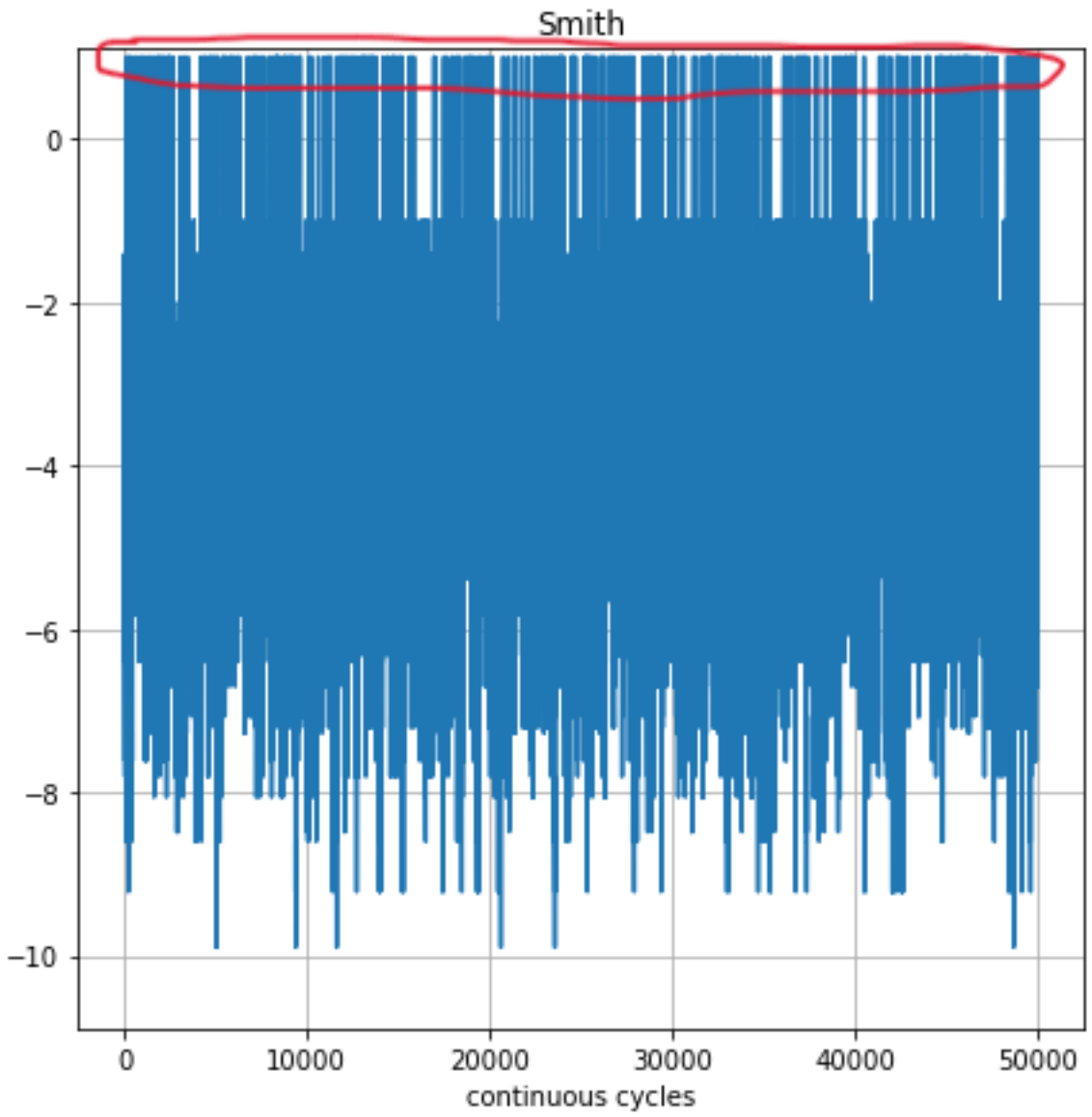
After the environment is initiated, the training will run for the specified amount of limits. The expected initial console

output can be seen below.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment Grid World: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment Grid World: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment Grid World: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment Grid World: Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: -- Training run 0 started...
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------

YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: -- Evaluation period 0 started...
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------

YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: -- Evaluation episode 0 started...
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training RL: ----------------------------------------
↪----------------------------------
...
```

After termination the local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- trained model.pkl



The image above shows that the agent in most cases can reach the goal.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Environment Model*
- *API Reference - RL Scenario and Training*

Howto RL-MB-003: MBRL on RobotHTM Environment

Ver. 2.0.2 (2023-03-10)

This module demonstrates model-based reinforcement learning (MBRL) with native algorithm and action planner using MPC.

You will learn:

1) How to set up an model-based agent with action planner
2) How to set up scenario for Robothtm and also with SB3 wrapper
3) How to run the scenario and train the agent
4) How to plot from the generated results

Prerequisites

Please install the following packages to run this examples properly:

- Pytorch
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_mb_003_robothtm_environment.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-12-17  0.0.0     MRD      Creation
## -- 2021-12-17  1.0.0     MRD      Released first version
## -- 2022-01-01  1.0.1     MRD      Refactoring due to new model implementation
## -- 2022-05-20  1.0.2     MRD      Add HTMEnvModel
## -- 2022-08-09  1.0.3     SY       Update due to introduction of ActionPlanner
## -- 2022-08-15  1.0.4     SY       - Renaming maturity to accuracy
## --                                - Utilize MPC from pool of objects
## -- 2022-10-13  1.0.5     SY       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-02-02  1.2.0     DA       Refactoring
## -- 2023-02-04  1.2.1     SY       Refactoring to avoid printing during unit test
## -- 2023-03-07  2.0.0     SY       Update due to MLPro-SL
## -- 2023-03-08  2.0.1     SY       Refactoring
## -- 2023-03-10  2.0.2     SY       Renumbering module
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 2.0.2 (2023-03-10)

This module demonstrates model-based reinforcement learning (MBRL) with native algorithm␣
↪and
action planner using MPC.
```

(continues on next page)

(continued from previous page)

```
You will learn:

1) How to set up an model-based agent with action planner

2) How to set up scenario for Robothtm and also with SB3 wrapper

3) How to run the scenario and train the agent

4) How to plot from the generated results

"""


import torch
import transformations
from mlpro.bf.plot import DataPlotting
from mlpro.bf.ml import *
from mlpro.rl import *
from mlpro.rl.models import *
from mlpro.rl.pool.envs.robotinhtm import RobotArm3D
from mlpro.rl.pool.envs.robotinhtm import RobotHTM
from stable_baselines3 import PPO
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from mlpro.sl.pool.afct.pytorch import *
from mlpro.sl import *
from mlpro.rl.pool.actionplanner.mpc import MPC
from pathlib import Path
from collections import deque




## ------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------
↪-----------

#1 Define htm_robotinhtm
class DHLayer(torch.nn.Module):
    """
    This class represents a layer architecture based on DH Parameter as the learnable␣
↪parameter
    and use them to construct the transformation matrix

    Parameters
    ----------
    p_in : int
        Number of Joints
    """
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
    def __init__(self, p_in):
        super(DHLayer, self).__init__()
        self._in = p_in

        self.register_parameter("alpha", torch.nn.Parameter((torch.rand(self._in, 1) - 0.
↪5) * 1))
        self.register_parameter("beta", torch.nn.Parameter((torch.rand(self._in, 1) - 0.
↪5) * 1))
        self.register_parameter("a", torch.nn.Parameter((torch.rand(self._in, 1) - 0.5)␣
↪* 1))
        self.register_parameter("b", torch.nn.Parameter((torch.rand(self._in, 1) - 0.5)␣
↪* 1))
        self.register_parameter("d", torch.nn.Parameter((torch.rand(self._in, 1) - 0.5)␣
↪* 1))


## ------------------------------------------------------------------------------------------
↪-----------
    def forward(self, p_in):
        batch_size = p_in.shape[0]

        l_alpha = self.alpha.repeat(batch_size, 1).reshape(batch_size, self._in)
        l_beta = self.beta.repeat(batch_size, 1).reshape(batch_size, self._in)
        l_tx = self.a.repeat(batch_size, 1, 1).reshape(batch_size, self._in, 1, 1)
        l_ty = self.b.repeat(batch_size, 1, 1).reshape(batch_size, self._in, 1, 1)
        l_tz = self.d.repeat(batch_size, 1, 1).reshape(batch_size, self._in, 1, 1)

        # Construct Z Transformations
        unit = torch.Tensor([[0.0, 0.0, 1.0]], device=p_in.device)
        trans_z = self.construct_transformation(unit, p_in, l_tz)

        # Construct Y Transformations
        unit = torch.Tensor([[0.0, 1.0, 0.0]])
        trans_y = self.construct_transformation(unit, l_beta, l_ty)

        # Construct X Transformations
        unit = torch.Tensor([[1.0, 0.0, 0.0]], device=p_in.device)
        trans_x = self.construct_transformation(unit, l_alpha, l_tx)

        # Construct DH Matrix
        # dh_mat = trans_z * trans_y * trans_x
        dh_mat = torch.matmul(trans_z, trans_y)
        dh_mat = torch.matmul(dh_mat, trans_x)

        return dh_mat


## ------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
    def construct_transformation(self, p_unit, p_angle, p_transl):
        """
        Construct Transformation Matrix

        Parameters
        ----------
        p_unit :
            Rotation and Translation Unit Vector
        p_angle :
            Rotation Angle
        p_transl :
            Translation

        Returns
        -------
        trans_mat :
            Transformations Matrix
        """

        # Create Rotation Matrix
        rot_mat = self.create_rot_mat(p_unit.to(p_angle.device), p_angle)

        # Create Translation Vector
        transl_vec = p_unit * p_transl

        # Combine Rotation Matrix and Translation Vector into Transformation Matrix
        trans_mat = self.trans_rot_to_transformations_mat(transl_vec, rot_mat)

        return trans_mat


## ------------------------------------------------------------------------------------------
↪-----------
    def trans_rot_to_transformations_mat(self, p_trans, p_rot):
        """
        Combine Translation Vector and Rotation Matrix into Transformation Matrix

        Parameters
        ----------
        p_trans :
            Translation Vector
        p_rot :
            Rotation Matrix

        Returns
        -------
        trans_mat :
            Transformation Matrix
        """
        batch_size = p_trans.shape[0]
        fixed_vec = torch.Tensor([[0.0, 0.0, 0.0, 1.0]], device=p_trans.device)
        trans_mat = torch.cat([p_rot, p_trans.permute(0, 1, 3, 2)], dim=3)
```

(continues on next page)

(continued from previous page)

```
        trans_mat = torch.cat([trans_mat, fixed_vec.repeat(batch_size, self._in, 1, 1)],␣
↪dim=2)
        return trans_mat


## ------------------------------------------------------------------------------------
↪-----------
    def create_trans_vec(self):
        pass


## ------------------------------------------------------------------------------------
↪-----------
    def create_rot_mat(self, p_unit, p_angle):
        """
        Create Rotation Matrix

        Parameters
        ----------
        p_unit :
            Rotation Unit Vector
        p_angle :
            Rotation Angle

        Returns
        -------
        rot_mat :
            Rotation Matrix
        """
        batch_size = p_angle.shape[0]
        masking_indices = torch.tensor([[3, 2, 1], [2, 3, 0], [1, 0, 3]], device=p_angle.
↪device)
        masking_cross = torch.Tensor([[0, -1, 1], [1, 0, -1], [-1, 1, 0]], device=p_
↪angle.device)

        unit = p_unit
        unit_stack = torch.Tensor([], device=p_angle.device)
        outer_prod = torch.Tensor([], device=p_angle.device)

        for i in range(self._in):
            outer_prod = torch.cat([outer_prod, torch.ger(unit[0], unit[0])])
            k_unit = torch.cat([unit, torch.Tensor([[0.0]], device=p_angle.device)],␣
↪dim=1)
            unit_store = k_unit[0][masking_indices] * masking_cross
            unit_stack = torch.cat([unit_stack, unit_store])

        unit = unit_stack.reshape(self._in, 3, 3).repeat(batch_size, 1, 1, 1)
        outer = outer_prod.reshape(self._in, 3, 3).repeat(batch_size, 1, 1, 1)

        rot_mat = unit * torch.sin(p_angle).reshape(batch_size, self._in, 1, 1) + (
                torch.eye(3, device=p_angle.device).repeat(batch_size, 1, 1, 1) - outer)␣
↪* \
```

(continues on next page)

(continued from previous page)

```
                torch.cos(p_angle).reshape(batch_size, self._in, 1, 1) + outer

        return rot_mat




## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------
class N3(torch.nn.Module):


## ------------------------------------------------------------------------------------------
↪-----------
    def __init__(self, n_in):
        super(N3, self).__init__()
        self.n_in = n_in
        self.added = 0


## ------------------------------------------------------------------------------------------
↪-----------
    def forward(self, I):
        BatchSize = I.shape[0]

        A = torch.eye(4)
        b = torch.tril(torch.ones(self.n_in + self.added + 1, self.n_in + self.added)).
↪flatten()
        c = torch.triu(torch.ones(self.n_in + self.added + 1, self.n_in + self.added),␣
↪diagonal=1).flatten()

        maskIlower = torch.einsum('ij,k->kij', A, b).reshape(self.n_in + self.added + 1,␣
↪self.n_in + self.added, 4, 4)
        maskIupper = torch.einsum('ij,k->kij', A, c).reshape(self.n_in + self.added + 1,␣
↪self.n_in + self.added, 4, 4)

        output1 = torch.matmul(maskIlower, I.reshape(BatchSize, 1, self.n_in + self.
↪added, 4, 4))
        output1 = torch.add(output1, maskIupper)
        output1 = output1.permute(0, 2, 1, 3, 4)

        output = torch.eye(4).repeat(BatchSize, self.n_in + self.added + 1, 1, 1)
        for outnum in range(self.n_in + self.added):
            output = torch.matmul(output, output1[:, outnum])

        return output
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------
class RobotHTMModel(torch.nn.Module):
    """
    Provide Forward Kinematic based on Neural Network with DH Layer.
    Predicts the end-effector position with given joint angles.
    """


## ------------------------------------------------------------------------------------------
↪-----------
    def __init__(self, p_in, p_t):
        super(RobotHTMModel, self).__init__()
        self._in = p_in
        self._t = p_t

        self.dh_layer = DHLayer(self._in)
        self.eef_layer = N3(self._in)


## ------------------------------------------------------------------------------------------
↪-----------
    def forward(self, p_in):
        batch_size=p_in.shape[0]
        new_i = p_in.reshape(batch_size,2,self._in) * torch.cat([torch.Tensor([self._t]).
↪repeat(1,self._in), torch.ones(1,self._in)])
        new_i = torch.sum(new_i,dim=1)

        trans = self.dh_layer(new_i)
        output = self.eef_layer(trans)

        return output




## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------
class RobothtmAFct(SLAdaptiveFunction, PyTorchHelperFunctions):
    C_NAME = "Robothtm Adaptive Function"
    C_BUFFER_CLS = PyTorchBuffer


## ------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
    def _hyperparameters_check(self):
        pass


## ------------------------------------------------------------------------------------
↪-----------
    def _setup_model(self):
        self.joint_num = self._output_space.get_num_dim() - 6
        net_model = RobotHTMModel(self.joint_num, 0.01)
        self.optimizer = torch.optim.Adam(net_model.parameters(), lr=3e-4)
        self.loss_dyn = torch.nn.MSELoss()
        self.train_model = True
        self.input_temp = None

        self.sim_env = RobotArm3D()

        joints = []

        jointType = []
        vectLinkLength = [[0, 0, 0], [0, 0, 0]]
        jointType.append("rz")
        for joint in range(self.joint_num - 1):
            vectLinkLength.append([0, 0.7, 0])
            jointType.append("rx")

        jointType.append("f")

        for x in range(len(jointType)):
            vectorLink = dict(x=vectLinkLength[x][0], y=vectLinkLength[x][1],␣
↪z=vectLinkLength[x][2])
            joint = dict(
                Joint_name="Joint %d" % x,
                Joint_type=jointType[x],
                Vector_link_length=vectorLink,
            )
            joints.append(joint)

        for robo in joints:
            self.sim_env.add_link_joint(
                lvector=torch.Tensor(
                    [
                        [
                            robo["Vector_link_length"]["x"],
                            robo["Vector_link_length"]["y"],
                            robo["Vector_link_length"]["z"],
                        ]
                    ]
                ),
                jointAxis=robo["Joint_type"],
                thetaInit=torch.Tensor([np.radians(0)]),
            )
```

(continues on next page)

(continued from previous page)

```
        self.sim_env.update_joint_coords()

        return net_model


## ------------------------------------------------------------------------------------
↪-----------
    def _input_preproc(self, p_input: torch.Tensor) -> torch.Tensor:
        input = torch.cat([p_input[0][6+self.joint_num:], p_input[0][6:6+self.joint_
↪num]])
        input = input.reshape(1,self.joint_num*2)
        self.input_temp = p_input[0][:3].reshape(1,3)

        return input


## ------------------------------------------------------------------------------------
↪-----------
    def _output_postproc(self, p_output: torch.Tensor) -> torch.Tensor:
        angles = torch.Tensor([])
        thets = torch.zeros(3)
        for idx in range(self.joint_num):
            angle = torch.Tensor(transformations.euler_from_matrix(p_output[-1][idx][:].
↪detach().numpy(), 'rxyz')) - thets
            thets = torch.Tensor(transformations.euler_from_matrix(p_output[-1][idx][:].
↪detach().numpy(), 'rxyz'))
            angles = torch.cat([angles, torch.norm(angle).reshape(1, 1)], dim=1)

        output = torch.cat([self.input_temp, p_output[-1][-1][:3, [-1]].reshape(1,3)],␣
↪dim=1)
        output = torch.cat([output, angles], dim=1)

        return output


## ------------------------------------------------------------------------------------
↪-----------
    def _adapt_online(self, p_input: Element, p_output: Element) -> bool:
        model_input = deque(p_input.get_values()[6:])
        model_input.rotate(self.joint_num)
        model_input = torch.Tensor([list(model_input)])

        self.sim_env.set_theta(torch.Tensor([p_output.get_values()[6 : 6 + self.joint_
↪num]]))
        self.sim_env.update_joint_coords()

        model_output = self.sim_env.get_homogeneous().reshape(self.joint_num+1,4,4)

        self._add_buffer(PyTorchIOElement(model_input, model_output))

        if self._buffer.get_internal_counter() % 100 != 0:
            return False
```

(continues on next page)

(continued from previous page)

```
        # Divide Test and Train
        if self.train_model:
            dataset_size = len(self._buffer)
            indices = list(range(dataset_size))
            split = int(np.floor(0.2 * dataset_size))
            np.random.seed(random.randint(1,1000))
            np.random.shuffle(indices)
            train_indices, test_indices = indices[split:], indices[:split]

            train_sampler = SubsetRandomSampler(train_indices)
            test_sampler = SubsetRandomSampler(test_indices)
            trainer = torch.utils.data.DataLoader(self._buffer, batch_size=100,␣
↪sampler=train_sampler)
            tester = torch.utils.data.DataLoader(self._buffer, batch_size=100,␣
↪sampler=test_sampler)

            # Training
            self._sl_model.train()

            for i, (In, Label) in enumerate(trainer):
                outputs = self._sl_model(In)
                loss = self.loss_dyn(outputs, Label)
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()

            test_loss = 0

            self._sl_model.eval()
            for i, (In, Label) in enumerate(tester):
                outputs = self._sl_model(In)
                loss = self.loss_dyn(outputs, Label)
                test_loss += loss.item()


            if test_loss/len(tester) < 5e-9:
                self.train_model = False

        return True


## ------------------------------------------------------------------------------------
↪-----------
    def _add_buffer(self, p_buffer_element: PyTorchIOElement):
        self._buffer.add_element(p_buffer_element)


## ------------------------------------------------------------------------------------
↪-----------
    def _map(self, p_input: Element, p_output: Element):
        # Input pre processing
```

(continues on next page)

(continued from previous page)

```
        input = self.input_preproc(p_input)

        # Make prediction
        output = self._sl_model(input)

        # Output post processing
        output = self.output_postproc(output)

        p_output.set_values(output)




## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------
class HTMEnvModel(EnvModel):
    C_NAME = "HTM Env Model"


## ------------------------------------------------------------------------------------------
↪-----------
    def __init__(
        self,
        p_num_joints=4,
        p_target_mode="Random",
        p_ada=True,
        p_logging=False,
    ):

        self._robot_htm = RobotHTM(
            p_num_joints=p_num_joints,
            p_target_mode=p_target_mode,
            p_logging=p_logging
        )

        # Setup Adaptive Function
        # HTM Function Here
        afct_strans = AFctSTrans(
            RobothtmAFct,
            p_state_space=self._robot_htm._state_space,
            p_action_space=self._robot_htm._action_space,
            p_threshold=1.8,
            p_buffer_size=20000,
            p_ada=p_ada,
            p_logging=p_logging,
        )

        EnvModel.__init__(
            self,
```

(continues on next page)

(continued from previous page)

```
            p_observation_space=self._robot_htm._state_space,
            p_action_space=self._robot_htm._action_space,
            p_latency=timedelta(seconds=self._robot_htm.dt),
            p_afct_strans=afct_strans,
            p_afct_reward=None,
            p_afct_success=None,
            p_afct_broken=None,
            p_ada=p_ada,
            p_init_states=self._robot_htm.get_state(),
            p_logging=p_logging,
        )

        self.reset()


## ------------------------------------------------------------------------------------
↪-----------
    def _reset(self, p_seed=None):
        self._robot_htm._reset(p_seed)
        self._state = State(self._state_space)
        self._state.set_values(self._robot_htm.get_state().get_values())
        return self._robot_htm._reset(p_seed)


## ------------------------------------------------------------------------------------
↪-----------
    def _compute_reward(self, p_state_old: State = None, p_state_new: State = None) ->␣
↪Reward:
        return self._robot_htm._compute_reward(p_state_old, p_state_new)




## ------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------
↪-----------
class ActualTraining(RLTraining):
    C_NAME = "Actual"




## ------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------
↪-----------

# 2 Implement RL Scenario for the actual environment to train the environment model
class ScenarioRobotHTMActual(RLScenario):
    C_NAME = "Matrix1"
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = RobotHTM(p_visualize=p_visualize, p_logging=p_logging)

        policy_kwargs = dict(activation_fn=torch.nn.Tanh,
                             net_arch=[dict(pi=[128, 128], vf=[128, 128])])

        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=100,
            env=None,
            _init_setup_model=False,
            policy_kwargs=policy_kwargs,
            device="cpu"
        )

        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
        )

        mb_training_param = dict(p_cycle_limit=100,
                                 p_cycles_per_epi_limit=100,
                                 p_max_stagnations=0,
                                 p_collect_states=False,
                                 p_collect_actions=False,
                                 p_collect_rewards=False,
                                 p_collect_training=False)

        # 1.2 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped,
            p_envmodel=HTMEnvModel(),
            p_em_acc_thsld=0.5,
            p_action_planner=MPC(p_logging=p_logging),
            p_predicting_horizon=5,
            p_controlling_horizon=2,
            p_planning_width=5,
            p_name="Smith1",
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
            **mb_training_param
        )
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------

# 3 Train agent in scenario
if __name__ == "__main__":
    # 2.1 Parameters for demo mode
    cycle_limit = 300000
    logging     = Log.C_LOG_ALL
    visualize   = True
    path        = str(Path.home())
    plotting    = True
else:
    # 2.2 Parameters for internal unit test
    cycle_limit = 100
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None
    plotting    = False


training = ActualTraining(
    p_scenario_cls=ScenarioRobotHTMActual,
    p_cycle_limit=cycle_limit,
    p_cycles_per_epi_limit=100,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_collect_training=True,
    p_path=path,
    p_logging=logging,
)

training.run()




## ------------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------------
↪-----------

# 4 Create Plotting Class
class MyDataPlotting(DataPlotting):
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
    def get_plots(self):
        """
        A function to plot data
        """
        for name in self.data.names:
            maxval = 0
            minval = 0
            if self.printing[name][0]:
                fig = plt.figure(figsize=(7, 7))
                raw = []
                label = []
                ax = fig.subplots(1, 1)
                ax.set_title(name)
                ax.grid(True, which="both", axis="both")
                for fr_id in self.data.frame_id[name]:
                    raw.append(np.sum(self.data.get_values(name, fr_id)))
                    if self.printing[name][1] == -1:
                        maxval = max(raw)
                        minval = min(raw)
                    else:
                        maxval = self.printing[name][2]
                        minval = self.printing[name][1]

                    label.append("%s" % fr_id)
                ax.plot(raw)
                ax.set_ylim(minval - (abs(minval) * 0.1), maxval + (abs(maxval) * 0.1))
                ax.set_xlabel("Episode")
                ax.legend(label, bbox_to_anchor=(1, 0.5), loc="center left")
                self.plots[0].append(name)
                self.plots[1].append(ax)
                if self.showing:
                    plt.show()
                else:
                    plt.close(fig)


# 5 Plotting using MLpro
if __name__ == "__main__":
    data_printing = {
        "Cycle": [False],
        "Day": [False],
        "Second": [False],
        "Microsecond": [False],
        "Smith1": [True, -1],
    }

    mem = training.get_results().ds_rewards
    mem_plot = MyDataPlotting(mem, p_showing=plotting, p_printing=data_printing)
    mem_plot.get_plots()
```

Results

After the environment is initiated, the training will run for the specified amount of limits. The expected initial console output can be seen below.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Training Actual: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment RobotHTM: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment RobotHTM: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment RobotHTM: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment RobotHTM: Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  SB3 Policy ????: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  SB3 Policy ????: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Smith1: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Smith1: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  SB3 Policy ????: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Matrix1: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Matrix1: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Training Actual: Training started (without␣
↪hyperparameter tuning)
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Results  RL: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training Actual: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training Actual: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training Actual: -- Training run 0 started...
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training Actual: ----------------------------------------
↪----------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Training Actual: ----------------------------------------
↪----------------------------------

YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Matrix1: Process time 0:00:00 : Scenario␣
↪reset with seed 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment RobotHTM: Reset
...
```

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Environment Model*
- *API Reference - RL Scenario and Training*

8.3.6 Advanced Training Techniques

Howto RL-ATT-001: Train and Reload Single Agent using Stagnation Detection (Gym)

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_att_001_train_and_reload_single_agent_gym_sd.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-03-04  1.0.0     DA       Creation as derivate of howto_rl_agent_011
## -------------------------------------------------------------------------------------------

"""
Ver. 1.0.0 (2023-03-04)

As in Howto RL AGENT 011, this module shows how to train a single agent and load it again to do some
extra cycles. In opposite to howto 011, stagnation detection is used to automatically end the
training if no further progress can be made.

You will learn:

1. How to use the RLScenario class of MLPro.

2. How to save a scenario after some run.

3. How to reload the saved scenario and re-run for additional cycles.

4. How to use stagnation detection to end the training automatically if there is no progress.

"""


import gym
from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from pathlib import Path



# 1 Implement your own RL scenario
class MyScenario (RLScenario):
    C_NAME = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        gym_env = gym.make('CartPole-v1')
```

(continues on next page)

(continued from previous page)

```
        self._env = WrEnvGYM2MLPro(gym_env, p_visualize=p_visualize, p_logging=p_logging)

        # 1.2 Setup Policy From SB3
        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=10,
            env=None,
            _init_setup_model=False,
            device="cpu",
            seed=1)

        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)

        # 1.4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )



if __name__ == '__main__':
    # Parameters for demo mode
    cycle_limit = 20000
    adaptation_limit = 0
    stagnation_limit = 5
    eval_frequency = 10
    eval_grp_size = 5
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
```

(continues on next page)

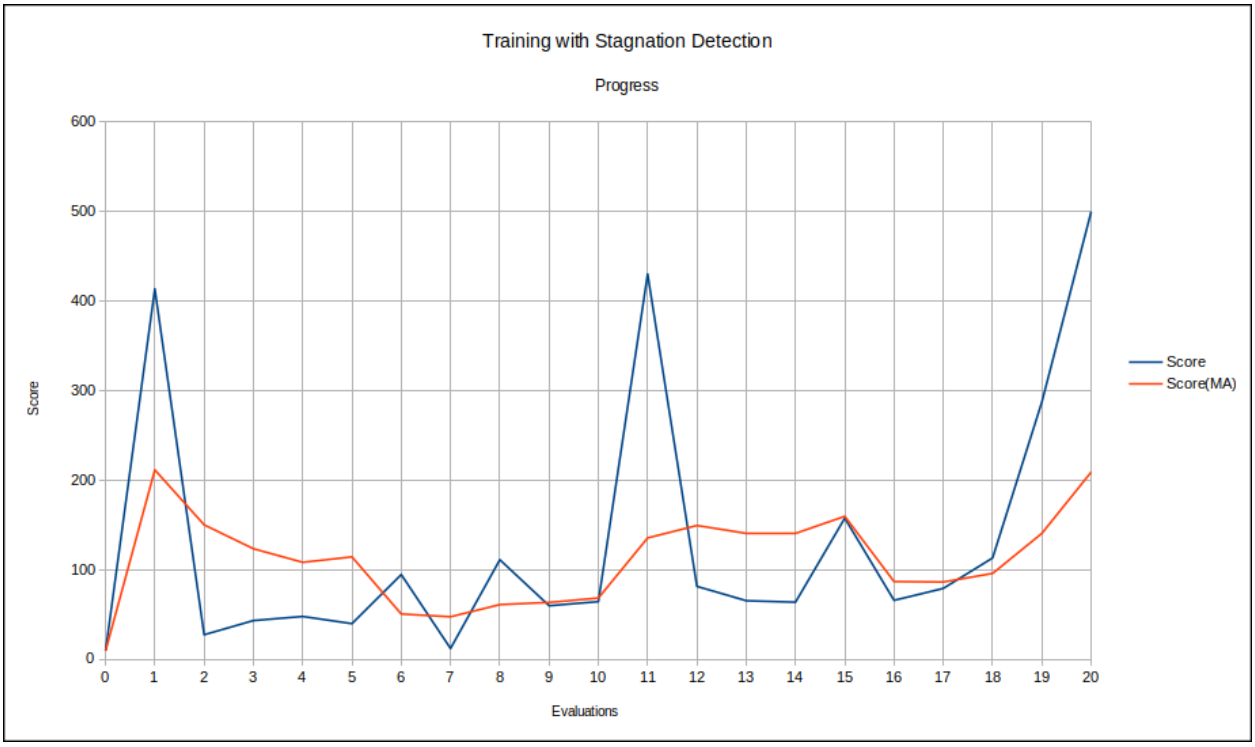
(continued from previous page)

```
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging )



# 3 Training
training.run()



# 4 Reload the scenario
if __name__ == '__main__':
    input( '\nTraining finished. Press ENTER to reload and run the scenario...\n')

scenario = MyScenario.load( p_path = training.get_training_path() + os.sep + 'scenario' )


# 5 Reset Scenario
scenario.reset()


# 6 Run Scenario
scenario.run()

if __name__ != '__main__':
    from shutil import rmtree
    rmtree(training.get_training_path())
else:
    input( '\nPress ENTER to finish...')
```

Results

The Gym Cartpole environment window appears during training and shows an improved control behavior after a while. After the training, the related scenario is reloaded and run for a further episode to demonstrate the final control behavior.

The training itself is terminated due to automatic stagnation detection. The chart below shows the training progress and the ending at the point of maximum possible reward:



After termination the local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- scenario

Cross Reference

- *MLPro-RL: Training*
- *Howto RL-AGENT-011: Train and Reload Single Agent (Gym)*
- *API Reference*

Howto RL-ATT-002: Train and Reload Single Agent using Stagnation Detection Cartpole Discrete (MuJoCo)

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------------
-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_att_002_train_and_reload_single_agent_mujoco_sd_cartpole_
discrete.py
## -------------------------------------------------------------------------------------------
-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-03-04  1.0.0     DA       Creation as derivate of howto_rl_agent_021
## -- 2023-03-07  1.0.1     MRD      Renamed
## -------------------------------------------------------------------------------------------
-----------

"""
Ver. 1.0.1 (2023-03-07)

As in Howto RL AGENT 021, this module shows how to train a single agent with SB3 Policy␣
on Discrete
Cartpole MuJoCo Environment. In opposite to howto 021, stagnation detection is used to␣
automatically
end the training if no further progress can be made.

You will learn:

1. How to use MLPro's RLScenario class.

2. How to create sb3 policy object.

3. How to create SB3 policy in MLPro.

4. How to setup and run RLTraining in MLPro.

"""


from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from mlpro.rl.pool.envs.cartpole import CartpoleMujocoDiscrete
from pathlib import Path


class MyScenario(RLScenario):
    C_NAME = "Matrix"

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = CartpoleMujocoDiscrete(p_logging=logging, p_visualize=visualize)

        # 1.2 Setup Policy From SB3
```

(continues on next page)

(continued from previous page)

```
        policy_sb3 = PPO(policy="MlpPolicy", n_steps=10, env=None, _init_setup_
↪model=False, device="cpu", seed=1)

        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
        )

        # 1.4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped, p_envmodel=None, p_name="Smith", p_ada=p_ada, p_
↪visualize=p_visualize, p_logging=p_logging
        )


# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # Parameters for demo mode
    cycle_limit = 20000
    adaptation_limit = 0
    stagnation_limit = 5
    eval_frequency = 10
    eval_grp_size = 5
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
```

(continues on next page)

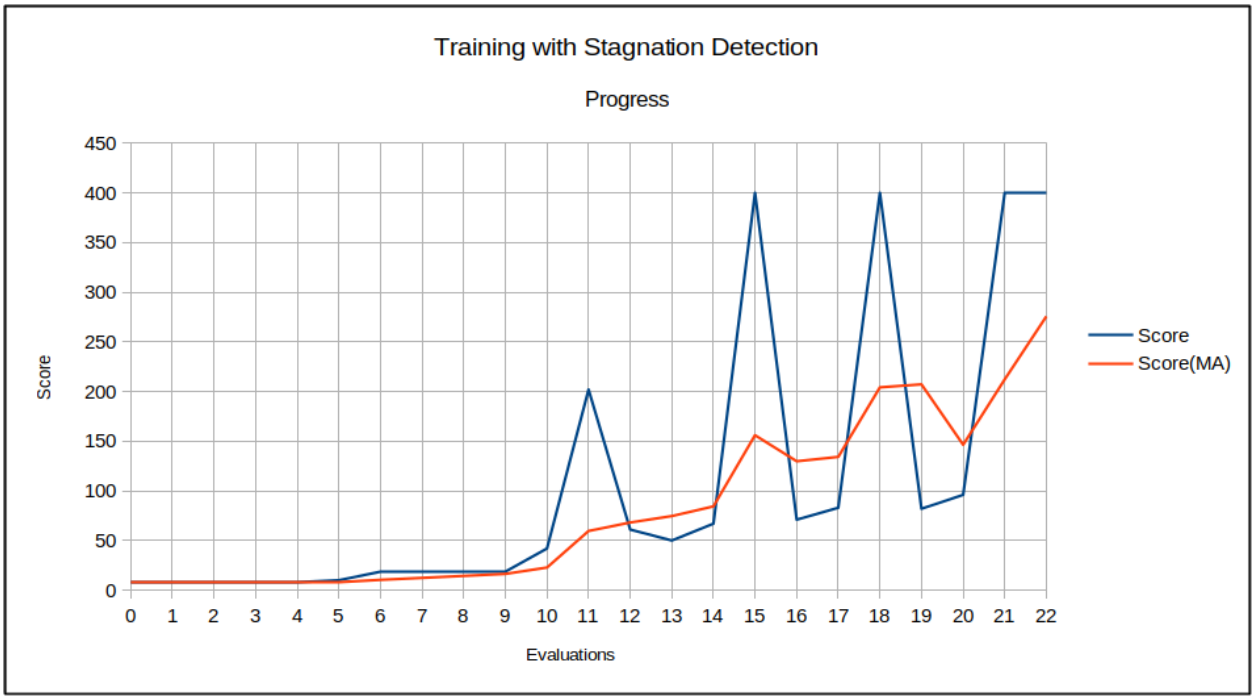
(continued from previous page)

```
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging,
)


# 3 Training
training.run()


# 4 Reload the scenario
if __name__ == '__main__':
    input( '\nTraining finished. Press ENTER to reload and run the scenario...\n')

scenario = MyScenario.load( p_path = training.get_training_path() + os.sep + 'scenario' )


# 5 Reset Scenario
scenario.reset()


# 6 Run Scenario
scenario.run()

if __name__ != '__main__':
    from shutil import rmtree
    rmtree(training.get_training_path())
else:
    input( '\nPress ENTER to finish...')
```

Results

The MuJoCo Cartpole environment window appears during training and shows an improved control behavior after a while. After the training, the related scenario is reloaded and run for a further episode to demonstrate the final control behavior.

The training itself is terminated due to automatic stagnation detection. The chart below shows the training progress and the ending at the point of maximum possible reward:



After termination the local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- scenario

Cross Reference

- *MLPro-RL: Training*
- *Howto RL-AGENT-021: Train and Reload Single Agent (MuJoCo)*
- *API Reference*

Howto RL-ATT-003: Train and Reload Single Agent using Stagnation Detection Cartpole Continuous (MuJoCo)

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## ----------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_att_003_train_and_reload_single_agent_mujoco_sd_cartpole_
↪continuous.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2023-03-07  1.0.0     MRD      Creation as derivate of howto_rl_agent_022
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.0 (2023-03-07)

As in Howto RL AGENT 022, this module shows how to train a single agent with SB3 Policy␣
↪on Continuous
Cartpole MuJoCo Environment. In opposite to howto 022, stagnation detection is used to␣
↪automatically
end the training if no further progress can be made.

You will learn:

1. How to use MLPro's RLScenario class.

2. How to create sb3 policy object.

3. How to create SB3 policy in MLPro.

4. How to setup and run RLTraining in MLPro.

"""


from stable_baselines3 import PPO
from mlpro.rl import *
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from mlpro.rl.pool.envs.cartpole import CartpoleMujocoContinuous
from pathlib import Path


class MyScenario(RLScenario):
    C_NAME = "Matrix"

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1.1 Setup environment
        self._env = CartpoleMujocoContinuous(p_logging=logging, p_visualize=visualize)

        # 1.2 Setup Policy From SB3
        policy_sb3 = PPO(policy="MlpPolicy", n_steps=10, env=None, _init_setup_
↪model=False, device="cpu", seed=1)
```

(continues on next page)

(continued from previous page)

```
        # 1.3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging,
        )

        # 1.4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped, p_envmodel=None, p_name="Smith", p_ada=p_ada, p_
↪visualize=p_visualize, p_logging=p_logging
        )


# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # Parameters for demo mode
    cycle_limit = 20000
    adaptation_limit = 0
    stagnation_limit = 5
    eval_frequency = 10
    eval_grp_size = 5
    logging = Log.C_LOG_WE
    visualize = True
    path = str(Path.home())

else:
    # Parameters for internal unit test
    cycle_limit = 50
    adaptation_limit = 5
    stagnation_limit = 5
    eval_frequency = 2
    eval_grp_size = 1
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = str(Path.home())


# 2 Create scenario and start training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_adaptation_limit=adaptation_limit,
    p_stagnation_limit=stagnation_limit,
    p_eval_frequency=eval_frequency,
    p_eval_grp_size=eval_grp_size,
    p_path=path,
    p_visualize=visualize,
```

(continues on next page)

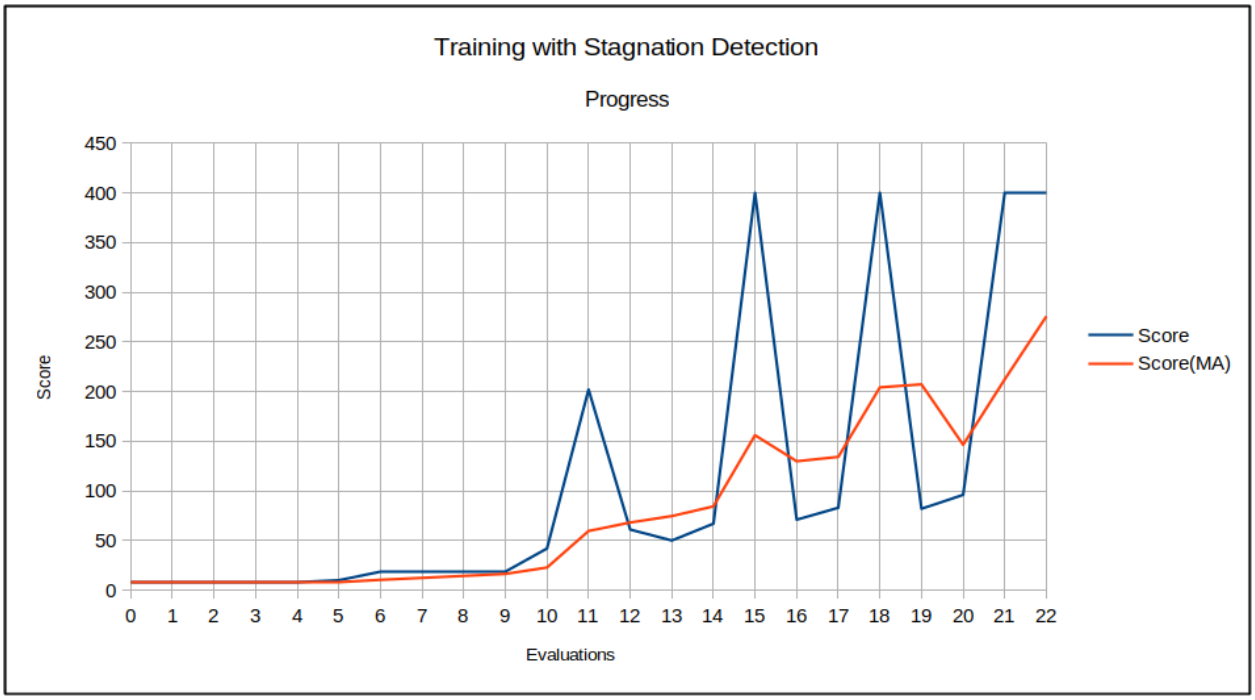
(continued from previous page)

```
    p_logging=logging,
)


# 3 Training
training.run()


# 4 Reload the scenario
if __name__ == '__main__':
    input( '\nTraining finished. Press ENTER to reload and run the scenario...\n')

scenario = MyScenario.load( p_path = training.get_training_path() + os.sep + 'scenario' )


# 5 Reset Scenario
scenario.reset()


# 6 Run Scenario
scenario.run()

if __name__ != '__main__':
    from shutil import rmtree
    rmtree(training.get_training_path())
else:
    input( '\nPress ENTER to finish...')
```

Results

The MuJoCo Cartpole environment window appears during training and shows an improved control behavior after a while. After the training, the related scenario is reloaded and run for a further episode to demonstrate the final control behavior.

The training itself is terminated due to automatic stagnation detection. The chart below shows the training progress and the ending at the point of maximum possible reward:



After termination the local result folder contains the training result files:

- agent_actions.csv
- env_rewards.csv
- env_states.csv
- evaluation.csv
- summary.csv
- scenario

Cross Reference

- *MLPro-RL: Training*
- *Howto RL-AGENT-022: Train and Reload Single Agent (MuJoCo)*
- *API Reference*

8.3.7 Hyperparameter Tuning Tools

Howto RL-HT-001: Hyperparameter Tuning using Hyperopt

Ver. 1.1.1 (2022-11-21)

This module demonstrates how to utilize wrapper class for Hyperopt in RL context.

You will learn:

1) How to set up a policy and its parameters
2) How to use Hyperopt wrapper.
3) How to tune the parameters using Hyperopt.

Prerequisites

Please install the following packages to run this examples properly:

- Hyperopt

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_ht_001_hyperopt.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-12-08  0.0.0     SY       Creation
## -- 2021-12-08  1.0.0     SY       Release of first version
## -- 2022-01-21  1.0.1     DA       Renaming: tupel -> tuple
## -- 2022-01-27  1.0.2     SY       Class WrHPTHyperopt enhancement
## -- 2022-02-25  1.0.3     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-10-12  1.0.4     DA       Renaming and minor fixes
## -- 2022-10-17  1.0.5     SY       Refactoring
## -- 2022-11-02  1.0.6     DA       Refactoring
## -- 2022-11-09  1.1.0     DA       Refactoring
## -- 2022-11-21  1.1.1     DA       Corrections on logging
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.1.1 (2022-11-21)

This module demonstrates how to utilize wrapper class for Hyperopt in RL context.

You will learn:

1) How to set up a policy and its parameters

2) How to use Hyperopt wrapper.

3) How to tune the parameters using Hyperopt.

"""


from mlpro.wrappers.hyperopt import *
from mlpro.rl.pool.envs.bglp import BGLP
from mlpro.rl import *
import random
from pathlib import Path
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------------

# 1. Create a policy and setup the hyperparameters
class myPolicy (Policy):

    C_NAME      = 'MyPolicy'

## -------------------------------------------------------------------------------------------
    def __init__(self, p_observation_space:MSpace, p_action_space:MSpace, p_buffer_size=1, p_ada=True, p_logging=Log.C_LOG_ALL):
        """
         Parameters:
            p_observation_space     Subspace of an environment that is observed by the policy
            p_action_space          Action space object
            p_buffer_size           Size of the buffer
            p_ada                   Boolean switch for adaptivity
            p_logging               Boolean switch for logging functionality
        """
        super().__init__( p_observation_space=p_observation_space, p_action_space=p_action_space, p_buffer_size=p_buffer_size, p_ada=p_ada, p_logging=p_logging)
        self._hyperparam_space  = HyperParamSpace()
        self._hyperparam_tuple  = None
        self._init_hyperparam()


## -------------------------------------------------------------------------------------------
    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)


## -------------------------------------------------------------------------------------------
    def _init_hyperparam(self):
        self._hyperparam_space.add_dim(HyperParam('num_states','Z', p_boundaries = [1,100]))
        self._hyperparam_space.add_dim(HyperParam('smoothing','R', p_boundaries = [0.1,0.5]))
        self._hyperparam_space.add_dim(HyperParam('lr_rate','R', p_boundaries = [0.001,0.1]))
        self._hyperparam_space.add_dim(HyperParam('buffer_size','Z', p_boundaries =[10000,100000]))
        self._hyperparam_space.add_dim(HyperParam('update_rate','Z', p_boundaries = [5,20]))
        self._hyperparam_space.add_dim(HyperParam('sampling_size','Z', p_boundaries =[64,256]))
        self._hyperparam_tuple = HyperParamTuple(self._hyperparam_space)
```

(continues on next page)

(continued from previous page)

```
        ids_ = self._hyperparam_tuple.get_dim_ids()
        self._hyperparam_tuple.set_value(ids_[0], 100)
        self._hyperparam_tuple.set_value(ids_[1], 0.035)
        self._hyperparam_tuple.set_value(ids_[2], 0.0001)
        self._hyperparam_tuple.set_value(ids_[3], 100000)
        self._hyperparam_tuple.set_value(ids_[4], 100)
        self._hyperparam_tuple.set_value(ids_[5], 256)


## ------------------------------------------------------------------------------------
↪-----------
    def compute_action(self, p_state: State) -> Action:
        my_action_values = np.zeros(self._action_space.get_num_dim())
        for d in range(self._action_space.get_num_dim()):
            self.set_random_seed(None)
            my_action_values[d] = random.random()
        return Action(self._id, self._action_space, my_action_values)


## ------------------------------------------------------------------------------------
↪-----------
    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')
        return False




## ------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------
↪-----------

# 2. Create a Scenario
class BGLP_Rnd(RLScenario):

    C_NAME      = 'BGLP_Dummy'

## ------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        self._env       = BGLP(p_logging=p_logging)
        self._agent     = MultiAgent(p_name='Dummy Policy', p_ada=1, p_logging=p_logging)
        state_space     = self._env.get_state_space()
        action_space    = self._env.get_action_space()


        # Agent 1
        _name         = 'BELT_CONVEYOR_A'
        _id           = 0
```

(continues on next page)

(continued from previous page)

```
        _ospace       = state_space.spawn([state_space.get_dim_ids()[0],state_space.get_
↪dim_ids()[1]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[0]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=Log.C_LOG_NOTHING)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging),
            p_weight=1.0
            )


        # Agent 2
        _name         = 'VACUUM_PUMP_B'
        _id           = 1
        _ospace       = state_space.spawn([state_space.get_dim_ids()[1],state_space.get_
↪dim_ids()[2]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[1]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=Log.C_LOG_NOTHING)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging),
            p_weight=1.0
            )


        # Agent 3
        _name         = 'VIBRATORY_CONVEYOR_B'
        _id           = 2
        _ospace       = state_space.spawn([state_space.get_dim_ids()[2],state_space.get_
↪dim_ids()[3]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[2]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=Log.C_LOG_NOTHING)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
```

(continues on next page)

(continued from previous page)

```
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging),
            p_weight=1.0
            )


        # Agent 4
        _name         = 'VACUUM_PUMP_C'
        _id           = 3
        _ospace       = state_space.spawn([state_space.get_dim_ids()[3],state_space.get_
↪dim_ids()[4]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[3]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=Log.C_LOG_NOTHING)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging),
            p_weight=1.0
            )


        # Agent 5
        _name         = 'ROTARY_FEEDER_C'
        _id           = 4
        _ospace       = state_space.spawn([state_space.get_dim_ids()[4],state_space.get_
↪dim_ids()[5]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[4]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=Log.C_LOG_NOTHING)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging),
            p_weight=1.0
            )

        return self._agent
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
## ------------------------------------------------------------------------------------------

if __name__ == "__main__":
    # Parameters for demo mode
    logging         = Log.C_LOG_ALL
    visualize       = False
    dest_path       = str(Path.home())
    cycle_limit     = 100
    cycle_per_ep    = 10
    eval_freq       = 2
    eval_grp_size   = 5
    adapt_limit     = 0
    stagnant_limit  = 5
    score_ma_hor    = 5

else:
    # Parameters for internal unit test
    logging         = Log.C_LOG_NOTHING
    visualize       = False
    dest_path       = None
    cycle_limit     = 3
    cycle_per_ep    = 1
    eval_freq       = 2
    eval_grp_size   = 1
    adapt_limit     = 0
    stagnant_limit  = 0
    score_ma_hor    = 0


# 3. Instantiate a hyperopt wrapper
myHyperopt = WrHPTHyperopt(p_logging=logging,
                           p_algo=WrHPTHyperopt.C_ALGO_TPE,
                           p_ids=None)


# 4. Train players in the scenario and turn the hyperparamter tuning on
training        = RLTraining(
    p_scenario_cls=BGLP_Rnd,
    p_cycle_limit=cycle_limit,
    p_cycles_per_epi_limit=cycle_per_ep,
    p_eval_frequency=eval_freq,
    p_eval_grp_size=eval_grp_size,
    p_adaptation_limit=adapt_limit,
    p_stagnation_limit=stagnant_limit,
    p_score_ma_horizon=score_ma_hor,
    p_hpt=myHyperopt,
```

(continues on next page)

(continued from previous page)

```
    p_hpt_trials=10,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_path=dest_path,
    p_logging=logging )

training.run()
```

Results

```
2023-02-12  16:44:58.495569  I  HyperParam Tuner "Hyperopt": Instantiated
2023-02-12  16:44:58.496565  I  HyperParam Tuner "Hyperopt": Wrapped package hyperopt␣
↪installed in version 0.2.7
2023-02-12  16:44:58.496565  I  HyperParam Tuner "Hyperopt": Hyperopt configuration is␣
↪successful
2023-02-12  16:44:58.496565  I  Training "RL": Instantiated
2023-02-12  16:44:58.497574  I  Training "RL": Training started (with hyperparameter␣
↪tuning)
2023-02-12  16:44:58.503565  I  HyperParam Tuner "Hyperopt": Spaces for hyperopt is ready
2023-02-12  16:44:58.503565  I  HyperParam Tuner "Hyperopt": ----------------------------
↪--------------------------------------------------

2023-02-12  16:44:58.536565
I  HyperParam Tuner "Hyperopt":
Trial number 0 has started

2023-02-12  16:44:58.537565
I  HyperParam Tuner "Hyperopt":
-------------------------------------------------------------------------------

2023-02-12  16:45:00.336566
I  HyperParam Tuner "Hyperopt":
Trial number 1 has finished

2023-02-12  16:45:00.337566
I  HyperParam Tuner "Hyperopt":
-------------------------------------------------------------------------------

2023-02-12  16:45:00.353564
I  HyperParam Tuner "Hyperopt":
Trial number 1 has started

2023-02-12  16:45:00.353564
I  HyperParam Tuner "Hyperopt":
-------------------------------------------------------------------------------

2023-02-12  16:45:02.055567
I  HyperParam Tuner "Hyperopt":
Trial number 2 has finished

2023-02-12  16:45:02.056566
```

(continues on next page)

(continued from previous page)

```
I  HyperParam Tuner "Hyperopt":
-------------------------------------------------------------------------------
```

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*
- *API Reference - Machine Learning*

Howto RL-HT-002: Hyperparameter Tuning using Optuna

Prerequisites

Please install the following packages to run this examples properly:

- Optuna

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_ht_002_optuna.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-03-24  0.0.0     SY       Creation
## -- 2022-03-24  1.0.0     SY       Release of first version
## -- 2022-04-05  1.0.1     SY       Add tuning recap visualization
## -- 2022-10-12  1.0.2     DA       Renaming and minor fixes
## -- 2022-10-17  1.0.3     SY       Refactoring
## -- 2022-11-02  1.0.4     DA       Refactoring
## -- 2022-11-09  1.1.0     DA       Refactoring
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.1.0 (2022-11-09)

This module demonstrates how to utilize wrapper class for Optuna in RL context.

You will learn:

1) How to set up a policy and its parameters

2) How to use optuna wrapper.

3) How to tune the parameters using optuna.
```

(continues on next page)

(continued from previous page)

```
"""


from mlpro.wrappers.optuna import *
from mlpro.rl.pool.envs.bglp import BGLP
from mlpro.rl import *
import random
from pathlib import Path




## -------------------------------------------------------------------------------------
## -------------------------------------------------------------------------------------

# 1. Create a policy and setup the hyperparameters
class myPolicy (Policy):

    C_NAME      = 'MyPolicy'

## -------------------------------------------------------------------------------------
    def __init__(self, p_observation_space:MSpace, p_action_space:MSpace, p_buffer_
↪size=1, p_ada=True, p_logging=True):
        """
         Parameters:
            p_observation_space     Subspace of an environment that is observed by the␣
↪policy
            p_action_space          Action space object
            p_buffer_size           Size of the buffer
            p_ada                   Boolean switch for adaptivity
            p_logging               Boolean switch for logging functionality
        """
        super().__init__(p_observation_space, p_action_space, p_buffer_size, p_ada, p_
↪logging)
        self._hyperparam_space  = HyperParamSpace()
        self._hyperparam_tuple  = None
        self._init_hyperparam()


## -------------------------------------------------------------------------------------
    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)


## -------------------------------------------------------------------------------------
    def _init_hyperparam(self):
        self._hyperparam_space.add_dim(HyperParam('num_states','Z', p_boundaries = [1,
```

(continues on next page)

(continued from previous page)

```
↪100]))
        self._hyperparam_space.add_dim(HyperParam('smoothing','R', p_boundaries = [0.1,0.
↪5]))
        self._hyperparam_space.add_dim(HyperParam('lr_rate','R', p_boundaries = [0.001,0.
↪1]))
        self._hyperparam_space.add_dim(HyperParam('buffer_size','Z', p_boundaries =␣
↪[10000,100000]))
        self._hyperparam_space.add_dim(HyperParam('update_rate','Z', p_boundaries = [5,
↪20]))
        self._hyperparam_space.add_dim(HyperParam('sampling_size','Z', p_boundaries =␣
↪[64,256]))
        self._hyperparam_tuple = HyperParamTuple(self._hyperparam_space)

        ids_ = self._hyperparam_tuple.get_dim_ids()
        self._hyperparam_tuple.set_value(ids_[0], 100)
        self._hyperparam_tuple.set_value(ids_[1], 0.035)
        self._hyperparam_tuple.set_value(ids_[2], 0.0001)
        self._hyperparam_tuple.set_value(ids_[3], 100000)
        self._hyperparam_tuple.set_value(ids_[4], 100)
        self._hyperparam_tuple.set_value(ids_[5], 256)


## ------------------------------------------------------------------------------------
↪-----------
    def compute_action(self, p_state: State) -> Action:
        my_action_values = np.zeros(self._action_space.get_num_dim())
        for d in range(self._action_space.get_num_dim()):
            self.set_random_seed(None)
            my_action_values[d] = random.random()
        return Action(self._id, self._action_space, my_action_values)


## ------------------------------------------------------------------------------------
↪-----------
    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')
        return False




## ------------------------------------------------------------------------------------
↪-----------
## ------------------------------------------------------------------------------------
↪-----------

# 2. Create a Scenario
class BGLP_Rnd(RLScenario):

    C_NAME      = 'BGLP_Dummy'
```

(continues on next page)

(continued from previous page)

```
## ------------------------------------------------------------------------------------------
↪-----------
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        self._env       = BGLP(p_logging=logging)
        self._agent     = MultiAgent(p_name='Dummy Policy', p_ada=1, p_logging=p_logging)
        state_space     = self._env.get_state_space()
        action_space    = self._env.get_action_space()


        # Agent 1
        _name         = 'BELT_CONVEYOR_A'
        _id           = 0
        _ospace       = state_space.spawn([state_space.get_dim_ids()[0],state_space.get_
↪dim_ids()[1]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[0]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=False)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=logging),
            p_weight=1.0
            )


        # Agent 2
        _name         = 'VACUUM_PUMP_B'
        _id           = 1
        _ospace       = state_space.spawn([state_space.get_dim_ids()[1],state_space.get_
↪dim_ids()[2]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[1]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=False)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=logging),
            p_weight=1.0
            )


        # Agent 3
```

(continues on next page)

(continued from previous page)

```
        _name         = 'VIBRATORY_CONVEYOR_B'
        _id           = 2
        _ospace       = state_space.spawn([state_space.get_dim_ids()[2],state_space.get_
↪dim_ids()[3]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[2]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=False)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=logging),
            p_weight=1.0
            )


        # Agent 4
        _name         = 'VACUUM_PUMP_C'
        _id           = 3
        _ospace       = state_space.spawn([state_space.get_dim_ids()[3],state_space.get_
↪dim_ids()[4]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[3]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=False)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=logging),
            p_weight=1.0
            )


        # Agent 5
        _name         = 'ROTARY_FEEDER_C'
        _id           = 4
        _ospace       = state_space.spawn([state_space.get_dim_ids()[4],state_space.get_
↪dim_ids()[5]])
        _aspace       = action_space.spawn([action_space.get_dim_ids()[4]])
        _policy       = myPolicy(p_observation_space=_ospace, p_action_space=_aspace, p_
↪buffer_size=1, p_ada=1, p_logging=False)
        self._agent.add_agent(
            p_agent=Agent(
                p_policy=_policy,
```

(continues on next page)

(continued from previous page)

```
                p_envmodel=None,
                p_name=_name,
                p_id=_id,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=logging),
            p_weight=1.0
            )

        return self._agent




## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------

if __name__ == "__main__":
    # Parameters for demo mode
    logging         = Log.C_LOG_ALL
    visualize       = False
    dest_path       = str(Path.home())
    cycle_limit     = 100
    cycle_per_ep    = 10
    eval_freq       = 2
    eval_grp_size   = 5
    adapt_limit     = 0
    stagnant_limit  = 5
    score_ma_hor    = 5

else:
    # Parameters for internal unit test
    logging         = Log.C_LOG_NOTHING
    visualize       = False
    dest_path       = None
    cycle_limit     = 3
    cycle_per_ep    = 1
    eval_freq       = 2
    eval_grp_size   = 1
    adapt_limit     = 0
    stagnant_limit  = 0
    score_ma_hor    = 0


# 3. Instantiate a hyperopt wrapper
myOptuna = WrHPTOptuna(p_logging=logging,
                       p_ids=None,
                       p_visualization=visualize)
```

(continues on next page)

(continued from previous page)

```
# 4. Train players in the scenario and turn the hyperparamter tuning on
training        = RLTraining(
    p_scenario_cls=BGLP_Rnd,
    p_cycle_limit=cycle_limit,
    p_cycles_per_epi_limit=cycle_per_ep,
    p_eval_frequency=eval_freq,
    p_eval_grp_size=eval_grp_size,
    p_adaptation_limit=adapt_limit,
    p_stagnation_limit=stagnant_limit,
    p_score_ma_horizon=score_ma_hor,
    p_hpt=myOptuna,
    p_hpt_trials=10,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_path=dest_path,
    p_logging=logging
)

training.run()
```

Results

```
2023-02-12  16:50:55.790961  I  Wrapper "Optuna": Instantiated
2023-02-12  16:50:56.033963  I  Wrapper "Optuna": Wrapped package optuna installed in␣
↪version 3.1.0
2023-02-12  16:50:56.033963  I  Wrapper "Optuna": Optuna configuration is successful
2023-02-12  16:50:56.033963  I  Training "RL": Instantiated
2023-02-12  16:50:56.033963  I  Training "RL": Training started (with hyperparameter␣
↪tuning)
2023-02-12  16:50:56.035961  I  Environment "BGLP": Instantiated
2023-02-12  16:50:56.035961  I  Environment "BGLP": Reset
2023-02-12  16:50:56.037961  I  Policy "MyPolicy 41746e2c-045e-485e-9dbf-7cbfc1acaddb":␣
↪Instantiated
2023-02-12  16:50:56.037961  I  Policy "MyPolicy 41746e2c-045e-485e-9dbf-7cbfc1acaddb":␣
↪Adaptivity switched on
2023-02-12  16:50:56.037961  I  Agent "BELT_CONVEYOR_A": Instantiated
2023-02-12  16:50:56.037961  I  Agent "BELT_CONVEYOR_A": Adaptivity switched on
2023-02-12  16:50:56.037961  I  Policy "MyPolicy 41746e2c-045e-485e-9dbf-7cbfc1acaddb":␣
↪Adaptivity switched on
2023-02-12  16:50:56.037961  I  Agent "BELT_CONVEYOR_A": Adaptivity switched on
2023-02-12  16:50:56.037961  I  Policy "MyPolicy 41746e2c-045e-485e-9dbf-7cbfc1acaddb":␣
↪Adaptivity switched on
2023-02-12  16:50:56.038960  I  Policy "MyPolicy 4d6eff7c-d873-4806-ad4d-b02cc70bb689":␣
↪Instantiated
2023-02-12  16:50:56.038960  I  Policy "MyPolicy 4d6eff7c-d873-4806-ad4d-b02cc70bb689":␣
↪Adaptivity switched on
2023-02-12  16:50:56.038960  I  Agent "VACUUM_PUMP_B": Instantiated
2023-02-12  16:50:56.039961  I  Agent "VACUUM_PUMP_B": Adaptivity switched on
2023-02-12  16:50:56.039961  I  Policy "MyPolicy 4d6eff7c-d873-4806-ad4d-b02cc70bb689":␣
↪Adaptivity switched on
```

(continues on next page)

(continued from previous page)

```
2023-02-12  16:50:56.039961  I  Agent "VACUUM_PUMP_B": Adaptivity switched on
2023-02-12  16:50:56.039961  I  Policy "MyPolicy 4d6eff7c-d873-4806-ad4d-b02cc70bb689":␣
↪Adaptivity switched on
2023-02-12  16:50:56.039961  I  Policy "MyPolicy 47697bb1-df82-412e-b11d-da9be4d71fbb":␣
↪Instantiated
2023-02-12  16:50:56.039961  I  Policy "MyPolicy 47697bb1-df82-412e-b11d-da9be4d71fbb":␣
↪Adaptivity switched on
2023-02-12  16:50:56.040959  I  Agent "VIBRATORY_CONVEYOR_B": Instantiated
2023-02-12  16:50:56.040959  I  Agent "VIBRATORY_CONVEYOR_B": Adaptivity switched on
2023-02-12  16:50:56.040959  I  Policy "MyPolicy 47697bb1-df82-412e-b11d-da9be4d71fbb":␣
↪Adaptivity switched on
2023-02-12  16:50:56.040959  I  Agent "VIBRATORY_CONVEYOR_B": Adaptivity switched on
2023-02-12  16:50:56.040959  I  Policy "MyPolicy 47697bb1-df82-412e-b11d-da9be4d71fbb":␣
↪Adaptivity switched on
2023-02-12  16:50:56.040959  I  Policy "MyPolicy 23203733-d4bc-44c4-a154-cad82824f622":␣
↪Instantiated
2023-02-12  16:50:56.040959  I  Policy "MyPolicy 23203733-d4bc-44c4-a154-cad82824f622":␣
↪Adaptivity switched on
2023-02-12  16:50:56.041961  I  Agent "VACUUM_PUMP_C": Instantiated
2023-02-12  16:50:56.041961  I  Agent "VACUUM_PUMP_C": Adaptivity switched on
2023-02-12  16:50:56.041961  I  Policy "MyPolicy 23203733-d4bc-44c4-a154-cad82824f622":␣
↪Adaptivity switched on
2023-02-12  16:50:56.041961  I  Agent "VACUUM_PUMP_C": Adaptivity switched on
2023-02-12  16:50:56.041961  I  Policy "MyPolicy 23203733-d4bc-44c4-a154-cad82824f622":␣
↪Adaptivity switched on
2023-02-12  16:50:56.041961  I  Policy "MyPolicy b821f2cc-2add-42e3-9326-7a37bef1ba3d":␣
↪Instantiated
2023-02-12  16:50:56.041961  I  Policy "MyPolicy b821f2cc-2add-42e3-9326-7a37bef1ba3d":␣
↪Adaptivity switched on
2023-02-12  16:50:56.042961  I  Agent "ROTARY_FEEDER_C": Instantiated
2023-02-12  16:50:56.042961  I  Agent "ROTARY_FEEDER_C": Adaptivity switched on
2023-02-12  16:50:56.042961  I  Policy "MyPolicy b821f2cc-2add-42e3-9326-7a37bef1ba3d":␣
↪Adaptivity switched on
2023-02-12  16:50:56.042961  I  Agent "ROTARY_FEEDER_C": Adaptivity switched on
2023-02-12  16:50:56.042961  I  Policy "MyPolicy b821f2cc-2add-42e3-9326-7a37bef1ba3d":␣
↪Adaptivity switched on
[I 2023-02-12 16:50:56,044] A new study created in memory with name: no-name-92b887d1-
↪0699-42f7-8d74-ddc065e13b15
C:\MLPro\MLPro\src\mlpro\wrappers\optuna.py:245: FutureWarning: suggest_uniform has been␣
↪deprecated in v3.0.0. This feature will be removed in v6.0.0. See https://github.com/
↪optuna/optuna/releases/tag/v3.0.0. Use :func:`~optuna.trial.Trial.suggest_float`␣
↪instead.
  parameters.append(trial.suggest_uniform(hp_object.get_name_short()+'_'+str(x),hp_low,
↪hp_high))
2023-02-12  16:50:56.046964  I  Wrapper "Optuna": Trial number 0 has started
2023-02-12  16:50:56.046964  I  Wrapper "Optuna": ----------------------------------------
↪----------------------------------------

2023-02-12  16:50:56.047961  I  Environment "BGLP": Instantiated
2023-02-12  16:50:56.047961  I  Environment "BGLP": Reset
2023-02-12  16:50:56.048960  I  Policy "MyPolicy 2343b475-8b8a-4fa9-9551-3f91751bf067":␣
↪Instantiated
```

(continues on next page)

(continued from previous page)

```
2023-02-12  16:50:56.048960  I  Policy "MyPolicy 2343b475-8b8a-4fa9-9551-3f91751bf067":␣
↪Adaptivity switched on
2023-02-12  16:50:56.049965  I  Agent "BELT_CONVEYOR_A": Instantiated
2023-02-12  16:50:56.049965  I  Agent "BELT_CONVEYOR_A": Adaptivity switched on
2023-02-12  16:50:56.049965  I  Policy "MyPolicy 2343b475-8b8a-4fa9-9551-3f91751bf067":␣
↪Adaptivity switched on
2023-02-12  16:50:56.049965  I  Agent "BELT_CONVEYOR_A": Adaptivity switched on
2023-02-12  16:50:56.049965  I  Policy "MyPolicy 2343b475-8b8a-4fa9-9551-3f91751bf067":␣
↪Adaptivity switched on
2023-02-12  16:50:56.049965  I  Policy "MyPolicy 00d3b281-3e2f-459a-8b96-f5d45b0ef172":␣
↪Instantiated
2023-02-12  16:50:56.049965  I  Policy "MyPolicy 00d3b281-3e2f-459a-8b96-f5d45b0ef172":␣
↪Adaptivity switched on
2023-02-12  16:50:56.050961  I  Agent "VACUUM_PUMP_B": Instantiated
2023-02-12  16:50:56.050961  I  Agent "VACUUM_PUMP_B": Adaptivity switched on
2023-02-12  16:50:56.050961  I  Policy "MyPolicy 00d3b281-3e2f-459a-8b96-f5d45b0ef172":␣
↪Adaptivity switched on
2023-02-12  16:50:56.050961  I  Agent "VACUUM_PUMP_B": Adaptivity switched on
2023-02-12  16:50:56.050961  I  Policy "MyPolicy 00d3b281-3e2f-459a-8b96-f5d45b0ef172":␣
↪Adaptivity switched on
2023-02-12  16:50:56.050961  I  Policy "MyPolicy 50ca0db6-d3f7-435b-82db-5273a2df4798":␣
↪Instantiated
2023-02-12  16:50:56.050961  I  Policy "MyPolicy 50ca0db6-d3f7-435b-82db-5273a2df4798":␣
↪Adaptivity switched on
2023-02-12  16:50:56.051965  I  Agent "VIBRATORY_CONVEYOR_B": Instantiated
2023-02-12  16:50:56.051965  I  Agent "VIBRATORY_CONVEYOR_B": Adaptivity switched on
2023-02-12  16:50:56.051965  I  Policy "MyPolicy 50ca0db6-d3f7-435b-82db-5273a2df4798":␣
↪Adaptivity switched on
2023-02-12  16:50:56.051965  I  Agent "VIBRATORY_CONVEYOR_B": Adaptivity switched on
2023-02-12  16:50:56.051965  I  Policy "MyPolicy 50ca0db6-d3f7-435b-82db-5273a2df4798":␣
↪Adaptivity switched on
2023-02-12  16:50:56.051965  I  Policy "MyPolicy 2ce4e729-cacd-47e9-9e55-4137858c7467":␣
↪Instantiated
2023-02-12  16:50:56.051965  I  Policy "MyPolicy 2ce4e729-cacd-47e9-9e55-4137858c7467":␣
↪Adaptivity switched on
2023-02-12  16:50:56.052964  I  Agent "VACUUM_PUMP_C": Instantiated
2023-02-12  16:50:56.052964  I  Agent "VACUUM_PUMP_C": Adaptivity switched on
2023-02-12  16:50:56.052964  I  Policy "MyPolicy 2ce4e729-cacd-47e9-9e55-4137858c7467":␣
↪Adaptivity switched on
2023-02-12  16:50:56.052964  I  Agent "VACUUM_PUMP_C": Adaptivity switched on
2023-02-12  16:50:56.052964  I  Policy "MyPolicy 2ce4e729-cacd-47e9-9e55-4137858c7467":␣
↪Adaptivity switched on
2023-02-12  16:50:56.052964  I  Policy "MyPolicy 41c9f1a1-4e3f-4819-a768-fd714135f20c":␣
↪Instantiated
2023-02-12  16:50:56.052964  I  Policy "MyPolicy 41c9f1a1-4e3f-4819-a768-fd714135f20c":␣
↪Adaptivity switched on
2023-02-12  16:50:56.053962  I  Agent "ROTARY_FEEDER_C": Instantiated
2023-02-12  16:50:56.053962  I  Agent "ROTARY_FEEDER_C": Adaptivity switched on
2023-02-12  16:50:56.053962  I  Policy "MyPolicy 41c9f1a1-4e3f-4819-a768-fd714135f20c":␣
↪Adaptivity switched on
2023-02-12  16:50:56.053962  I  Agent "ROTARY_FEEDER_C": Adaptivity switched on
2023-02-12  16:50:56.053962  I  Policy "MyPolicy 41c9f1a1-4e3f-4819-a768-fd714135f20c":␣
```

(continues on next page)

(continued from previous page)

```
↪Adaptivity switched on
2023-02-12  16:50:56.065960  I  Wrapper "Optuna": New parameters for optuna tuner is␣
↪ready
2023-02-12  16:50:56.065960  I  Wrapper "Optuna": ----------------------------------------
↪----------------------------------------
```

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - RL Scenario and Training*
- *API Reference - Machine Learning*

8.3.8 Wrappers

Howto RL-WP-001: MLPro to OpenAI Gym

Ver. 1.0.8 (2023-03-02)

This module shows how to wrap a native MLPro environment class to OpenAI Gym environment.

You will learn:

1. How to setup an MLPro environment.
2. How to wrap MLPro's native Environment class to the Gym environment object.

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_wp_001_mlpro_environment_to_gym_environment.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-09-30  0.0.0     SY       Creation
## -- 2021-09-30  1.0.0     SY       Released first version
## -- 2021-10-04  1.0.1     DA       Minor fixes
## -- 2021-12-22  1.0.2     DA       Cleaned up a bit
## -- 2022-03-21  1.0.3     MRD      Use Gym Env Checker
## -- 2022-05-30  1.0.4     DA       Little refactoring
## -- 2022-07-28  1.0.5     SY       Update due to the latest introduction of Gym 0.25
## -- 2022-10-14  1.0.6     SY       Refactoring
## -- 2022-11-02  1.0.7     SY       Unable logging in unit test model
## -- 2023-03-02  1.0.8     LSB      Refactoring
```

(continues on next page)

(continued from previous page)

```
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.8 (2023-03-02)

This module shows how to wrap a native MLPro environment class to OpenAI Gym environment.

You will learn:

1. How to setup an MLPro environment.

2. How to wrap MLPro's native Environment class to the Gym environment object.
"""


from mlpro.bf.various import Log
from mlpro.wrappers.openai_gym import WrEnvMLPro2GYM
from mlpro.rl.pool.envs.gridworld import GridWorld
from gym.utils.env_checker import check_env


if __name__ == "__main__":
    logging = Log.C_LOG_ALL
else:
    logging = Log.C_LOG_NOTHING

# 1. Set up MLPro native environment
mlpro_env = GridWorld(p_logging=logging)

# 2. Wrap the MLPro environment to gym compatible environment
env = WrEnvMLPro2GYM(mlpro_env,
                     p_state_space=None,
                     p_action_space=None,
                     p_new_step_api=False,
                     p_logging=logging)

# 3. Check whether the environment is valid
check_env(env)
```

Results

The native MLPro GridWorld environment will be wrapped to a OpenAI Gym environment. By making use of Gym's environment checker, we could confirm the success of the environment wrapping.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Start processing action
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Actions of agent 0 = [3.
```

(continues on next page)

(continued from previous page)

```
↪415721893310547, -7.9934492111206055]
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment GridWorld: Action processing finished␣
↪successfully
...
```

There will be several more lines of action processing logs due to the nature of the environment checker. When there is no detected failure, the environment is successfully wrapped.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Wrapper OpenAI Gym*

Howto RL-WP-002: MLPro to PettingZoo

Ver. 1.0.6 (2023-03-02)

This module shows how to wrap mlpro's Environment class to petting zoo compatible.

1. How to setup an MLPro environment.
2. How to wrap MLPro's native envrionment to a Petting Zoo environment.

Prerequisites

Please install the following packages to run this examples properly:

- PettingZoo

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_wp_002_mlpro_environment_to_petting_zoo_environment.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-10-02  0.0.0     SY       Creation
## -- 2021-10-02  1.0.0     SY       Released first version
## -- 2021-10-04  1.0.1     DA       Minor fix
## -- 2021-11-15  1.0.2     DA       Refactoring
## -- 2021-12-03  1.0.3     DA       Refactoring
## -- 2022-10-14  1.0.4     SY       Refactoring
## -- 2022-11-02  1.0.5     SY       Unable logging in unit test model
## -- 2023-03-02  1.0.6     LSB      Refactoring
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.0.6 (2023-03-02)
```

(continues on next page)

(continued from previous page)

```
This module shows how to wrap mlpro's Environment class to petting zoo compatible.

1. How to setup an MLPro environment.

2. How to wrap MLPro's native envrionment to a Petting Zoo environment.
"""


from mlpro.bf.various import Log
from mlpro.wrappers.pettingzoo import WrEnvMLPro2PZoo
from mlpro.rl.pool.envs.bglp import BGLP
from pettingzoo.test import api_test


if __name__ == "__main__":
    logging = Log.C_LOG_ALL
else:
    logging = Log.C_LOG_NOTHING

# 1. Set up MLPro native environment
mlpro_env = BGLP(p_logging=logging)

# 2. Wrap the MLPro environment to PettingZoo compatible environment
env = WrEnvMLPro2PZoo(mlpro_env,
                      p_num_agents=5,
                      p_state_space=None,
                      p_action_space=None,
                      p_logging=logging).pzoo_env

# 3. Check whether the environment is valid
try:
    api_test(env, num_cycles=10, verbose_progress=False)
    print("test completed")
    assert True
except:
    print("test failed")
    assert False
```

Results

The Bulk Good Laboratory Plant (BGLP) environment will be wrapped to a PettingZoo compliant environment.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Reset
Starting API test
...
Passed API test
test completed
```

There are several lines of action processing logs due to the API tests. When there is no detected failure, the environment

is successfully wrapped.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Wrapper PettingZoo*

Howto RL-WP-003: Run Multi-Agent on PettingZoo Environment

Ver. 1.4.0 (2023-02-21)

This module shows how to run an own policy inside the MLPro standard agent model with a wrapped Petting Zoo environment.

You will learn:

1) How to set up a scenario for a Petting Zoo environment in MLPro
2) How to run the scenario
3) How to save, reload and rerun a scenario

Prerequisites

Please install the following packages to run this examples properly:

- PettingZoo

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_wp_003_run_multiagent_with_own_policy_on_petting_zoo_environment.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-08-26  0.0.0     SY       Creation
## -- 2021-08-27  1.0.0     SY       Released first version
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library name
## -- 2021-09-23  1.1.0     SY       Updated wrapper WrEnvPZoo class, provides two different envs
## -- 2021-09-29  1.1.1     SY       Change name: WrEnvPZoo to WrEnvPZOO2MLPro
## -- 2021-10-06  1.1.2     DA       Refactoring
## -- 2021-10-18  1.1.3     DA       Refactoring
## -- 2021-11-15  1.1.4     DA       Refactoring
## -- 2021-11-15  1.1.4     DA       Refactoring
## -- 2021-11-16  1.1.5     DA       Added explicit scenario reset with constant seeding
## -- 2021-12-03  1.1.6     DA       Refactoring
## -- 2022-02-25  1.1.7     SY       Refactoring due to auto generated ID in class Dimension
## -- 2022-05-19  1.1.8     SY       Utilize RandomGenerator
```

(continues on next page)

(continued from previous page)

```
## -- 2022-05-30  1.1.9     DA       Cleaned up/rearranged a bit
## -- 2022-05-30  1.1.8     SY       Update pistonball_v5 to pistonball_v6
## -- 2022-10-08  1.2.0     SY       Turn off render: causing error due to pzoo ver 1.22.
↪0
## -- 2022-10-14  1.2.1     SY       Refactoring
## -- 2022-11-01  1.2.2     DA       Refactoring
## -- 2022-11-02  1.2.3     SY       Unable logging in unit test model and bug fixing
## -- 2022-11-07  1.3.0     DA       Refactoring
## -- 2023-02-21  1.4.0     DA       Added save + reload + rerun steps to demonstrate/
↪validate
## --                                 persistence of pettingzoo scenarios
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.4.0 (2023-02-21)

This module shows how to run an own policy inside the MLPro standard agent model with a␣
↪wrapped
Petting Zoo environment.

You will learn:

1) How to set up a scenario for a Petting Zoo environment in MLPro

2) How to run the scenario

3) How to save, reload and rerun a scenario

"""


from pathlib import Path
from pettingzoo.butterfly import pistonball_v6
from pettingzoo.classic import connect_four_v3
from mlpro.bf.math import *
from mlpro.rl import *
from mlpro.wrappers.pettingzoo import WrEnvPZOO2MLPro
from mlpro.rl.pool.policies.randomgenerator import RandomGenerator




# 1 RL Scenario based on PettingZoo Pistonball environment
class PBScenario (RLScenario):
    """
    Reference : https://www.pettingzoo.ml/butterfly/pistonball
    """

    C_NAME      = 'Pistonball V6'
```

(continues on next page)

(continued from previous page)

```
    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        if p_visualize:
            zoo_env         = pistonball_v6.env(render_mode="human")
        else:
            zoo_env         = pistonball_v6.env(render_mode="ansi")
        self._env           = WrEnvPZOO2MLPro(zoo_env, p_visualize=p_visualize, p_
↪logging=p_logging)

        multi_agent         = MultiAgent(p_name='Pistonball_agents', p_ada=1, p_
↪visualize=p_visualize, p_logging=p_logging)
        agent_idx           = 0
        for k in self._env._zoo_env.action_spaces:
            agent_name      = "Agent_"+str(agent_idx)
            as_ids          = self._env.get_action_space().get_dim_ids()
            agent_ospace    = self._env.get_state_space()
            agent_asspace   = self._env.get_action_space().spawn([as_ids[agent_idx]])
            agent           = Agent(p_policy=RandomGenerator(p_observation_space=agent_
↪ospace,
                                                             p_action_space=agent_
↪asspace,
                                                             p_buffer_size=10,
                                                             p_ada=p_ada,
                                                             p_visualize=p_visualize,
                                                             p_logging=p_logging
                                                             ),
                                    p_envmodel=None,
                                    p_id=agent_idx,
                                    p_name=agent_name,
                                    p_ada=p_ada,
                                    p_visualize=p_visualize,
                                    p_logging=p_logging
                                    )
            multi_agent.add_agent(p_agent=agent)
            agent_idx += 1

        return multi_agent



# 2 Alternative RL Scenario based on PettingZoo Connect Four environment
class C4Scenario (RLScenario):
    """
    https://www.pettingzoo.ml/classic/connect_four
    """

    C_NAME      = 'Connect Four V3'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        if p_visualize:
            zoo_env         = connect_four_v3.env(render_mode="human")
        else:
            zoo_env         = connect_four_v3.env(render_mode="ansi")
```

(continues on next page)

(continued from previous page)

```
        self._env           = WrEnvPZOO2MLPro(zoo_env, p_visualize=p_visualize, p_
↪logging=p_logging)

        multi_agent         = MultiAgent(p_name='Connect4_Agents', p_ada=1, p_
↪visualize=p_visualize, p_logging=p_logging)
        agent_idx           = 0
        for k in self._env._zoo_env.action_spaces:
            agent_name      = "Agent_"+str(agent_idx)
            as_ids          = self._env.get_action_space().get_dim_ids()
            agent_sspace    = self._env.get_state_space()
            agent_asspace   = self._env.get_action_space().spawn([as_ids[agent_idx]])
            agent           = Agent(p_policy=RandomGenerator(p_observation_space=agent_
↪sspace,
                                                             p_action_space=agent_
↪asspace,
                                                             p_buffer_size=10,
                                                             p_ada=p_ada,
                                                             p_visualize=p_visualize,
                                                             p_logging=p_logging
                                                             ),
                                    p_envmodel=None,
                                    p_id=agent_idx,
                                    p_name=agent_name,
                                    p_ada=p_ada,
                                    p_visualize=p_visualize,
                                    p_logging=p_logging
                                    )
            multi_agent.add_agent(p_agent=agent)
            agent_idx += 1

        return multi_agent



# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    logging     = Log.C_LOG_ALL
    visualize   = True
    now         = datetime.now()
    path        = str(Path.home()) + os.sep + '%04d-%02d-%02d  %02d.%02d.%02d Howto RL␣
↪WP 003' % (now.year, now.month, now.day, now.hour, now.minute, now.second)
else:
    # 3.2 Parameters for internal unit test
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None


# 3.3 Instantiate one of two prepared demo scenarios
myscenario  = PBScenario(
        p_mode=Mode.C_MODE_SIM,
```

(continues on next page)

(continued from previous page)

```
        p_ada=True,
        p_cycle_limit=100,
        p_visualize=visualize,
        p_logging=logging
)

# myscenario  = C4Scenario(
#         p_mode=Mode.C_MODE_SIM,
#         p_ada=True,
#         p_cycle_limit=100,
#         p_visualize=visualize,
#         p_logging=logging
# )


# 3.4 Reset and run the scenario
myscenario.reset(1)
myscenario.run()


if __name__ == '__main__':
    # 3.5 In demo mode we save, reload and rerun the entire scenario to demonstrate␣
↪persistence
    myscenario.save(path, 'dummy')
    input('\nPress ENTER to reload and run again...\n')
    myscenario = PBScenario.load(path + os.sep + 'scenario')
    myscenario.reset(1)
    myscenario.run()
```

We use the Petting Zoo environment Pistonball as default testing environment in this example. However, in step 3.3 you can also change the environment into Connect Four.

Results

By running the example code, the environment window appears and the runtime log is dumped to the terminal.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Petting Zoo Env Env "connect_four_v3": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Petting Zoo Env Env "connect_four_v3": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Petting Zoo Env Env "connect_four_v3": Operation mode␣
↪set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_0: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_0: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_0: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Agent 0 Agent_0 added.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
```

(continues on next page)

(continued from previous page)

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_1: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_1: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_1: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Agent 1 Agent_1 added.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Process time 0:00:00 :␣
↪Scenario reset with seed 1
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Init vizualization for all␣
↪agents...
....
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Start vizualization for all␣
↪agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Process time 0:00:12 End of␣
↪processing
```

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Wrapper PettingZoo*

Howto RL-WP-004: Train an Agent with SB3

Ver. 1.1.2 (2023-02-13)

This module shows how to train agent with SB3 Wrapper for On- and Off-Policy Algorithms

You will learn:

1) How to set up a scenario with SB3 policy
2) How to run the scenario

Prerequisites

Please install the following packages to run this examples properly:

- OpenAI Gym
- Stable-Baselines3

Executable code

```
## -------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_wp_004_train_agent_with_sb3_policy.py
## -------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
```

(continues on next page)

(continued from previous page)

```
## -- 2021-09-29  0.0.0     MRD       Creation
## -- 2021-10-07  1.0.0     MRD       Released first version
## -- 2021-10-08  1.0.1     DA        Take over the cycle limit from the environment
## -- 2021-10-18  1.0.2     DA        Refactoring
## -- 2021-10-18  1.0.3     MRD       SB3 Off Policy Wrapper DQN, DDPG, SAC
## -- 2021-11-15  1.0.4     DA        Refactoring
## -- 2021-12-03  1.0.5     DA        Refactoring
## -- 2021-12-07  1.0.6     DA        Refactoring
## -- 2022-02-25  1.0.7     SY        Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-07-20  1.0.8     SY        Update due to the latest introduction of Gym 0.25
## -- 2022-10-14  1.0.9     SY        Refactoring
## -- 2022-11-07  1.1.0     DA        Refactoring
## -- 2023-01-14  1.1.1     MRD       Removing default parameter new_step_api and render_
↪mode for gym
## -- 2023-02-13  1.1.2     DA        Optimization of dark mode
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.1.2 (2023-02-13)

This module shows how to train agent with SB3 Wrapper for On- and Off-Policy Algorithms

You will learn:

1) How to set up a scenario with SB3 policy

2) How to run the scenario

"""


import gym
from stable_baselines3 import A2C, PPO, DQN, DDPG, SAC
from mlpro.rl.models import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from collections import deque
from pathlib import Path


# 1 Implement your own RL scenario
class MyScenario(RLScenario):
    C_NAME = 'Howto-RL-WP-004'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        # 1 Setup environment
        # self._env   = RobotHTM(p_logging=False)
        gym_env = gym.make('CartPole-v1')
        self._env = WrEnvGYM2MLPro(gym_env, p_visualize=p_visualize, p_logging=p_logging)
```

(continues on next page)

(continued from previous page)

```
        # 2 Instantiate Policy From SB3
        # env is set to None, it will be set up later inside the wrapper
        # _init_setup_model is set to False, the _setup_model() will be called inside
        # the wrapper manually

        # A2C
        # policy_sb3 = A2C(
        #             policy="MlpPolicy",
        #             env=None,
        #             use_rms_prop=False,
        #             _init_setup_model=False,
        #             device="cpu")

        # PPO
        policy_sb3 = PPO(
            policy="MlpPolicy",
            n_steps=5,
            env=None,
            _init_setup_model=False,
            device="cpu")

        # DQN Discrete only
        # policy_sb3 = DQN(
        #             policy="MlpPolicy",
        #             env=None,
        #             _init_setup_model=False,
        #             device="cpu")

        # DDPG Continuous only
        # policy_sb3 = DDPG(
        #             policy="MlpPolicy",
        #             env=None,
        #             _init_setup_model=False,
        #             device="cpu")

        # SAC Continuous only
        # policy_sb3 = SAC(
        #             policy="MlpPolicy",
        #             env=None,
        #             _init_setup_model=False,
        #             device="cpu")

        # 3 Wrap the policy
        policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)
```

(continues on next page)

(continued from previous page)

```
        # 4 Setup standard single-agent with own policy
        return Agent(
            p_policy=policy_wrapped,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )


# 2 Create scenario and start training
if __name__ == "__main__":
    # 2.1 Parameters for demo mode
    logging = Log.C_LOG_ALL
    cycle_limit = 1000
    visualize = True
    path = str(Path.home())

else:
    # 2.2 Parameters for internal unit test
    logging = Log.C_LOG_NOTHING
    cycle_limit = 50
    visualize = False
    path = None

# 2.3 Create and run training object
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_max_adaptations=0,
    p_max_stagnations=0,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging)

training.run()
```

Results

An output similar to the one present in *Howto RL-AGENT-002* will show up, making use of the wrapped training algorithm.

Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Wrapper SB3*

Howto RL-WP-005: Validation SB3 Wrapper (On-Policy)

Prerequisites

Please install the following packages to run this examples properly:

- Pytorch
- OpenAI Gym
- Stable-Baselines3
- Panda

Executable code

```
## -------------------------------------------------------------------------------------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_wp_005_validation_wrapped_sb3_on_policy.py
## -------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-10-27  0.0.0     MRD      Creation
## -- 2021-10-27  1.0.0     MRD      Released first version
## -- 2021-11-16  1.0.1     DA       Refactoring
## -- 2021-12-07  1.0.2     DA       Refactoring
## -- 2021-12-20  1.0.3     DA       Refactoring
## -- 2021-12-23  1.0.4     MRD      Small change on custom _reset Wrapper
## -- 2021-12-24  1.0.5     DA       Replaced separtor in log line by Training.C_LOG_
SEPARATOR
## -- 2022-02-27  1.0.6     SY       Refactoring due to auto generated ID in class␣
Dimension
## -- 2022-03-21  1.0.7     WB       Rewrite module description
## -- 2022-07-20  1.0.8     SY       Update due to the latest introduction of Gym 0.25
## -- 2022-10-14  1.0.9     SY       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-01-14  1.1.1     MRD      Removing default parameter new_step_api and render_
mode for gym
## -- 2023-02-02  1.2.0     DA       Refactoring
## -- 2023-02-04  1.2.1     SY       Refactoring to avoid printing during unit test
## -- 2023-02-13  1.2.2     DA       Optimization of dark mode
## -------------------------------------------------------------------------------------

"""
Ver. 1.2.2 (2023-02-13)

This module shows comparison between native and wrapped SB3 policy (On-policy).
"""


import gym
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
import torch
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import BaseCallback
from mlpro.bf.plot import DataPlotting
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from pathlib import Path



# 1 Parameter
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    logging     = Log.C_LOG_ALL
    visualize   = True
    path        = str(Path.home())
    cycle_limit = 1000

else:
    # 1.2 Parameters for internal unit test
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None
    cycle_limit = 10

mva_window = 1
buffer_size = 100
policy_kwargs = dict(activation_fn=torch.nn.Tanh,
                     net_arch=[dict(pi=[10, 10], vf=[10, 10])])



# 2 Implement your own RL scenario
class MyScenario(RLScenario):
    C_NAME = 'Howto-RL-WP-005'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:

        class CustomWrapperFixedSeed(WrEnvGYM2MLPro):
            def _reset(self, p_seed=None):
                self.log(self.C_LOG_TYPE_I, 'Reset')

                # 1 Reset Gym environment and determine initial state
                observation = self._gym_env.reset()
                obs = DataObject(observation)

                # 2 Create state object from Gym observation
                state = State(self._state_space)
                state.set_values(obs.get_data())
                state.set_success(True)
                self._set_state(state)
```

(continues on next page)

(continued from previous page)

```
        # 1 Setup environment
        gym_env = gym.make('CartPole-v1')
        gym_env.seed(1)
        # self._env   = mlpro_env
        self._env = CustomWrapperFixedSeed(gym_env, p_logging=p_logging)

        # 2 Instatiate Policy From SB3
        # env is set to None, it will be set up later inside the wrapper
        # _init_setup_model is set to False, the _setup_model() will be called inside
        # the wrapper manually

        # PPO
        policy_sb3 = PPO(
            policy="MlpPolicy",
            env=None,
            n_steps=buffer_size,
            _init_setup_model=False,
            policy_kwargs=policy_kwargs,
            device="cpu",
            seed=1)

        # 3 Wrap the policy
        self.policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)

        # 4 Setup standard single-agent with own policy
        return Agent(
            p_policy=self.policy_wrapped,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )


# 3 Instantiate training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_collect_eval=True,
    p_path=path,
```

(continues on next page)

(continued from previous page)

```
    p_visualize=visualize,
    p_logging=logging)


# 4 Train SB3 Wrapper
training.run()


# 5 Create Plotting Class
class MyDataPlotting(DataPlotting):
    def get_plots(self):
        """
        A function to plot data
        """
        for name in self.data.names:
            maxval = 0
            minval = 0
            if self.printing[name][0]:
                fig = plt.figure(figsize=(7, 7))
                raw = []
                label = []
                ax = fig.subplots(1, 1)
                ax.set_title(name)
                ax.grid(True, which="both", axis="both")
                for fr_id in self.data.frame_id[name]:
                    raw.append(np.sum(self.data.get_values(name, fr_id)))
                    if self.printing[name][1] == -1:
                        maxval = max(raw)
                        minval = min(raw)
                    else:
                        maxval = self.printing[name][2]
                        minval = self.printing[name][1]

                    label.append("%s" % fr_id)
                ax.plot(raw)
                ax.set_ylim(minval - (abs(minval) * 0.1), maxval + (maxval * 0.1))
                ax.set_xlabel("Episode")
                ax.legend(label, bbox_to_anchor=(1, 0.5), loc="center left")
                self.plots[0].append(name)
                self.plots[1].append(ax)
                if self.showing:
                    plt.show()
                else:
                    plt.close(fig)


# 6 Plotting 1 MLpro
data_printing = {"Cycle": [False],
                 "Day": [False],
                 "Second": [False],
                 "Microsecond": [False],
                 "Smith": [True, -1]}
```

(continues on next page)

(continued from previous page)

```
mem = training.get_results().ds_rewards
mem_plot = MyDataPlotting(mem, p_showing=False, p_printing=data_printing)
mem_plot.get_plots()
wrapper_plot = mem_plot.plots


# 7 Create Callback for the SB3 Training
class CustomCallback(BaseCallback, Log):
    """
    A custom callback that derives from ``BaseCallback``.

    :param verbose: (int) Verbosity level 0: not output 1: info 2: debug
    """

    C_TYPE = 'Wrapper'
    C_NAME = 'SB3 Policy'

    def __init__(self, p_verbose=0, p_logging=False):
        Log.__init__(self, p_logging=p_logging)
        super(CustomCallback, self).__init__(p_verbose)
        reward_space = Set()
        reward_space.add_dim(Dimension("Native"))
        self.ds_rewards = RLDataStoring(reward_space)
        self.episode_num = 0
        self.total_cycle = 0
        self.cycles = 0
        self.plots = None
        Log.__init__(self, p_logging=p_logging)

        self.continue_training = True
        self.rewards_cnt = []

    def _on_training_start(self) -> None:
        self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR)
        self.log(self.C_LOG_TYPE_I, '-- Episode', self.episode_num, 'started...')
        self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR, '\n')
        self.ds_rewards.add_episode(self.episode_num)

    def _on_step(self) -> bool:
        # With Cycle Limit
        self.ds_rewards.memorize_row(self.total_cycle, timedelta(0, 0, 0), self.locals.
↪get("rewards"))
        self.total_cycle += 1
        self.cycles += 1
        if self.locals.get("infos")[0]:
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR)
            self.log(self.C_LOG_TYPE_I, '-- Episode', self.episode_num, 'finished after',
↪ self.total_cycle + 1,
                     'cycles')
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR, '\n\n')
            self.episode_num += 1
```

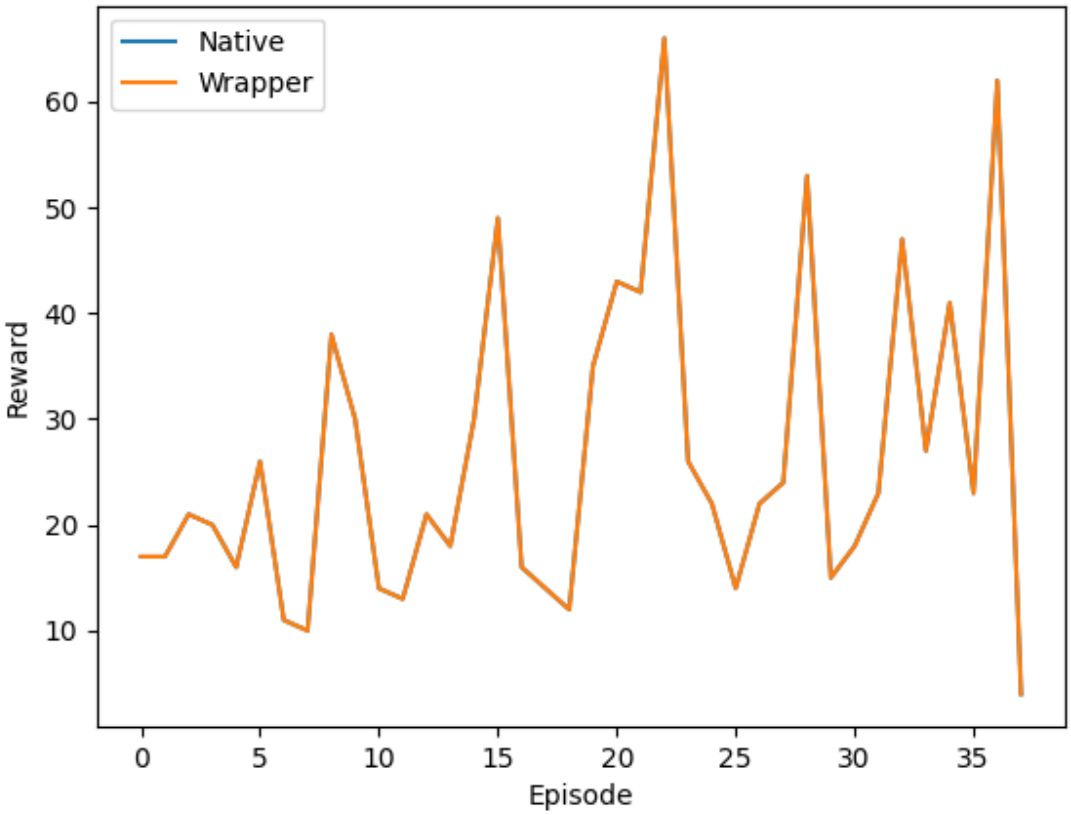
(continues on next page)

(continued from previous page)

```
            self.total_cycle = 0
            self.ds_rewards.add_episode(self.episode_num)
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR)
            self.log(self.C_LOG_TYPE_I, '-- Episode', self.episode_num, 'started...')
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR, '\n')

        return True

    def _on_training_end(self) -> None:
        data_printing = {"Cycle": [False],
                         "Day": [False],
                         "Second": [False],
                         "Microsecond": [False],
                         "Native": [True, -1]}
        mem_plot = MyDataPlotting(self.ds_rewards, p_showing=False, p_printing=data_
↪printing)
        mem_plot.get_plots()
        self.plots = mem_plot.plots


# 8 Run the SB3 Training Native
gym_env = gym.make('CartPole-v1')
gym_env.seed(1)
policy_sb3 = PPO(
    policy="MlpPolicy",
    env=gym_env,
    n_steps=buffer_size,
    verbose=0,
    policy_kwargs=policy_kwargs,
    device="cpu",
    seed=1)

cus_callback = CustomCallback(p_logging=logging)
policy_sb3.learn(total_timesteps=1000, callback=cus_callback)
native_plot = cus_callback.plots


# 9 Difference Plot
native_ydata = native_plot[1][0].lines[0].get_ydata()
wrapper_ydata = wrapper_plot[1][0].lines[0].get_ydata()
smoothed_native = pd.Series.rolling(pd.Series(native_ydata), mva_window).mean()
smoothed_native = [elem for elem in smoothed_native]
smoothed_wrapper = pd.Series.rolling(pd.Series(wrapper_ydata), mva_window).mean()
smoothed_wrapper = [elem for elem in smoothed_wrapper]
plt.plot(smoothed_native, label="Native")
plt.plot(smoothed_wrapper, label="Wrapper")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.legend()

if __name__ == "__main__":
    plt.show()
```

Results

The result plot shows that MLPro's wrapper for Stable Baselines 3 behaves neutrally.



Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Wrapper SB3*

Howto RL-WP-006: Validation SB3 Wrapper (Off-Policy)

Prerequisites

Please install the following packages to run this examples properly:

- Pytorch
- OpenAI Gym
- Stable-Baselines3
- Panda

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_wp_006_validation_wrapped_sb3_off_policy.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2022-01-11  0.0.0     MRD      Creation
## -- 2022-01-18  1.0.0     MRD      Released first version
## -- 2022-02-27  1.0.1     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-07-20  1.0.2     SY       Update due to the latest introduction of Gym 0.25
## -- 2022-10-14  1.0.3     SY       Refactoring
## -- 2022-11-07  1.1.0     DA       Refactoring
## -- 2023-01-14  1.1.1     MRD      Removing default parameter new_step_api and render_
↪mode for gym
## -- 2023-02-02  1.2.0     DA       Refactoring
## -- 2023-02-04  1.2.1     SY       Refactoring to avoid printing during unit test
## -- 2023-02-13  1.2.2     DA       Optimization of dark mode
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.2.2 (2023-02-13)

This module shows comparison between native and wrapped SB3 policy (Off-policy).
"""


import gym
import pandas as pd
import torch
from stable_baselines3 import DQN
from stable_baselines3.common.callbacks import BaseCallback
from mlpro.bf.plot import DataPlotting
from mlpro.rl import *
from mlpro.wrappers.openai_gym import WrEnvGYM2MLPro
from mlpro.wrappers.sb3 import WrPolicySB32MLPro
from pathlib import Path



# 1 Parameter
if __name__ == "__main__":
    # 1.1 Parameters for demo mode
    logging = Log.C_LOG_WE
    visualize = False
    path = str(Path.home())
    cycle_limit = 1200
```

(continues on next page)

(continued from previous page)

```
else:
    # 1.2 Parameters for internal unit test
    logging = Log.C_LOG_NOTHING
    visualize = False
    path = None
    cycle_limit = 10

mva_window = 1
buffer_size = 100
policy_kwargs = dict(activation_fn=torch.nn.ReLU,
                     net_arch=[10])


# 2 Implement your own RL scenario
class MyScenario(RLScenario):
    C_NAME = 'Howto-RL-WP-006'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:

        class CustomWrapperFixedSeed(WrEnvGYM2MLPro):
            def _reset(self, p_seed=None):
                self.log(self.C_LOG_TYPE_I, 'Reset')

                # 1 Reset Gym environment and determine initial state
                observation = self._gym_env.reset()
                obs = DataObject(observation)

                # 2 Create state object from Gym observation
                state = State(self._state_space)
                state.set_values(obs.get_data())
                self._set_state(state)

        # 1 Setup environment
        gym_env = gym.make('CartPole-v1')
        gym_env.seed(2)
        self._env = CustomWrapperFixedSeed(gym_env, p_logging=p_logging)

        # 2 Instatiate Policy From SB3
        # DQN
        policy_sb3 = DQN(
            policy="MlpPolicy",
            learning_starts=12,
            buffer_size=24,
            env=None,
            _init_setup_model=False,
            policy_kwargs=policy_kwargs,
            device="cpu",
            seed=2)

        # 3 Wrap the policy
        self.policy_wrapped = WrPolicySB32MLPro(
            p_sb3_policy=policy_sb3,
```

(continues on next page)

(continued from previous page)

```
            p_cycle_limit=self._cycle_limit,
            p_observation_space=self._env.get_state_space(),
            p_action_space=self._env.get_action_space(),
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging)

        # 4 Setup standard single-agent with own policy
        return Agent(
            p_policy=self.policy_wrapped,
            p_envmodel=None,
            p_name='Smith',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )


# 3 Instantiate training
training = RLTraining(
    p_scenario_cls=MyScenario,
    p_cycle_limit=cycle_limit,
    p_collect_states=True,
    p_collect_actions=True,
    p_collect_rewards=True,
    p_collect_eval=True,
    p_path=path,
    p_visualize=visualize,
    p_logging=logging)


# 4 Train SB3 Wrapper
training.run()


# 5 Create Plotting Class
class MyDataPlotting(DataPlotting):
    def get_plots(self):
        """
        A function to plot data
        """
        for name in self.data.names:
            maxval = 0
            minval = 0
            if self.printing[name][0]:
                fig = plt.figure(figsize=(7, 7))
                raw = []
                label = []
                ax = fig.subplots(1, 1)
                ax.set_title(name)
                ax.grid(True, which="both", axis="both")
                for fr_id in self.data.frame_id[name]:
```

(continues on next page)

(continued from previous page)

```
                    raw.append(np.sum(self.data.get_values(name, fr_id)))
                    if self.printing[name][1] == -1:
                        maxval = max(raw)
                        minval = min(raw)
                    else:
                        maxval = self.printing[name][2]
                        minval = self.printing[name][1]

                    label.append("%s" % fr_id)
                ax.plot(raw)
                ax.set_ylim(minval - (abs(minval) * 0.1), maxval + (maxval * 0.1))
                ax.set_xlabel("Episode")
                ax.legend(label, bbox_to_anchor=(1, 0.5), loc="center left")
                self.plots[0].append(name)
                self.plots[1].append(ax)
                if self.showing:
                    plt.show()
                else:
                    plt.close(fig)


# 6 Plotting 1 MLpro
data_printing = {"Cycle": [False],
                 "Day": [False],
                 "Second": [False],
                 "Microsecond": [False],
                 "Smith": [True, -1]}

mem = training.get_results().ds_rewards
mem_plot = MyDataPlotting(mem, p_showing=False, p_printing=data_printing)
mem_plot.get_plots()
wrapper_plot = mem_plot.plots


# 7 Create Callback for the SB3 Training
class CustomCallback(BaseCallback, Log):
    """
    A custom callback that derives from ``BaseCallback``.

    :param verbose: (int) Verbosity level 0: not output 1: info 2: debug
    """

    C_TYPE = 'Wrapper'
    C_NAME = 'SB3 Policy'

    def __init__(self, p_verbose=0, p_logging=False):
        Log.__init__(self, p_logging=p_logging)
        super(CustomCallback, self).__init__(p_verbose)
        reward_space = Set()
        reward_space.add_dim(Dimension("Native"))
        self.ds_rewards = RLDataStoring(reward_space)
        self.episode_num = 0
```

(continues on next page)

(continued from previous page)

```
        self.total_cycle = 0
        self.cycles = 0
        self.plots = None
        self.new_episodes = False
        Log.__init__(self, p_logging=p_logging)

        self.continue_training = True
        self.rewards_cnt = []

    def _on_training_start(self) -> None:
        self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR)
        self.log(self.C_LOG_TYPE_I, '-- Episode', self.episode_num, 'started...')
        self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR, '\n')
        self.ds_rewards.add_episode(self.episode_num)

    def _on_step(self) -> bool:
        if self.new_episodes:
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR)
            self.log(self.C_LOG_TYPE_I, '-- Episode', self.episode_num, 'finished after',
↪ self.total_cycle, 'cycles')
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR, '\n\n')
            self.episode_num += 1
            self.total_cycle = 0
            self.ds_rewards.add_episode(self.episode_num)
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR)
            self.log(self.C_LOG_TYPE_I, '-- Episode', self.episode_num, 'started...')
            self.log(self.C_LOG_TYPE_I, Training.C_LOG_SEPARATOR, '\n')
            self.new_episodes = False
        # With Cycle Limit
        self.ds_rewards.memorize_row(self.total_cycle, timedelta(0, 0, 0), self.locals.
↪get("rewards"))
        self.total_cycle += 1
        self.cycles += 1
        if self.locals.get("infos")[0]:
            self.new_episodes = True

        return True

    def _on_training_end(self) -> None:
        self.log(self.C_LOG_TYPE_I, 'Training cycle limit', self.cycles, 'reached')
        data_printing = {"Cycle": [False],
                         "Day": [False],
                         "Second": [False],
                         "Microsecond": [False],
                         "Native": [True, -1]}
        mem_plot = MyDataPlotting(self.ds_rewards, p_showing=False, p_printing=data_
↪printing)
        mem_plot.get_plots()
        self.plots = mem_plot.plots


# 8 Run the SB3 Training Native
```

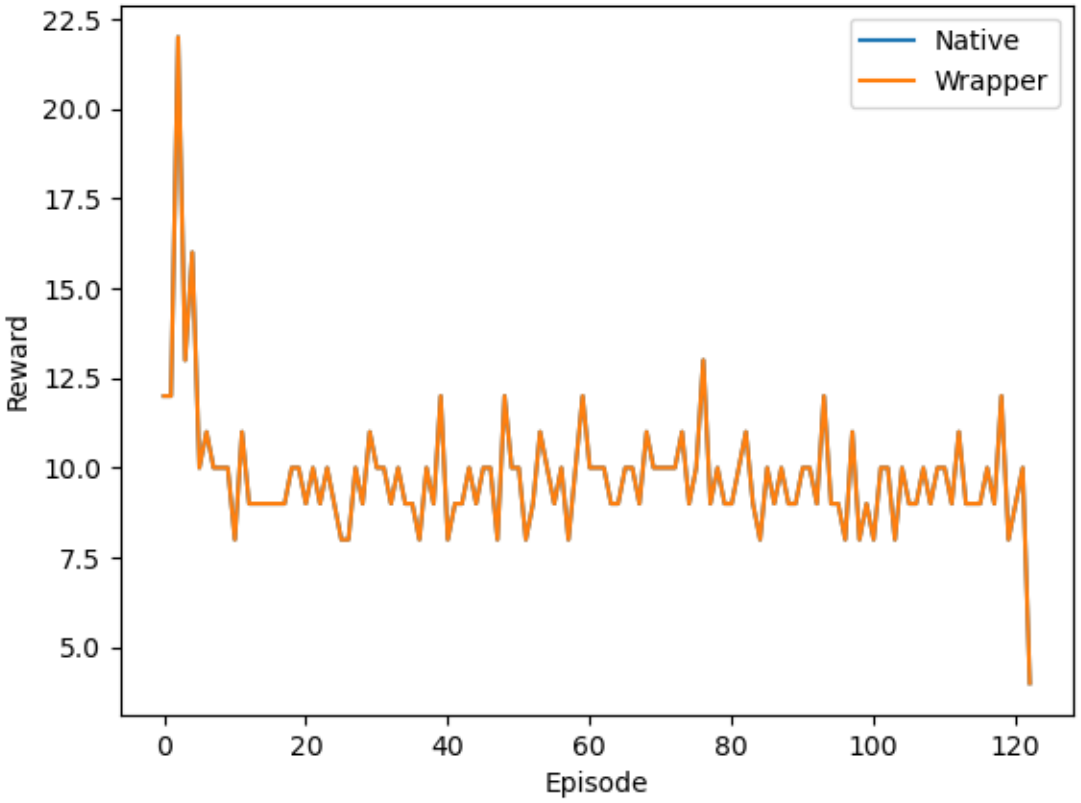
(continues on next page)

(continued from previous page)

```
gym_env = gym.make('CartPole-v1')
gym_env.seed(2)
policy_sb3 = DQN(
    policy="MlpPolicy",
    learning_starts=12,
    buffer_size=24,
    env=gym_env,
    policy_kwargs=policy_kwargs,
    device="cpu",
    seed=2)

cus_callback = CustomCallback(p_logging=logging)
policy_sb3.learn(total_timesteps=1200, callback=cus_callback)
native_plot = cus_callback.plots


# 9 Difference Plot
native_ydata = native_plot[1][0].lines[0].get_ydata()
wrapper_ydata = wrapper_plot[1][0].lines[0].get_ydata()
smoothed_native = pd.Series.rolling(pd.Series(native_ydata), mva_window).mean()
smoothed_native = [elem for elem in smoothed_native]
smoothed_wrapper = pd.Series.rolling(pd.Series(wrapper_ydata), mva_window).mean()
smoothed_wrapper = [elem for elem in smoothed_wrapper]
plt.plot(smoothed_native, label="Native")
plt.plot(smoothed_wrapper, label="Wrapper")
plt.xlabel("Episode")
plt.ylabel("Reward")
plt.legend()

if __name__ == "__main__":
    plt.show()
```

Results

The result plot shows that MLPro's wrapper for Stable Baselines 3 behaves neutrally.



Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*
- *API Reference - Wrapper SB3*

8.3.9 User Interaction

Howto RL-UI-001: SciUI - Reinforcement Learning Cockpit

Ver. 0.1.3 (2023-03-02)

SciUI template for a Reinforcement Learning simulation.

You will learn:

1. How to use MLPro's SciUI RL extension for reinforcement learning.

Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- Matplotlib
- Tkinter

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.rl.examples
## -- Module  : howto_rl_ui_001_reinforcement_learning_cockpit.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-07-27  0.0.0     SY       Creation
## -- 2021-10-07  0.1.0     SY       Release of first draft
## -- 2022-10-12  0.1.1     DA       Renaming and shift to sub-framework mlpro-rl
## -- 2022-10-14  0.1.2     SY       Refactoring
## -- 2023-03-02  0.1.3     LSB      Refactoring
## -------------------------------------------------------------------------------------------
↪-----------

"""
Ver. 0.1.3 (2023-03-02)

SciUI template for a Reinforcement Learning simulation.

You will learn:

1. How to use MLPro's SciUI RL extension for reinforcement learning.
"""



from mlpro.bf.ui.sciui.framework import *
from mlpro.bf.math import *
from mlpro.rl.sciui_rl import RLInteractiveUI




## -------------------------------------------------------------------------------------------
↪-----------
## -------------------------------------------------------------------------------------------
↪-----------
class SciUI_RL(SciUIScenario):

    C_NAME          = 'Reinforcement Learning'
    C_VERSION       = '1.0.0'
    C_RELEASED      = True
    C_VISIBLE       = True

## -------------------------------------------------------------------------------------------
↪-----------
    def init_component(self):
        super().init_component()
```

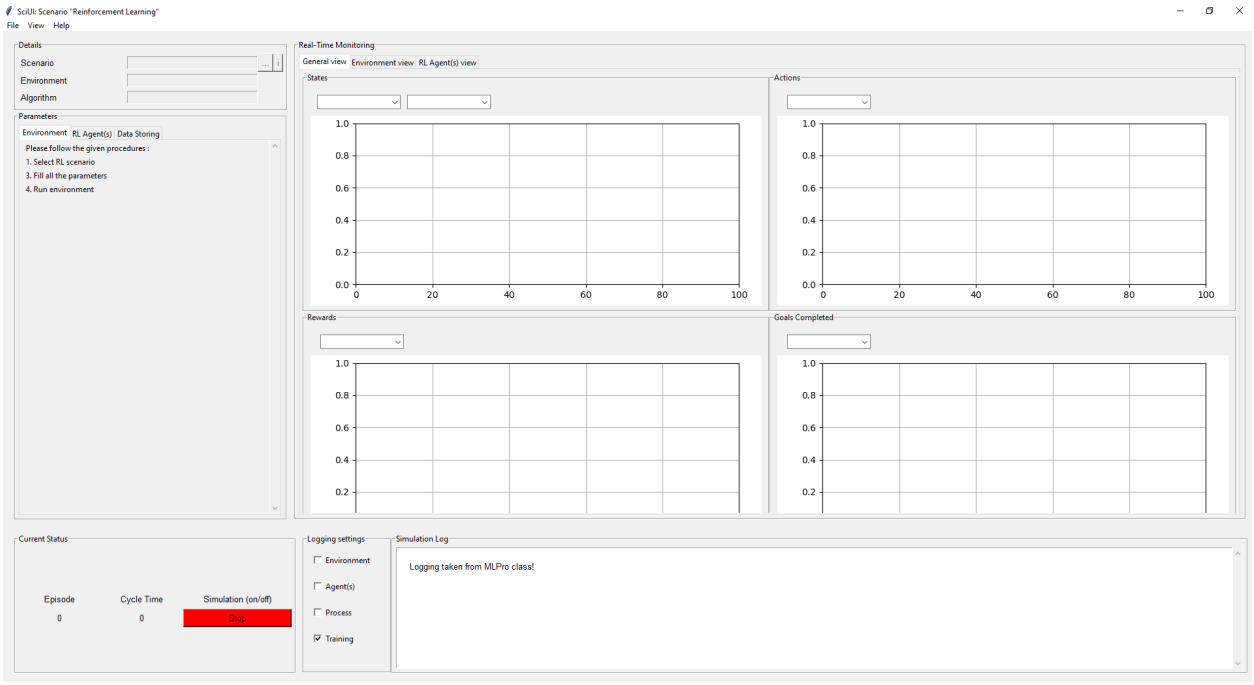
(continues on next page)

(continued from previous page)

```
        RLInteractiveUI.enrich_shared_db(self.shared_db)

        self.add_component(RLInteractiveUI(p_shared_db=self.shared_db, p_row=0, p_col=0,
                                           p_padx=5, p_logging=self._level, p_refresh_
↪rate=100))



if (__name__ == '__main__'):
    from mlpro.bf.ui.sciui.main import SciUI
    SciUI()
```

Results

This UI is not ready-to-use and still under development. The current cockpit is shown as follow,



Cross Reference

- *API Reference - RL Agent*
- *API Reference - RL Environments*

8.4 MLPro-GT - Game Theory

The following examples demonstrate various functionalities of MLPro-GT:

8.4.1 Dynamic Programming

Howto GT-001: Run Multi-Player with Own Policy

Ver. 1.2.0 (2022-11-07)

This module shows how to run an own multi-player with the enhanced multi-action game board MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up your own players' policies
2) How to set up your own game in dynamic programming, including players and game board interaction
3) How to run the game

Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_dp_001_run_multi_player_with_own_policy_on_multicartpole_game_
↪board.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-06-06  0.0.0     DA       Creation
## -- 2021-06-06  1.0.0     DA       Release of first version
## -- 2021-08-28  1.0.1     DA       Adjustments after changings on rl models
## -- 2021-09-11  1.0.1     MRD      Change Header information to match our new library␣
↪name
## -- 2021-10-06  1.0.2     DA       Adjustments after changings on rl models
## -- 2021-11-15  1.1.0     DA       Refactoring
## -- 2022-02-25  1.1.1     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-10-13  1.1.2     SY       Refactoring
## -- 2022-11-01  1.1.3     DA       Refactoring
## -- 2022-11-02  1.1.4     DA       Refactoring
## -- 2022-11-07  1.2.0     DA       Refactoring
## -------------------------------------------------------------------------------------------
↪-----------
```

(continues on next page)

(continued from previous page)

```
"""
Ver. 1.2.0 (2022-11-07)

This module shows how to run an own multi-player with the enhanced multi-action game␣
↪board
MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up your own players' policies

2) How to set up your own game in dynamic programming, including players and game board␣
↪interaction

3) How to run the game

"""



from mlpro.rl import *
from mlpro.gt import *
from mlpro.gt.pool.boards.multicartpole import MultiCartPolePGT
import random
import numpy as np




# 1 Implement your own player policy
class MyPolicy (Policy):

    C_NAME      = 'MyPolicy'

    def set_random_seed(self, p_seed=None):
        random.seed(p_seed)


    def compute_action(self, p_state: State) -> Action:
        # 1.1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 1.2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = random.random()

        # 1.3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)


    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        # 1.4 Adapting the internal policy is up to you...
```

(continues on next page)

(continued from previous page)

```
        self.log(self.C_LOG_TYPE_W, 'Sorry, I am a stupid agent...')

        # 1.5 Only return True if something has been adapted...
        return False



# 2 Implement your own game
class MyGame (Game):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:

        # 2.1 Setup Multi-Player Environment (consisting of 3 OpenAI Gym Cartpole envs)
        self._env   = MultiCartPolePGT(p_num_envs=3, p_visualize=p_visualize, p_
↪logging=p_logging)


        # 2.2 Setup Multi-Player

        # 2.2.1 Create empty Multi-Player
        multi_player = MultiPlayer(
            p_name='Human Beings',
            p_ada=True,
            p_visualize=p_visualize,
            p_logging=p_logging
        )

        # 2.2.2 Add Single-Player #1 with own policy (controlling sub-environment #1)
        ss_ids = self._env.get_state_space().get_dim_ids()
        as_ids = self._env.get_action_space().get_dim_ids()
        multi_player.add_player(
            p_player=Player(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[0],ss_
↪ids[1],ss_ids[2],ss_ids[3]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[0]]),
                    p_ada=True,
                    p_visualize=p_visualize,
                    p_logging=p_logging
                ),
                p_name='Neo',
                p_id=0,
                p_ada=True,
                p_visualize=p_visualize,
                p_logging=p_logging
            ),
            p_weight=0.3
        )
```

(continues on next page)

(continued from previous page)

```
        # 2.2.3 Add Single-Player #2 with own policy (controlling sub-environments #2,#3)
        multi_player.add_player(
            p_player=Player(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[4],ss_
↪ids[5],ss_ids[6],ss_ids[7],ss_ids[8],ss_ids[9],ss_ids[10],ss_ids[11]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[1],as_
↪ids[2]]),
                    p_ada=True,
                    p_visualize=p_visualize,
                    p_logging=p_logging
                ),
                p_name='Trinity',
                p_id=1,
                p_ada=True,
                p_visualize=p_visualize,
                p_logging=p_logging
            ),
            p_weight=0.7
        )

        # 2.3 Adaptive ML model (here: our multi-player) is returned
        return multi_player



# 3 Create game and run some cycles

if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    logging     = Log.C_LOG_ALL
    visualize   = True

else:
    # 3.2 Parameters for internal unit test
    logging     = Log.C_LOG_NOTHING
    visualize   = False

# 3.3 Run the game
mygame  = MyGame(
    p_mode=Mode.C_MODE_SIM,
    p_ada=True,
    p_cycle_limit=200,
    p_visualize=visualize,
    p_logging=logging )

mygame.run()
```

Results

Three Gym Cartpole game board windows should appear and the following output should be expected in the console.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Operation mode␣
↪set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1": Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1" (0): Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1" (1): Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  OpenAI Gym Env Env "CartPole-v1" (2): Reset
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player Neo: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player Neo: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player Neo: Adaptivity switched on
```

(continues on next page)

(continued from previous page)

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Player 0 Neo added.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player Trinity: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player Trinity: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player Trinity: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Player 1 Trinity added.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:00 Start of processing
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:00 : Start of cycle 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:00 : Agent computes␣
↪action...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Start of action computation␣
↪for all agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player 0 Neo: Action computation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player 0 Neo: Action computation finished
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player 1 Trinity: Action computation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player 1 Trinity: Action computation finished
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: End of action computation for␣
↪all agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:00 : Env processes action.
↪..
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Start␣
↪processing action
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Actions of␣
↪agent 0 = [0.02821633]
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Actions of␣
↪agent 1 = [0.5796828  0.73351315]
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Potential Game Board MultiCartPole(PGT): Action␣
↪processing finished successfully
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:01 : Agent adapts policy..
↪.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Adaptation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Start of adaptation for all␣
↪agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Start adaption for agent 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player 0 Neo: Adaptation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Policy MyPolicy: Sorry, I am a stupid agent...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Start adaption for agent 1
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Player 1 Trinity: Adaptation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy MyPolicy: Adaptation started
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Policy MyPolicy: Sorry, I am a stupid agent...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: End of adaptation for all␣
↪agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:01 : End of cycle 0
...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Player Human Beings: Start vizualization for all␣
```

(continues on next page)

(continued from previous page)

```
↪agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Game Matrix: Process time 0:00:12 End of processing
```

Cross Reference

- *API Reference: Game Theory*

Howto GT-002: Train Multi-Player

Ver. 1.3.0 (2022-11-07)

This module shows how to train an own multi-player with the enhanced multi-action game board MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up your own players' policies
2) How to set up your own game in dynamic programming, including players and game board interaction
3) How to run the GT training and train your own players

Prerequisites

Please install the following packages to run this examples properly:

- NumPy
- OpenAI Gym

Executable code

```
## -------------------------------------------------------------------------------------------
↪-----------
## -- Project : MLPro - A Synoptic Framework for Standardized Machine Learning Tasks
## -- Package : mlpro.gt.examples
## -- Module  : howto_gt_dp_002_train_own_multi_player_on_multicartpole_game_board.py
## -------------------------------------------------------------------------------------------
↪-----------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-06-06  0.0.0     DA       Creation
## -- 2021-06-06  1.0.0     DA       Release of first version
## -- 2021-07-01  1.1.0     DA       Extended by data logging/storing (user home␣
↪directory)
## -- 2021-07-06  1.1.1     SY       Bugfix due to method Training.save_data() update
## -- 2021-08-28  1.1.2     DA       Adjustments after changings on rl models
## -- 2021-09-11  1.1.2     MRD      Change Header information to match our new library␣
↪name
## -- 2021-09-28  1.1.3     SY       Adjustment due to implementation of SAR Buffer on␣
↪player
## -- 2021-10-06  1.1.4     DA       Refactoring
## -- 2021-11-16  1.2.0     DA       Refactoring
## -- 2021-12-07  1.2.1     DA       Refactoring
## -- 2022-02-25  1.2.2     SY       Refactoring due to auto generated ID in class␣
↪Dimension
## -- 2022-10-13  1.2.3     SY       Refactoring
```

(continues on next page)

(continued from previous page)

```
## -- 2022-11-01  1.2.4     DA       Refactoring
## -- 2022-11-02  1.2.5     DA       Refactoring
## -- 2022-11-07  1.3.0     DA       Refactoring
## -------------------------------------------------------------------------------------
↪-----------

"""
Ver. 1.3.0 (2022-11-07)

This module shows how to train an own multi-player with the enhanced multi-action
game board MultiCartPole based on the OpenAI Gym CartPole environment.

You will learn:

1) How to set up your own players' policies

2) How to set up your own game in dynamic programming, including players and game board␣
↪interaction

3) How to run the GT training and train your own players

"""


from mlpro.rl import *
from mlpro.gt import *
from mlpro.gt.pool.boards.multicartpole import MultiCartPolePGT
import random
import numpy as np
from pathlib import Path
import os
from datetime import datetime




# 1 Implement your own player policy
class MyPolicy(Policy):

    C_NAME      = 'MyPolicy'

    def compute_action(self, p_state: State) -> Action:
        # 1 Create a numpy array for your action values
        my_action_values = np.zeros(self._action_space.get_num_dim())

        # 2 Computing action values is up to you...
        for d in range(self._action_space.get_num_dim()):
            my_action_values[d] = random.random()

        # 3 Return an action object with your values
        return Action(self._id, self._action_space, my_action_values)
```

(continues on next page)

(continued from previous page)

```
    def _adapt(self, p_sars_elem:SARSElement) -> bool:
        # 1 Adapting the internal policy is up to you...
        self.log(self.C_LOG_TYPE_I, 'Sorry, I am a stupid agent...')

        # 2 Only return True if something has been adapted...
        return False



# 2 Implement your own game
class MyGame(Game):

    C_NAME      = 'Matrix'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:

        # 1 Setup Multi-Player Environment (consisting of 3 OpenAI Gym Cartpole envs)
        self._env   = MultiCartPolePGT(p_num_envs=3, p_visualize=p_visualize, p_
↪logging=p_logging)


        # 2 Setup Multi-Player

        # 2.1 Create empty Multi-Player
        multi_player = MultiPlayer(
            p_name='Human Beings',
            p_ada=p_ada,
            p_visualize=p_visualize,
            p_logging=p_logging
        )

        # 2.2 Add Single-Player #1 with own policy (controlling sub-environment #1)
        ss_ids = self._env.get_state_space().get_dim_ids()
        as_ids = self._env.get_action_space().get_dim_ids()
        multi_player.add_player(
            p_player=Player(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[0],ss_
↪ids[1],ss_ids[2],ss_ids[3]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[0]]),
                    p_buffer_size=1,
                    p_ada=p_ada,
                    p_visualize=p_visualize,
                    p_logging=p_logging
                ),
                p_name='Neo',
                p_id=0,
                p_ada=p_ada,
                p_visualize=p_visualize,
```

(continues on next page)

(continued from previous page)

```
                p_logging=p_logging
            ),
            p_weight=0.3
        )


        # 2.2 Add Single-Player #2 with own policy (controlling sub-environments #2,#3)
        multi_player.add_player(
            p_player=Player(
                p_policy=MyPolicy(
                    p_observation_space=self._env.get_state_space().spawn([ss_ids[4],ss_
↪ids[5],ss_ids[6],ss_ids[7],ss_ids[8],ss_ids[9],ss_ids[10],ss_ids[11]]),
                    p_action_space=self._env.get_action_space().spawn([as_ids[1],as_
↪ids[2]]),
                    p_buffer_size=1,
                    p_ada=p_ada,
                    p_visualize=p_visualize,
                    p_logging=p_logging
                ),
                p_name='Trinity',
                p_id=1,
                p_ada=p_ada,
                p_visualize=p_visualize,
                p_logging=p_logging
            ),
            p_weight=0.7
        )


        # 2.3 Return multi-player as adaptive model
        return multi_player




# 3 Create game and run some cycles

if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    cycle_limit = 200
    logging     = Log.C_LOG_WE
    visualize   = True
    path        = str(Path.home())

else:
    # 3.2 Parameters for internal unit test
    cycle_limit = 10
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None
```

(continues on next page)

(continued from previous page)

```
# 3.3 Create and run training object
training = GTTraining(
        p_game_cls=MyGame,
        p_cycle_limit=cycle_limit,
        p_path=path,
        p_visualize=visualize,
        p_logging=logging )

training.run()
```

Results After the multiple game boards are initialised, the console will be filled with training logs and the final training result should show up at the end of the script.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training Results of run 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Scenario          : Game Matrix
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Model             : Multi-Player Human␣
↪Beings
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Start time stamp  : YYYY-MM-DD  HH:MM:SS.
↪SSSSSS
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- End time stamp    : YYYY-MM-DD  HH:MM:SS.
↪SSSSSS
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Duration          : 0:00:12.329561
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Start cycle id    : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- End cycle id      : 199
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training cycles   : 200
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Evaluation cycles : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Adaptations       : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- High score        : None
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Results stored in : "C:\Users\%username%\
↪YYYY-MM-DD  HH:MM:SS Training GT"
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Training Episodes : 14
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: -- Evaluations       : 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
YYYY-MM-DD  HH:MM:SS.SSSSSS  W  Results  RL: ----------------------------------------
↪---------------------------------
```

Cross Reference

- *API Reference: Game Theory*

8.5 MLPro-OA - Online Adaptivity

Coming soon…

CHAPTER

NINE

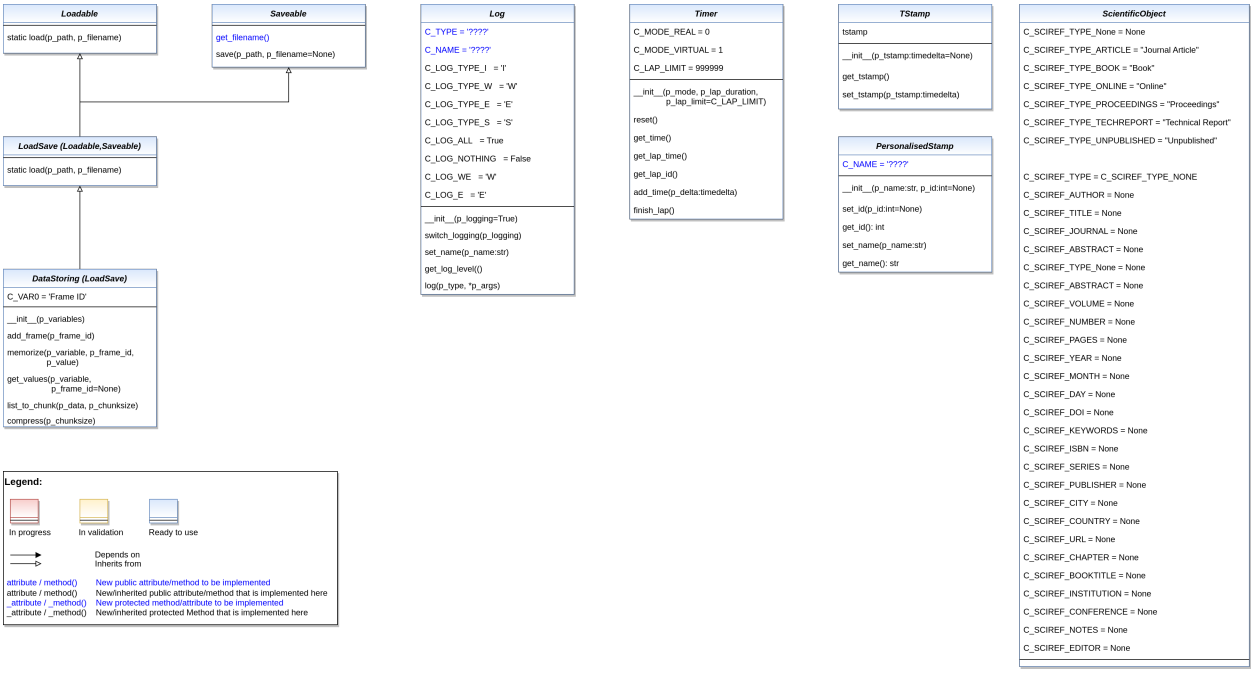
A2 - API REFERENCE

9.1 Core Functions

9.1.1 MLPro-BF - Basic Functions

Layer 0 - Elementary Functions

BF-VARIOUS - Various Functions



Ver. 2.0.0 (2023-02-22)

This module provides various classes with elementry functionalities for reuse in higher level classes. For example: logging, load/save, timer...

class mlpro.bf.various.**Loadable**

Bases: object

This abstract class adds the ability to be loadable to inherited classes.

static load(*p_path*, *p_filename*)

Loads content from the given path and file name. If file does not exist, it returns None.

Parameters

- **file** (*p_path Path that contains the*) –
- **name** (*p_filename File*) –

Returns
A loaded object, if file content was loaded successfully. None otherwise.

class mlpro.bf.various.**Saveable**

Bases: object

This abstract class adds the ability to be saveable to inherited classes. The filename can be generated internally by implementing the method get_filename() or provided from outside otherwise.

C_SUFFIX = '.pkl'

generate_filename()

To be redefined in case of use of internal generated file names.

Return type
Returns an internal unique filename.

save(*p_path*, *p_filename=None*) → bool

Saves content to the given path and file name. If file name is None, a unique file name will be generated by calling method generate_filename(). If it returns False then the saving method is failed. For saving the object, the custom method _save is called. This contains a default implementation based on pickle/dill and can be redefined on demand.

Parameters

- **saved** (*p_path Path where file will be*) –
- **generated)** (*p_filename File name (if None an internal filename will be*) –

Return type
True, if file content was saved successfully. False otherwise.

class mlpro.bf.various.**LoadSave**

Bases: *Loadable*, *Saveable*

This abstract class adds the ability to be loadable and saveable to inherited classes. The filename can be generated internally by implementing the method generate_filename() or provided from outside otherwise. See classes Loadable and Saveable for further information.

class mlpro.bf.various.**Log**(*p_logging=True*)

Bases: object

This class adds elementry log functionality to inherited classes.

Parameters
p_logging – Log level (see constants C_LOG_*). Default: Log.C_LOG_ALL

C_TYPE = '????'

C_NAME = '????'

C_LOG_TYPE_I = 'I'

C_LOG_TYPE_W = 'W'

C_LOG_TYPE_E = 'E'

C_LOG_TYPE_S = 'S'

C_LOG_TYPES = ['I', 'W', 'E', 'S']

C_COL_WARNING = '\x1b[93m'

C_COL_ERROR = '\x1b[91m'

C_COL_SUCCESS = '\x1b[32m'

C_COL_RESET = '\x1b[0m'

C_LOG_ALL = True

C_LOG_NOTHING = False

C_LOG_WE = 'W'

C_LOG_E = 'E'

C_LOG_LEVELS = [True, False, 'W', 'E']

C_INST_MSG = True

get_name() → str

set_name(*p_name: str*)

switch_logging(*p_logging*)

Sets new log level.

Parameters

p_logging – Log level (constant C_LOG_LEVELS contains valid values)

get_log_level()

log(*p_type*, **p_args*)

Writes log line to standard output in format: yyyy-mm-dd hh:mm:ss.mmmmmm [p_type C_TYPE C_NAME]: [p_args]

Parameters

- **entry** (*p_type type of log*) –
- **informations** (*p_args log*) –

Returns

Nothing

class mlpro.bf.various.Timer(*p_mode: int*, *p_lap_duration: timedelta | None = None*, *p_lap_limit: int = 999999*)

Bases: object

Timer class in two time modes (real/virtual) and with simple lap management.

Parameters

- **p_mode** (*int*) – C_MODE_REAL for real time mode or C_MODE_VIRTUAL for virtual time mode
- **p_lap_duration** (*timedelta = None*) – Optional duration of a single lap.

- **p_lap_limit** (*int = C_LAP_LIMIT*) – Maximum number of laps until the lap counter restarts with 0

C_MODE_REAL = 0

C_MODE_VIRTUAL = 1

C_LAP_LIMIT = 999999

reset() → None

Resets timer.

Returns
Nothing

get_time() → timedelta

get_lap_time() → timedelta

get_lap_id()

add_time(*p_delta: timedelta*)

finish_lap() → bool

Finishes the current lap. In timer mode C_MODE_REAL the remaining time until the end of the lap will be paused.

Returns
True, if the remaining time to the next lap was positive. False, if the timer timed out.

class mlpro.bf.various.**TStamp**(*p_tstamp: timedelta | None = None*)

Bases: object

This class provides elementry time stamp functionality for inherited classes.

get_tstamp() → timedelta

set_tstamp(*p_tstamp: timedelta*)

class mlpro.bf.various.**ScientificObject**

Bases: object

This class provides elementary functionality for storing a scientific reference.

C_SCIREF_TYPE_NONE = None

C_SCIREF_TYPE_ARTICLE = 'Journal Article'

C_SCIREF_TYPE_BOOK = 'Book'

C_SCIREF_TYPE_ONLINE = 'Online'

C_SCIREF_TYPE_PROCEEDINGS = 'Proceedings'

C_SCIREF_TYPE_TECHREPORT = 'Technical Report'

C_SCIREF_TYPE_UNPUBLISHED = 'Unpublished'

C_SCIREF_TYPE = None

C_SCIREF_AUTHOR = None

C_SCIREF_TITLE = None

C_SCIREF_JOURNAL = None

C_SCIREF_ABSTRACT = None

C_SCIREF_VOLUME = None

C_SCIREF_NUMBER = None

C_SCIREF_PAGES = None

C_SCIREF_YEAR = None

C_SCIREF_MONTH = None

C_SCIREF_DAY = None

C_SCIREF_DOI = None

C_SCIREF_KEYWORDS = None

C_SCIREF_ISBN = None

C_SCIREF_SERIES = None

C_SCIREF_PUBLISHER = None

C_SCIREF_CITY = None

C_SCIREF_COUNTRY = None

C_SCIREF_URL = None

C_SCIREF_CHAPTER = None

C_SCIREF_BOOKTITLE = None

C_SCIREF_INSTITUTION = None

C_SCIREF_CONFERENCE = None

C_SCIREF_NOTES = None

C_SCIREF_EDITOR = None

class mlpro.bf.various.**PersonalisedStamp**(*p_name: str*, *p_id: int | None = None*)

Bases: object

This class serves as a base class of label to set up a name and id for another class.

Parameters

- **p_name** (*str*) – name of the created class.
- **p_id** (*int*) – unique id of the created class. Default: None.

C_NAME

name of the created class. Default: ‘’.

Type
str

C_NAME = ''

set_id(*p_id: int | None = None*)

This method provides a functionality to set an unique ID.

Parameters
p_id (*int, optional*) – An unique ID. Default: None.

get_id() → str

This method provides a functionality to get the defined unique ID.

Returns
The unique ID.

Return type
str

set_name(*p_name: str*)

This method provides a functionality to set an unique name.

Parameters
p_name (*str*) – An unique name.

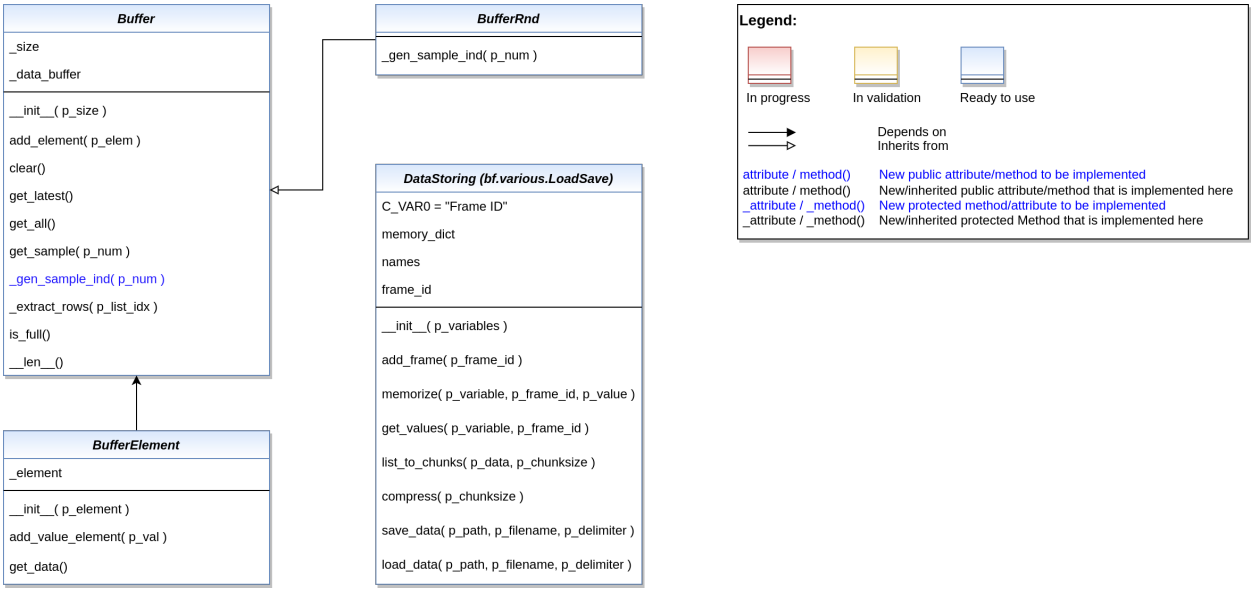
get_name() → str

This method provides a functionality to get the unique name.

Returns
The unique name of the related component.

Return type
str

BF-DATA - Data Management



Ver. 1.3.2 (2023-02-09)

This module provides various elementary data management classes.

class mlpro.bf.data.**DataStoring**(*p_variables*)

Bases: *LoadSave*

This class provides a functionality to store values of variables during training/simulation.

C_VAR0 = 'Frame ID'

add_frame(*p_frame_id*)

To store unique sections in a variable (e.g episodes in RL, etc.)

memorize(*p_variable*, *p_frame_id*, *p_value*)

To store a particular variable into a memory

get_values(*p_variable*, *p_frame_id=None*)

To obtain value from the memory

list_to_chunks(*p_data*, *p_chunksize*)

compress(*p_chunksize*)

save_data(*p_path*, *p_filename=None*, *p_delimiter='\t'*) → bool

To save stored data in memory_dict as a readable file format

load_data(*p_path*, *p_filename*, *p_delimiter='\t'*) → bool

To load data from a readable file format and store them into the DataStoring class format

class mlpro.bf.data.**BufferElement**(*p_element: dict*)

Bases: object

Base class implementation for buffer element

add_value_element(*p_val: dict*)

Adding new value to the element container

Parameters

p_val (*dict*) – Elements in dictionary

get_data()

Get the buffer element.

Returns

Returns the buffer element.

class mlpro.bf.data.**Buffer**(*p_size=1*)

Bases: object

Base class implementation for buffer management.

add_element(*p_elem:* BufferElement)

Add element to the buffer.

Parameters

p_elem (BufferElement) – Element of Buffer

clear()

Resets buffer.

get_latest()

Returns latest buffered element.

get_all()

Return all buffered elements.

get_sample(*p_num: int*)

Sample some element from the buffer.

Parameters
p_num (*int*) – Number of sample

Returns
Samples in dictionary

is_full() → bool

Check if the buffer is full.

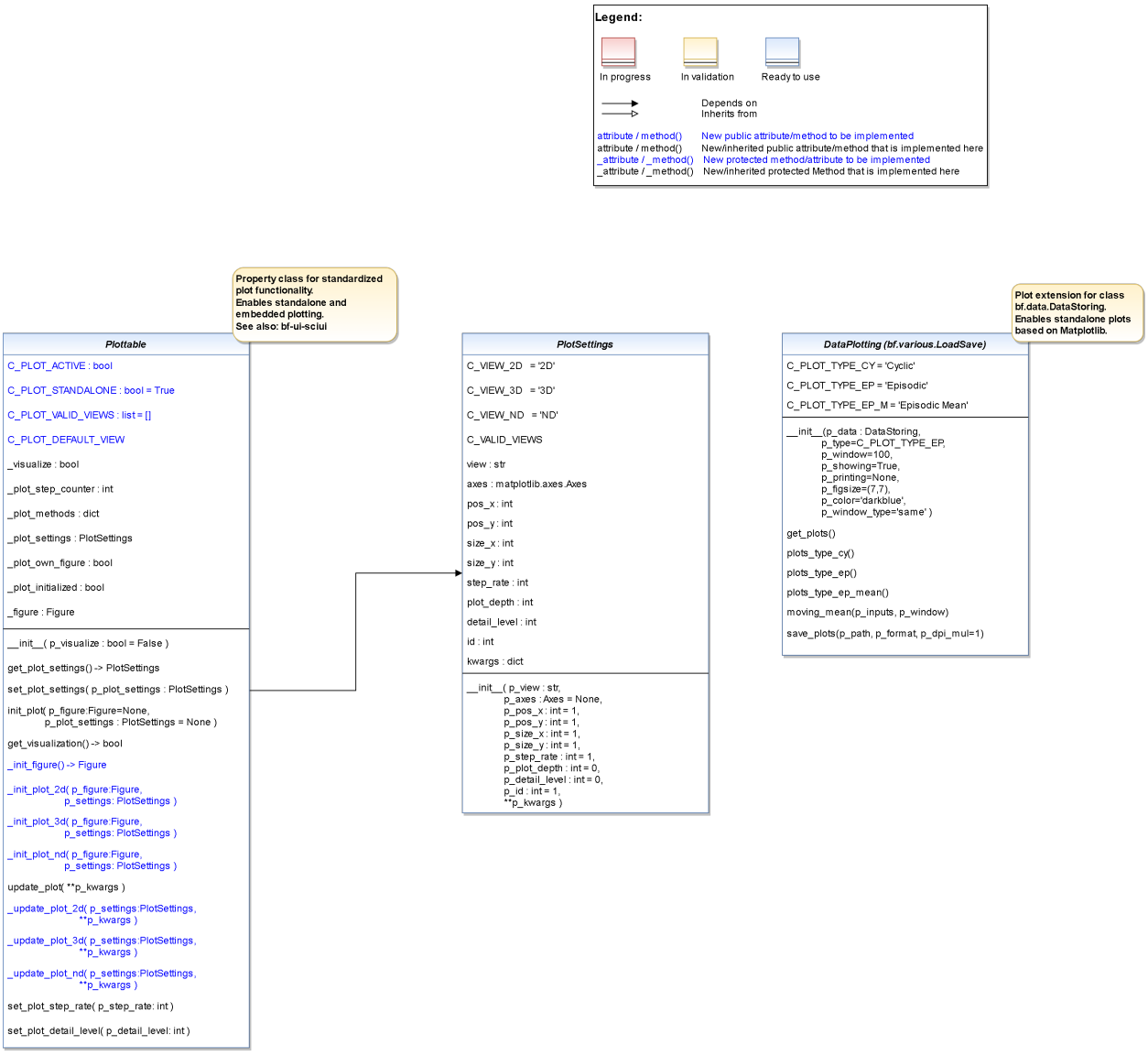
Returns
True, if the buffer is full

class mlpro.bf.data.**BufferRnd**(*p_size=1*)

Bases: *Buffer*

Buffer implmentation with random sampling

BF-PLOT - Plotting and Visualization



Ver. 2.8.2 (2023-02-17)

This module provides various classes related to data plotting.

class mlpro.bf.plot.**PlotSettings**(*p_view: str*, *p_axes: Axes | None = None*, *p_pos_x: int = 1*, *p_pos_y: int = 1*, *p_size_x: int = 1*, *p_size_y: int = 1*, *p_step_rate: int = 1*, *p_horizon: int = 0*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_force_fg: bool = True*, *p_id: int = 1*, ***p_kwargs*)

Bases: `object`

Class to specify the context of a subplot.

Parameters

- **p_view** (*str*) – ID of the view (see constants C_VIEW_*)
- **p_axes** (*Axes*) – Optional Matplotlib Axes object as destination for plot outputs. Default is None.

- **p_pos_x** (*int*) – Optional x position of a subplot within a Matplotlib figure. Default = 1.
- **p_pos_y** (*int*) – Optional y position of a subplot within a Matplotlib figure. Default = 1.
- **p_size_x** (*int*) – Relative size factor in x direction. Default = 1.
- **p_size_y** (*int*) – Relative size factor in y direction. Default = 1.
- **p_step_rate** (*int*) – Optional step rate. Decides after how many calls of the update_plot() method the custom methods _update_plot() carries out an output. Default = 1.
- **p_horizon** (*int*) – Optional plot horizon. A value > 0 limits the number of data entities that are shown in the plot. Default = 0.
- **p_plot_depth** (*int*) – Optional plot depth in case of hierarchical plotting. A value of 0 means that the plot depth is unlimited. Default = 0.
- **p_detail_level** (*int*) – Optional detail level. Default = 0.
- **p_force_fg** (*bool*) – Optional boolean flag. If True, the releated window is kept in foreground. Default = True.
- **p_id** (*int*) – Optional unique id of the subplot within the figure. Default = 1.
- **p_kwargs** (*dict*) – Further optional named parameters.

C_VIEW_2D = '2D'

C_VIEW_3D = '3D'

C_VIEW_ND = 'ND'

C_VALID_VIEWS = ['2D', '3D', 'ND']

class mlpro.bf.plot.**Plottable**(*p_visualize: bool = False*)

Bases: object

Property class that inherits the ability to be plottable. The class is prepared for plotting with MatPlotLib but not restricted to it. Three different views are supported:

2D: 2-dimensional plot 3D: 3-dimensional plot ND: Multidimensional plot

See class Plotsettings for further detais.

Parameters
p_visualize (*bool*) – Boolean switch for visualisation. Default = False.

C_PLOT_ACTIVE

Custom attribute to turn on or off the plot functionality. Must be turned on explicitely.

Type
bool

C_PLOT_STANDALONE

Custom attribute to be set to True, if the plot needs a separate subplot or False if the plot can be added to an existing subplot.

Type
bool = True

C_PLOT_VALID_VIEWS

Custom list of views that are supported/implemented (see class PlotSettings)

Type
list = [*PlotSettings*]

C_PLOT_DEFAULT_VIEW

Custom attribute for the default view. See class PlotSettings for more details.

Type
str = ‘’

C_PLOT_ACTIVE: bool = False

C_PLOT_STANDALONE: bool = True

C_PLOT_VALID_VIEWS: list = []

C_PLOT_DEFAULT_VIEW: str = 'ND'

get_plot_settings() → *PlotSettings*

set_plot_settings(*p_plot_settings:* PlotSettings)

Sets plot settings in advance (before initialization of plot).

Parameters
p_plot_settings (PlotSettings) – New PlotSettings to be set. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

get_visualization() → bool

force_fg()

Internal use.

update_plot(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

set_plot_step_rate(*p_step_rate: int*)

set_plot_detail_level(*p_detail_level: int*)

class mlpro.bf.plot.**DataPlotting**(*p_data:* DataStoring, *p_type='Episodic'*, *p_window=100*, *p_showing=True*, *p_printing=None*, *p_figsize=(7, 7)*, *p_color='darkblue'*, *p_window_type='same'*)

Bases: *LoadSave*

This class provides a functionality to plot the stored values of variables.

Parameters

- **p_data** (DataStoring) – Data object with stored variables values.
- **p_type** (*str, optional*) – Type of plot. The default is C_PLOT_TYPE_EP.
- **p_window** (*int, optional*) – Moving average parameter. The default is 100.

- **p_showing** (*Bool, optional*) – Showing graphs after they are generated. The default is True.
- **p_printing** (*dict, optional*) – Additional important parameters for plotting. [0] = Bool : Whether the stored values is plotted. [1] = Float : Min. value on graph. [2] = Float : Max. value on graph. Set to -1, if you want to set min/max value according to the stored values. Example = {"p_variable_1" : [True,0,-1],

  "p_variable_2" : [True,-0.5,10]}.

  The default is None.
- **p_figsize** (*int, optional*) – Frame size. The default is (7,7).
- **p_color** (*str, optional*) – Line colors. The default is "darkblue".
- **p_window_type** (*str, optional*) – Plotting type for moving average. The default is 'same'. Options: 'same', 'full', 'valid'

C_PLOT_TYPE_CY

one of the plotting types, which plot the graph with multiple lines according to the number of frames.

Type
str

C_PLOT_TYPE_EP

one of the plotting types, which plot the graph everything in one line regardless the number of frames.

Type
str

C_PLOT_TYPE_EP_M

one of the plotting types, which plot only the mean value of each variable for each frame.

Type
str

C_PLOT_TYPE_CY = 'Cyclic'

C_PLOT_TYPE_EP = 'Episodic'

C_PLOT_TYPE_EP_M = 'Episodic Mean'

get_plots()

A function to plot data.

plots_type_cy()

A function to plot data per cycle.

plots_type_ep()

A function to plot data per frame by extending the cyclic plots in one plot.

plots_type_ep_mean()

A function to plot data per frame according to its mean value.

moving_mean(*p_inputs*, *p_window*)

This method creates a series of averages of different subsets of the full data set.

Parameters

- **p_inputs** (*list of floats*) – input dataset.
- **p_window** (*int*) – moving average parameter.

Returns
outputs – transformed data set.

Return type
list of floats

`save_plots`(*p_path*, *p_format*, *p_dpi_mul=1*)

This method is used to save generated plots.

Parameters

- **p_path** (*str*) – Path where file will be saved.
- **p_format** (*str*) – Format of the saved file. Options: 'eps', 'jpg', 'png', 'pdf', 'svg'.
- **p_dpi_mul** (*int, optional*) – Saving plots parameter. The default is 1.

Returns
True, if plots where saved successfully. False otherwise..

Return type
bool

BF-EXCEPTIONS - Exceptions

Ver. 1.0.2 (2021-12-12)

This module provides exception classes.

exception mlpro.bf.exceptions.**ParamError**

Bases: Exception

To be raised on a parameter error…

exception mlpro.bf.exceptions.**ImplementationError**

Bases: Exception

To be raised on an implementation error in a child class of MLPro…

exception mlpro.bf.exceptions.**Error**

Bases: Exception

To be raised on an error…

BF-UI-SCIUI - SciUI Application Class

Ver. 0.6.0 (2021-07-05)

Provides the SciUI application class.

class mlpro.bf.ui.sciui.main.**SciUI**(*p_fullscreen=False*, *p_autoscan_scenarios=True*, *p_start_immediately=True*, *p_logging=False*)

Bases: *SciUIWindow*

C_TYPE = 'Application'

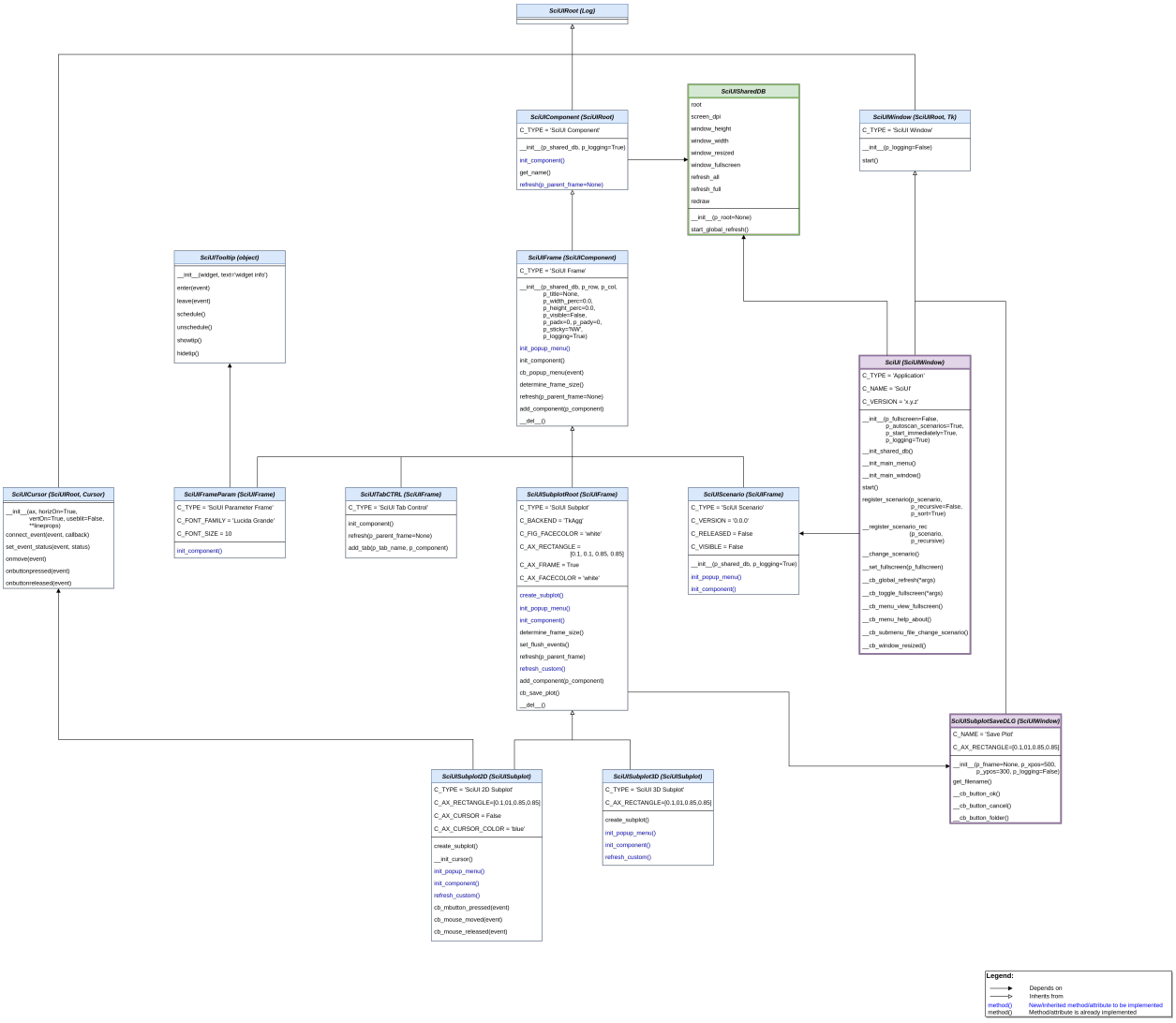
C_NAME = 'SciUI'

C_VERSION = '0.6.0'

start()

register_scenario(*p_scenario:* SciUIScenario, *p_recursive=False*, *p_sort=True*)

BF-UI-SCIUI - SciUI Framework



Ver. 1.1.1 (2022-01-06)

SciUI framework classes to be reused in own SciUI scenarios. Needs Matplotlib 3.3 or higher.

class mlpro.bf.ui.sciui.framework.**SciUISharedDB**(*p_root=None*)

Bases: object

Container for scenario-internal data exchange and communication. Can be extended while runtime by consuming classes.

start_global_refresh()

class mlpro.bf.ui.sciui.framework.**SciUIRoot**(*p_logging=True*)

Bases: *Log*

SciUI root class with overarching properties.

class mlpro.bf.ui.sciui.framework.**SciUIWindow**(*p_logging=False*)

Bases: *SciUIRoot*, Tk

Root class for SciUI window apllications.

C_TYPE = 'SciUI Window'

C_NAME = '????'

start()

class mlpro.bf.ui.sciui.framework.**SciUICursor**(*ax*, *horizOn=True*, *vertOn=True*, *useblit=False*, ***lineprops*)

Bases: *SciUIRoot*, Cursor

Enriched matplotlib cursor widget.

connect_event(*event*, *callback*)

Connect a callback function with an event.

This should be used in lieu of `figure.canvas.mpl_connect` since this function stores callback ids for later clean up.

set_event_status(*event*, *status*)

onmove(*event*)

Internal event handler to draw the cursor when the mouse moves.

onbuttonpressed(*event*)

onbuttonreleased(*event*)

class mlpro.bf.ui.sciui.framework.**SciUITooltip**(*widget*, *text='widget info'*)

Bases: object

Enriched tooltip class.

enter(*event=None*)

leave(*event=None*)

schedule()

unschedule()

showtip(*event=None*)

hidetip()

class mlpro.bf.ui.sciui.framework.**SciUIComponent**(*p_shared_db:* SciUISharedDB, *p_logging=True*)

Bases: *SciUIRoot*

Elementry screen object in SciUI framework.

C_TYPE = 'SciUI Component'

C_NAME = ''

init_component()

Initialization of component-specific elements at instance creation time. To be redefined.

get_name()

refresh(*p_parent_frame=None*)

Refresh of all component-specific elements. To be redefined. Please call super().refresh() at the beginning of your own implementation.

Parameters
object (*p_parent_frame Parent frame*) –

class mlpro.bf.ui.sciui.framework.**SciUIFrame**(*p_shared_db:* SciUISharedDB, *p_row*, *p_col*, *p_title=None*, *p_width_perc=0.0*, *p_height_perc=0.0*, *p_visible=False*, *p_padx=5*, *p_pady=0*, *p_sticky='NW'*, *p_logging=True*)

Bases: *SciUIComponent*

Enriched wrapper class for the Tkinter (Label-)Frame class, based on the Tkinter grid positioning model.

C_TYPE = 'SciUI Frame'

C_NAME = ''

init_popup_menu()

Initializes popup menu of the component. To be redefined.

init_component()

Initialization of component-specific elements at instance creation time. To be redefined.

cb_popup_menu(*p_event*)

determine_frame_size()

refresh(*p_parent_frame=None*)

Refresh of all component-specific elements. To be redefined. Please call super().refresh() at the beginning of your own implementation.

Parameters
object (*p_parent_frame Parent frame*) –

add_component(*p_component:* SciUIComponent)

class mlpro.bf.ui.sciui.framework.**SciUITabCTRL**(*p_shared_db:* SciUISharedDB, *p_row*, *p_col*, *p_title=None*, *p_width_perc=0.0*, *p_height_perc=0.0*, *p_visible=False*, *p_padx=5*, *p_pady=0*, *p_sticky='NW'*, *p_logging=True*)

Bases: *SciUIFrame*

Enriched wrapper class for the Tkinter tab control.

C_TYPE = 'SciUI Tabs'

init_component()

Initialization of component-specific elements at instance creation time. To be redefined.

refresh(*p_parent_frame*)

Refresh of all component-specific elements. To be redefined. Please call super().refresh() at the beginning of your own implementation.

Parameters
object (*p_parent_frame Parent frame*) –

add_component(*p_component:* SciUIComponent)

add_tab(*p_tab_name*, *p_component:* SciUIComponent)

class mlpro.bf.ui.sciui.framework.**SciUISubplotSaveDLG**(*p_fname=None*, *p_xpos=500*, *p_ypos=300*, *p_logging=False*)

Bases: *SciUIWindow*

Small SciUI window application to choose folder and file name for saving a SciUI subplot.

C_NAME = 'Save Plot'

C_FONT_FAMILY = 'Lucida Grande'

C_FONT_SIZE = 10

C_FILENAME = 'myplot'

C_SUFFIXES = ['.pdf', '.png', '.svg']

get_filename()

class mlpro.bf.ui.sciui.framework.**SciUISubplotRoot**(*p_shared_db:* SciUISharedDB, *p_row*, *p_col*, *p_title=None*, *p_width_perc=0.0*, *p_height_perc=0.0*, *p_visible=False*, *p_padx=5*, *p_pady=0*, *p_sticky='NW'*, *p_logging=True*)

Bases: *SciUIFrame*

Root class for specialized frame classes that embedd a matplotlib figure with one subplot into a Tkinter frame. Not intended for direct reuse. Please use inherited classes SciUISubplot2D, SciUISubplot3D instead.

C_TYPE = 'SciUI Subplot'

C_BACKEND = 'TkAgg'

C_FIG_FACECOLOR = 'white'

C_AX_RECTANGLE = [0.1, 0.1, 0.85, 0.85]

C_AX_FRAME = True

C_AX_FACECOLOR = 'white'

create_subplot()

Internally used to create a suitable subplot. New subplot needs to be bound to self.ax. To be redefined.

init_popup_menu()

Initializes popup menu of the component. To be redefined.

init_component()

Initialization of component-specific elements at instance creation time. To be redefined. Please call super().init_component() at beginning of your implementation.

determine_frame_size()

set_flush_events(*p_flush: bool*)

refresh(*p_parent_frame*)

Refresh of all component-specific elements. To be redefined. Please call super().refresh() at the beginning of your own implementation.

Parameters
object (*p_parent_frame Parent frame*) –

refresh_custom()

Additional refresh activities. To be redefined.

add_component(*p_component*)

Adding further subcomponents is disabled here.

cb_save_plot()

class mlpro.bf.ui.sciui.framework.**SciUISubplot2D**(*p_shared_db: SciUISharedDB*, *p_row*, *p_col*, *p_title=None*, *p_width_perc=0.0*, *p_height_perc=0.0*, *p_visible=False*, *p_padx=5*, *p_pady=0*, *p_sticky='NW'*, *p_logging=True*)

Bases: *SciUISubplotRoot*

Specialized frame class that embedds a 2D matplotlib figure with one subplot into a Tkinter frame. A cross hair cursor functinality with mouse event handling can be switched on/off via class constant C_AX_CURSOR.

C_TYPE = 'SciUI 2D Subplot'

C_AX_RECTANGLE = [0.1, 0.1, 0.85, 0.85]

C_AX_CURSOR = False

C_AX_CURSOR_COLOR = 'blue'

create_subplot()

Internally used to create a suitable subplot. New subplot needs to be bound to self.ax. To be redefined.

cb_mbutton_pressed(*event*)

cb_mouse_moved(*event*)

cb_mbutton_released(*event*)

class mlpro.bf.ui.sciui.framework.**SciUISubplot3D**(*p_shared_db: SciUISharedDB*, *p_row*, *p_col*, *p_title=None*, *p_width_perc=0.0*, *p_height_perc=0.0*, *p_visible=False*, *p_padx=5*, *p_pady=0*, *p_sticky='NW'*, *p_logging=True*)

Bases: *SciUISubplotRoot*

Specialized frame class that embedds a 3D matplotlib figure with one subplot into a Tkinter frame.

C_TYPE = 'SciUI 3D Subplot'

C_AX_RECTANGLE = [0.1, 0.1, 0.85, 0.85]

create_subplot()

Internally used to create a suitable subplot. New subplot needs to be bound to self.ax. To be redefined.

class mlpro.bf.ui.sciui.framework.**SciUIFrameParam**(*p_shared_db: SciUISharedDB*, *p_row*, *p_col*, *p_title=None*, *p_width_perc=0.0*, *p_height_perc=0.0*, *p_visible=False*, *p_padx=5*, *p_pady=0*, *p_sticky='NW'*, *p_logging=True*)

Bases: *SciUIFrame*

Class for parameter/text frames.

C_TYPE = 'SciUI Parameter Frame'

C_NAME = ''

C_FONT_FAMILY = 'Lucida Grande'

C_FONT_SIZE = 10

init_component()

Initialization of component-specific elements at instance creation time. To be redefined.

class mlpro.bf.ui.sciui.framework.**SciUIScenario**(*p_shared_db:* SciUISharedDB, *p_logging=True*)

Bases: *SciUIFrame*

Top level class for an entire SciUI scenario that can be registered by the SciUI application class. SciUI scenarios are visible and chooseable if the switches C_RELEASED and C_VISIBLE are set to True.

C_TYPE = 'SciUI Scenario'

C_NAME = '????'

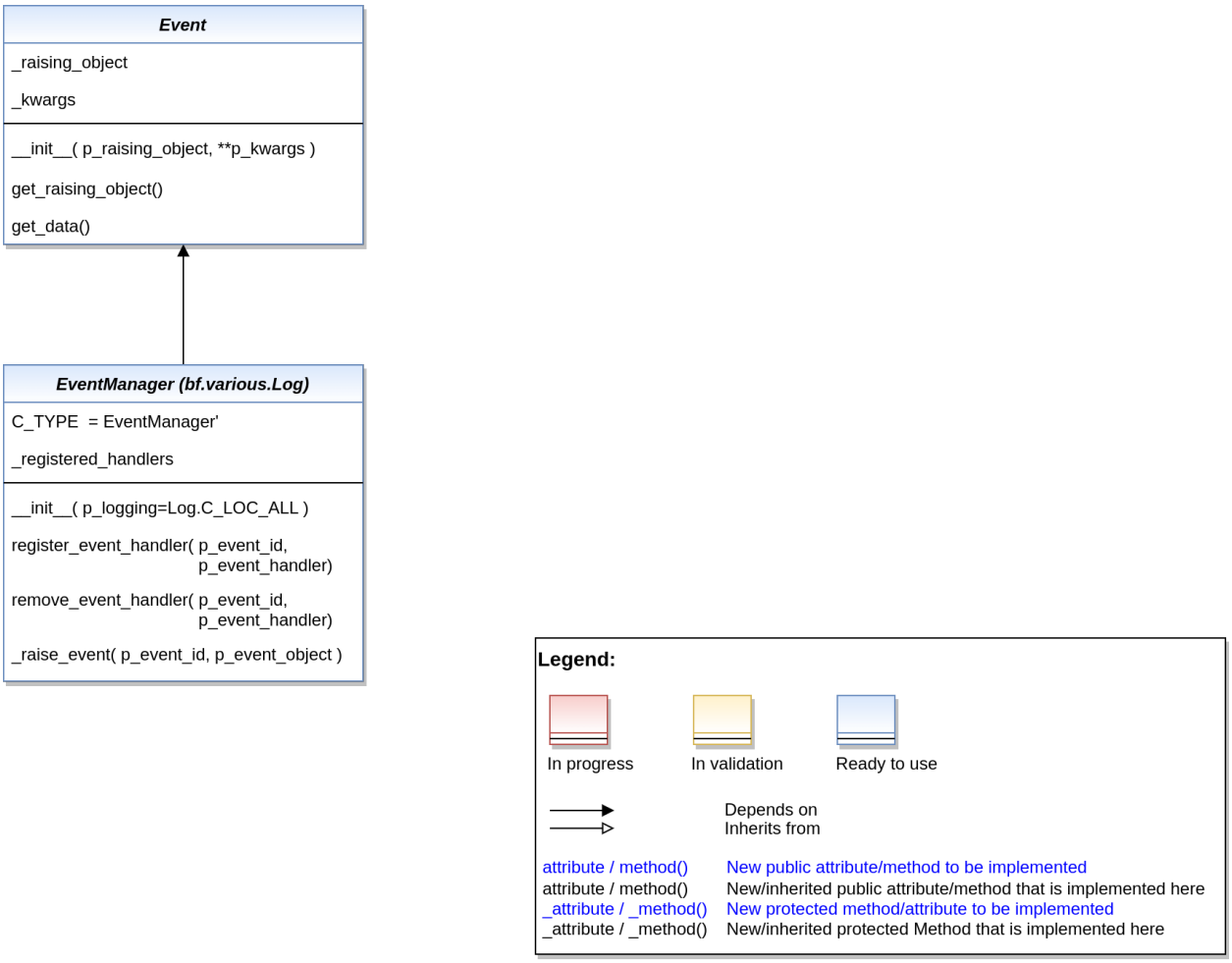
C_VERSION = '0.0.0'

C_RELEASED = False

C_VISIBLE = False

Layer 1 - Computation

BF-EVENTS - Event Handling



Ver. 1.1.1 (2023-02-15)

This module provides classes for event handling. To this regard, the property class Eventmanager is provided to add event functionality to child classes by inheritence.

class mlpro.bf.events.**Event**(*p_raising_object*, ***p_kwargs*)

Bases: `object`

Root class for events. It is ready to use and transfers the raising object and further key/value data to the event handler.

Parameters

- **p_raising_object** – Reference to object that raised the event.
- ****p_kwargs** – List of named parameters

get_raising_object()

get_data()

class mlpro.bf.events.**EventManager**(*p_logging=True*)

Bases: *Log*

This property class provides universal event management functionalities to be inherited to child classes.

Parameters
p_logging – Log level (see constants of class Log). Default: Log.C_LOG_ALL

C_TYPE = 'EventManager'

register_event_handler(*p_event_id: str*, *p_event_handler*)

Registers an event handler.

Parameters

- **p_event_id** (*str*) – Unique event id
- **p_event_handler** – Reference to an event handler method with parameters p_event_id and p_event_object:Event

remove_event_handler(*p_event_id: str*, *p_event_handler*)

Removes an already registered event handler.

Parameters

- **p_event_id** – Unique event id
- **p_event_handler** – Reference to an event handler method.

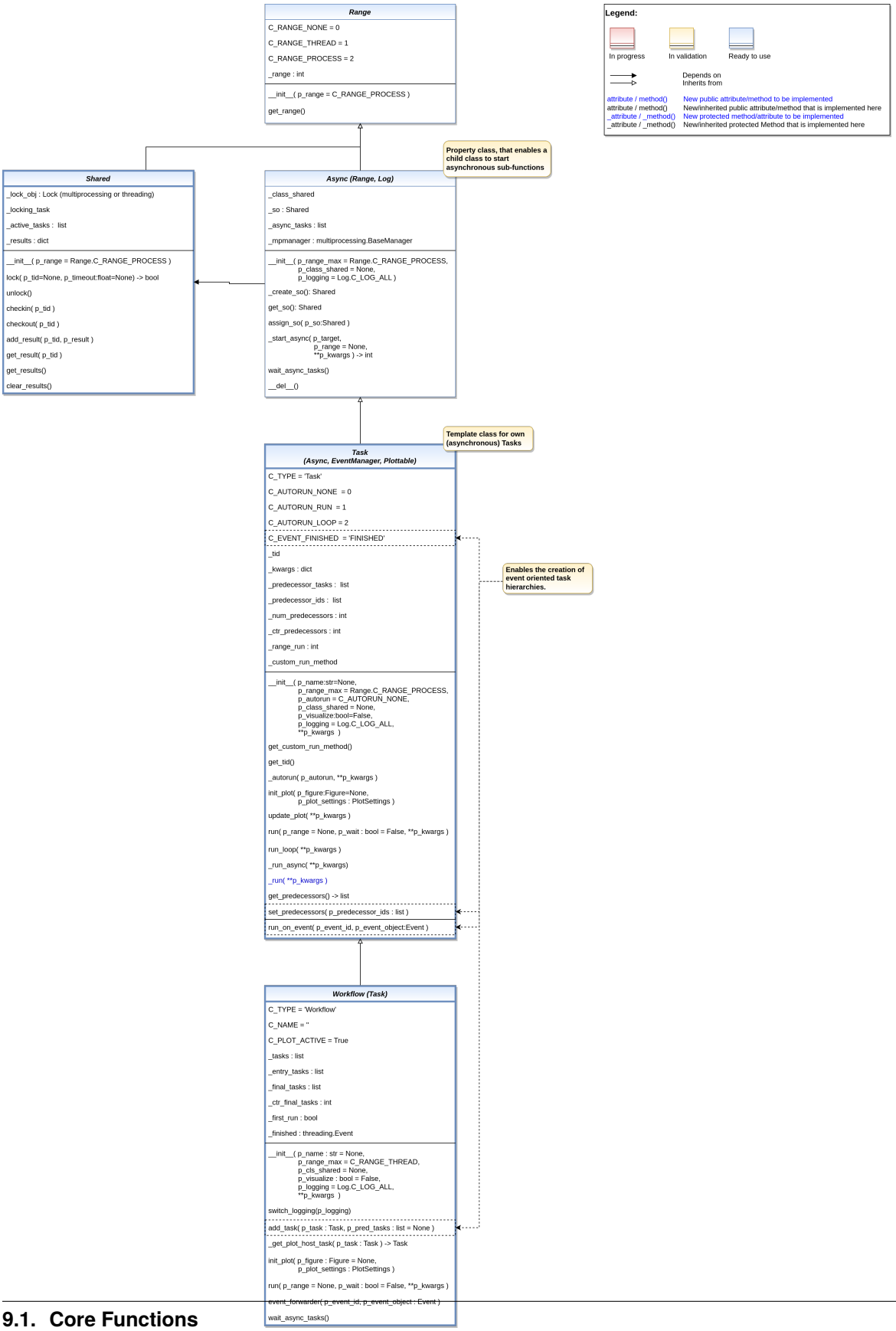
_raise_event(*p_event_id: str*, *p_event_object:* Event)

Raises an event and calls all registered handlers. To be used inside an event manager class.

Parameters

- **p_event_id** (*str*) – Unique event id
- **p_event_object** (Event) – Event object with further context informations

BF-MT - Multitasking



This module provides classes for multitasking with optional interprocess communication (IPC) based on shared objects. Multitasking in MLPro combines multrithreading and multiprocessing and simplifies parallel programming.

Annotation to multitasking: Standard Python package multiprocessing uses pickle for serialization. This leads to problems with more complex objects. That was the reason to opt for the more flexible package multiprocess, which is a fork of multiprocessing and uses dill for serialization.

See also: https://stackoverflow.com/questions/40234771/replace-pickle-in-python-multiprocessing-lib

class mlpro.bf.mt.**Range**(*p_range: int = 2*)

Bases: object

Property class that defines the possible ranges of asynchronous execution supported by MLPro.

Parameters
p_range (*int*) – Range of asynchonicity

C_RANGE_NONE

Synchronous execution only.

Type
int

C_RANGE_THREAD

Asynchronous execution as separate thread within the current process.

Type
int

C_RANGE_PROCESS

Asynchronous execution as separate process within the current machine.

Type
int

C_VALID_RANGES

List of valid ranges.

Type
list

C_RANGE_NONE = 0

C_RANGE_THREAD = 1

C_RANGE_PROCESS = 2

C_VALID_RANGES = [0, 1, 2]

get_range() → int

class mlpro.bf.mt.**Shared**(*p_range: int = 2*)

Bases: *Range*

Template class for shared objects. It is ready to use and the default class for IPC. It provides elementary mechanisms for access control and messaging.

It is also possible to inherit and enrich a child class for special needs but please beware that at least in multiprocessing mode (p_range=Range.C_RANGE_PROCESS) a direct access to attributes is not possible. Child classes should generally provide suitable methods for access to attribues.

Parameters
p_range (*int*) – Range of asynchonicity

C_MSG_TYPE_DATA = 0

C_MSG_TYPE_TERM = 1

lock(*p_tid=None*, *p_timeout: float | None = None*) → bool

Locks the shared object for a specific process.

Parameters

- **p_tid** – Unique task id. If None then the internal locking mechanism is disabled.
- **p_timeout** (*float*) – Optional timeout in seconds. If None, timeout is infinite.
- **Returns** –
- **True** –
- **otherwise.** (*if shared object was locked successfully. False*) –

unlock()

Unlocks the shared object.

checkin(*p_tid*)

Registers a task.

Parameters
p_tid – Task id.

checkout(*p_tid*)

Unregisters a task.

Parameters
p_tid – Task id.

add_result(*p_tid*, *p_result*)

Adds a result for a task.

Parameters

- **p_tid** – Task id.
- **p_result** – Any kind of result data.

get_result(*p_tid*)

Returns the result data of a task.

Parameters
p_tid – Task id.

Returns
Result data of a task.

Return type
task_results

get_results()

Returns reference to internal dictionary of results

Returns
results – Dictionary of results

Return type
dict

clear_results()

Clears internal dictionary of results

_range: int

class mlpro.bf.mt.**Async**(*p_range_max: int = 2*, *p_class_shared=None*, *p_logging=True*)

Bases: *Range*, *Log*

Property class that enables child classes to run sub-tasks asynchronously. Depending on the given range a task can be executed as a separate thread in the same process or a separate process on the same machine.

Parameters

- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_class_shared** – Optional class for a shared object (class Shared or a child class of Shared)
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL

_create_so(*p_range: int*, *p_class_shared*) → *Shared*

Internal use. Creates a suitable shared object for the given range.

Parameters

- **p_range** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_class_shared** – Class for a shared object (class Shared or a child class of Shared)

Returns
so – A new shared object

Return type
Shared

get_so() → *Shared*

Returns the associated shared object.

Returns
so – Shared object of type Shared (or inherited)

Return type
Shared

assign_so(*p_so:* Shared)

Assigns an existing shared object to the task. The task takes over the range of asynchronicity of the shared object if it is less than the current one of the task.

Parameters
p_so (Shared) – Shared object.

_start_async(*p_target*, *p_range: int | None = None*, ***p_kwargs*) → int

Starts a method or a new instance of a given class asynchronously. If neither a method nor a class is specified, a new instance of the current class is created asynchronously.

Parameters

- **p_target** – A class, method or function to be executed (a)synchronously depending on the actual range

- **p_range** (*int*) – Optional deviating range of asynchonicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Oterwise the minimum range of both is taken.
- **p_kwargs** (*dictionary*) – Parameters to be handed over to asynchonous method/instance

Returns
range – Actual range of asynchronicity

Return type
int

wait_async_tasks()

Waits until all internal asynchonous tasks are finished.

_range: int

class mlpro.bf.mt.**Task**(*p_name: str | None = None*, *p_range_max: int = 1*, *p_autorun=0*, *p_class_shared=None*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *Async*, *EventManager*, *Plottable*

Template class for a task, that can run things - and even itself - asynchronously in a thread or process. Tasks can run standalone or as part of a workflow (see class Workflow). The integrated event manager allows callbacks on specific events inside the same process(!).

Parameters

- **p_name** (*str*) – Optional name of the task. Default is None.
- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_THREAD.
- **p_autorun** (*int*) – On value C_AUTORUN_RUN method run() is called imediately during instantiation. On vaule C_AUTORUN_LOOP method run_loop() is called. Value C_AUTORUN_NONE (default) causes an object instantiation without starting further actions.
- **p_class_shared** – Optional class for a shared object (class Shared or a child class of Shared)
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*dict*) – Further optional named parameters.

C_TYPE = 'Task'

C_AUTORUN_NONE = 0

C_AUTURUN_RUN = 1

C_AUTORUN_LOOP = 2

C_EVENT_FINISHED = 'FINISHED'

_get_custom_run_method()

get_tid()

Returns unique task id.

_autorun(*p_autorun*, ***p_kwargs*)

Internal method to automate a single or looped run.

Parameters

- **p_autorun** (*int*) – On value C_AUTORUN_RUN method run() is called imediately during instantiation. On vaule C_AUTORUN_LOOP method run_loop() is called. Value C_AUTORUN_NONE (default) causes an object instantiation without starting further actions.
- **p_kwargs** (*dict*) – Further parameters handed over to method run().

run(*p_range: int | None = None*, *p_wait: bool = False*, ***p_kwargs*)

Executes the task specific actions implemented in custom method _run(). At the end event C_EVENT_FINISHED is raised to start subsequent actions (p_wait=True).

Parameters

- **p_range** (*int*) – Optional deviating range of asynchonicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Oterwise the minimum range of both is taken.
- **p_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p_kwargs** (*dict*) – Further parameters handed over to custom method _run().

_run_async(***p_kwargs*)

Internally used by method run(). It runs the custom method _run() and raises event C_EVENT_FINISHED.

Parameters
p_kwargs (*dict*) – Custom parameters.

_run(***p_kwargs*)

Custom method that is called (asynchronously) by method run().

Parameters
p_kwargs (*dict*) – Custom parameters.

run_loop(***p_kwargs*)

Executes method run() in a loop, until a message of type Shared.C_MSG_TYPE_TERM is sent to the task.

Parameters
p_kwargs (*dict*) – Parameters for method run()

get_predecessors() → list

set_predecessors(*p_predecessor_tasks: list*)

Used by class Workflow to inform a task about it's number of predecessor tasks. See method run_on_event().

Parameters
p_predecessor_ids (*list*) – List of ids of predecessor tasks in a workflow.

run_on_event(*p_event_id*, *p_event_object:* Event)

Can be used as event handler - in particular for other tasks in a workflow in combination with event C_EVENT_FINISHED. Method self.run() is called if the last predecessor task in a workflow has raised event C_EVENT_FINISHED.

Parameters

- **p_event_id** – Event id.
- **p_event_object** (Event) – Event object with further context informations.

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(***p_kwargs*)

Updates the plot.

Parameters

****p_kwargs** – Implementation-specific plot data and/or parameters.

_range: int

_so: *Shared*

class mlpro.bf.mt.**Workflow**(*p_name: str | None = None*, *p_range_max=1*, *p_class_shared=None*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *Task*

Ready-to-use container class for task groups. Objects of type Task (or inherited) can be added and chained to sequences or hierarchies of tasks.

Parameters

- **p_name** (*str*) – Optional name of the task. Default is None.
- **p_range_max** (*int*) – Range of asynchonicity. See class Range. Default is Range.C_RANGE_THREAD.
- **p_class_shared** – Optional class for a shared object (class Shared or a child class of Shared)
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*dict*) – Further optional named parameters handed over to every task within.

C_TYPE = 'Workflow'

C_NAME = ''

C_PLOT_ACTIVE: bool = True

switch_logging(*p_logging*)

Sets log level for the workflow and all tasks inside.

Parameters

p_logging – Log level (see constants of class Log).

add_task(*p_task:* Task, *p_pred_tasks: list | None = None*)

Adds a task to the workflow.

Parameters

- **p_task** (Task) – Task object to be added.
- **p_pred_tasks** (*list*) – Optional list of predecessor task objects

_get_plot_host_task(*p_task:* Task) → *Task*

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot of a workflow. The method creates a host figure for all tasks if no external host figure is parameterized. The sub-plots of the tasks are autmatically arranged within the host figure.

See method init_plot() of class mlpro.bf.plot.Plottable for further details.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

run(*p_range: int | None = None*, *p_wait: bool = False*, ***p_kwargs*)

Executes all tasks of the workflow. At the end event C_EVENT_FINISHED is raised to start subsequent actions (p_wait=True).

Parameters

- **p_range** (*int*) – Optional deviating range of asynchonicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Oterwise the minimum range of both is taken.
- **p_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p_kwargs** (*dict*) – Further parameters handed over to custom method _run().

event_forwarder(*p_event_id*, *p_event_object:* Event)

Internally used to raise event C_EVENT_FINISHED on workflow level if all final tasks have been finished.

Parameters

- **p_event_id** – Event id.
- **p_event_object** (Event) – Event object with further context informations.

wait_async_tasks()

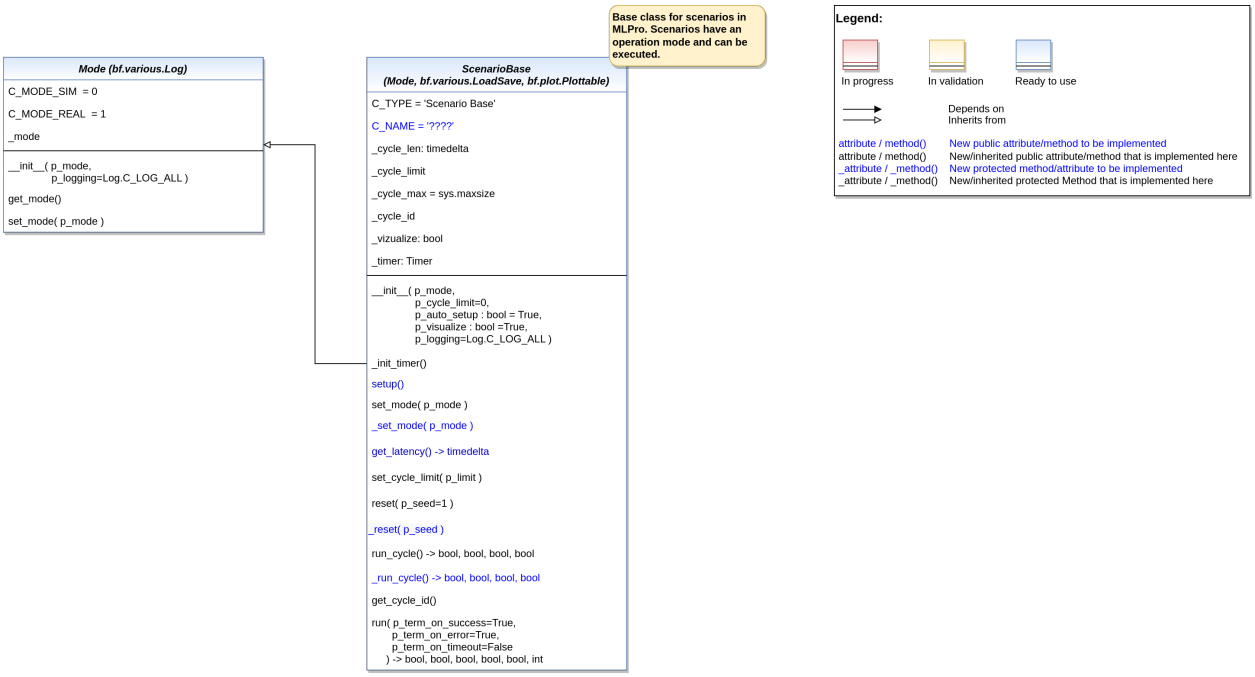
Waits until all tasks are finished.

_range: *int*

_so: *Shared*

_plot_settings: *PlotSettings*

BF-OPS - Operations



Ver. 1.2.2 (2022-11-21)

This module provides classes for operation.

class mlpro.bf.ops.**Mode**(*p_mode*, *p_logging=True*)

Bases: *Log*

Property class that adds a mode and related methods to a child class.

Parameters

- **p_mode** – Operation mode. Valid values are stored in constant C_VALID_MODES.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL

C_MODE_SIM = 0

Simulation mode.

C_MODE_REAL = 1

Real operation mode.

C_VALID_MODES

List of valid modes.

Type
list

C_MODE_INITIAL = -1

C_MODE_SIM = 0

C_MODE_REAL = 1

C_VALID_MODES = [0, 1]

get_mode()

Returns current mode.

set_mode(*p_mode*)

Sets new mode.

Parameters

p_mode – Operation mode. Valid values are stored in constant C_VALID_MODES.

class mlpro.bf.ops.**ScenarioBase**(*p_mode*, *p_cycle_limit=0*, *p_auto_setup: bool = True*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *Mode*, *LoadSave*, *Plottable*

Base class for executable scenarios in MLPro. To be inherited and specialized in higher layers.

The following key features are included:

- Operation mode
- Cycle management
- Timer
- Latency

Parameters

- **p_mode** – Operation mode. See Mode.C_VALID_MODES for valid values. Default = Mode.C_MODE_SIM.
- **p_cycle_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).
- **p_auto_setup** (*bool*) – If True custom method setup() is called after initialization.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = True.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.

C_TYPE = 'Scenario Base'

C_NAME = '????'

_init_timer()

setup()

Custom method to set up all components of the scenario.

set_mode(*p_mode*)

Sets operation mode of the scenario. Custom method _set_mode() is called.

Parameter

p_mode

Operation mode. See class bf.ops.Mode for further details.

_set_mode(*p_mode*)

Custom method to set the operation mode of components of the scenario. See method set_mode() for further details.

Parameter

p_mode
Operation mode. See class bf.ops.Mode for further details.

`get_latency()` → timedelta

Returns the latency of the scenario. To be implemented in child class.

`set_cycle_limit(p_limit)`

Sets the maximum number of cycles to run.

Parameters
p_cycle_limit (*int*) – Maximum number of cycles. Default = 0 (no limit).

`reset(p_seed=1)`

Resets the scenario and especially the ML model inside. Internal random generators are seed with the given value. Custom reset actions can be implemented in method _reset().

Parameters
p_seed (*int*) – Seed value for internal random generator

`_reset(p_seed)`

Custom method to reset the components of the scenario and to set the given random seed value. See method reset() for further details.

Parameters
p_seed (*int*) – Seed value for internal random generator

`run_cycle()`

Runs a single process cycle.

Returns

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **timeout** (*bool*) – True on timeout. False otherwise.
- **cycle_limit** (*bool*) – True, if cycle limit has reached. False otherwise.
- **adapted** (*bool*) – True, if something within the scenario has adapted something in this cycle. False otherwise.
- **end_of_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

`_run_cycle()`

Custom implementation of a single process cycle. To be redefined.

Returns

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **adapted** (*bool*) – True, if something within the scenario has adapted something in this cycle. False otherwise.
- **end_of_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

get_cycle_id()

Returns current cycle id.

run(*p_term_on_success: bool = True*, *p_term_on_error: bool = True*, *p_term_on_timeout: bool = False*)

Runs the scenario as a sequence of single process steps until a terminating event occures.

Parameters

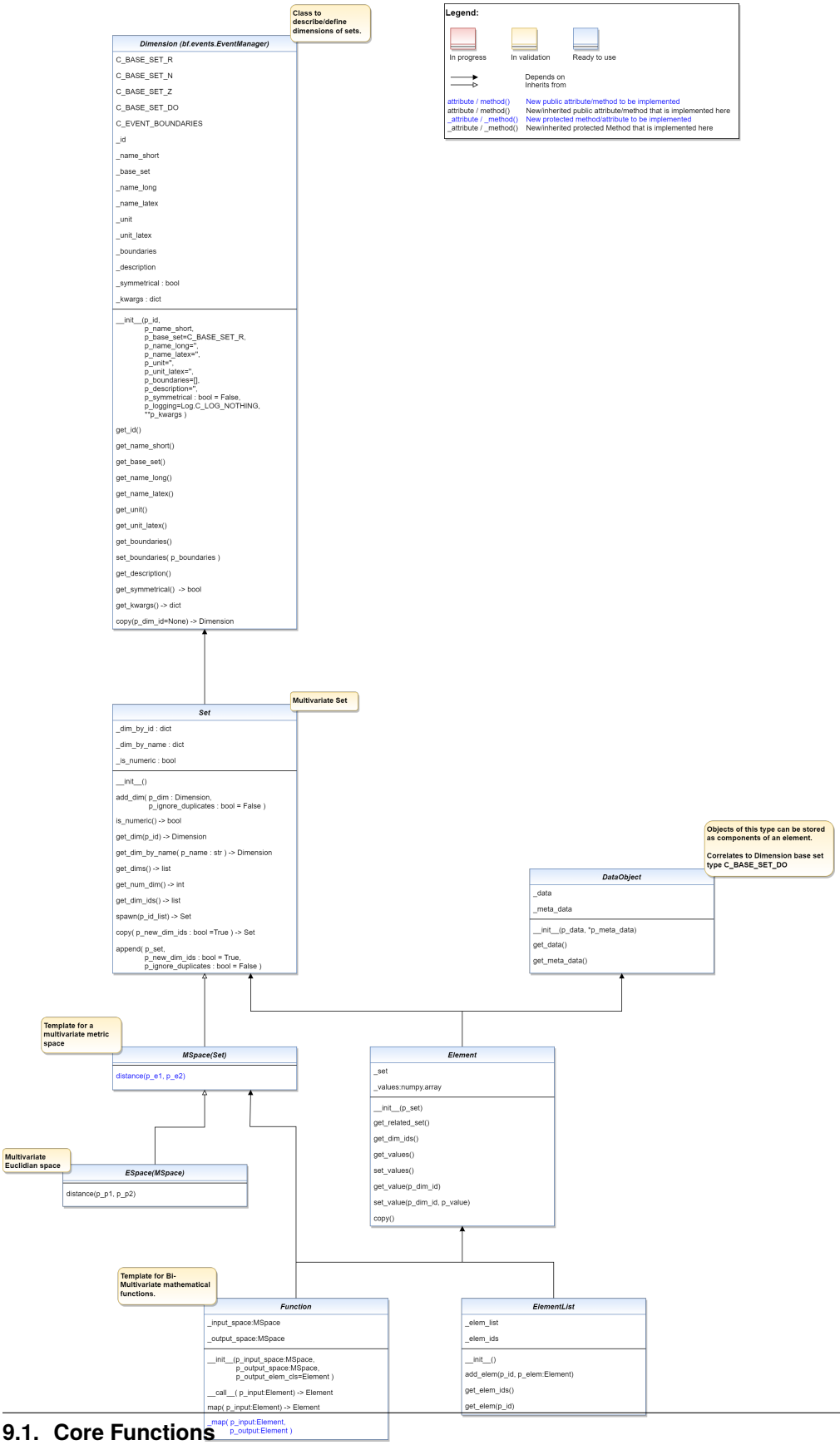
- **p_term_on_success** (*bool*) – If True, the run terminates on success. Default = True.
- **p_term_on_error** (*bool*) – If True, the run terminates on error. Default = True.
- **p_term_on_timeout** (*bool*) – If True, the run terminates on timeout. Default = False.

Returns

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **timeout** (*bool*) – True on timeout. False otherwise.
- **cycle_limit** (*bool*) – True, if cycle limit has reached. False otherwise.
- **adapted** (*bool*) – True, if ml model adapted something. False otherwise.
- **end_of_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.
- **num_cycles** (*int*) – Number of cycles.

Layer 2 - Mathemathics

BF-MATH - Basic Mathematics



This module provides basic mathematical classes.

class mlpro.bf.math.basics.**Dimension**(*p_name_short*, *p_base_set='R'*, *p_name_long=''*, *p_name_latex=''*, *p_unit=''*, *p_unit_latex=''*, *p_boundaries: list = []*, *p_description=''*, *p_symmetrical: bool = False*, *p_logging=False*, ***p_kwargs*)

Bases: *EventManager*

Objects of this type specify properties of a dimension of a set.

Parameters:

p_name_short
[str] Short name of dimension

p_base_set
Base set of dimension. See constants C_BASE_SET_*. Default = C_BASE_SET_R.

p_name_long :str
Long name of dimension (optional)

p_name_latex
[str] LaTeX name of dimension (optional)

p_unit
[str] Unit (optional)

p_unit_latex
[str] LaTeX code of unit (optional)

p_boundaries
[list] List with minimum and maximum value (optional)

p_description
[str] Description of dimension (optional)

p_symmetrical
[bool] Information about the symmetry of the dimension (optional, default is False)

p_logging
Log level (see constants of class Log). Default: Log.C_LOG_ALL

p_kwargs
[dict] Further optional keyword parameters.

C_TYPE = 'Dimension'

C_BASE_SET_R = 'R'

C_BASE_SET_N = 'N'

C_BASE_SET_Z = 'Z'

C_BASE_SET_DO = 'DO'

C_EVENT_BOUNDARIES = 'BOUNDARIES'

get_id()

get_name_short()

get_base_set()

get_name_long()

get_name_latex()

get_unit()

get_unit_latex()

get_boundaries()

set_boundaries(*p_boundaries: list*)

Sets new boundaries with respect to the symmmetry and raises event C_EVENT_BOUNDARIES.

Parameters

p_boundaries (*list*) – New boundaries (lower and upper value)

get_description()

get_symmetrical() → bool

get_kwargs() → dict

copy()

class mlpro.bf.math.basics.**Set**

Bases: object

Objects of this type describe a (multivariate) set in a mathematical sense.

C_NUMERIC_BASE_SETS = ['N', 'Z', 'R']

add_dim(*p_dim:* Dimension, *p_ignore_duplicates: bool = False*)

Raises the dimensionality of the set by adding a new dimension.

Parameters

- **p_dim** (Dimension) – Dimension to be added.
- **p_ignore_duplicates** (*bool*) – If True, duplicated short names of dimensions are accepted. Default = False.

is_numeric() → bool

Returns True if the set consists of numeric dimensions only.

get_dim(*p_id*) → *Dimension*

Returns the dimension specified by it's unique id.

get_dim_by_name(*p_name*) → *Dimension*

get_dims() → list

" Returns all dimensions.

get_num_dim()

Returns the dimensionality of the set (=number of dimensions of the set).

get_dim_ids()

Returns the unique ids of the related dimensions.

spawn(*p_id_list: list*)

Spawns a new class with same type and a subset of dimensions specified by an index list.

Parameters
adopted (*p_id_list List of indices of dimensions to be*) –

Returns
New object with subset of dimensions

copy(*p_new_dim_ids: bool = True*)

append(*p_set*, *p_new_dim_ids: bool = True*, *p_ignore_duplicates: bool = False*)

class mlpro.bf.math.basics.**DataObject**(*p_data*, **p_meta_data*)

Bases: object

Container class for (big) data objects of any type with optional additional meta data.

get_data()

get_meta_data() → tuple

class mlpro.bf.math.basics.**Element**(*p_set:* Set)

Bases: object

Element of a (multivariate) set.

Parameters
p_set (Set) – Underlying set.

get_related_set() → *Set*

set_related_set(*p_set:* Set)

get_dim_ids() → list

get_values()

set_values(*p_values*)

Overwrites the values of all components of the element.

Parameters
dimensions. (*p_values Something iterable with same length as number of element*) –

get_value(*p_dim_id*)

set_value(*p_dim_id*, *p_value*)

copy()

class mlpro.bf.math.basics.**ElementList**

Bases: object

List of Element objects.

add_elem(*p_id*, *p_elem:* Element)

Adds an element object under it's id in the internal element list.

Parameters

- **element** (*p_id Unique id of the*) –

- **added** (*p_elem Element object to be*) –

get_elem_ids() → list

get_elem(*p_id*) → *Element*

class mlpro.bf.math.basics.**MSpace**

Bases: *Set*

Objects of this type represent a metric space. The method distance implements the metric of the space.

distance(*p_e1:* Element, *p_e2:* Element)

class mlpro.bf.math.basics.**ESpace**

Bases: *MSpace*

Objects of this type represent an Euclidian space. The distance method implements the Euclidian norm.

distance(*p_e1:* Element, *p_e2:* Element)

class mlpro.bf.math.basics.**Function**(*p_input_space: ~mlpro.bf.math.basics.MSpace*, *p_output_space: ~mlpro.bf.math.basics.MSpace*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*)

Bases: object

Model class for an elementary bi-multivariate mathematical function that maps elements of a multivariate input space to elements of a multivariate output space.

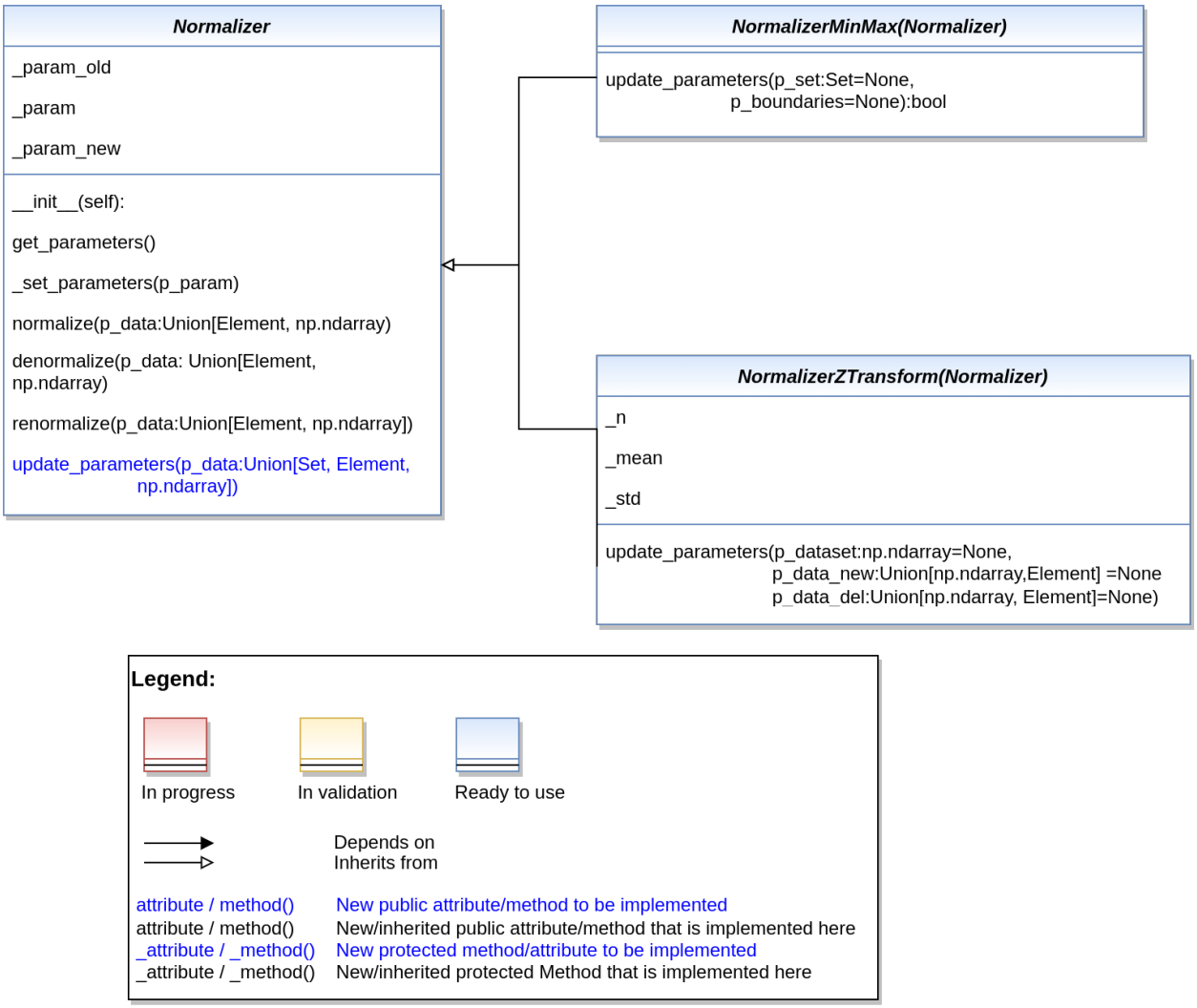
map(*p_input:* Element) → *Element*

Alternative method to map an input to an output. Actually, it refers to method __call__(). Redefining this method has no effect. See method __call__() and constructor for further details.

_map(*p_input:* Element, *p_output:* Element)

Custom method for own mapping algorithm. See methods __call__() and map() for further details.

BF-MATH-NORMALIZERS - Normalizers



Ver. 1.0.14 (2023-02-13) This module provides base class for Normalizers and normalizer objects including MinMax normalization and normalization by Z transformation.

class mlpro.bf.math.normalizers.**Normalizer**

Bases: `object`

Base template class for normalizer objects.

_set_parameters(*p_param*)

custom method to set the normalization parameters

Parameters

p_set (`Set`) – Set related to the elements to be normalized

Returns

boolean – Returns true after setting the parameters

Return type
True

normalize(*p_data:* Element | *ndarray*)

Method to normalize a data (Element/ndarray) element based on MinMax or Z-transformation

Parameters
p_data (*Element or a numpy array*) – Data element to be normalized

Returns
element – Normalized Data

Return type
Element or numpy array

denormalize(*p_data:* Element | *ndarray*)

Method to denormalize a data (Element/ndarray) element based on MinMax or Z-transformation

Parameters
p_data (*Element or a numpy array*) – Data element to be denormalized

Returns
element – Denormalized Data

Return type
Element or numpy array

renormalize(*p_data:* Element | *ndarray*)

Method to denormalize and renormalize an element based on old and current normalization parameters.

Parameters
p_data (*Element or numpy array*) – Element to be renormalized.

Returns
renormalized_element – Renormalized Data

Return type
Element or numpy array

update_parameters(*p_data:* Set | Element | *ndarray*)

Custom method to update normalization parameters.

Parameters
p_data – arguments specific to normalization parameters. Check the normalizer objects for specific parameters

class mlpro.bf.math.normalizers.**NormalizerMinMax**

Bases: *Normalizer*

Class to normalize elements based on MinMax normalization.

update_parameters(*p_set:* Set | *None = None*, *p_boundaries: list* | *ndarray* | *None = None*)

Method to update the normalization parameters of MinMax normalizer.

Parameters

- **p_set** (*Set*) – Set related to the elements to be normalized
- **p_boundaries** (*ndarray*) – array consisting of boundaries related to the dimension of the array

class mlpro.bf.math.normalizers.**NormalizerZTrans**

Bases: *Normalizer*

Class for Normalization based on Z transformation.

update_parameters(*p_dataset: ndarray | None = None*, *p_data_new:* Element *| ndarray | None = None*, *p_data_del:* Element *| ndarray | None = None*)

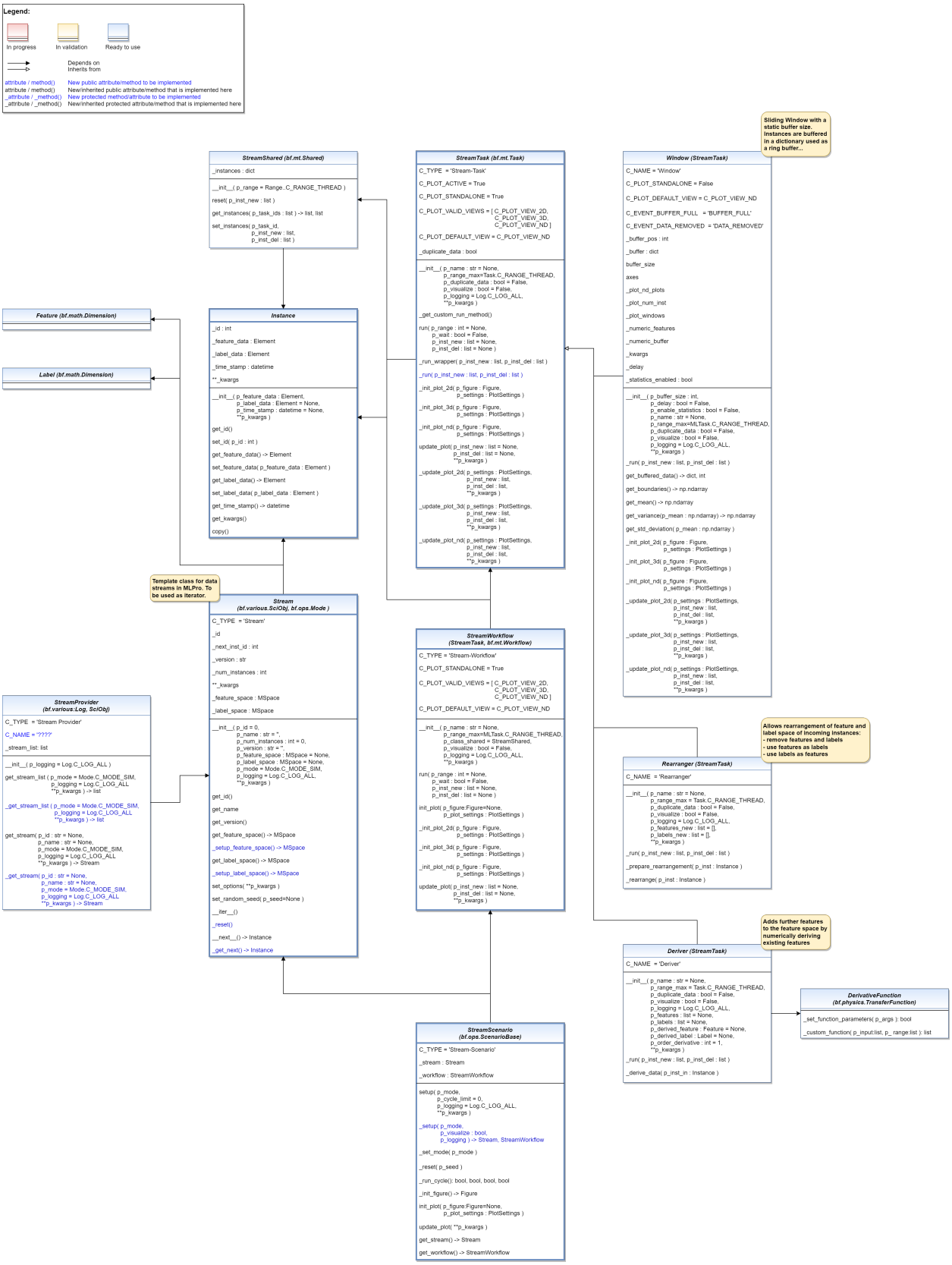
Method to update the normalization parameters for Z transformer

Parameters

- **p_dataset** (*numpy array*) – Dataset related to the elements to be normalized. Using this parameter will reset the normalization parameters based on the dataset provided.
- **p_data_new** (Element *or numpy array*) – New element to update the normalization parameters. Using this parameter will set/update the normalization parameters based on the data provided.
- **p_data_del** (Element *or Numpy array*) – Old element that is replaced with the new element.

Layer 3 - Application Support

BF-STREAMS - Stream Processing



This module provides classes for standardized stream processing.

class mlpro.bf.streams.models.**Feature**(*p_name_short*, *p_base_set='R'*, *p_name_long=''*, *p_name_latex=''*, *p_unit=''*, *p_unit_latex=''*, *p_boundaries: list = []*, *p_description=''*, *p_symmetrical: bool = False*, *p_logging=False*, ***p_kwargs*)

Bases: *Dimension*

class mlpro.bf.streams.models.**Label**(*p_name_short*, *p_base_set='R'*, *p_name_long=''*, *p_name_latex=''*, *p_unit=''*, *p_unit_latex=''*, *p_boundaries: list = []*, *p_description=''*, *p_symmetrical: bool = False*, *p_logging=False*, ***p_kwargs*)

Bases: *Dimension*

class mlpro.bf.streams.models.**Instance**(*p_feature_data:* Element, *p_label_data:* Element | *None = None*, *p_time_stamp: datetime | None = None*, ***p_kwargs*)

Bases: `object`

Instance class to store the current instance and the corresponding labels of the stream

Parameters

- **p_feature_data** (Element) – Feature data of the instance.
- **p_label_data** (Element) – Optional label data of the instance.
- **p_time_stamp** (*datetime*) – Optional time stamp of the instance.
- **p_kwargs** (*dict*) – Further optional named parameters.

C_TYPE = 'Instance'

get_id()

set_id(*p_id: int*)

get_feature_data() → *Element*

set_feature_data(*p_feature_data:* Element)

get_label_data() → *Element*

set_label_data(*p_label_data:* Element)

get_time_stamp()

get_kwargs()

copy()

class mlpro.bf.streams.models.**StreamShared**(*p_range: int = 2*)

Bases: *Shared*

Template class for shared objects in the context of stream processing.

_inst_new

List of new instances of a process cycle. At the beginning of a cycle it contains the incoming instance of a stream. The list evolves due to the manipulations of the stream tasks.

Type
list

_inst_del

List of instances to be removed. At the beginning of a cycle it is empty. The list evolves due to the manipulations of the stream tasks.

Type
list

reset(*p_inst_new: list*)

Resets the shared object and prepares the processing of the given set of new instances.

Parameters
p_inst_new (*list*) – List of new instances to be processed.

get_instances(*p_task_ids: list*)

Provides the result instances of all given task ids.

Parameters
p_task_ids (*list*) – List of task ids.

Returns

- **inst_new** (*list*) – List of new instances of all given task ids.
- **inst_del** (*list*) – List of instances to be deleted of all given task ids.

set_instances(*p_task_id*, *p_inst_new: list*, *p_inst_del: list*)

Stores result instances of a task in the shared object.

Parameters

- **p_task_id** – Id of related task.
- **p_inst_new** (*list*) – List of new instances.
- **p_inst_del** (*list*) – List of instances to be deleted.

_range: int

class mlpro.bf.streams.models.**Stream**(*p_id=0*, *p_name: str = ''*, *p_num_instances: int = 0*, *p_version: str = ''*, *p_feature_space:* MSpace | *None = None*, *p_label_space:* MSpace | *None = None*, *p_mode=0*, *p_logging=True*, ***p_kwargs*)

Bases: *Mode*, *LoadSave*, *ScientificObject*

Template class for data streams. Objects of this type can be used as iterators.

Parameters

- **p_id** – Optional id of the stream. Default = 0.
- **p_name** (*str*) – Optional name of the stream. Default = ''.
- **p_num_instances** (*int*) – Optional number of instances in the stream. Default = 0.
- **p_version** (*str*) – Optional version of the stream. Default = ''.
- **p_feature_space** (MSpace) – Optional feature space. Default = None.
- **p_label_space** (MSpace) – Optional label space. Default = None.
- **p_mode** – Operation mode. Default: Mode.C_MODE_SIM.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.
- **p_kwargs** (*dict*) – Further stream specific parameters.

C_TYPE = 'Stream'

get_id()

Returns the id of the stream.

get_name() → str

Returns the name of the stream.

Returns
stream_name – Name of the stream.

Return type
str

get_url() → str

Returns the URL of the scientific source/reference.

Returns
url – URL of the scientific source/reference.

Return type
str

get_num_instances() → int

Returns the number of instances of the stream.

Returns
num_inst – Number of instances of the stream. If 0 the number is unknown.

Return type
int

get_feature_space() → *MSpace*

Returns the feature space of the stream.

Returns
feature_space – Feature space of the stream.

Return type
MSpace

_setup_feature_space() → *MSpace*

Custom method to set up the feature space of the stream. It is called by method get_feature_space().

Returns
feature_space – Feature space of the stream.

Return type
MSpace

get_label_space() → *MSpace*

Returns the label space of the stream.

Returns
label_space – Label space of the stream.

Return type
MSpace

_setup_label_space() → *MSpace*

Custom method to set up the label space of the stream. It is called by method get_label_space().

Returns
label_space – Label space of the stream.

Return type
MSpace

`set_options`(***p_kwargs*)

Method to set specific options for the stream. The possible options depend on the stream provider and stream itself.

`set_random_seed`(*p_seed=None*)

Resets the internal random generator using the given seed.

`_reset`()

Custom reset method for data stream. See method __iter__() for more details.

`_get_next`() → *Instance*

Custom method to determine the next data stream instance. At the end of the stream exception StopIteration is to be raised. See method __next__() for more details.

Returns
instance – Next instance of data stream or None.

Return type
Instance

`class` mlpro.bf.streams.models.`StreamProvider`(*p_logging=True*)

Bases: *Log*, *ScientificObject*

Template class for stream providers.

Parameters
p_logging – Log level (see constants of class Log). Default: Log.C_LOG_ALL

`C_TYPE = 'Stream Provider'`

`get_stream_list`(*p_mode=0*, *p_logging=True*, ***p_kwargs*) → list

Gets a list of provided streams by calling custom method _get_stream_list().

Parameters

- **p_mode** – Operation mode. Default: Mode.C_MODE_SIM.
- **p_logging** – Log level of stream objects (see constants of class Log). Default: Log.C_LOG_ALL.
- **p_kwargs** (*dict*) – Further stream specific parameters.

Returns
stream_list – List of provided streams.

Return type
list

`_get_stream_list`(*p_mode=0*, *p_logging=True*, ***p_kwargs*) → list

Custom method to get the list of provided streams. See method get_stream_list() for further details.

Parameters

- **p_mode** – Operation mode. Default: Mode.C_MODE_SIM.
- **p_logging** – Log level of stream objects (see constants of class Log). Default: Log.C_LOG_ALL.

- **p_kwargs** (*dict*) – Further stream specific parameters.

Returns
stream_list – List of provided streams.

Return type
list

get_stream(*p_id: str | None = None*, *p_name: str | None = None*, *p_mode=0*, *p_logging=True*, ***p_kwargs*) → *Stream*

Returns stream with the specified id by calling custom method _get_stream().

Parameters

- **p_id** (*str*) – Optional Id of the requested stream. Default = None.
- **p_name** (*str*) – Optional name of the requested stream. Default = None.
- **p_mode** – Operation mode. Default: Mode.C_MODE_SIM.
- **p_logging** – Log level of stream object (see constants of class Log). Default: Log.C_LOG_ALL.
- **p_kwargs** (*dict*) – Further stream specific parameters.

Returns
s – Stream object or None in case of an error.

Return type
Stream

_get_stream(*p_id: str | None = None*, *p_name: str | None = None*, *p_mode=0*, *p_logging=True*, ***p_kwargs*) → *Stream*

Custom method to get the specified stream. See method get_stream() for further details.

Parameters

- **p_id** (*str*) – Optional Id of the requested stream. Default = None.
- **p_name** (*str*) – Optional name of the requested stream. Default = None.
- **p_mode** – Operation mode. Default: Mode.C_MODE_SIM.
- **p_logging** – Log level of stream object (see constants of class Log). Default: Log.C_LOG_ALL.
- **p_kwargs** (*dict*) – Further stream specific parameters.

Returns
s – Stream object or None in case of an error.

Return type
Stream

class mlpro.bf.streams.models.**StreamTask**(*p_name: str | None = None*, *p_range_max=1*, *p_duplicate_data: bool = False*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *Task*

Template class for stream-based tasks.

Parameters

- **p_name** (*str*) – Optional name of the task. Default is None.

- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_duplicate_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*dict*) – Further optional named parameters.

C_TYPE = 'Stream-Task'

C_PLOT_ACTIVE: bool = True

C_PLOT_STANDALONE: bool = True

C_PLOT_VALID_VIEWS: list = ['2D', '3D', 'ND']

C_PLOT_DEFAULT_VIEW: str = 'ND'

C_PLOT_ND_XLABEL_INST = 'Instance index'

C_PLOT_ND_XLABEL_TIME = 'Time index'

C_PLOT_ND_YLABEL = 'Feature Data'

_get_custom_run_method()

run(*p_range: int | None = None*, *p_wait: bool = False*, *p_inst_new: list | None = None*, *p_inst_del: list | None = None*)

Executes the specific actions of the task implemented in custom method _run(). At the end event C_EVENT_FINISHED is raised to start subsequent actions.

Parameters

- **p_range** (*int*) – Optional deviating range of asynchonicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Oterwise the minimum range of both is taken.
- **p_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p_inst_new** (*list*) – Optional list of new stream instances to be processed. If None, the list of the shared object is used instead. Default = None.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed. If None, the list of the shared object is used instead. Default = None.

_run_wrapper(*p_inst_new: list*, *p_inst_del: list*)

Internal use.

_run(*p_inst_new: list*, *p_inst_del: list*)

Custom method that is called by method run().

Parameters

- **p_inst_new** (*set*) – Set of new stream instances to be processed.
- **p_inst_del** (*set*) – Set of obsolete stream instances to be removed.

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

_init_plot_2d(*p_figure: Figure*, *p_settings:* PlotSettings)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

_init_plot_3d(*p_figure: Figure*, *p_settings:* PlotSettings)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

_init_plot_nd(*p_figure: Figure*, *p_settings:* PlotSettings)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

_finalize_plot_view(*p_inst_ref:* Instance)

update_plot(*p_inst_new: list | None = None*, *p_inst_del: list | None = None*, ***p_kwargs*)

Specialized definition of method update_plot() of class mlpro.bf.plot.Plottable.

Parameters

- **p_inst_new** (*list*) – List of new stream instances to be plotted.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed.
- **p_kwargs** (*dict*) – Further optional plot parameters.

_update_plot_2d(*p_settings:* PlotSettings, *p_inst_new: list*, *p_inst_del: list*, ***p_kwargs*)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

Parameters

- **p_settings** (PlotSettings) – Object with further plot settings.
- **p_inst_new** (*list*) – List of new stream instances to be plotted.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed.
- **p_kwargs** (*dict*) – Further optional plot parameters.

_update_plot_3d(*p_settings:* PlotSettings, *p_inst_new: list*, *p_inst_del: list*, ***p_kwargs*)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

Parameters

- **p_settings** (PlotSettings) – Object with further plot settings.
- **p_inst_new** (*list*) – List of new stream instances to be plotted.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed.
- **p_kwargs** (*dict*) – Further optional plot parameters.

_update_plot_nd(*p_settings:* PlotSettings, *p_inst_new: list*, *p_inst_del: list*, ***p_kwargs*)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

Parameters

- **p_settings** (PlotSettings) – Object with further plot settings.

- **p_inst_new** (*list*) – List of new stream instances to be plotted.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed.
- **p_kwargs** (*dict*) – Further optional plot parameters.

_range: int

_so: *Shared*

_plot_settings: *PlotSettings*

class mlpro.bf.streams.models.**StreamWorkflow**(*p_name: str | None = None*, *p_range_max=1*, *p_class_shared=<class 'mlpro.bf.streams.models.StreamShared'>*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *StreamTask*, *Workflow*

Workflow for stream processing. See class bf.mt.Workflow for further details.

Parameters

- **p_name** (*str*) – Optional name of the task. Default is None.
- **p_range_max** (*int*) – Range of asynchonicity. See class Range. Default is Range.C_RANGE_THREAD.
- **p_class_shared** – Optional class for a shared object (class StreamShared or a child class of StreamShared). Default = StreamShared
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*dict*) – Further optional named parameters handed over to every task within.

C_TYPE = 'Stream-Workflow'

C_PLOT_ACTIVE: bool = True

run(*p_range: int | None = None*, *p_wait: bool = False*, *p_inst_new: list | None = None*, *p_inst_del: list | None = None*)

Runs all stream tasks according to their predecessor relations.

Parameters

- **p_range** (*int*) – Optional deviating range of asynchonicity. See class Range. Default is None what means that the maximum range defined during instantiation is taken. Oterwise the minimum range of both is taken.
- **p_wait** (*bool*) – If True, the method waits until all (a)synchronous tasks are finished.
- **p_inst_new** (*list*) – Optional list of new stream instances to be processed. If None, the list of the shared object is used instead. Default = None.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed. If None, the list of the shared object is used instead. Default = None.

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot of a workflow. The method creates a host figure for all tasks if no external host figure is parameterized. The sub-plots of the tasks are autmatically arranged within the host figure.

See method init_plot() of class mlpro.bf.plot.Plottable for further details.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

_init_plot_2d(*p_figure: Figure*, *p_settings:* PlotSettings)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

_init_plot_3d(*p_figure: Figure*, *p_settings:* PlotSettings)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

_init_plot_nd(*p_figure: Figure*, *p_settings:* PlotSettings)

Default implementation for stream tasks. See class mlpro.bf.plot.Plottable for more details.

update_plot(*p_inst_new: list | None = None*, *p_inst_del: list | None = None*, ***p_kwargs*)

Specialized definition of method update_plot() of class mlpro.bf.plot.Plottable.

Parameters

- **p_inst_new** (*list*) – List of new stream instances to be plotted.
- **p_inst_del** (*list*) – List of obsolete stream instances to be removed.
- **p_kwargs** (*dict*) – Further optional plot parameters.

_range: int

_so: *Shared*

_plot_settings: *PlotSettings*

class mlpro.bf.streams.models.**StreamScenario**(*p_mode*, *p_cycle_limit=0*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *ScenarioBase*

Template class for stream based scenarios.

Parameters

- **p_mode** – Operation mode. See Mode.C_VALID_MODES for valid values. Default = Mode.C_MODE_SIM.
- **p_cycle_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.

C_TYPE = 'Stream-Scenario'

C_PLOT_ACTIVE: bool = True

setup()

Specialized method to set up a stream scenario. It is automatically called by the constructor and calls in turn the custom method _setup().

_setup(*p_mode*, *p_visualize: bool*, *p_logging*)

Custom method to set up a stream scenario consisting of a stream and a processing stream workflow.

Parameters

- **p_mode** – Operation mode. See Mode.C_VALID_MODES for valid values. Default = Mode.C_MODE_SIM.
- **p_visualize** (*bool*) – Boolean switch for visualisation.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.

Returns

- **stream** (*Stream*) – A stream object.
- **workflow** (*StreamWorkflow*) – A stream workflow object.

_set_mode(*p_mode*)

Custom method to set the operation mode of components of the scenario. See method set_mode() for further details.

Parameter

p_mode
Operation mode. See class bf.ops.Mode for further details.

_reset(*p_seed*)

Custom method to reset the components of the scenario and to set the given random seed value. See method reset() for further details.

Parameters
p_seed (*int*) – Seed value for internal random generator

get_latency() → timedelta

Returns the latency of the scenario. To be implemented in child class.

_run_cycle()

Gets next instance from the stream and lets process it by the stream workflow.

Returns

- **success** (*bool*) – True on success. False otherwise.
- **error** (*bool*) – True on error. False otherwise.
- **adapted** (*bool*) – True, if something within the scenario has adapted something in this cycle. False otherwise.
- **end_of_data** (*bool*) – True, if the end of the related data source has been reached. False otherwise.

_init_figure() → Figure

Custom method to initialize a suitable standalone Matplotlib figure.

Returns
figure – Matplotlib figure object to host the subplot(s)

Return type
Matplotlib.figure.Figure

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(***p_kwargs*)

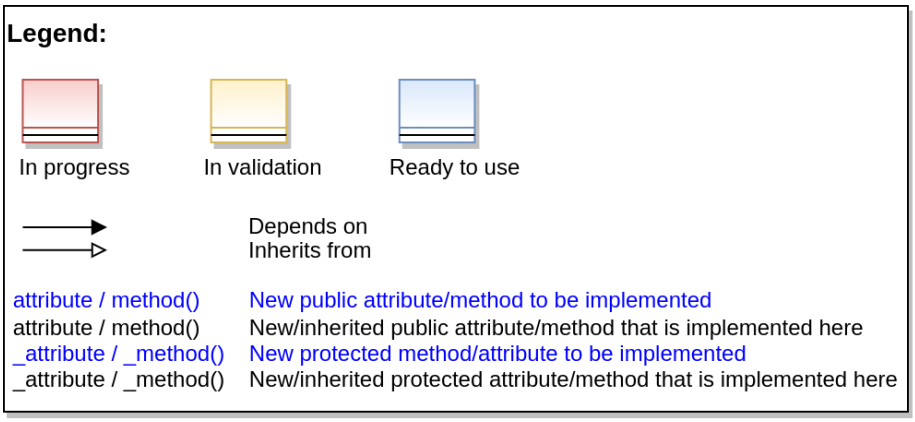
Plot updates take place during workflow/task processing and are disabled here…

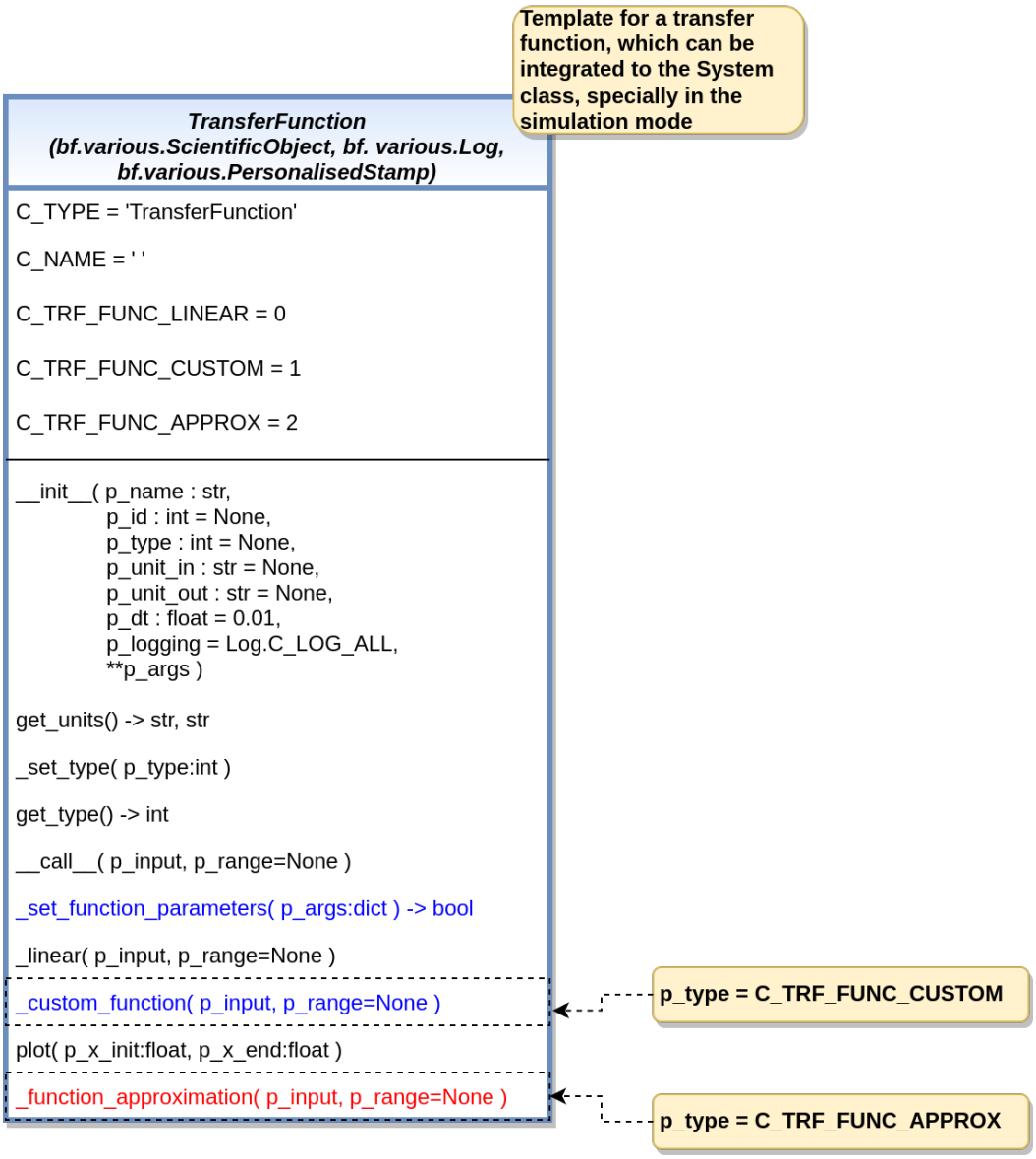
get_stream() → *Stream*

_plot_settings: *PlotSettings*

get_workflow() → *StreamWorkflow*

BF-PHYSICS - Physics





Ver. 1.0.0 (2023-02-04)

This module provides models and templates for physics.

class mlpro.bf.physics.basics.**TransferFunction**(*p_name: str*, *p_id: int | None = None*, *p_type: int | None = None*, *p_unit_in: str | None = None*, *p_unit_out: str | None = None*, *p_dt: float = 0.01*, *p_logging=True*, ***p_args*)

Bases: *ScientificObject*, *Log*, *PersonalisedStamp*

This class serves as a base class of transfer functions, which provides the main attributes of a transfer function. By default, there are several ready-to-use transfer function types available. If none of them suits to your transfer function, then you can also select a 'custom' type of transfer function and design your own function. Another possibility is to use a function approximation functionality provided by MLPro (coming soon).

Parameters

- **p_name** (*str*) – name of the transfer function.
- **p_id** (*int*) – unique id of the transfer function. Default: None.
- **p_type** (*int*) – type of the transfer function. Default: None.
- **p_unit_in** (*str*) – unit of the transfer function's input. Default: None.
- **p_unit_out** (*str*) – unit of the transfer function's output. Default: None.
- **p_dt** (*float*) – delta time. Default: 0.01.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.
- **p_args** (*dict*) – extra parameter for each specific transfer function.

C_TYPE

type of the base class. Default: 'TransferFunction'.

Type
str

C_NAME

name of the transfer function. Default: ''.

Type
str

C_TRF_FUNC_LINEAR

linear function. Default: 0.

Type
int

C_TRF_FUNC_CUSTOM

custom transfer function. Default: 1.

Type
int

C_TRF_FUNC_APPROX

function approximation. Default: 2.

Type
int

C_TYPE = 'TransferFunction'

C_TRF_FUNC_LINEAR = 0

C_TRF_FUNC_CUSTOM = 1

C_TRF_FUNC_APPROX = 2

C_NAME = ''

get_units()

This method provides a functionality to get the SI units of the input and output data.

Returns

- **self._unit_in** (*str*) – the SI unit of the input data.
- **self._unit_out** (*str*) – the SI unit of the output data.

_set_type(*p_type: int*)

This method provides a functionality to set the type of the transfer function.

Parameters
p_type (*int*) – the type of the transfer function.

get_type() → int

This method provides a functionality to get the type of the transfer function.

Returns
the type of the transfer function.

Return type
int

_set_function_parameters(*p_args: dict*) → bool

This method provides a functionality to set the parameters of the transfer function.

Parameters
p_args (*dict*) – set of parameters of the transfer function.

Returns
true means no parameters are missing.

Return type
bool

_linear(*p_input: float*, *p_range=None*) → float

This method provides a functionality for linear transfer function.

Formula –> y = mx+b y = output m = slope x = input b = y-intercept

Parameters

- **p_input** (*float*) – input value.
- **p_range** – range of the calculation. None means 0. Default: None.

Returns
output value.

Return type
float

_custom_function(*p_input*, *p_range=None*)

This function represents the template to create a custom function and must be redefined.

Parameters

- **p_input** – input value.
- **p_range** – range of the calculation. None means 0. Default: None.

Returns
output value.

Return type
float

plot(*p_x_init: float*, *p_x_end: float*)

This methods provides functionality to plot the defined function within a range.

Parameters

- **p_x_init** (*float*) – The initial value of the input (x-axis).
- **p_x_end** (*float*) – The end value of the input (x-axis).

_function_approximation(*p_input*, *p_range=None*)

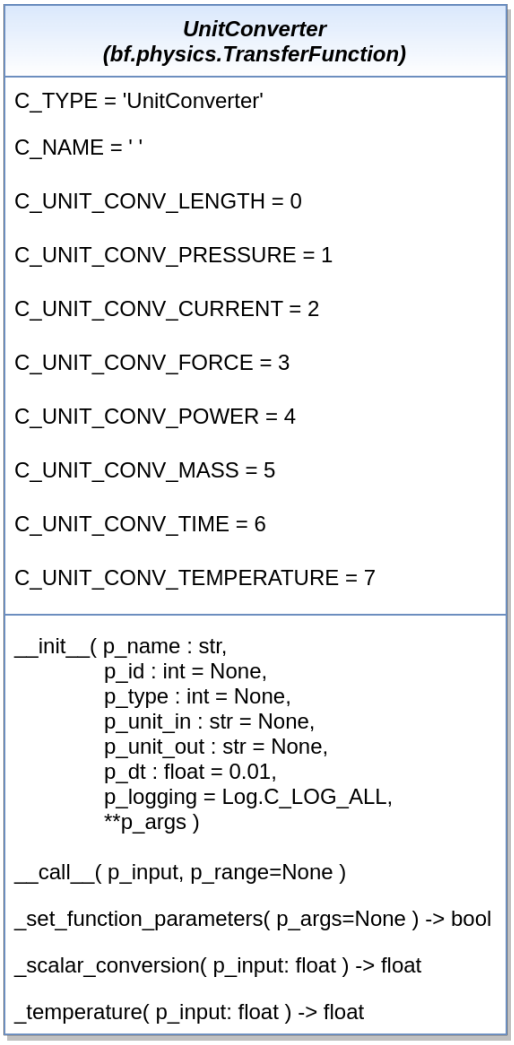
The function approximation is not yet ready (coming soon).

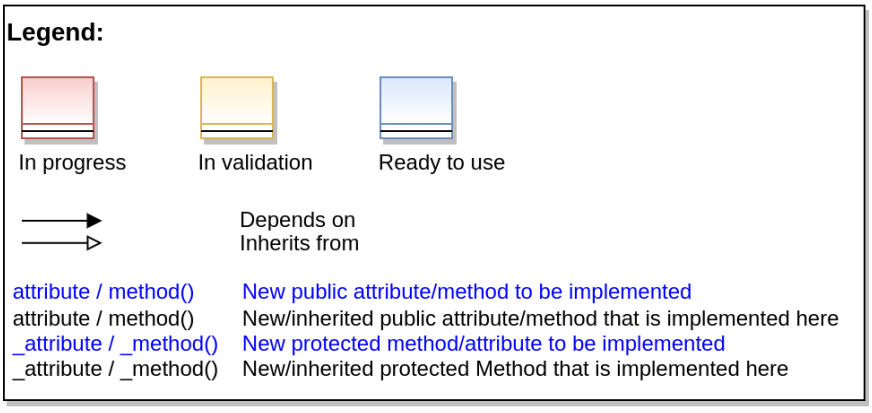
Parameters
p_input (*TYPE*) – DESCRIPTION.

Returns
DESCRIPTION.

Return type
bool

BF-PHYSICS-UNITCONVERTER - Unit Converters





This module provides models for unit conversions.

class mlpro.bf.physics.unitconverter.**UnitConverter**(*p_name: str*, *p_id: int | None = None*, *p_type: int | None = None*, *p_unit_in: str | None = None*, *p_unit_out: str | None = None*, *p_logging=True*, ***p_args*)

Bases: *TransferFunction*

This class serves as a base class of unit converters, which inherits the main attributes from a transfer function. By default, there are several ready-to-use unit converters available, such as, Length, Temperature, Pressure, Electric Current, Force, Power, Mass, and Time.

Parameters

- **p_name** (*str*) – name of the unit converter.
- **p_id** (*int*) – unique id of the unit converter. Default: None.
- **p_type** (*int*) – type of the unit converter. Default: None.
- **p_unit_in** (*str*) – unit of the unit converter's input. Default: None.
- **p_unit_out** (*str*) – unit of the unit converter's output. Default: None.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.

C_TYPE

type of the base class. Default: 'UnitConverter'.

Type
str

C_NAME

name of the unit converter. Default: ''.

Type
str

C_UNIT_CONV_LENGTH

unit converter for length. Default: 0.

Type
int

C_UNIT_CONV_PRESSURE

unit converter for pressure. Default: 1.

Type
int

C_UNIT_CONV_CURRENT

unit converter for electric current. Default: 2.

Type
int

C_UNIT_CONV_FORCE

unit converter for force. Default: 3.

Type
int

C_UNIT_CONV_POWER

unit converter for power. Default: 4.

Type
int

C_UNIT_CONV_MASS

unit converter for mass. Default: 5.

Type
int

C_UNIT_CONV_TIME

unit converter for time. Default: 6.

Type
int

C_UNIT_CONV_TEMPERATURE

unit converter for temperature. Default: 7.

Type
int

C_TYPE = 'UnitConverter'

C_NAME = ''

C_UNIT_CONV_LENGTH = 0

C_UNIT_CONV_PRESSURE = 1

C_UNIT_CONV_CURRENT = 2

C_UNIT_CONV_FORCE = 3

C_UNIT_CONV_POWER = 4

C_UNIT_CONV_MASS = 5

C_UNIT_CONV_TIME = 6

C_UNIT_CONV_TEMPERATURE = 7

_set_function_parameters(*p_args=None*) → bool

This method provides a functionality to set the parameters of the unit converter.

Parameters
p_args – not necessary for a unit converter. Default: None.

Returns
true means initialization is successful.

Return type
bool

_scalar_conversion(*p_input: float*) → float

This method provides a scalar conversion functionality.

Parameters
p_input (*float*) – input value.

Returns
output value.

Return type
float

`_temperature`(*p_input: float*) → float

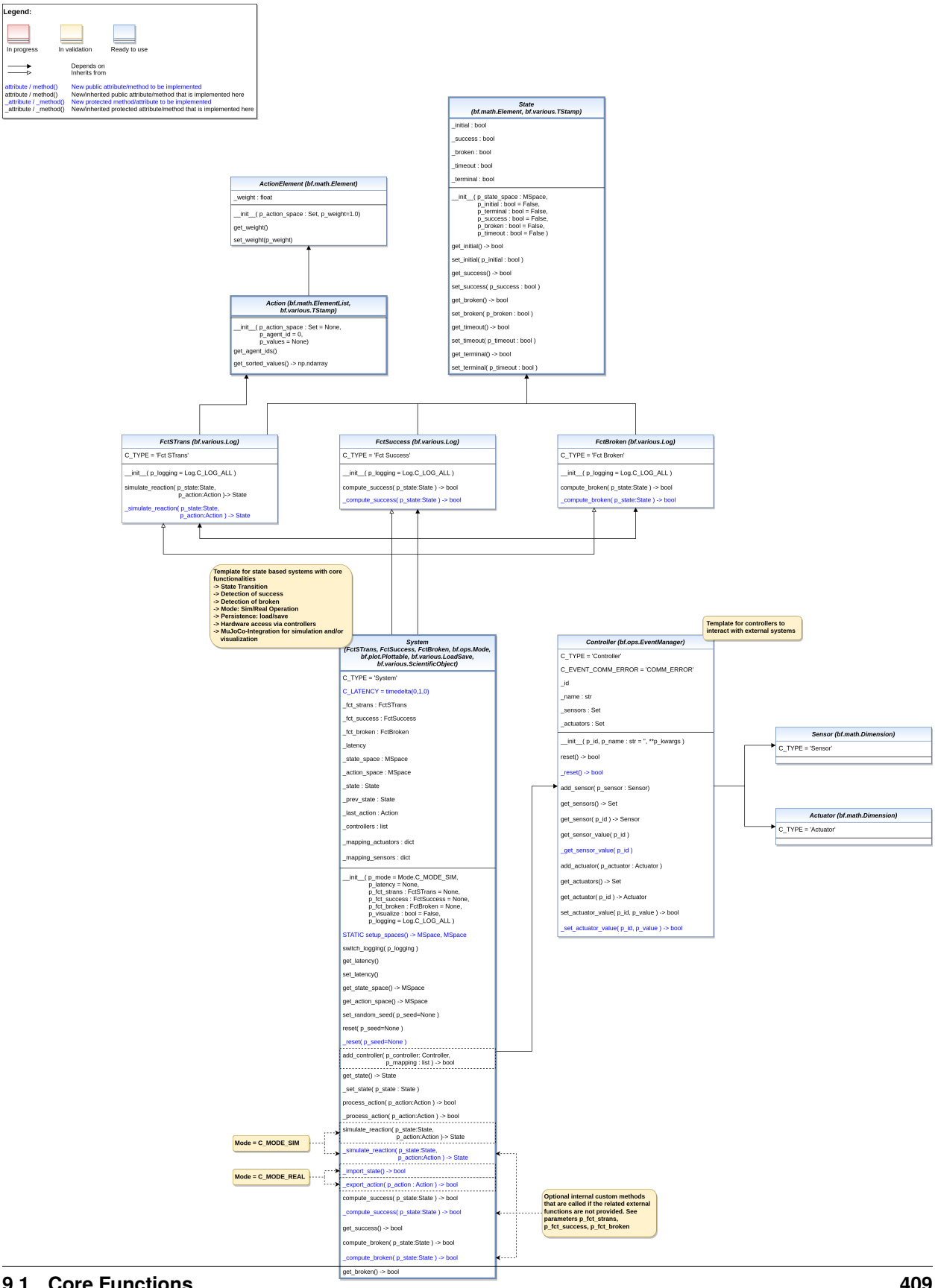
This method provides a temperature conversion functionality.

Parameters
p_input (*float*) – input value.

Returns
output value.

Return type
float

BF-SYSTEMS - State-based Systems



This module provides models and templates for state based systems.

class mlpro.bf.systems.basics.**State**(*p_state_space:* MSpace, *p_initial: bool = False*, *p_terminal: bool = False*, *p_success: bool = False*, *p_broken: bool = False*, *p_timeout: bool = False*)

Bases: *Element*, *TStamp*

State of a system as an element of a given state space. Additionally, the state can be labeled with various properties.

Parameters

- **p_state_space** (MSpace) – State space of the related system.
- **p_initial** (*bool*) – This optional flag signals that the state is the first one after a reset. Default=False.
- **p_terminal** (*bool*) – This optional flag labels the state as a terminal state. Default=False.
- **p_success** (*bool*) – This optional flag labels the state as an objective state. Default=False.
- **p_broken** (*bool*) – This optional flag labels the state as a final error state. Default=False.
- **p_timeout** (*bool*) – This optional flag signals that the cycle limit of an episode has been reached. Default=False.

get_initial() → bool

set_initial(*p_initial: bool*)

get_success() → bool

set_success(*p_success: bool*)

get_broken() → bool

set_broken(*p_broken: bool*)

get_timeout() → bool

set_timeout(*p_timeout: bool*)

get_terminal() → bool

set_terminal(*p_terminal: bool*)

class mlpro.bf.systems.basics.**ActionElement**(*p_action_space:* Set, *p_weight: float = 1.0*)

Bases: *Element*

Single entry of an action. See class Action for further details.

Parameters

- **p_action_space** (Set) – Related action space.
- **p_weight** (*float*) – Weight of action element. Default = 1.0.

get_weight()

set_weight(*p_weight*)

class mlpro.bf.systems.basics.**Action**(*p_agent_id=0*, *p_action_space:* Set | *None* = *None*, *p_values:* *ndarray* | *None* = *None*)

Bases: *ElementList*, *TStamp*

Objects of this class represent actions of (multi-)agents. Every element of the internal list is related to an agent, and its partial subsection. Action values for the first agent can be added while object instantiation. Action values of further agents can be added by using method self.add_elem().

Parameters

- **p_agent_id** – Unique id of (first) agent to be added
- **p_action_space** (Set) – Action space of (first) agent to be added
- **p_values** (*np.ndarray*) – Action values of (first) agent to be added

get_agent_ids()

get_sorted_values() → ndarray

class mlpro.bf.systems.basics.**FctSTrans**(*p_logging=True*)

Bases: *Log*

Template class for state transition functions.

Parameters
p_logging – Log level (see class Log for more details). Default = Log.C_LOG_ALL.

C_TYPE = 'Fct STrans'

simulate_reaction(*p_state:* State, *p_action:* Action) → *State*

Simulates a state transition based on a state and action. Custom method _simulate_reaction() is called.

Parameters

- **p_state** (State) – System state.
- **p_action** (Action) – Action to be processed.

Returns
new_state – Result state after state transition.

Return type
State

_simulate_reaction(*p_state:* State, *p_action:* Action) → *State*

Custom method for a simulated state transition. See method simulate_reaction() for further details.

class mlpro.bf.systems.basics.**FctSuccess**(*p_logging=True*)

Bases: *Log*

Template class for functions that determine whether or not a state is a success state.

Parameters
p_logging – Log level (see class Log for more details). Default = Log.C_LOG_ALL.

C_TYPE = 'Fct Success'

compute_success(*p_state:* State) → bool

Assesses the given state regarding success criteria. Custom method _compute_success() is called.

Parameters
p_state (State) – System state.

Returns
success – True, if given state is a success state. False otherwise.

Return type
bool

`_compute_success`(*p_state:* State) → bool

Custom method for assessment for success. See method compute_success() for further details.

class mlpro.bf.systems.basics.**FctBroken**(*p_logging=True*)

Bases: *Log*

Template class for functions that determine whether or not a state is a broken state.

Parameters
p_logging – Log level (see class Log for more details). Default = Log.C_LOG_ALL.

`C_TYPE = 'Fct Broken'`

`compute_broken`(*p_state:* State) → bool

Assesses the given state regarding breakdown criteria. Custom method _compute_success() is called.

Parameters
p_state (State) – System state.

Returns
broken – True, if given state is a breakdown state. False otherwise.

Return type
bool

`_compute_broken`(*p_state:* State) → bool

Custom method for assessment for breakdown. See method compute_broken() for further details.

class mlpro.bf.systems.basics.**Sensor**(*p_name_short*, *p_base_set='R'*, *p_name_long=''*, *p_name_latex=''*, *p_unit=''*, *p_unit_latex=''*, *p_boundaries: list = []*, *p_description=''*, *p_symmetrical: bool = False*, *p_logging=False*, ***p_kwargs*)

Bases: *Dimension*

Template for a sensor.

`C_TYPE = 'Sensor'`

class mlpro.bf.systems.basics.**Actuator**(*p_name_short*, *p_base_set='R'*, *p_name_long=''*, *p_name_latex=''*, *p_unit=''*, *p_unit_latex=''*, *p_boundaries: list = []*, *p_description=''*, *p_symmetrical: bool = False*, *p_logging=False*, ***p_kwargs*)

Bases: *Dimension*

Template for an actuator.

`C_TYPE = 'Actuator'`

class mlpro.bf.systems.basics.**Controller**(*p_id*, *p_name: str = ''*, *p_logging: bool = True*, ***p_kwargs*)

Bases: *EventManager*

Template for a controller that enables access to sensors and actuators.

Parameters

- **p_id** – Unique id of the controller.

- **p_name** (*str*) – Optional name of the controller.
- **p_logging** – Log level (see class Log for more details). Default = Log.C_LOG_ALL.
- **p_kwargs** (*dict*) – Further keyword arguments specific to the controller.

C_EVENT_COMM_ERROR

Event that is raised on a communication error

C_TYPE = 'Controller'

C_EVENT_COMM_ERROR = 'COMM_ERROR'

reset() → bool

Resets the controller by calling custom method _reset().

Returns
result – True, if successful. False otherwise. Additionally event C_EVENT_COMM_ERROR is raised.

Return type
bool

_reset() → bool

Custom reset method.

Returns
result – True, if successful. False otherwise.

Return type
bool

add_sensor(*p_sensor:* Sensor)

Adds a sensor to the controller.

Parameters
p_sensor (Sensor) – Sensor object to be added.

get_sensors() → *Set*

Returns the internal set of sensors.

Returns
sensors – Set of sensors.

Return type
Set

get_sensor(*p_id*) → *Sensor*

Returns a sensor.

get_sensor_value(*p_id*)

Determines the value of a sensor by calling custom method _get_sensor_value().

Parameters
p_id – Id of the sensor.

Returns
Current value of the sensor or None on a communication error. In that case, event C_EVENT_COMM_ERROR is raised additionally.

Return type
value

_get_sensor_value(*p_id*)

Custom method to get a sensor value. See method get_sensor_value() for further details.

Parameters
p_id – Id of the sensor.

Returns
Current value of the sensor or None on a communication error.

Return type
value

add_actuator(*p_actuator:* Actuator)

Adds an actuator to the controller.

Parameters
p_actuator (Actuator) – Actuator object to be added.

get_actuators() → *Set*

Returns the internal set of actuators.

Returns
actuators – Set of actuators.

Return type
Set

get_actuator(*p_id*) → *Actuator*

Returns an actuator.

set_actuator_value(*p_id*, *p_value*) → bool

Sets the value of an actuator by calling custom method _set_actuatur_value().

Parameters

- **p_id** – Id of the actuator.
- **p_value** – New actuator value.

Returns
successful – True, if successful. False otherwise. In that case, event C_EVENT_COMM_ERROR is raised additionally.

Return type
bool

_set_actuator_value(*p_id*, *p_value*) → bool

Custom method to set an actuator value. See method set_sensor_value() for further details.

Parameters

- **p_id** – Id of the actuator.
- **p_value** – New actuator value.

Returns
successful – True, if successful. False otherwise.

Return type
bool

class mlpro.bf.systems.basics.**System**(*p_mode=0*, *p_latency: timedelta | None = None*, *p_fct_strans: FctSTrans | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_name=None*, *p_frame_skip: int = 1*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf: tuple = (None, None, None)*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *FctSTrans*, *FctSuccess*, *FctBroken*, *Mode*, *Plottable*, *LoadSave*, *ScientificObject*

Base class for state based systems.

Parameters

- **p_mode** – Mode of the system. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of the system. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_fct_strans** (FctSTrans) – Optional external function for state transition.
- **p_fct_success** (FctSuccess) – Optional external function for state evaluation 'success'.
- **p_fct_broken** (FctBroken) – Optional external function for state evaluation 'broken'.
- **p_mujoco_file** – Path to XML file for MuJoCo model.
- **p_frame_skip** (*int*) – Frame to be skipped every step. Default = 1.
- **p_state_mapping** – State mapping if the MLPro state and MuJoCo state have different naming.
- **p_action_mapping** – Action mapping if the MLPro action and MuJoCo action have different naming.
- **p_use_radian** (*bool*) – Use radian if the action and the state based on radian unit. Default = True.
- **p_camera_conf** (*tuple*) – Default camera configuration on MuJoCo Simulation (xyz position, elevation, distance).
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see class Log for more details). Default = Log.C_LOG_ALL.

_latency

Latency of the system.

Type
timedelta

_state

Current state of system.

Type
State

_prev_state

Previous state of system.

Type
State

_last_action

Last action.

Type
Action

_fct_strans

Internal state transition function.

Type
FctSTrans

_fct_success

Internal function for state evaluation 'success'.

Type
FctSuccess

_fct_broken

Internal function for state evaluation 'broken'.

Type
FctBroken

C_TYPE = 'System'

C_LATENCY = datetime.timedelta(seconds=1)

C_PLOT_ACTIVE: bool = True

static setup_spaces()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

static load(*p_path*, *p_filename*)

Loads content from the given path and file name. If file does not exist, it returns None.

Parameters

- **file** (*p_path Path that contains the*) –
- **name** (*p_filename File*) –

Returns
A loaded object, if file content was loaded successfully. None otherwise.

_save(*p_path*, *p_filename*) → bool

Custom method for saving an object. The default implementation is based on pickle/dill. Redefine on demand. See method save() for further details.

switch_logging(*p_logging*)

Sets new log level.

Parameters
p_logging – Log level (constant C_LOG_LEVELS contains valid values)

get_latency() → timedelta

Returns latency of the system.

set_latency(*p_latency: timedelta | None = None*) → None

Sets latency of the system. If p_latency is None latency will be reset to internal value of attribute C_LATENCY.

Parameters
p_latency (*timedelta*) – New latency value

get_state_space() → *MSpace*

get_action_space() → *MSpace*

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

Parameters
p_seed (*int*) – Seed parameter for an internal random generator

reset(*p_seed=None*) → None

Resets the system to an initial state. If MuJoCo is not used, the custom method _reset() is called.

Parameters
p_seed (*int*) – Seed parameter for an internal random generator

_reset(*p_seed=None*) → None

Custom method to reset the system to an initial/defined state. Use method _set_status() to set the state.

Parameters
p_seed (*int*) – Seed parameter for an internal random generator

add_controller(*p_controller:* Controller, *p_mapping: list*) → bool

Adds a controller and a related mapping of states and actions to sensors and actuators.

Parameters

- **p_controller** (Controller) – Controller object to be added.
- **p_mapping** (*list*) – A list of mapping tuples following the syntax ([Type = 'S' or 'A'], [Name of state/action] [Name of sensor/actuator])

Returns
successful – True, if controller and related mapping was added successfully. False otherwise.

Return type
bool

get_state() → *State*

Returns current state of the system.

_set_state(*p_state:* State)

Explicitly sets the current state of the system. Internal use only.

process_action(*p_action:* Action) → bool

Processes a state transition based on the current state and a given action. The state transition itself is implemented in child classes in the custom method _process_action().

Parameters
p_action (Action) – Action to be processed

Returns
success – True, if action processing was successfull. False otherwise.

Return type
bool

_process_action(*p_action:* Action) → bool

Internal custom method for state transition with default implementation. To be redefined in a child class on demand. See method process_action() for further details.

simulate_reaction(*p_state:* State | *None* = *None*, *p_action:* Action | *None* = *None*) → *State*

Simulates a state transition based on a state and an action. The simulation step itself is carried out either by an internal custom implementation in method _simulate_reaction() or by an embedded external function.

Parameters

- **p_state** (State) – Current state.
- **p_action** (Action) – Action.

Returns
Subsequent state after transition

Return type
State

_simulate_reaction(*p_state:* State, *p_action:* Action) → *State*

Custom method for a simulated state transition. Implement this method if no external state transition function is used. See method simulate_reaction() for further details.

action_to_mujoco(*p_mlpro_action*)

Action conversion method from converting MLPro action to MuJoCo action.

_action_to_mujoco(*p_mlpro_action*)

Custom method for to do transition between MuJoCo state and MLPro state. Implement this method if the MLPro state has different dimension from MuJoCo state.

Parameters
p_mujoco_state (*Numpy*) – MLPro action.

Returns
Modified MLPro action

Return type
Numpy

state_from_mujoco(*p_mujoco_state*)

State conversion method from converting MuJoCo state to MLPro state.

_state_from_mujoco(*p_mujoco_state*)

Custom method for to do transition between MuJoCo state and MLPro state. Implement this method if the MLPro state has different dimension from MuJoCo state.

Parameters
p_mujoco_state (*Numpy*) – MuJoCo state.

Returns
Modified MuJoCo state

Return type
Numpy

_import_state() → bool

_export_action(*p_action:* Action) → bool

compute_success(*p_state:* State) → bool

Assesses the given state whether it is a 'success' state. Assessment is carried out either by a custom implementation in method _compute_success() or by an embedded external function.

Parameters
p_state (State) – State to be assessed.

Returns
success – True, if the given state is a 'success' state. False otherwise.

Return type
bool

_compute_success(*p_state:* State) → bool

Custom method for assessment for success. Implement this method if no external function is used. See method compute_success() for further details.

get_success() → bool

compute_broken(*p_state:* State) → bool

Assesses the given state whether it is a 'broken' state. Assessment is carried out either by a custom implementation in method _compute_broken() or by an embedded external function.

Parameters
p_state (State) – State to be assessed.

Returns
broken – True, if the given state is a 'broken' state. False otherwise.

Return type
bool

_compute_broken(*p_state:* State) → bool

Custom method for assessment for breakdown. Implement this method if no external function is used. See method compute_broken() for further details.

get_broken() → bool

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

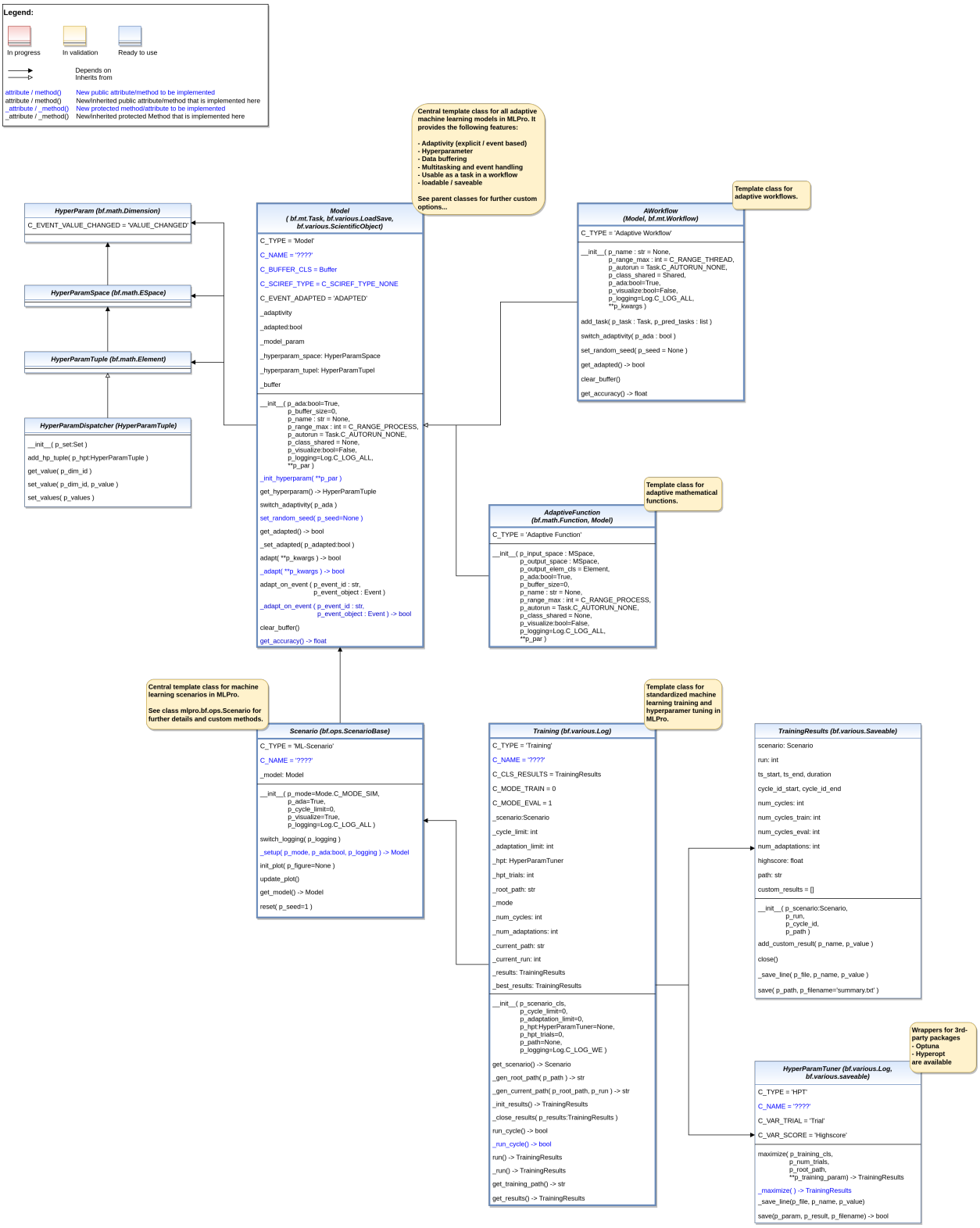
update_plot(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

Layer 4 - Machine Learning

BF-ML - Machine Learning



Ver. 2.1.3 (2023-03-10)

This module provides the fundamental templates and processes for machine learning in MLPro.

class mlpro.bf.ml.basics.**HyperParam**(*p_name_short*, *p_base_set='R'*, *p_name_long=''*, *p_name_latex=''*, *p_unit=''*, *p_unit_latex=''*, *p_boundaries: list = []*, *p_description=''*, *p_symmetrical: bool = False*, *p_logging=False*, ***p_kwargs*)

Bases: *Dimension*

Hyperparameter definition class. See class Dimension for further descriptions.

C_EVENT_VALUE_CHANGED = 'VALUE_CHANGED'

class mlpro.bf.ml.basics.**HyperParamSpace**

Bases: *ESpace*

Hyperparameter space, which is just an Euclidian space.

class mlpro.bf.ml.basics.**HyperParamTuple**(*p_set:* Set)

Bases: *Element*

Tuple of hyperparameters, which is an element of a hyperparameter space

set_value(*p_dim_id*, *p_value*)

class mlpro.bf.ml.basics.**HyperParamDispatcher**(*p_set:* Set)

Bases: *HyperParamTuple*

To dispatch multiple hp tuples into one tuple

add_hp_tuple(*p_hpt:* HyperParamTuple)

get_value(*p_dim_id*)

set_value(*p_dim_id*, *p_value*)

get_values()

set_values(*p_values*)

Overwrites the values of all components of the element.

Parameters

dimensions. (*p_values Something iterable with same length as number of element*) –

class mlpro.bf.ml.basics.**Model**(*p_ada: bool = True*, *p_buffer_size: int = 0*, *p_name: str | None = None*, *p_range_max: int = 2*, *p_autorun=0*, *p_class_shared=None*, *p_visualize: bool = False*, *p_logging=True*, ***p_par*)

Bases: *Task*, *LoadSave*, *ScientificObject*

Fundamental template class for adaptive ML models. Supports in particular

- Adaptivity (explicit and/or event based)
- Hyperparameter management
- Data buffering
- Multitasking
- Plotting
- Scientific referencing on source code level

Parameters

- **p_ada** (*bool*) – Boolean switch for adaptivitiy. Default = True.

- **p_buffer_size** (*int*) – Initial size of internal data buffer. Defaut = 0 (no buffering).
- **p_name** (*str*) – Optional name of the model. Default is None.
- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_autorun** (*int*) – On value C_AUTORUN_RUN method run() is called imediately during instantiation. On vaule C_AUTORUN_LOOP method run_loop() is called. Value C_AUTORUN_NONE (default) causes an object instantiation without starting further actions.
- **p_class_shared** – Optional class for a shared object (class Shared or a child class of it)
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_par** (*Dict*) – Further model specific hyperparameters (to be defined in child class).

C_TYPE = 'Model'

C_NAME = '????'

C_EVENT_ADAPTED = 'ADAPTED'

C_BUFFER_CLS

alias of *Buffer*

C_SCIREF_TYPE = None

_init_hyperparam(***p_par*)

Implementation specific hyperparameters can be added here. Please follow these steps: a) Add each hyperparameter as an object of type HyperParam to the internal hyperparameter

space object self._hyperparam_space

b) Create hyperparameter tuple and bind to self._hyperparam_tuple

c) Set default value for each hyperparameter

Parameters
p_par (*Dict*) – Further model specific hyperparameters, that are passed through constructor.

get_hyperparam() → *HyperParamTuple*

Returns the internal hyperparameter tuple to get access to single values.

switch_adaptivity(*p_ada: bool*)

Switches adaption functionality on/off.

Parameters
p_ada (*bool*) – Boolean switch for adaptivity

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

get_adapted() → bool

Returns True, if the model was adapted at least once. False otherwise.

_set_adapted(*p_adapted: bool*)

Sets the adapted flag. Internal use only.

adapt(***p_kwargs*) → bool

Adapts the model by calling the custom method _adapt().

Parameters
p_kwargs (*dict*) – All parameters that are needed for the adaption. Depends on the specific higher context.

Returns
adapted – True, if something has been adapted. False otherwise.

Return type
bool

_adapt(***p_kwargs*) → bool

Custom implementation of the adaptation algorithm. Please specify the parameters needed by your implementation. This method will be called by public method adapt() if adaptivity is switched on.

Parameters
p_kwargs (*dict*) – All parameters that are needed for the adaption. Please replace by concrete parameter definitions that meet the needs of your algorithm.

Returns
adapted – True, if something has been adapted. False otherwise.

Return type
bool

adapt_on_event(*p_event_id: str*, *p_event_object:* Event)

Method to be used as event handler for event-based adaptations. Calls custom method _adapt_on_event() and updates the internal adaptation state.

Parameters

- **p_event_id** (*str*) – Event id.
- **p_event_object** (Event) – Object with further context informations about the event.

_adapt_on_event(*p_event_id: str*, *p_event_object:* Event) → bool

Custom method to be used for event-based adaptation. See method adapt_on_event().

Parameters

- **p_event_id** (*str*) – Event id.
- **p_event_object** (Event) – Object with further context informations about the event.

Returns
adapted – True, if something was adapted. False otherwise.

Return type
bool

clear_buffer()

Clears internal buffer (if buffering is active).

get_accuracy() → float

Determines the accuracy of the model.

Returns
accuracy – Accuracy of the model as a scalar value in interval [0,1]

Return type
float

_range: int

_so: *Shared*

_plot_settings: *PlotSettings*

class mlpro.bf.ml.basics.**AWorkflow**(*p_name: str | None = None*, *p_range_max=1*, *p_class_shared=<class 'mlpro.bf.mt.Shared'>*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *Model*, *Workflow*

Adaptive workflow based on a workflow and an adaptive ml model.

Parameters

- **p_name** (*str*) – Optional name of the workflow. Default is None.
- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_class_shared** – Optional class for a shared object (class OAShared or a child class of OAShared)
- **p_ada** (*bool*) – Boolean switch for adaptivitiy. Default = True.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*dict*) – Further optional named parameters.

C_TYPE = 'Adaptive Workflow'

add_task(*p_task:* Task, *p_pred_tasks: list | None = None*)

Adds a task to the workflow.

Parameters

- **p_task** (Task) – Task object to be added.
- **p_pred_tasks** (*list*) – Optional list of predecessor task objects

switch_adaptivity(*p_ada: bool*)

Switches adaption functionality on/off.

Parameters
p_ada (*bool*) – Boolean switch for adaptivity

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

get_adapted() → bool

Returns True, if the model was adapted at least once. False otherwise.

clear_buffer()

Clears internal buffer (if buffering is active).

get_accuracy()

Determines the accuracy of the model.

Returns
accuracy – Accuracy of the model as a scalar value in interval [0,1]

Return type
float

_range: int

_so: *Shared*

_plot_settings: *PlotSettings*

class mlpro.bf.ml.basics.Scenario(*p_mode=0*, *p_ada: bool = True*, *p_cycle_limit: int = 0*, *p_auto_setup: bool = True*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *ScenarioBase*

Template class for a common ML scenario with an adaptive model inside. To be inherited and specialized in higher ML subtopic layers. See class bf.ops.ScenarioBase for further details and custom methods.

The following key features are included:

- Operation mode
- Cycle management
- Timer
- Latency
- Explicit handling of an adaptive ML model inside

Parameters

- **p_mode** – Operation mode. See bf.ops.Mode.C_VALID_MODES for valid values. Default = Mode.C_MODE_SIM.
- **p_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p_cycle_limit** (*int*) – Maximum number of cycles. Default = 0 (no limit).
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = True.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.

C_TYPE = 'ML-Scenario'

C_NAME = '????'

switch_logging(*p_logging*)

Sets new log level.

Parameters
p_logging – Log level (constant C_LOG_LEVELS contains valid values)

setup()

Custom method to set up all components of the scenario.

_setup(*p_mode*, *p_ada: bool*, *p_visualize: bool*, *p_logging*) → *Model*

Custom setup of ML scenario.

Parameters

- **p_mode** – Operation mode. See Mode.C_VALID_MODES for valid values. Default = Mode.C_MODE_SIM
- **p_ada** (*bool*) – Boolean switch for adaptivity.
- **p_visualize** (*bool*) – Boolean switch for visualisation.
- **p_logging** – Log level (see constants of class Log).

Returns
model – Adaptive model inside the ML scenario

Return type
Model

init_plot(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

get_model() → *Model*

Returns the adaptive model object inside the scenario.

reset(*p_seed=1*)

Resets the scenario and especially the ML model inside. Internal random generators are seed with the given value. Custom reset actions can be implemented in method _reset().

Parameters
p_seed (*int*) – Seed value for internal random generator

_plot_settings: *PlotSettings*

class mlpro.bf.ml.basics.**TrainingResults**(*p_scenario:* Scenario, *p_run*, *p_cycle_id*, *p_logging='W'*)

Bases: *Log*, *Saveable*

Results of a training (see class Training).

Parameters

- **p_scenario** (Scenario) – Related scenario.
- **p_run** (*int*) – Run id.
- **p_cycle_id** (*int*) – Id of first cycle of this run.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL

C_TYPE = 'Results '

add_custom_result(*p_name*, *p_value*)

close()

log_results()

_log_results()

_save_line(*p_file*, *p_name*, *p_value*)

save(*p_path*, *p_filename='summary.csv'*) → bool

Saves a training summary in the given path.

Parameters

- **p_path** (*str*) – Destination folder
- **p_filename** (*string*) – Name of summary file. Default = 'summary.csv'

Returns

success – True, if summary file was created successfully. False otherwise.

Return type

bool

class mlpro.bf.ml.basics.**HyperParamTuner**(*p_logging=True*)

Bases: *Log*, *Saveable*

Template class for hyperparameter tuning (HPT).

C_TYPE = 'HyperParam Tuner'

C_NAME = '????'

C_VAR_TRIAL = 'Trial'

C_VAR_SCORE = 'Highscore'

maximize(*p_training_cls*, *p_num_trials*, *p_root_path*, ***p_training_param*) → *TrainingResults*

...

Parameters

- **p_training_cls** – Training class to be instantiated/executed
- **p_num_trials** (*str*) – Number of trials
- **p_num_trials** – Root path of the training class
- **p_training_param** (*dictionary*) – Training parameters

Returns

results – Training results of the best tuned model (see class TrainingResults).

Return type

TrainingResults

_maximize() → *TrainingResults*

_save_line(*p_file*, *p_name*, *p_value*)

save(*p_param*, *p_result*, *p_filename='best_parameters.csv'*) → bool

Saves the best result of the hyperparameter tuning in the root path.

Parameters

- **p_param** (*dict*) – A dictionary that consists of list of best parameters
- **p_result** (*float*) – Highest score
- **p_filename** (*str*) – Name of summary file. Default = 'best_parameters.csv'

Returns

success – True, if summary file was created successfully. False otherwise.

Return type

bool

class mlpro.bf.ml.basics.**Training**(***p_kwargs*)

Bases: *Log*

Template class for a ML training and hyperparameter tuning.

Parameters

- **p_scenario_cls** – Name of ML scenario class, compatible to/inherited from class Scenario.
- **p_cycle_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.
- **p_adaptation_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p_hpt** (HyperParamTuner) – Optional hyperparameter tuner (see class HyperParamTuner). Default = None.
- **p_hpt_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0.
- **p_path** (*str*) – Optional destination path to store training data. Default = None.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_WE.

C_TYPE = 'Training'

C_NAME = '????'

C_CLS_RESULTS

alias of *TrainingResults*

C_MODE_TRAIN = 0

C_MODE_EVAL = 1

C_LOG_SEPARATOR =
'--'

get_scenario() → *Scenario*

_gen_root_path(*p_path*) → str

_gen_current_path(*p_root_path*, *p_run*) → str

_init_results() → *TrainingResults*

_close_results(*p_results:* TrainingResults)

run_cycle() → bool

Runs a single training cycle.

Returns
termination_event – True, if training run has finished. False otherwise.

Return type
bool

_run_cycle() → bool

Single custom trainig cycle to be redefined. Custom training results can be added to using self._results.add_custom_result(p_name, p_value).

Returns
True, if training has finished. False otherwise.

Return type
bool

run() → *TrainingResults*

Runs a training and returns the results of the best trained/tuned agent.

Returns
Object with training results.

Return type
TrainingResults

_run() → *TrainingResults*

get_results() → *TrainingResults*

get_training_path() → str

class mlpro.bf.ml.basics.**AdaptiveFunction**(*p_input_space: ~mlpro.bf.math.basics.MSpace, p_output_space: ~mlpro.bf.math.basics.MSpace, p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>, p_ada: bool = True, p_buffer_size: int = 0, p_name: str | None = None, p_range_max: int = 2, p_autorun=0, p_class_shared=None, p_visualize: bool = False, p_logging=True, **p_par*)

Bases: *Function*, *Model*

Template class for an adaptive bi-multivariate mathematical function. The kind of adaptation (learning paradigm) is to be specified in child classes.

Parameters

- **p_input_space** (MSpace) – Input space of function
- **p_output_space** (MSpace) – Output space of function
- **p_output_elem_cls** – Output element class (compatible to/inherited from class Element)
- **p_ada** (*bool*) – Boolean switch for adaptivitiy. Default = True.
- **p_buffer_size** (*int*) – Initial size of internal data buffer. Defaut = 0 (no buffering).
- **p_name** (*str*) – Optional name of the model. Default is None.

- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_autorun** (*int*) – On value C_AUTORUN_RUN method run() is called imediately during instantiation. On vaule C_AUTORUN_LOOP method run_loop() is called. Value C_AUTORUN_NONE (default) causes an object instantiation without starting further actions.
- **p_class_shared** – Optional class for a shared object (class Shared or a child class of it)
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_par** (*Dict*) – Further model specific hyperparameters (to be defined in chhild class).

_range: int

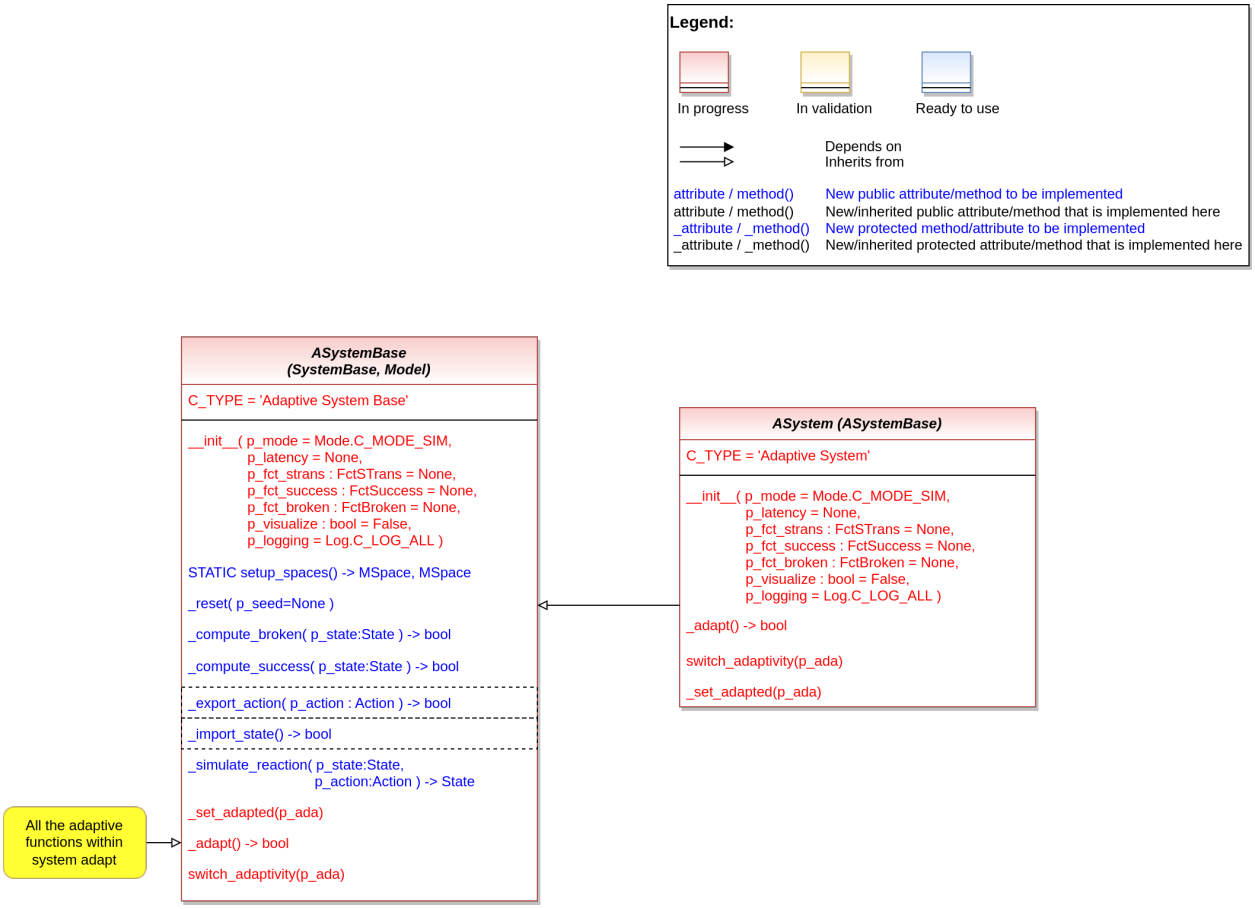
_so: *Shared*

_plot_settings: *PlotSettings*

C_TYPE = 'Adaptive Function'

C_NAME = '????'

BF-ML-SYSTEMS - Adaptive State-based Systems



9.1.2 MLPro-SL - Supervised Learning

SL - Basics


content/04_appendices/appendix2/sub/core/mlpro_sl/images/MLPro-SL-basics_class_diagram.drawio.png
Ver. 0.4.3 (2023-03-10)

This module provides model classes for supervised learning tasks.

class mlpro.sl.basics.**SLAdaptiveFunction**(*p_input_space: ~mlpro.bf.math.basics.MSpace*, *p_output_space: ~mlpro.bf.math.basics.MSpace*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_ada: bool = True*, *p_buffer_size: int = 0*, *p_name: str | None = None*, *p_range_max: int = 2*, *p_autorun=0*, *p_class_shared=None*, *p_visualize: bool = False*, *p_logging=True*, ***p_par*)

Bases: *AdaptiveFunction*

Template class for an adaptive bi-multivariate mathematical function that adapts by supervised learning.

Parameters

- **p_input_space** (*MSpace*) – Input space of function
- **p_output_space** (*MSpace*) – Output space of function
- **p_output_elem_cls** – Output element class (compatible to/inherited from class Element)
- **p_threshold** (*float*) – Threshold for the difference between a setpoint and a computed output. Computed outputs with a difference less than this threshold will be assessed as 'good' outputs. Default = 0.
- **p_ada** (*bool*) – Boolean switch for adaptivitiy. Default = True.
- **p_buffer_size** (*int*) – Initial size of internal data buffer. Defaut = 0 (no buffering).
- **p_name** (*str*) – Optional name of the model. Default is None.
- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_autorun** (*int*) – On value C_AUTORUN_RUN method run() is called imediately during instantiation. On vaule C_AUTORUN_LOOP method run_loop() is called. Value C_AUTORUN_NONE (default) causes an object instantiation without starting further actions.
- **p_class_shared** – Optional class for a shared object (class Shared or a child class of it)
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_par** (*Dict*) – Further model specific hyperparameters (to be defined in chhild class).

C_TYPE = 'Adaptive Function (SL)'

C_NAME = '????'

get_model()

A method to get the supervised learning network.

Return type
A set up supervised learning model

adapt(*p_input:* Element | *None* = *None*, *p_output:* Element | *None* = *None*, *p_dataset=None*) → bool

Adaption by supervised learning.

Parameters

- **p_input** (Element) – Abscissa/input element object (type Element)
- **p_output** (Element) – Setpoint ordinate/output element (type Element)
- **p_dataset** – A set of data for offline learning

get_accuracy()

Returns the accuracy of the adaptive function. The accuracy is defined as the relation between the number of successful mapped inputs and the total number of mappings since the last adaptation.

class mlpro.sl.basics.**SLScenario**(*p_mode=0*, *p_ada: bool = True*, *p_cycle_limit: int = 0*, *p_visualize: bool = True*, *_logging=True*)

Bases: *Scenario*

To be designed.

C_TYPE = 'SL-Scenario'

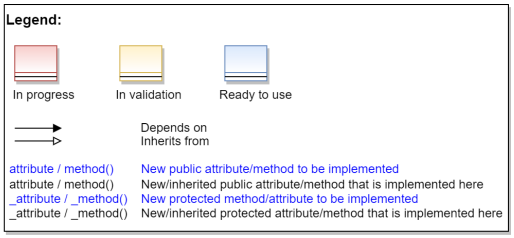
class mlpro.sl.basics.**SLTraining**(***p_kwargs*)

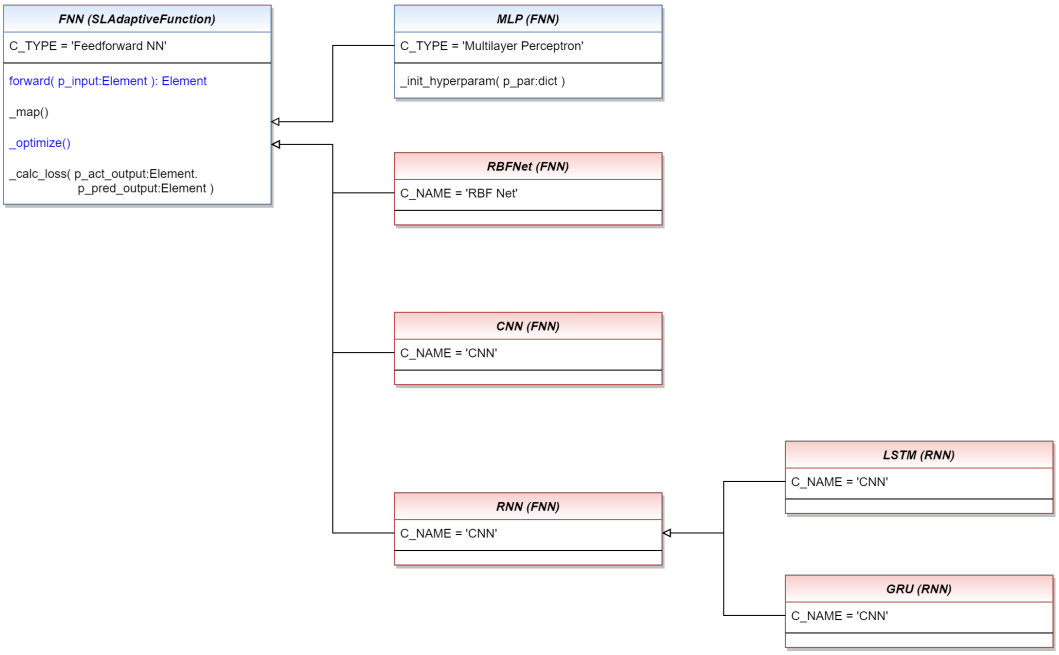
Bases: *Training*

To be designed.

C_NAME = 'SL'

SL - Feedforward Neural Network





Ver. 1.1.0 (2023-03-10)

This module provides model classes of feedforward neural networks for supervised learning tasks.

class mlpro.sl.fnn.**FNN**(*p_input_space: ~mlpro.bf.math.basics.MSpace*, *p_output_space: ~mlpro.bf.math.basics.MSpace*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_ada: bool = True*, *p_buffer_size: int = 0*, *p_name: str | None = None*, *p_range_max: int = 2*, *p_autorun=0*, *p_class_shared=None*, *p_visualize: bool = False*, *p_logging=True*, ***p_par*)

Bases: *SLAdaptiveFunction*

This class provides the base class of feedforward neural networks.

C_TYPE = 'Feedforward NN'

forward(*p_input:* Element) → *Element*

Custom forward propagation in neural networks to generate some output that can be called by an external method. Please redefine.

Parameters

p_input (Element) – Input data

Returns

output – Output data

Return type
Element

class mlpro.sl.fnn.**MLP**(*p_input_space: ~mlpro.bf.math.basics.MSpace*, *p_output_space: ~mlpro.bf.math.basics.MSpace*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_ada: bool = True*, *p_buffer_size: int = 0*, *p_name: str | None = None*, *p_range_max: int = 2*, *p_autorun=0*, *p_class_shared=None*, *p_visualize: bool = False*, *p_logging=True*, ***p_par*)

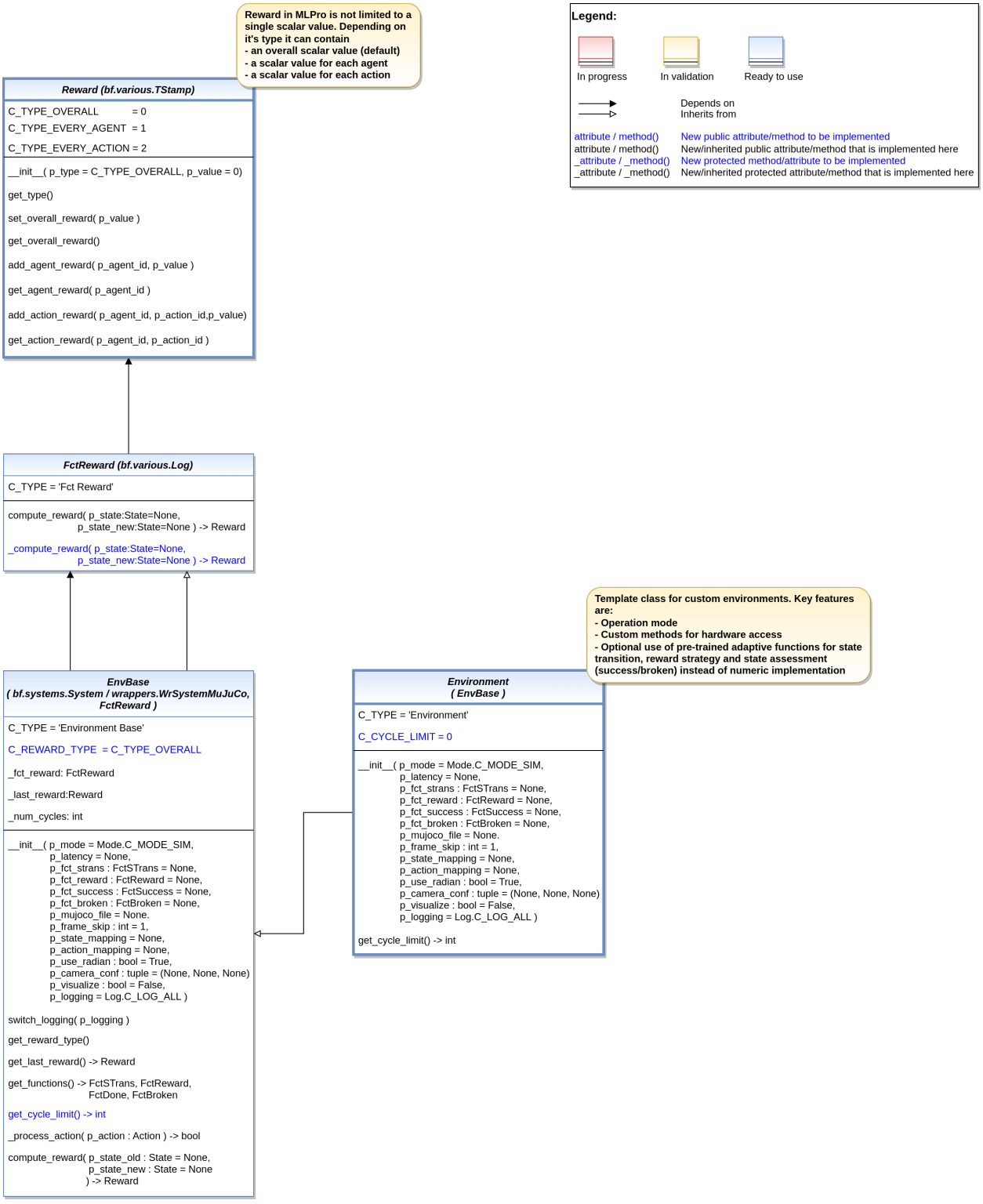
Bases: *FNN*

This class provides the base class of multilayer perceptron.

C_TYPE = 'Multilayer Perceptron'

9.1.3 MLPro-RL - Reinforcement Learning

RL-ENV - Environments



Ver. 1.7.3 (2023-02-13)

This module provides model classes for environments.

class mlpro.rl.models_env.**Reward**(*p_type=0*, *p_value=0*)

Bases: *TStamp*

Objects of this class represent rewards of environments. The internal structure depends on the reward type. Three types are supported as listed below.

Parameters

- **p_type** – Reward type (default: C_TYPE_OVERALL)
- **p_value** – Overall reward value (reward type C_TYPE_OVERALL only)

C_TYPE_OVERALL = 0

C_TYPE_EVERY_AGENT = 1

C_TYPE_EVERY_ACTION = 2

C_VALID_TYPES = [0, 1, 2]

get_type()

is_rewarded(*p_agent_id*) → bool

set_overall_reward(*p_reward*) → bool

get_overall_reward()

add_agent_reward(*p_agent_id*, *p_reward*) → bool

get_agent_reward(*p_agent_id*)

add_action_reward(*p_agent_id*, *p_action_id*, *p_reward*) → bool

get_action_reward(*p_agent_id*, *p_action_id*)

class mlpro.rl.models_env.**FctReward**(*p_logging=True*)

Bases: *Log*

Template class for reward functions.

Parameters

p_logging – Log level (see class Log for more details).

C_TYPE = 'Fct Reward'

compute_reward(*p_state_old:* State | *None = None*, *p_state_new:* State | *None = None*) → *Reward*

Computes a reward based on a predecessor and successor state. Custom method _compute_reward() is called.

Parameters

- **p_state_old** (State) – Predecessor state.
- **p_state_new** (State) – Successor state.

Returns

r – Reward

Return type

Reward

class mlpro.rl.models_env.**EnvBase**(*p_mode=0*, *p_latency: timedelta | None = None*, *p_fct_strans: FctSTrans | None = None*, *p_fct_reward: FctReward | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_frame_skip: int = 1*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf: tuple = (None, None, None)*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *System*, *FctReward*

Base class for all environment classes. It defines the interface and elementary properties for an environment in the context of reinforcement learning.

Parameters

- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_fct_strans** (FctSTrans) – Optional external function for state transition.
- **p_fct_reward** (FctReward) – Optional external function for reward computation.
- **p_fct_success** (FctSuccess) – Optional external function for state evaluation 'success'.
- **p_fct_broken** (FctBroken) – Optional external function for state evaluation 'broken'.
- **p_mujoco_file** – Path to XML file for MuJoCo model.
- **p_frame_skip** (*int*) – Frame to be skipped every step. Default = 1.
- **p_state_mapping** – State mapping if the MLPro state and MuJoCo state have different naming.
- **p_action_mapping** – Action mapping if the MLPro action and MuJoCo action have different naming.
- **p_use_radian** (*bool*) – Use radian if the action and the state based on radian unit. Default = True.
- **p_camera_conf** (*tuple*) – Default camera configuration on MuJoCo Simulation (xyz position, elevation, distance).
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see class Log for more details).

_latency

Latency of the environment.

Type
timedelta

_state

Current state of environment.

Type
State

_state

Previous state of environment.

Type
State

_last_action

Last action.

Type
Action

_last_reward

Last reward.

Type
Reward

_afct_strans

Internal adaptive state transition function.

Type
AFctSTrans

_afct_reward

Internal adaptive reward function.

Type
AFctReward

_afct_success

Internal adaptive function for state evaluation 'success'.

Type
AFctSuccess

_afct_broken

Internal adaptive function for state evaluation 'broken'.

Type
AFctBroken

C_TYPE = 'Environment Base'

C_REWARD_TYPE = 0

switch_logging(*p_logging*)

Sets new log level.

Parameters
p_logging – Log level (constant C_LOG_LEVELS contains valid values)

get_reward_type()

get_last_reward() → *Reward*

get_functions()

get_cycle_limit() → int

Returns limit of cycles per training episode. To be implemented in child classes.

compute_reward(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*) → *Reward*

Computes a reward for the state transition, given by two successive states. The reward computation itself is carried out either by a custom implementation in method _compute_reward() or by an embedded adaptive function.

Parameters

- **p_state_old** (State) – Optional state before transition. If None the internal previous state of the environment is used.
- **p_state_new** (State) – Optional tate after transition. If None the internal current state of the environment is used.

Returns
Reward object.

Return type
Reward

class mlpro.rl.models_env.**Environment**(*p_mode=0*, *p_latency: timedelta | None = None*, *p_fct_strans:* FctSTrans *| None = None*, *p_fct_reward:* FctReward *| None = None*, *p_fct_success:* FctSuccess *| None = None*, *p_fct_broken:* FctBroken *| None = None*, *p_mujoco_file=None*, *p_frame_skip: int = 1*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf: tuple = (None, None, None)*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *EnvBase*

This class represents the central environment model to be reused/inherited in own rl projects.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_fct_strans** (FctSTrans) – Optional external function for state transition.
- **p_fct_reward** (FctReward) – Optional external function for reward computation.
- **p_fct_success** (FctSuccess) – Optional external function for state evaluation 'success'.
- **p_fct_broken** (FctBroken) – Optional external function for state evaluation 'broken'.
- **p_mujoco_file** – Path to XML file for MuJoCo model.
- **p_frame_skip** (*int*) – Frame to be skipped every step. Default = 1.
- **p_state_mapping** – State mapping if the MLPro state and MuJoCo state have different naming.
- **p_action_mapping** – Action mapping if the MLPro action and MuJoCo action have different naming.
- **p_use_radian** (*bool*) – Use radian if the action and the state based on radian unit. Default = True.
- **p_camera_conf** (*tuple*) – Default camera configuration on MuJoCo Simulation (xyz position, elevation, distance).
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.
- **p_logging** – Log level (see class Log for more details)

C_TYPE = 'Environment'

C_CYCLE_LIMIT = 0

static setup_spaces()

Static template method to set up and return state and action space of environment.

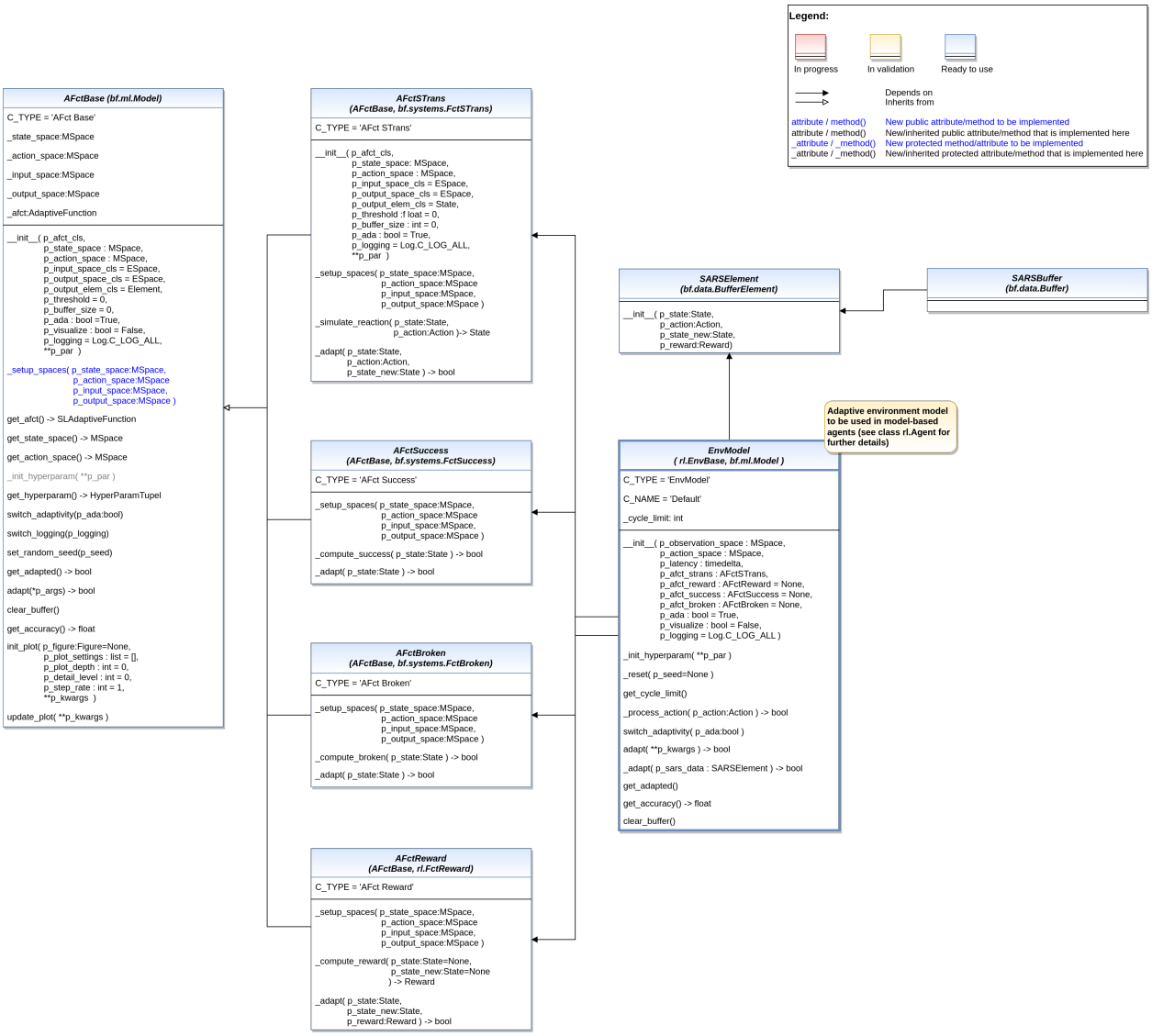
Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

get_cycle_limit() → int

Returns limit of cycles per training episode.

RL-ENV-ADA - Environment Models



Ver. 1.0.2 (2023-03-10)

This module provides model classes for adaptive environment models.

class mlpro.rl.models_env_ada.**AFctBase**(*p_afct_cls*, *p_state_space: ~mlpro.bf.math.basics.MSpace*, *p_action_space: ~mlpro.bf.math.basics.MSpace*, *p_input_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_buffer_size=0*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *Model*

Base class for all special adaptive functions (state transition, reward, success, broken).

Parameters

- **p_afct_cls** – Adaptive function class (compatible to class mlpro.sl.SLAdaptiveFunction)
- **p_state_space** (*MSpace*) – State space of an environment or observation space of an agent
- **p_action_space** (*MSpace*) – Action space of an environment or agent
- **p_input_space_cls** – Space class that is used for the generated input space of the embedded adaptive function (compatible to class MSpace)
- **p_output_space_cls** – Space class that is used for the generated output space of the embedded adaptive function (compatible to class MSpace)
- **p_output_elem_cls** – Output element class (compatible to/inherited from class Element)
- **p_threshold** (*float*) – Threshold for the difference between a set point and a computed output. Computed outputs with a difference less than this threshold will be assessed as 'good' outputs. Default = 0.
- **p_buffer_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*Dict*) – Further model specific parameters (to be specified in child class).

_state_space

State space

Type
MSpace

_action_space

Action space

Type
MSpace

_input_space

Input space of embedded adaptive function

Type
MSpace

_output_space

Output space oof embedded adaptive function

Type
MSpace

_afct

Embedded adaptive function

Type
SLAdaptiveFunction

C_TYPE = 'AFct Base'

get_afct() → *SLAdaptiveFunction*

get_state_space() → *MSpace*

get_action_space() → *MSpace*

get_hyperparam() → *HyperParamTuple*

Returns the internal hyperparameter tuple to get access to single values.

switch_adaptivity(*p_ada: bool*)

Switches adaption functionality on/off.

Parameters
p_ada (*bool*) – Boolean switch for adaptivity

switch_logging(*p_logging*)

Sets new log level.

Parameters
p_logging – Log level (constant C_LOG_LEVELS contains valid values)

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

get_adapted() → bool

Returns True, if the model was adapted at least once. False otherwise.

clear_buffer()

Clears internal buffer (if buffering is active).

get_accuracy()

Determines the accuracy of the model.

Returns
accuracy – Accuracy of the model as a scalar value in interval [0,1]

Return type
float

init_plot(*p_figure: Figure | None = None*, *p_plot_settings: list = []*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_step_rate: int = 0*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(**p_kwargs)

Updates the plot.

Parameters

****p_kwargs** – Implementation-specific plot data and/or parameters.

class mlpro.rl.models_env_ada.**AFctSTrans**(*p_afct_cls*, *p_state_space: ~mlpro.bf.math.basics.MSpace*, *p_action_space: ~mlpro.bf.math.basics.MSpace*, *p_input_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_elem_cls=<class 'mlpro.bf.systems.basics.State'>*, *p_threshold=0*, *p_buffer_size=0*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_par*)

Bases: *AFctBase*, *FctSTrans*

Online adaptive version of a state transition function. See parent classes for further details.

C_TYPE = 'AFct STrans'

class mlpro.rl.models_env_ada.**AFctSuccess**(*p_afct_cls*, *p_state_space: ~mlpro.bf.math.basics.MSpace*, *p_action_space: ~mlpro.bf.math.basics.MSpace*, *p_input_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_buffer_size=0*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *AFctBase*, *FctSuccess*

Online adaptive version of a function that determine whether or not a state is a success state. See parent classes for further details.

C_TYPE = 'AFct Success'

class mlpro.rl.models_env_ada.**AFctBroken**(*p_afct_cls*, *p_state_space: ~mlpro.bf.math.basics.MSpace*, *p_action_space: ~mlpro.bf.math.basics.MSpace*, *p_input_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_buffer_size=0*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *AFctBase*, *FctBroken*

Online adaptive version of a function that determine whether or not a state is a broken state. See parent classes for further details.

C_TYPE = 'AFct Broken'

class mlpro.rl.models_env_ada.**AFctReward**(*p_afct_cls*, *p_state_space: ~mlpro.bf.math.basics.MSpace*, *p_action_space: ~mlpro.bf.math.basics.MSpace*, *p_input_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_space_cls=<class 'mlpro.bf.math.basics.ESpace'>*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_buffer_size=0*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *AFctBase*, *FctReward*

Online adaptive version of a reward function. See parent classes for further details.

C_TYPE = 'AFct Reward'

class mlpro.rl.models_env_ada.**SARSElement**(*p_state:* State, *p_action:* Action, *p_reward:* Reward, *p_state_new:* State)

Bases: *BufferElement*

Element of a SARSBuffer.

class mlpro.rl.models_env_ada.**SARSBuffer**(*p_size=1*)

Bases: *Buffer*

State-Action-Reward-State-Buffer in dictionary.

class mlpro.rl.models_env_ada.**EnvModel**(*p_observation_space:* MSpace, *p_action_space:* MSpace, *p_latency: timedelta*, *p_afct_strans:* AFctSTrans, *p_afct_reward:* AFctReward | *None* = *None*, *p_afct_success:* AFctSuccess | *None* = *None*, *p_afct_broken:* AFctBroken | *None* = *None*, *p_ada: bool* = *True*, *p_init_states:* State | *None* = *None*, *p_visualize: bool* = *False*, *p_logging=True*)

Bases: *EnvBase*, *Model*

Environment model class as part of a model-based agent.

Parameters

- **p_observation_space** (MSpace) – Observation space of related agent.
- **p_action_space** (MSpace) – Action space of related agent.
- **p_latency** (*timedelta*) – Latency of related environment.
- **p_afct_strans** (AFctSTrans) – Mandatory external adaptive function for state transition.
- **p_afct_reward** (AFctReward) – Optional external adaptive function for reward computation.
- **p_afct_success** (AFctSuccess) – Optional external adaptive function for state assessment 'success'.
- **p_afct_broken** (AFctBroken) – Optional external adaptive function for state assessment 'broken'.
- **p_ada** (*bool*) – Boolean switch for adaptivity.
- **p_init_states** (State) – Initial state of the env models.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see class Log for more details).

C_TYPE = 'EnvModel'

C_NAME = 'Default'

static **setup_spaces**()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

get_cycle_limit() → int

Returns limit of cycles per training episode.

switch_adaptivity(*p_ada: bool*)

Switches adaption functionality on/off.

Parameters

p_ada (*bool*) – Boolean switch for adaptivity

adapt(***p_kwargs*) → bool

Reactivated adaptation mechanism. See method Model.adapt() for further details.

get_adapted() → bool

Returns True, if the model was adapted at least once. False otherwise.

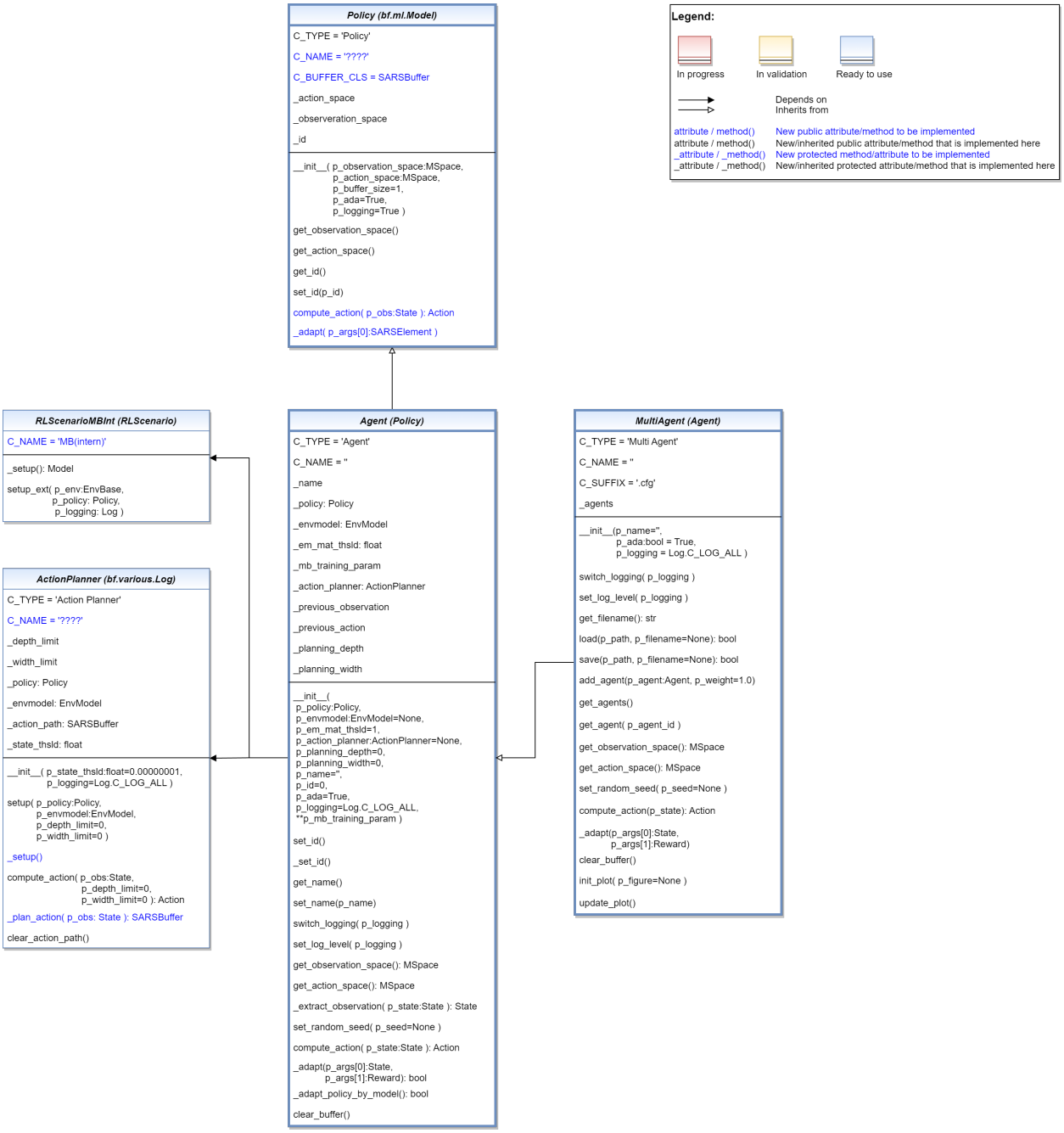
get_accuracy()

Returns accuracy of environment model as average accuracy of the embedded adaptive functions.

clear_buffer()

Clears internal buffer (if buffering is active).

RL-AGENTS - Agents



Ver. 1.6.7 (2023-03-10)

This module provides model classes for policies, model-free and model-based agents and multi-agents.

class mlpro.rl.models_agents.**Policy**(*p_observation_space:* MSpace, *p_action_space:* MSpace, *p_buffer_size=1*, *p_ada=True*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *Model*

This class represents the policy of a single-agent. It is adaptive and can be trained with State-Action-Reward (SAR) data that will be expected as a SAR buffer object. The three main learning paradigms of machine learning

to train a policy are supported:

a) Training by Supervised Learning The entire SAR data set inside the SAR buffer shall be adapted.

b) Training by Reinforcement Learning The latest SAR data record inside the SAR buffer shall be adapted.

c) Training by Unsupervised Learning All state data inside the SAR buffer shall be adapted.

Furthermore, a policy class can compute actions from states.

Hyperparameters of the policy should be stored in the internal object self._hp_list, so that they can be tuned from outside. Optionally a policy-specific callback method can be called on changes. For more information see class HyperParameterList.

Parameters

- **p_observation_space** (MSpace) – Subspace of an environment that is observed by the policy.
- **p_action_space** (MSpace) – Action space object.
- **p_buffer_size** (*int*) – Size of internal buffer. Default = 1.
- **p_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Policy'

C_NAME = '????'

C_BUFFER_CLS

alias of *SARSBuffer*

get_observation_space() → *MSpace*

get_action_space() → *MSpace*

get_id()

set_id(*p_id*)

compute_action(*p_obs:* State) → *Action*

Specific action computation method to be redefined.

Parameters
p_obs (State) – Observation data.

Returns
action – Action object.

Return type
Action

class mlpro.rl.models_agents.**ActionPlanner**(*p_state_thsld=1e-08*, *p_logging=True*)

Bases: *Log*

Template class for action planning algorithms to be used as part of model-based planning agents. The goal is to find the shortest sequence of actions that leads to a maximum reward.

Parameters

- **p_state_thsld** (*float*) – Threshold for metric difference between two states to be equal. Default = 0.00000001.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Action Planner'

setup(*p_policy:* Policy, *p_envmodel:* EnvModel, *p_prediction_horizon=0*, *p_control_horizon=0*, *p_width_limit=0*)

Setup of action planner object in concrete planning scenario. Must be called before first planning. Optional custom method _setup() is called at the end.

Parameters

- **p_policy** (Policy) – Policy of an agent.
- **p_envmodel** (EnvModel) – Environment model.
- **p_prediction_horizon** (*int*) – Optional static maximum planning depth (=length of action path to be predicted). Can be overridden by method compute_action(). Default=0.
- **p_control_horizon** (*int*) – The length of predicted action path to be applied. Can be overridden by method compute_action(). Default=0.
- **p_width_limit** (*int*) – Optional static maximum planning width (=number of alternative actions per planning level). Can be overridden by method compute_action(). Default=0.

compute_action(*p_obs:* State, *p_prediction_horizon=0*, *p_control_horizon=0*, *p_width_limit=0*) → *Action*

Computes a path of actions with defined length that maximizes the reward of the given environment model. The planning algorithm itself is to be implemented in the custom method _plan_action().

Parameters

- **p_obs** (State) – Observation data.
- **p_prediction_horizon** (*int*) – Optional dynamic maximum planning depth (=length of action path to be predicted) that overrides the static limit of method setup(). Default=0 (no override).
- **p_control_horizon** (*int*) – The length of predicted action path to be applied that overrides the static limit of method setup(). Default=0 (no override).
- **p_width_limit** (*int*) – Optional dynamic maximum planning width (=number of alternative actions per planning level) that overrides the static limit of method setup(). Default=0 (no override).

Returns
action – Best action as result of the planning process.

Return type
Action

clear_action_path()

class mlpro.rl.models_agents.**RLScenarioMBInt**(*p_mode=0*, *p_ada: bool = True*, *p_cycle_limit=0*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *RLScenario*

Internal use in class Agent. Intended for the training of the policy with the environment model of a model-based (single) agent.

C_NAME = 'MB(intern)'

setup_ext(*p_env:* EnvBase, *p_policy:* Policy, *p_logging:* Log)

class mlpro.rl.models_agents.**Agent**(*p_policy:* Policy, *p_envmodel:* EnvModel | *None* = *None*, *p_em_acc_thsld=0.9*, *p_action_planner:* ActionPlanner | *None* = *None*, *p_predicting_horizon=0*, *p_controlling_horizon=0*, *p_planning_width=0*, *p_name=''*, *p_id=0*, *p_ada=True*, *p_visualize: bool = True*, *p_logging=True*, ***p_mb_training_param*)

Bases: *Policy*

This class represents a single agent model.

Parameters

- **p_policy** (Policy) – Policy object.
- **p_envmodel** (EnvModel) – Optional environment model object. Default = None.
- **p_em_acc_thsld** (*float*) – Optional threshold for environment model accuracy (whether the envmodel is 'good' enough to be used to train the policy). Default = 0.9.
- **p_action_planner** (ActionPlanner) – Optional action planner object (obligatory for model based agents). Default = None.
- **p_predicting_horizon** (*int*) – Optional predicting horizon (obligatory for model based agents). Default = 0.
- **p_controlling_horizon** (*int*) – Optional controlling (obligatory for model based agents). Default = 0.
- **p_planning_width** (*int*) – Optional planning width (obligatory for model based agents). Default = 0.
- **p_name** (*str*) – Optional name of agent. Default = ''.
- **p_id** (*int*) – Optional unique agent id (especially important for multi-agent scenarios). Default = 0.
- **p_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_ALL.
- **p_mb_training_param** (*dict*) – Optional parameters for internal policy training with environment model (see parameters of class RLTraining). Hyperparameter tuning and data logging is not supported here. The suitable scenario class is added internally.

C_TYPE = 'Agent'

C_NAME = ''

set_id(*p_id*)

The unique agent id will be defined while instantiation and can't be changed anymore.

get_name()

set_name(*p_name*)

switch_logging(*p_logging*)

Sets new log level.

Parameters

p_logging – Log level (constant C_LOG_LEVELS contains valid values)

switch_adaptivity(*p_ada: bool*)

Switches adaption functionality on/off.

Parameters
p_ada (*bool*) – Boolean switch for adaptivity

set_log_level(*p_level*)

get_observation_space() → *MSpace*

get_action_space() → *MSpace*

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

compute_action(*p_state:* State) → *Action*

Default implementation of a single agent.

Parameters
p_state (State) – State of the related environment.

Returns
action – Action object.

Return type
Action

clear_buffer()

Clears internal buffer (if buffering is active).

class mlpro.rl.models_agents.**MultiAgent**(*p_name=''*, *p_ada=True*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *Agent*

Multi-Agent.

Parameters

- **p_name** (*str*) – Name of agent. Default = ‘’.
- **p_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Multi-Agent'

C_NAME = ''

C_SUFFIX = '.cfg'

switch_logging(*p_logging*) → None

Sets new log level.

Parameters
p_logging – Log level (constant C_LOG_LEVELS contains valid values)

switch_adaptivity(*p_ada: bool*)

Switches adaption functionality on/off.

Parameters
p_ada (*bool*) – Boolean switch for adaptivity

set_log_level(*p_level*)

add_agent(*p_agent: Agent*, *p_weight=1.0*) → None

Adds agent object to internal list of agents.

Parameters

- **p_agent** (Agent) – Agent object to be added.
- **p_weight** (*float*) – Optional weight for the agent. Default = 1.0.

get_agents()

get_agent(*p_agent_id*)

Returns information of a single agent.

Returns
agent_info – agent_info[0] is the agent object itself and agent_info[1] it's weight

Return type
list

get_observation_space() → *MSpace*

get_action_space() → *MSpace*

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

compute_action(*p_state: State*) → *Action*

Default implementation of a single agent.

Parameters
p_state (State) – State of the related environment.

Returns
action – Action object.

Return type
Action

clear_buffer()

Clears internal buffer (if buffering is active).

init_plot(*p_figure: Figure | None = None*, *p_plot_settings: list = Ellipsis*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_step_rate: int = 0*, ***p_kwargs*)

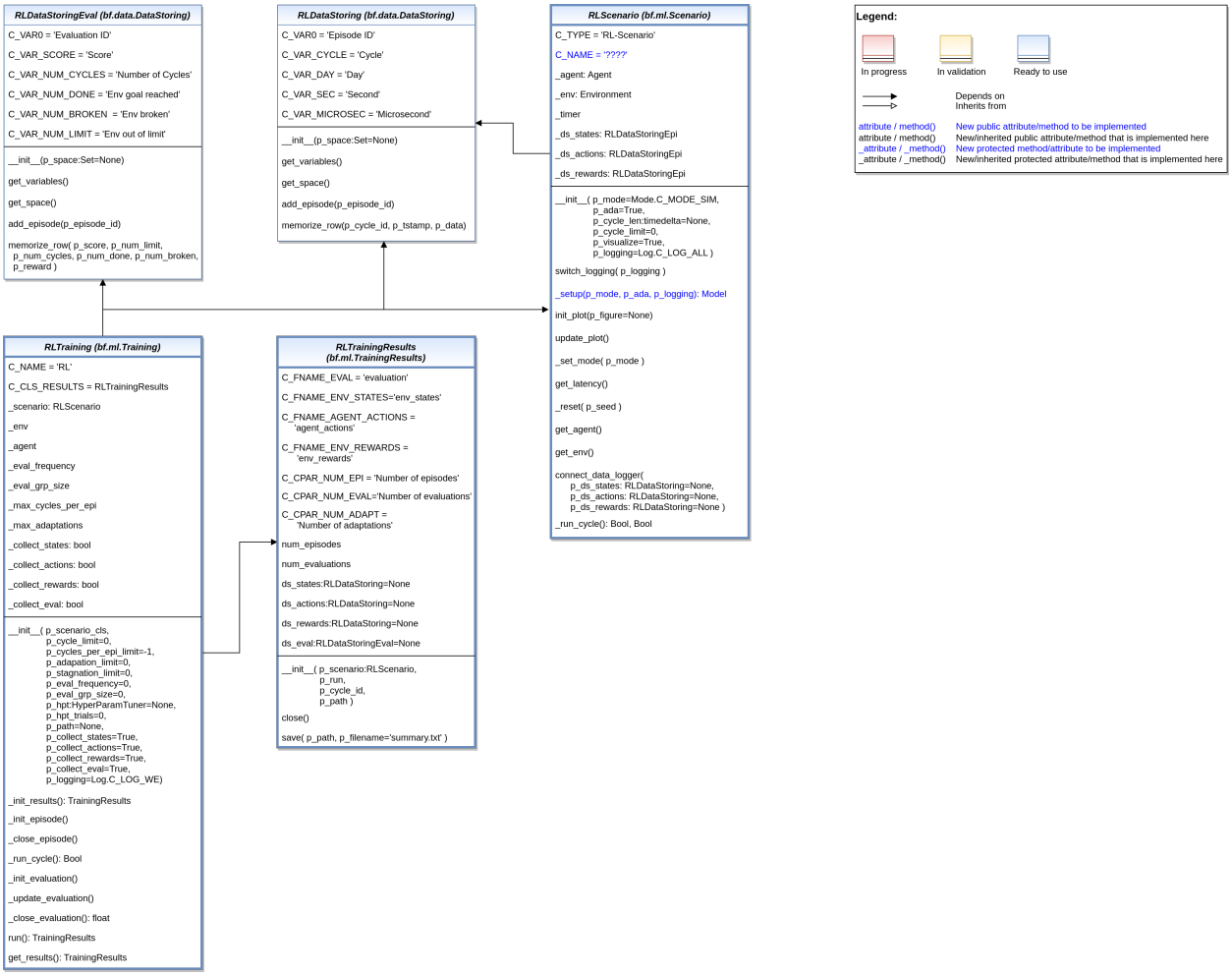
Doesn't support embedded plot of underlying agent hierarchy.

update_plot(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

RL-TRAIN - Scenarios, Training and Tuning



Ver. 1.9.1 (2023-03-09)

This module provides model classes to define and run rl scenarios and to train agents inside them.

class mlpro.rl.models_train.**RLDataStoring**(*p_space:* Set | *None* = *None*)

Bases: *DataStoring*

Derivative of basic class DataStoring that is specialized to store episodic training data in the context of reinforcement learning.

Parameters

p_space (Set) – Space object that provides dimensional information for raw data. If None, a training header data object will be instantiated.

C_VAR0 = 'Episode ID'

C_VAR_CYCLE = 'Cycle'

C_VAR_DAY = 'Day'

C_VAR_SEC = 'Second'

C_VAR_MICROSEC = 'Microsecond'

get_variables()

get_space()

add_episode(*p_episode_id*)

memorize_row(*p_cycle_id*, *p_tstamp: timedelta*, *p_data*)

Memorizes an episodic data row.

Parameters

- **id** (*p_cycle_id Cycle*) –
- **stamp** (*p_tstamp Time*) –
- **space** (*p_data Data that meet the dimensionality of the related*) –

class mlpro.rl.models_train.**RLDataStoringEval**(*p_space: Set*)

Bases: *DataStoring*

Derivative of basic class DataStoring that is specialized to store evaluation data of a training in the context of reinforcement learning.

Parameters

- **p_space** (Set) – Set object that provides dimensional information for raw data. If None a training header data object will
- **instantiated.** (*be*) –

C_VAR0 = 'Evaluation ID'

C_VAR_SCORE = 'Score'

C_VAR_SCORE_MA = 'Score(MA)'

C_VAR_SCORE_UNTIL_STAG = 'Score until Stagnation'

C_VAR_SCORE_MA_UNTIL_STAG = 'Score(MA) until Stagnation'

C_VAR_NUM_CYCLES = 'Cycles'

C_VAR_NUM_SUCCESS = 'Successes'

C_VAR_NUM_BROKEN = 'Crashes'

C_VAR_NUM_LIMIT = 'Timeouts'

C_VAR_NUM_ADAPT = 'Adaptations'

get_variables()

get_space()

add_evaluation(*p_evaluation_id*)

memorize_row(*p_score*, *p_score_ma*, *p_num_limit*, *p_num_cycles*, *p_num_success*, *p_num_broken*, *p_num_adaptations*, *p_reward*, *p_score_until_stag=None*, *p_score_ma_until_stag=None*)

Memorizes an evaluation data row.

Parameters

- **p_score** (*float*) – Score value of current evaluation.

- **p_score_ma** (*float*) – Moving average score value.
- **p_num_limit** (*int*) – Number of episodes in timeout.
- **p_num_cycles** (*int*) – Number of evaluation cycles.
- **p_num_success** (*int*) – Number of states that were labeled as successfully.
- **p_num_broken** (*int*) – Number of states that were labeled as broken.
- **p_num_adaptations** (*int*) – Number of adaptations in the last training period.
- **p_reward** (*list*) – Episode Reward
- **p_score_until_stag** (*float*) – Optional score value of current evaluation until first stagnation. Default = None.
- **p_score_ma_until_stag** (*float*) – Optional moving average score value until first stagnation. Default = None.

class mlpro.rl.models_train.**RLScenario**(*p_mode=0*, *p_ada: bool = True*, *p_cycle_limit=0*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *Scenario*

Template class for an RL scenario consisting of an environment and an agent.

Parameters

- **p_mode** – Operation mode. See bf.ops.Mode.C_VALID_MODES for valid values. Default = Mode.C_MODE_SIM.
- **p_ada** (*bool*) – Boolean switch for adaptivitiy. Default = True.
- **p_cycle_limit** (*int*) – Maximum number of cycles (0=no limit, -1=get from env). Default = 0.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_TYPE

Constant class type for logging: 'RL-Scenario'.

Type
str

C_NAME

Constant custom name for logging. To be set in own child class.

Type
str

C_TYPE = 'RL-Scenario'

C_NAME = '????'

static load(*p_path*)

Loads content from the given path and file name. If file does not exist, it returns None.

Parameters

- **file** (*p_path Path that contains the*) –
- **name** (*p_filename File*) –

Returns
A loaded object, if file content was loaded successfully. None otherwise.

`switch_logging`(*p_logging*)

Sets new log level.

Parameters
p_logging – Log level (constant C_LOG_LEVELS contains valid values)

`init_plot`(*p_figure: Figure | None = None*, *p_plot_settings:* PlotSettings *| None = None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

`update_plot`(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

`get_latency`() → timedelta

Returns the latency of the scenario. To be implemented in child class.

`get_agent`()

`get_env`()

`connect_data_logger`(*p_ds_states:* RLDataStoring *| None = None*, *p_ds_actions:* RLDataStoring *| None = None*, *p_ds_rewards:* RLDataStoring *| None = None*)

`class` mlpro.rl.models_train.**RLTrainingResults**(*p_scenario:* RLScenario, *p_run*, *p_cycle_id*, *p_logging='W'*)

Bases: *TrainingResults*

Results of a RL training.

Parameters

- **p_scenario** (RLScenario) – Related reinforcement learning scenario.
- **p_run** (*int*) – Run id.
- **p_cycle_id** (*int*) – Id of first cycle of this run.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL

`C_NAME = 'RL'`

`C_FNAME_EVAL = 'evaluation'`

`C_FNAME_ENV_STATES = 'env_states'`

`C_FNAME_AGENT_ACTIONS = 'agent_actions'`

`C_FNAME_ENV_REWARDS = 'env_rewards'`

C_CPAR_NUM_EPI = 'Training Episodes'

C_CPAR_NUM_EVAL = 'Evaluations'

close()

save(*p_path*, *p_filename='summary.csv'*) → bool

Saves a training summary in the given path.

Parameters

- **p_path** (*str*) – Destination folder
- **p_filename** (*string*) – Name of summary file. Default = 'summary.csv'

Returns

success – True, if summary file was created successfully. False otherwise.

Return type

bool

class mlpro.rl.models_train.**RLTraining**(***p_kwargs*)

Bases: *Training*

This class performs an episodic training on a (multi-)agent in a given environment. Both are expected as parts of a reinforcement learning scenario (see class RLScenario for more details). The class optionally collects all relevant data like environment states and rewards or agents actions. Furthermore, overarching training data will be collected.

Parameters

- **p_scenario_cls** – Name of RL scenario class, compatible to/inherited from class RLScenario.
- **p_cycle_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.
- **p_cycles_per_epi_limit** (*int*) – Optional limit of cycles per episode (0=no limit, -1=get environment limit). Default = -1.
- **p_adaptation_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p_eval_frequency** (*int*) – Optional evaluation frequency (0=no evaluation). Default = 0.
- **p_eval_grp_size** (*int*) – Number of evaluation episodes (eval group). Default = 0.
- **p_score_ma_horizon** (*int*) – Horizon length for moving average score computation. Default = 5.
- **p_stagnation_limit** (*int*) – Optional limit of consecutive evaluations without training progress. Base is the moving average score. Default = 0.
- **p_stagnation_entry** (*int*) – Optional number of evaluations before the stagnation detection starts. Default = 0.
- **p_end_at_stagnation** (*bool*) – If True, the training ends when stagnation has beed detected. Default = True.
- **p_hpt** (HyperParamTuner) – Optional hyperparameter tuner (see class mlpro.bf.ml.HyperParamTuner). Default = None.
- **p_hpt_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0. Must be > 0 if p_hpt is supplied.
- **p_path** (*str*) – Optional destination path to store training data. Default = None.

- **p_collect_states** (*bool*) – If True, the environment states will be collected. Default = True.
- **p_collect_actions** (*bool*) – If True, the agent actions will be collected. Default = True.
- **p_collect_rewards** (*bool*) – If True, the environment reward will be collected. Default = True.
- **p_collect_eval** (*bool*) – If True, global evaluation data will be collected. Default = True.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

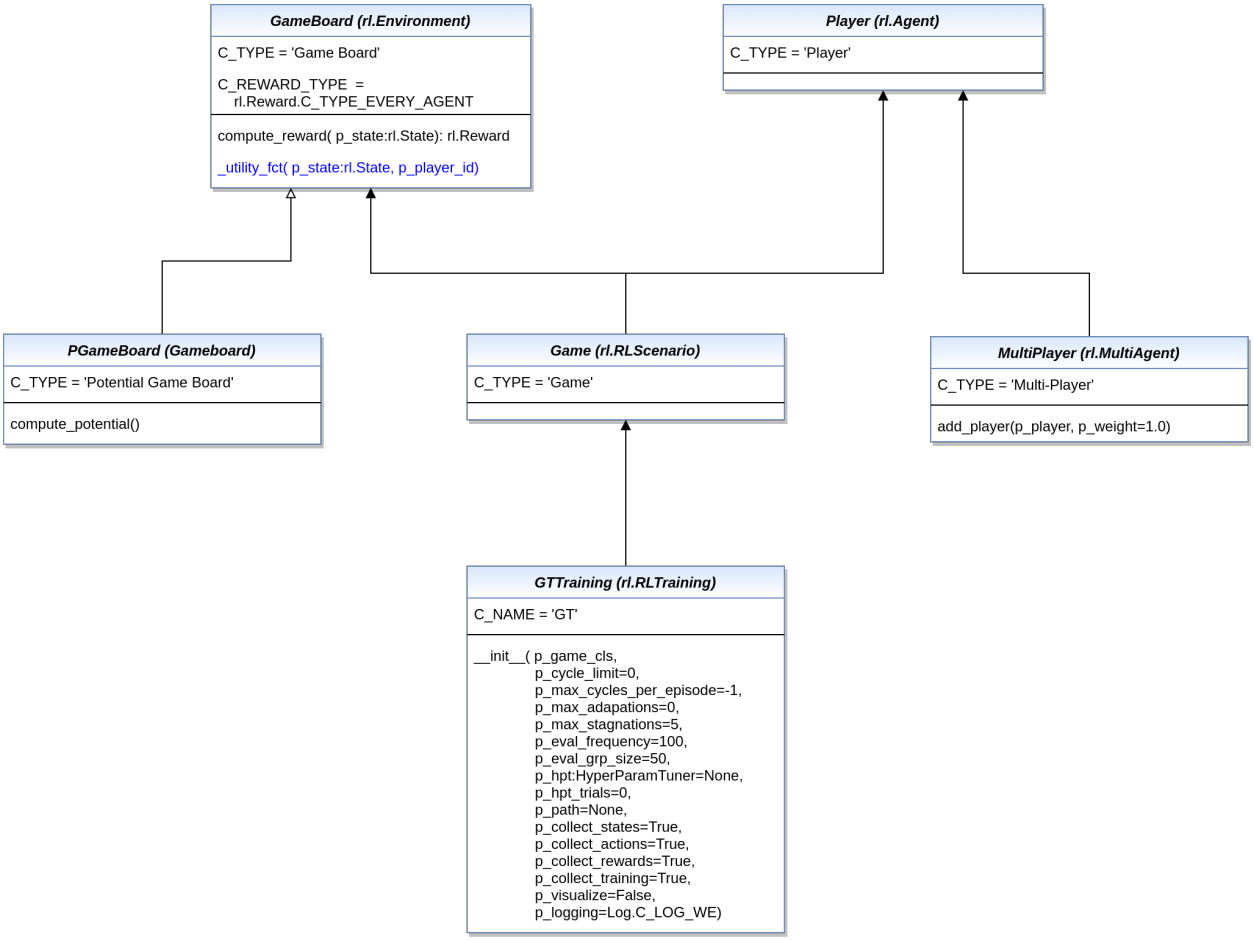
C_NAME = 'RL'

C_CLS_RESULTS

alias of *RLTrainingResults*

9.1.4 MLPro-GT - Game Theory

GT - Basics



Ver. 1.2.0 (2022-11-01)

This module provides model classes for tasks related to cooperative Game Theory.

class mlpro.gt.models.**GameBoard**(*p_mode=0*, *p_latency: timedelta | None = None*, *p_fct_strans: FctSTrans | None = None*, *p_fct_reward: FctReward | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_frame_skip: int = 1*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf: tuple = (None, None, None)*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *Environment*

Model class for a game theoretical game board. See super class for more information.

C_TYPE = 'Game Board'

C_REWARD_TYPE = 1

class mlpro.gt.models.**PGameBoard**(*p_mode=0*, *p_latency: timedelta | None = None*, *p_fct_strans: FctSTrans | None = None*, *p_fct_reward: FctReward | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_frame_skip: int = 1*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf: tuple = (None, None, None)*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *GameBoard*

Model class for a potential game theoretical game board. See super class for more information.

C_TYPE = 'Potential Game Board'

compute_potential()

Computes (weighted) potential level of the game board.

class mlpro.gt.models.**Player**(*p_policy: Policy*, *p_envmodel: EnvModel | None = None*, *p_em_acc_thsld=0.9*, *p_action_planner: ActionPlanner | None = None*, *p_predicting_horizon=0*, *p_controlling_horizon=0*, *p_planning_width=0*, *p_name=''*, *p_id=0*, *p_ada=True*, *p_visualize: bool = True*, *p_logging=True*, ***p_mb_training_param*)

Bases: *Agent*

This class implements a game theoretical player model. See super class for more information.

C_TYPE = 'Player'

class mlpro.gt.models.**MultiPlayer**(*p_name=''*, *p_ada=True*, *p_visualize: bool = False*, *p_logging=True*)

Bases: *MultiAgent*

This class implements a game theoretical model for a team of players. See super class for more information.

C_TYPE = 'Multi-Player'

add_player(*p_player: Player*, *p_weight=1.0*) → None

class mlpro.gt.models.**Game**(*p_mode=0*, *p_ada: bool = True*, *p_cycle_limit=0*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *RLScenario*

This class implements a game consisting of a game board and a (multi-)player. See super class for more information.

C_TYPE = 'Game'

class mlpro.gt.models.**GTTraining**(***p_kwargs*)

Bases: *RLTraining*

This class implements a standardized episodical training process. See super class for more information.

Parameters

- **p_game_cls** – Name of GT game class, compatible to/inherited from class Game.
- **p_cycle_limit** (*int*) – Maximum number of training cycles (0=no limit). Default = 0.
- **p_cycles_per_epi_limit** (*int*) – Optional limit of cycles per episode (0=no limit, -1=get environment limit). Default = -1.
- **p_adaptation_limit** (*int*) – Maximum number of adaptations (0=no limit). Default = 0.
- **p_stagnation_limit** (*int*) – Optional limit of consecutive evaluations without training progress. Default = 0.
- **p_eval_frequency** (*int*) – Optional evaluation frequency (0=no evaluation). Default = 0.
- **p_eval_grp_size** (*int*) – Number of evaluation episodes (eval group). Default = 0.
- **p_hpt** (HyperParamTuner) – Optional hyperparameter tuner (see class mlpro.bf.ml.HyperParamTuner). Default = None.
- **p_hpt_trials** (*int*) – Optional number of hyperparameter tuning trials. Default = 0. Must be > 0 if p_hpt is supplied.
- **p_path** (*str*) – Optional destination path to store training data. Default = None.
- **p_collect_states** (*bool*) – If True, the environment states will be collected. Default = True.
- **p_collect_actions** (*bool*) – If True, the agent actions will be collected. Default = True.
- **p_collect_rewards** (*bool*) – If True, the environment reward will be collected. Default = True.
- **p_collect_training** (*bool*) – If True, global training data will be collected. Default = True.
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = False.
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_NAME = 'GT'

9.1.5 MLPro-OA - Online Adaptivity

Coming soon…

9.2 Pool Objects

9.2.1 MLPro-BF - Basic Functions

BF-STREAMS - Stream Processing

Native Sample Streams

2D Random Point Clouds

Ver. 1.0.0 (2022-12-15)

This module provides the native stream class StreamMLProStaticClouds2D. This stream provides 1000 instances with 2-dimensional random feature data placed around four fixed center points.

class mlpro.bf.streams.streams.clouds2d_static.**StreamMLProStaticClouds2D**(*p_logging=True*)

Bases: StreamMLProBase

This demo stream provides 1000 2-dimensional instances randomly positioned around four fixed centers.

C_ID = 'StaticClouds2D'

C_NAME = 'Static Clouds 2D'

C_VERSION = '1.0.0'

C_NUM_INSTANCES = 1000

C_SCIREF_ABSTRACT = 'Demo stream provides 1000 2-dimensional instances randomly positioned around four fixed centers.'

C_BOUNDARIES = [-10, 10]

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

3D Random Point Clouds

Ver. 1.0.0 (2022-12-15)

This module provides the native stream class StreamMLProStaticClouds3D. This stream provides 2000 instances with 3-dimensional random feature data placed around eight fixed center points.

class mlpro.bf.streams.streams.clouds3d_static.**StreamMLProStaticClouds3D**(*p_logging=True*)

Bases: StreamMLProBase

This demo stream provides 2000 3-dimensional instances randomly positioned around four fixed centers.

C_ID = 'StaticClouds3D'

C_NAME = 'Static Clouds 3D'

C_VERSION = '1.0.0'

C_NUM_INSTANCES = 2000

C_SCIREF_ABSTRACT = 'Demo stream provides 1000 2-dimensional instances randomly positioned around four fixed centers.'

C_BOUNDARIES = [-10, 10]

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

2D Double Spiral

Ver. 1.0.0 (2022-12-14)

This module provides the native stream class DoubleSpiral2D. It provides 721 instances with 2-dimensional feature data that follow a double spiral pattern.

class mlpro.bf.streams.streams.doublespiral2d.**DoubleSpiral2D**(*p_logging=True*)

Bases: StreamMLProBase

C_ID = 'DoubleSpiral2D'

C_NAME = 'Double Spiral 2D x 721'

C_VERSION = '1.0.0'

C_NUM_INSTANCES = 721

C_SCIREF_ABSTRACT = 'This benchmark test generates 721 2-dimensional inputs positioned in a double spiral.'

C_BOUNDARIES = [-10, 10]

10D Random Instances

Ver. 1.0.0 (2022-12-13)

This module provides the native stream class StreamMLProRnd10D. This stream provides 1000 instances with 10-dimensional random feature data and 2-dimensional random label data.

class mlpro.bf.streams.streams.rnd10d.**StreamMLProRnd10D**(*p_logging=True*)

Bases: StreamMLProBase

Demo stream consisting of 1000 instances with 10-dimensional random feature data and 2-dimensional label data. All values are in range defined by attribute C_BOUNDARIES.

C_NUM_INSTANCES = 1000

Number of instances.

C_BOUNDARIES = [-10,10]

Boundaries for all random values.

C_ID = 'Rnd10Dx1000'

C_NAME = 'Random 10D x 1000'

C_VERSION = '1.0.0'

C_NUM_INSTANCES = 1000

C_SCIREF_ABSTRACT = 'Demo stream of 1000 instances with 10-dimensional random feature data and 2-dimensional label data.'

C_BOUNDARIES = [-10, 10]

set_random_seed(*p_seed=None*)

Resets the internal random generator using the given seed.

Stream Tasks

Deriver

Ver. 1.0.0 (2023-02-05)

This module provides a stream task class Deriver to derive the data of instances.

class mlpro.bf.streams.tasks.deriver.**Deriver**(*p_name: str | None = None, p_range_max=1, p_duplicate_data: bool = False, p_visualize: bool = False, p_logging=True, p_features: list | None = None, p_labels: list | None = None, p_derived_feature:* Feature *| None = None, p_derived_label:* Label *| None = None, p_order_derivative: int = 1, **p_kwargs*)

Bases: *StreamTask*

This stream task extend the feature and label data of incoming instances with a derivation of a pre-selected feature.

Parameters

- **p_name** (*str*) – Optional name of the task. Default is None.
- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_duplicate_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_features** (*list*) – The list of current features in the stream. Default = None.
- **p_labels** (*list*) – The list of current labels in the stream. Default = None.
- **p_derived_feature** (Feature) – A pre-selected feature that would like to be derived. Default = None.
- **p_derived_label** (Feature) – A correspondig label of the derived feature. Default = None.
- **p_order_derivative** (*int*) – The derivative of order n. Default = None.
- **p_kwargs** (*dict*) – Further optional named parameters.

C_NAME = 'Deriver'

C_PLOT_STANDALONE: bool = True

class mlpro.bf.streams.tasks.deriver.**DerivativeFunction**(*p_name: str*, *p_id: int | None = None*, *p_type: int | None = None*, *p_unit_in: str | None = None*, *p_unit_out: str | None = None*, *p_dt: float = 0.01*, *p_logging=True*, ***p_args*)

Bases: *TransferFunction*

Rearranger

Ver. 1.0.3 (2022-12-19)

This module provides a stream task class Rearranger to rearrange the feature and label space of instances.

class mlpro.bf.streams.tasks.rearranger.**Rearranger**(*p_name: str | None = None*, *p_range_max=1*, *p_duplicate_data: bool = False*, *p_visualize: bool = False*, *p_logging=True*, *p_features_new: list = []*, *p_labels_new: list = []*, ***p_kwargs*)

Bases: *StreamTask*

This stream task rearrange the feature and/or label data of incoming instances. To this regard, two additional parameters p_features_new and p_labels_new describe the dimensions of the feature/label space of the resulting instances.

Parameters

- **p_name** (*str*) – Optional name of the task. Default is None.
- **p_range_max** (*int*) – Maximum range of asynchonicity. See class Range. Default is Range.C_RANGE_PROCESS.
- **p_duplicate_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_features_new** (*list*) – List of resulting features that are described as tuples ('F' or 'L', list[Dimension]). The first component specifies the origin ('F' = feature space, 'L' = label space). The second component is a list of dimension objects.
- **p_labels_new** (*list[Label]*) – List of resulting labels that are described as tuples ('F' or 'L', list[Dimension]).
- **p_kwargs** (*dict*) – Further optional named parameters.

C_NAME = 'Rearranger'

C_PLOT_STANDALONE: bool = True

Windows

Ver. 1.1.5 (2023-02-02)

This module provides pool of window objects further used in the context of online adaptivity.

class mlpro.bf.streams.tasks.windows.**Window**(*p_buffer_size: int*, *p_delay: bool = False*, *p_enable_statistics: bool = False*, *p_name: str | None = None*, *p_range_max=1*, *p_duplicate_data: bool = False*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *StreamTask*

This is the base class for window implementations

Parameters

- **p_buffer_size** (*int*) – the size/length of the buffer/window.
- **p_delay** (*bool, optional*) – Set to true if full buffer is desired before passing the window data to next step. Default is false.
- **p_name** (*str, optional*) – Name of the Window. Default is None.
- **-Optional** (*p_logging*) – Maximum range of task parallelism for window task. Default is set to multithread.
- **p_duplicate_data** (*bool*) – If True, instances will be duplicated before processing. Default = False.
- **p_ada** (*bool, optional*) – Adaptivity property of object. Default is True.
- **-Optional** – Log level for the object. Default is log everything.

C_NAME = 'Window'

C_PLOT_STANDALONE: bool = False

C_PLOT_IN_WINDOW = 'In Window'

C_PLOT_OUTSIDE_WINDOW = 'Out Window'

C_EVENT_BUFFER_FULL = 'BUFFER_FULL'

C_EVENT_DATA_REMOVED = 'DATA_REMOVED'

get_buffered_data()

Method to fetch the date from the window buffer

Returns

- **buffer** (*dict*) – the buffered data in the form of dictionary
- **buffer_pos** (*int*) – the latest buffer position

get_boundaries()

Method to get the current boundaries of the Window

Returns
boundaries – Returns the current window boundaries in the form of a Numpy array.

Return type
np.ndarray

`get_mean()`

Method to get the mean of the data in the Window.

Returns
mean – Returns the mean of the current data in the window in the form of a Numpy array.

Return type
np.ndarray

`get_variance()`

Method to get the variance of the data in the Window.

Returns
variance – Returns the variance of the current data in the window as a numpy array.

Return type
np.ndarray

`get_std_deviation()`

Method to get the standard deviation of the data in the window.

Returns
std – Returns the standard deviation of the data in the window as a numpy array.

Return type
np.ndarray

BF-SYSTEMS - Systems

Double Pendulum System

The Double Pendulum System is an implementation of a classic control problem of Double Pendulum system. The dynamics of the system are based on the Double Pendulum implementation by Matplotlib. The double pendulum is a system of two poles, with the inner pole connected to a fixed point at one end and to outer pole at other end. The native implementation of Double Pendulum consists of an input motor providing the torque in either directions to actuate the system.

class mlpro.bf.systems.pool.doublependulum.**DoublePendulumSystemRoot**(*p_mode=0*, *p_latency=None*, *p_max_torque=20*, *p_l1=1.0*, *p_l2=1.0*, *p_m1=1.0*, *p_m2=1.0*, *p_init_angles='random'*, *p_g=9.8*, *p_fct_strans: FctSTrans | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_frame_skip=None*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf=None*, *p_history_length=5*, *p_visualize: bool = False*, *p_random_range: list | None = None*, *p_balancing_range: list = (-0.2, 0.2)*, *p_swinging_outer_pole_range=(0.2, 0.5)*, *p_break_swinging: bool = False*, *p_logging=True*)

Bases: *System*

This is the root double pendulum environment class inherited from Environment class with four dimensional state space and underlying implementation of the Double Pendulum dynamics, default reward strategy.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_max_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p_init_angles** (*str, optional*) – C_ANGLES_UP starts the pendulum in an upright position C_ANGLES_DOWN starts the pendulum in a downward position

C_ANGLES_RND starts the pendulum from a random position.

- **p_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p_history_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p_fct_strans** (FctSTrans, *optional*) – The custom State transition function.
- **p_fct_success** (FctSuccess, *optional*) – The custom Success Function.
- **p_fct_broken** (FctBroken, *optional*) – The custom Broken Function.
- **p_mujoco_file** (*optional*) – The corresponding mujoco file
- **p_frame_skip** (*optional*) – Number of frames to be skipped for visualization.
- **p_state_mapping** (*optional*) – State mapping configurations.
- **p_action_mapping** (*optional*) – Action mapping configurations.
- **p_camera_conf** (*optional*) – Camera configurations for mujoco specific visualization.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_random_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p_balancing*) – The boundaries for state space of environment in balancing region
- **p_swinging_outer_pole_range** – The boundaries for state space of environment in swinging of outer pole region
- **p_break_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_NAME = 'DoublePendulumSystemRoot'

C_SCIREF_TYPE = 'Online'

C_SCIREF_AUTHOR = 'John Hunter, Darren Dale, Eric Firing, Michael Droettboom and the Matplotlib development team'

C_SCIREF_TITLE = 'The Double Pendulum Problem'

C_SCIREF_URL = 'https://matplotlib.org/stable/gallery/animation/double_pendulum.html'

C_PLOT_ACTIVE: bool = True

C_PLOT_DEFAULT_VIEW: str = '2D'

C_CYCLE_LIMIT = 0

C_LATENCY = datetime.timedelta(microseconds=40000)

C_ANGLES_UP = 'up'

C_ANGLES_DOWN = 'down'

C_ANGLES_RND = 'random'

C_VALID_ANGLES = ['up', 'down', 'random']

C_THRSH_GOAL = 0

C_ANI_FRAME = 30

C_ANI_STEP = 0.001

setup_spaces()

Method to setup the spaces for the Double Pendulum root environment. This method sets up four dimensional Euclidean space for the root DP environment.

class mlpro.bf.systems.pool.doublependulum.**DoublePendulumSystemS4**(*p_mode=0*, *p_latency=None*, *p_max_torque=20*, *p_l1=1.0*, *p_l2=1.0*, *p_m1=1.0*, *p_m2=1.0*, *p_init_angles='random'*, *p_g=9.8*, *p_fct_strans: FctSTrans | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_frame_skip=None*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf=None*, *p_history_length=5*, *p_visualize: bool = False*, *p_random_range: list | None = None*, *p_balancing_range: list = (-0.2, 0.2)*, *p_swinging_outer_pole_range=(0.2, 0.5)*, *p_break_swinging: bool = False*, *p_logging=True*)

Bases: *DoublePendulumSystemRoot*

This is the Double Pendulum Static 4 dimensional environment that inherits from the double pendulum root class, inheriting the dynamics and default reward strategy.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_max_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25

- **p_init_angles** (*str, optional*) – C_ANGLES_UP starts the pendulum in an upright position C_ANGLES_DOWN starts the pendulum in a downward position C_ANGLES_RND starts the pendulum from a random position.
- **p_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p_history_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_plot_level** (*int*) – Types and number of plots to be plotted. Default = ALL C_PLOT_DEPTH_ENV only plots the environment C_PLOT_DEPTH_REWARD only plots the reward C_PLOT_ALL plots both reward and the environment
- **p_rst_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p_rst_swinging** – Reward strategy to be used for the swinging region of the environment
- **p_reward_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p_reward_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p_reward_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p_random_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p_balancing*) – The boundaries for state space of environment in balancing region
- **p_break_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_NAME = 'DoublePendulumSystemS4'

class mlpro.bf.systems.pool.doublependulum.**DoublePendulumSystemS7**(*p_mode=0*, *p_latency=None*, *p_max_torque=20*, *p_l1=1.0*, *p_l2=1.0*, *p_m1=1.0*, *p_m2=1.0*, *p_init_angles='random'*, *p_g=9.8*, *p_fct_strans: FctSTrans | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_mujoco_file=None*, *p_frame_skip=None*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf=None*, *p_history_length=5*, *p_visualize: bool = False*, *p_random_range: list | None = None*, *p_balancing_range: list = (-0.2, 0.2)*, *p_swinging_outer_pole_range=(0.2, 0.5)*, *p_break_swinging: bool = False*, *p_logging=True*)

Bases: *DoublePendulumSystemS4*

This is the classic implementation of Double Pendulum with 7 dimensional state space including derived accelerations of both the poles and the input torque. The dynamics of the system are inherited from the Double Pendulum Root class.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_max_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p_init_angles** (*str, optional*) – C_ANGLES_UP starts the pendulum in an upright position C_ANGLES_DOWN starts the pendulum in a downward position C_ANGLES_RND starts the pendulum from a random position.
- **p_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p_history_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.

- **p_plot_level** (*int*) – Types and number of plots to be plotted. Default = ALL C_PLOT_DEPTH_ENV only plots the environment C_PLOT_DEPTH_REWARD only plots the reward C_PLOT_ALL plots both reward and the environment
- **p_rst_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p_rst_swinging** – Reward strategy to be used for the swinging region of the environment
- **p_reward_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p_reward_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p_reward_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p_random_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p_balancing*) – The boundaries for state space of environment in balancing region
- **p_break_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_NAME = 'DoublePendulumSystemS7'

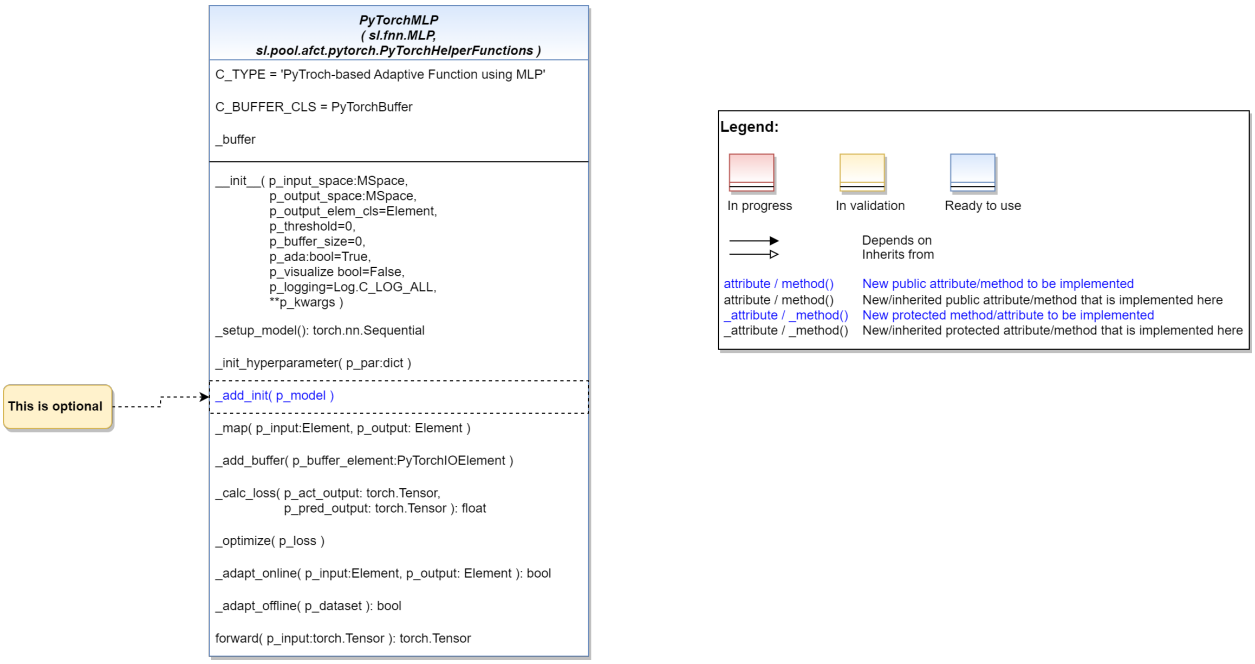
setup_spaces()

Method to set up the state and action spaces of the classic Double Pendulum Environment. Inheriting from the root class, this method adds 3 dimensions for accelerations and torque respectively.

9.2.2 MLPro-SL - Supervised Learning

Adaptive Functions

PyTorch-based MLP



Ver. 1.1.0 (2023-03-10)

This module provides a template ready-to-use MLP model using PyTorch.

class mlpro.sl.pool.afct.fnn.pytorch.mlp.**PyTorchMLP**(*p_input_space: ~mlpro.bf.math.basics.MSpace*, *p_output_space: ~mlpro.bf.math.basics.MSpace*, *p_output_elem_cls=<class 'mlpro.bf.math.basics.Element'>*, *p_threshold=0*, *p_buffer_size=0*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, ***p_kwargs*)

Bases: *MLP*, *PyTorchHelperFunctions*

Template class for an adaptive bi-multivariate mathematical function that adapts by supervised learning using PyTorch-based MLP.

Parameters

- **p_input_space** (MSpace) – Input space of function
- **p_output_space** (MSpace) – Output space of function
- **p_output_elem_cls** – Output element class (compatible to/inherited from class Element)
- **p_threshold** (*float*) – Threshold for the difference between a setpoint and a computed output. Computed outputs with a difference less than this threshold will be assessed as 'good' outputs. Default = 0.
- **p_buffer_size** (*int*) – Initial size of internal data buffer. Default = 0 (no buffering).
- **p_ada** (*bool*) – Boolean switch for adaptivity. Default = True.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL
- **p_kwargs** (*Dict*) – Further model specific parameters (to be specified in child class).

C_TYPE = 'PyTorch-based Adaptive Function using MLP'

C_BUFFER_CLS

alias of *PyTorchBuffer*

forward(*p_input: Tensor*) → Tensor

Forward propagation in neural networks to generate some output using PyTorch.

Parameters

p_input (Element) – Input data

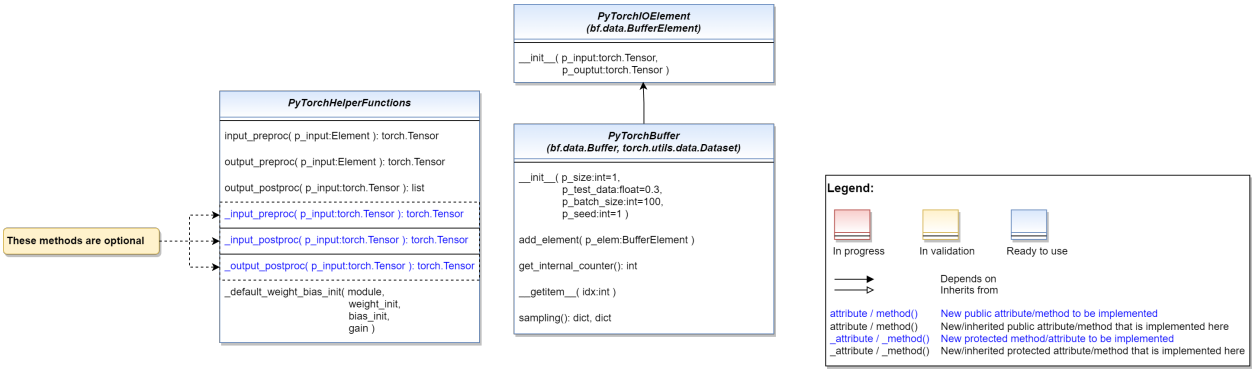
Returns

output – Output data

Return type

Element

PyTorch Helper Functions



Ver. 3.0.2 (2023-03-10)

This a helper module for supervised learning models using PyTorch.

class mlpro.sl.pool.afct.pytorch.**PyTorchIOElement**(*p_input: Tensor*, *p_output: Tensor*)

Bases: *BufferElement*

This class provides a buffer element for PyTorch based SLNetwork.

Parameters

- **p_input** (Element) – Abscissa/input element object (type Element)
- **p_output** (Element) – Setpoint ordinate/output element (type Element)

class mlpro.sl.pool.afct.pytorch.**PyTorchBuffer**(*p_size: int = 1*, *p_test_data: float = 0.3*, *p_batch_size: int = 100*, *p_seed: int = 1*)

Bases: *Buffer*, Dataset

This class provides buffer functionalities for PyTorch based SLNetwork and also using several built-in PyTorch functionalities.

Parameters

- **p_size** (*int*) – the buffer size. Default = 1.

- **p_test_data** (*float*) – the proportion of testing data within the sampled data. Default = 0.3.
- **p_batch_size** (*int*) – the batch size for a sample. Default = 100.
- **p_seed** (*int*) – the seeding for randomizer in the buffer, optional. Default = 1.

add_element(*p_elem:* BufferElement)

This method has a functionality to add an element to the buffer.

Parameters
p_elem (BufferElement) – an element of the buffer

get_internal_counter() → int

This method has a functionality to get the number of elements being added to the buffer.

sampling()

This method has a functionality to sample from the buffer using built-in PyTorch functionalities.

class mlpro.sl.pool.afct.pytorch.**PyTorchHelperFunctions**

Bases: object

PyTorch Helper Functions in MLPro-SL.

input_preproc(*p_input:* Element) → Tensor

This method has a functionality to transform input data in the form of Element to torch.Tensor for pre-processing.

Parameters
p_input (Element) – Input data in the form of Element.

Returns
input – Input data in the form of torch.Tensor.

Return type
torch.Tensor

output_preproc(*p_output:* Element) → Tensor

This method has a functionality to transform output data in the form of Element to torch.Tensor for pre-processing.

Parameters
p_output (Element) – Output data in the form of Element.

Returns
output – Output data in the form of torch.Tensor.

Return type
torch.Tensor

output_postproc(*p_output: Tensor*) → list

This method has a functionality to transform output data in the form of torch.Tensor to a list for post-processing.

Parameters
p_output (*torch.Tensor*) – Output data in the form of torch.Tensor.

Returns
output – Output data in the form of list.

Return type
list

9.2.3 MLPro-RL - Reinforcement Learning

Action Planners

MPC - Model Predictive Control

Ver. 1.1.1 (2023-02-04)

This module provides a default implementation of model predictive control (MPC).

class mlpro.rl.pool.actionplanner.mpc.**MPC**(*p_range_max=0*, *p_state_thsld=1e-08*, *p_logging=True*)

Bases: *ActionPlanner*, *ScientificObject*, *Async*

Template class for MPC to be used as part of model-based planning agents. The goal is to find the best sequence of actions that leads to a maximum reward.

Parameters

- **p_range** (*int*) – Range of asynchonicity.
- **p_state_thsld** (*float*) – Threshold for metric difference between two states to be equal. Default = 0.00000001.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Model Predictive Control'

execute(***p_kwargs*)

Environments

Bulk Goods Plant

Ver. 2.3.0 (2023-02-22)

This module provides an RL environment of Bulk Good Laboratory Plant (BGLP).

class mlpro.rl.pool.envs.bglp.**Actuator**(*minpower*, *maxpower*, *minaction*, *maxaction*, *masscoeff*)

Bases: object

This class serves as a parent class of different types of actuators, which provides the main attributes of an actuator in the BGLP environment.

Parameters

- **minpower** (*float*) – minimum power of an actuator.
- **maxpower** (*float*) – maximum power of an actuator.
- **minaction** (*float*) – minimum action of an actuator.
- **maxaction** (*float*) – maximum action of an actuator.
- **masscoeff** (*float*) – mass transport coefficient of an actuator.

reg_a

list of existing actuators in the environment.

Type
list of objects

`idx_a`

length of reg_a.

Type
int

`power_max`

maximum power of an actuator.

Type
float

`power_min`

minimum power of an actuator.

Type
float

`power_coeff`

power coefficient of an actuator, if necessary.

Type
float

`action_max`

maximum action of an actuator.

Type
float

`action_min`

minimum action of an actuator.

Type
float

`mass_coeff`

mass transport coefficient of an actuator.

Type
float

`t_activated`

a time indicator about an actuator is activated.

Type
float

`t_end`

a time indicator about the end of an activation sequence of the actuator.

Type
float

`status`

status of an actuator, false means inactive and true means active.

Type
bool

`cur_mass_transport`

current transported mass of an actuator.

Type
float

`cur_power`

current power consumption of an actuator.

Type
float

`cur_action`

current taken action of an actuator in RL context.

Type
float

`cur_speed`

current speed of an actuator.

Type
float

`type_`

a short name for an actuator, usually 3 capital letters (e.g VAC, BLT).

Type
str

reg_a = [<mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>]

power_coeff = 0

t_activated = 0

t_end = 0

status = False

type_ = ''

idx_a = 0

power_max = 0

power_min = 0

action_min = 0

action_max = 0

mass_coeff = 0

cur_mass_transport = 0

cur_power = 0

cur_action = 0

cur_speed = 0

class mlpro.rl.pool.envs.bglp.**VacuumPump**(*name*, *minpower*, *maxpower*, *minaction*, *maxaction*, *masscoeff*)

Bases: *Actuator*

This class inherits Actuator class and serves as a child class of Actuator. This class represents a type of actuators in the BGLP environment, namely Vacuum Pump. Vacuum Pumps are mostly used to transport material from mini hoppers to silos. However, the parameter of each vacuum pump can be dissimilar to each other based on their settings.

Parameters

- **name** (*str*) – specific name or id of a vacuum pump (e.g. Vac_A, etc.).
- **minpower** (*float*) – minimum power of a vacuum pump.
- **maxpower** (*float*) – maximum power of a vacuum pump.
- **minaction** (*float*) – minimum action of a vacuum pump.
- **maxaction** (*float*) – maximum action of a vacuum pump.
- **masscoeff** (*float*) – mass transport coefficient of a vacuum pump.

reg_v

list of existing vacuum pumps.

Type
list of objects

idx_v

length of reg_v.

Type
int

name

specific name or id of a vacuum pump.

Type
str

t_end_max

maximum end of activation time of a vacuum pump with respect to current time.

Type
float

`reg_v = [<mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>, <mlpro.rl.pool.envs.bglp.VacuumPump object>]`

`t_end_max = 0`

`idx_v = 0`

`name = ''`

`start_t`(*now*, *duration*, *overwrite=False*)

This method calculates the activation time and the end of activation time of the vacuum pump. This method is called, if the vacuum pump would like to be activated or updated.

Parameters

- **now** (*float*) – current time of the system.
- **duration** (*float*) – duration of the vacuum pump being activated or the action by an agent in RL context.
- **overwrite** (*bool, optional*) – To indicate whether the current operation can be overwritten or not.

`calc_mass`(*now*)

This method calculates the transported mass flow by the vacuum pump for a time step.

Parameters

now (*float*) – current time of the system.

Returns

cur_mass_transport – current transported mass.

Return type

float

`calc_power`()

This method calculates the power consumption of a vacuum pump.

Returns

cur_power – current power consumption.

Return type

float

update(*now*)

This method calculates whether a vacuum pump must be deactived or not.

Parameters
now (*float*) – current time of the system.

deactivate()

This method is used to deactivate a vacuum pump.

class mlpro.rl.pool.envs.bglp.**Belt**(*name*, *actiontype*, *minpower*, *maxpower*, *minaction*, *maxaction*, *masscoeff*)

Bases: *Actuator*

This class inherits Actuator class and serves as a child class of Actuator. This class represents a type of actuators in the BGLP environment, namely Belt. This class can be used for Conveyor Belt, Rotary Feeder, Vibratory Conveyor, or similar type of actuators. Belts are mostly used to transport material from silos to hoppers. However, the parameter of each actuator can be dissimilar to each other based on their settings.

Parameters

- **name** (*str*) – specific name or id of an actuator (e.g. Belt_A, etc.).
- **actiontype** (*str*) – "C" for continuous action, "B" for binary action.
- **minpower** (*float*) – minimum power of an actuator.
- **maxpower** (*float*) – maximum power of an actuator.
- **minaction** (*float*) – minimum action of an actuator.
- **maxaction** (*float*) – maximum action of an actuator.
- **masscoeff** (*float*) – mass transport coefficient of an actuator.

reg_b

list of existing actuators.

Type
list of objects

idx_b

length of reg_b.

Type
int

name

specific name or id of an actuator.

Type
str

actiontype

"C" for continuous action, "B" for binary action.

Type
str

speed

speed of an actuator.

Type
float

reg_b = [<mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>, <mlpro.rl.pool.envs.bglp.Belt object>]

idx_b = 0

name = ''

actiontype = ''

speed = 0

start_t(*now*, *duration*, *speed*, *overwrite=False*)

This method calculates the activation time and the end of activation time of the belt. This method is called, if the belt would like to be activated or updated.

Parameters

- **now** (*float*) – current time of the system.
- **duration** (*float*) – duration of the belt being activated, which being defined by the time set.
- **speed** (*float*) – speed of the belt or the action by an agent in RL context.
- **overwrite** (*bool, optional*) – To indicate whether the current operation can be overwritten or not.

calc_mass(*now*)

This method calculates the transported mass flow by the belt for a time step.

Parameters

now (*float*) – current time of the system.

Returns

cur_mass_transport – current transported mass.

Return type

float

calc_power()

This method calculates the power consumption of a belt.

Returns
cur_power – current power consumption.

Return type
float

`update`(*now*)

This method calculates whether a belt must be deactived or not.

Parameters
now (*float*) – current time of the system.

`deactivate`()

This method is used to deactivate a belt.

`class mlpro.rl.pool.envs.bglp.Reservoir`(*vol_max*, *vol_init_abs=0*)

Bases: `object`

This class serves as a parent class of different types of reservoirs, which provides the main attributes of a buffer in the BGLP environment.

Parameters

- **vol_max** (*float*) – maximum volume of a reservoir.
- **vol_init_abs** (*float, optional*) – initial volume of a reservoir. The default is 0.

`reg_r`

list of existing reservoirs in the environment.

Type
list of objects

`idx_r`

length of reg_r.

Type
int

`vol_max`

maximum volume of a reservoir.

Type
float

`vol_init_abs`

initial volume of a reservoir.

Type
float

`vol_cur_abs`

current volume of a reservoir.

Type
float

`vol_cur_rel`

current volume of a reservoir in percentage.

Type
float

change

volume change of a reservoir in a time step.

Type

float

```
reg_r = [<mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo
object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Hopper
object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper
object>]
```

`idx_r = []`

`vol_max = 0`

`vol_init_abs = 0`

`vol_cur_abs = 0`

`vol_cur_rel = 0`

`change = 0`

set_change(*vol_change*)

This method sets up a volume change of a reservoir.

Parameters
vol_change (*float*) – volume change of a reservoir in a time step.

update()

This method calculates the current volume of a reservoir after volume changes are made.

class mlpro.rl.pool.envs.bglp.**Silo**(*name*, *vol_max*, *vol_cur=0*, *mode='abs'*)

Bases: *Reservoir*

This class inherits Reservoir class and serves as a child class of Reservoir. This class represents a type of buffers in the BGLP environment, namely Silo. Silos are used to temporary stored the transported materials. However, the parameter of each silo can be dissimilar to each other based on their settings.

Parameters

- **name** (*str*) – specific name or id of a silo (e.g. Silo_A, etc.).
- **vol_max** (*float*) – maximum capacity of a silo.
- **vol_cur** (*float*) – current volume of a silo.
- **mode** (*str, optional*) – mode of measuring the current volume. "abs" means absolute value, "rel" means percentage. The default is "abs".

reg_s

list of existing silos.

Type
list of objects

idx_s

length of reg_s.

Type
int

name

specific name or id of a silo.

Type
str

type_

a short name for a silo, usually 3 capital letters (e.g SIL).

Type
str

reg_s = [<mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>, <mlpro.rl.pool.envs.bglp.Silo object>]

name = ''

idx_s = []

type_ = ''

class mlpro.rl.pool.envs.bglp.**Hopper**(*name*, *vol_max*, *vol_cur=0*, *mode='abs'*)

Bases: `Reservoir`

This class inherits Reservoir class and serves as a child class of Reservoir. This class represents a type of buffers in the BGLP environment, namely Hopper. Hoppers are used to temporary stored the transported materials. However, the parameter of each hopper can be dissimilar to each other based on their settings.

Parameters

- **name** (*str*) – specific name or id of a hopper (e.g. Hop_A, etc.).
- **vol_max** (*float*) – maximum capacity of a hopper.
- **vol_cur** (*float*) – current volume of a hopper.
- **mode** (*str, optional*) – mode of measuring the current volume. “abs” means absolute value, “rel” means percentage. The default is “abs”.

reg_h

list of existing silos.

Type

list of objects

idx_h

length of reg_h.

Type

int

name

specific name or id of a hopper.

Type
str

type_

a short name for a hopper, usually 3 capital letters (e.g HOP).

Type
str

reg_h = [<mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>, <mlpro.rl.pool.envs.bglp.Hopper object>]

name = ''

idx_h = []

type_ = ''

class mlpro.rl.pool.envs.bglp.**BGLP**(*p_reward_type=1*, *p_visualize: bool = False*, *p_logging=True*, *t_step=0.5*, *t_set=10.0*, *demand=0.1*, *lr_margin=1.0*, *lr_demand=4.0*, *lr_power=0.001*, *margin_p=[0.2, 0.8, 4]*, *prod_target=10000*, *prod_scenario='continuous'*, *cycle_limit=0*)

Bases: *Environment*

This is the main class of BGLP environment that inherits Environment class from MLPro.

Parameters

- **p_reward_type** (Reward, *optional*) – rewarding type. The default is Reward.C_TYPE_EVERY_AGENT.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** (Log, *optional*) – logging functionalities. The default is Log.C_LOG_ALL.
- **t_step** (*float, optional*) – time for each time step (in seconds). The default is 0.5.
- **t_set** (*float, optional*) – time set for one horizon (in seconds). The default is 10.0.
- **demand** (*float, optional*) – the constant output flow from the system (in L/s). The default is 0.1.
- **lr_margin** (*float, optional*) – the learning rate for margin parameter, related to implemented reward function. The default is 1.0.

- **lr_demand** (*float, optional*) – the learning rate for production demand, related to implemented reward function. The default is 4.0.
- **lr_power** (*float, optional*) – the learning rate for power consumption, related to implemented reward function. The default is 0.0010.
- **margin_p** (*list of floats, optional*) – the margin parameter of reservoirs [low, high, multplicator]. The default is [0.2,0.8,4].
- **prod_target** (*float, optional*) – the production target for batch operation (in L). The default is 10000.
- **prod_scenario** (*str, optional*) – 'batch' means batch production scenario and 'continuous' means continuous production scenario. The default is 'continuous'.

C_NAME

name of the environment.

Type
str

C_LATENCY

latency.

Type
timedelta()

C_INFINITY

infinity.

Type
np.finfo()

C_REWARD_TYPE

rewarding type.

Type
Reward

sils

list of existing silos.

Type
list of objects

hops

list of existing hoppers.

Type
list of objects

ress

list of existing reservoirs.

Type
list of objects

blts

list of existing belts.

Type
list of objects

`acts`
: list of existing actuators.

  Type
  list of objects

`vacs`
: list of existing vacuum pumps.

  Type
  list of objects

`con_act_to_res`
: connection between actuators and reservoirs and setup the sequence.

  Type
  list of lists of int

`m_param`
: the margin parameter of reservoirs [low, high, multplicator].

  Type
  list of floats

`_demand`
: the constant output flow from the system (in L/s).

  Type
  float

`t`
: current time of the system (in seconds).

  Type
  float

`t_step`
: time for each time step (in seconds).

  Type
  float

`lr_margin`
: the learning rate for margin parameter.

  Type
  float

`lr_demand`
: the learning rate for production demand.

  Type
  float

`lr_power`
: the learning rate for power consumption.

  Type
  float

`overflow`

current overflow.

Type
list of floats

`power`

current power consumption.

Type
list of floats

`transport`

current transported mass flow.

Type
list of floats

`reward`

current rewards.

Type
list of floats

`levels_init`

initial level of reservoirs.

Type
float

`overflow_t`

current overflow for a specific buffer in a specific time.

Type
float

`demand_t`

current demand for a specific buffer in a specific time.

Type
float

`power_t`

current power consumption for a specific actuator in a specific time.

Type
float

`transport_t`

current transported material by a specific actuaor in a specific time.

Type
float

`margin_t`

current margin for a specific buffer in a specific time.

Type
float

prod_reached

current production reached in L for batch operation.

Type
float

prod_target

the production target for batch operation (in L).

Type
float

prod_scenario

'batch' means batch production scenario and 'continuous' means continuous production scenario.

Type
str

cycle_limit

the number of cycle limit.

Type
int

```
C_NAME = 'BGLP'

C_LATENCY = datetime.timedelta(seconds=1)

C_INFINITY = 3.4028235e+38

C_REWARD_TYPE = 1

con_act_to_res = []

m_param = []

prod_reached = 0

C_CYCLE_LIMIT = 0

t = 0

t_step = 0

lr_margin = 0

lr_demand = 0

lr_power = 0

prod_target = 0

prod_scenario = 0

levels_init = 0

sils = []

hops = []

ress = []
```

blts = []

vacs = []

acts = []

overflow = []

demand = []

power = []

transport = []

overflow_t = 0

demand_t = 0

power_t = 0

transport_t = 0

margin_t = 0

reward = []

static setup_spaces()

This method is used to setup action and state spaces of the system.

The actions and states are normalized between 0 to 1. For the actions, 0 means minimum action and 1 means maximum action. Meanwhile, for the states, 0 means minimum capacity (empty) and 1 means maximum capacity (full)

Returns

- **state_space** (*ESpace()*) – state space of the system.
- **action_space** (*ESpace()*) – action space of the system.

collect_substates() → *State*

This method is called during resetting the environment.

Returns
state – current states.

Return type
State

calc_mass_flows()

This method calculates the mass flow transported by actuators.

calc_power()

This method calculates the power consumptions per actuator.

calc_margin()

This method calculates margin level for every reservoir.

set_volume_changes(*demandval*)

This method sets up volume changes for every reservoir.

`update_levels()`

This method updates the current level of reservoirs.

`reset_levels()`

This method resets reservoirs.

`reset_actuators()`

This method resets actuators.

`update_actuators()`

This method updates actuators.

`update`(*demandval*)

This method sets up volume changes, updates reservoirs' level, and updates actuators.

Parameters
demandval (*float*) – production outflow target in L/s.

`get_status`(*t*, *demandval*)

This method calculates overflow, demand, power, transport, and margin. This function will be called every time step.

Parameters

- **t** (*float*) – current time in sec.
- **demandval** (*float*) – production outflow target in L/s.

Returns

- **overflow** (*list of floats*) – overflow levels.
- **demand** (*list of floats*) – demand fulfilled.
- **power** (*list of floats*) – power consumptions.
- **transport** (*list of floats*) – transported materials.
- **margin** (*list of floats*) – margin levels.

`set_actions`(*actions*)

This method sets up actions for actuators. This function will be called every time set.

`calc_state()`

This method obtains current levels of reservoirs.

Returns
levels – level of each reservoir.

Return type
list of floats

`calc_reward()`

This method calculates the reward. This method can be redifined! The current reward function is suitable for continuous operation and scalar reward for individual agents.

Returns
reward – reward for each agent.

Return type
list of floats

init_plot(*p_figure=None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot()

Updates the plot.

Parameters

****p_kwargs** – Implementation-specific plot data and/or parameters.

Double Pendulum

The Double Pendulum environment is an implementation of a classic control problem of Double Pendulum system. The dynamics of the system are based on the Double Pendulum implementation by Matplotlib. The double pendulum is a system of two poles, with the inner pole connected to a fixed point at one end and to outer pole at other end. The native implementation of Double Pendulum consists of an input motor providing the torque in either directions to actuate the system.

class mlpro.rl.pool.envs.doublependulum.**DoublePendulumRoot**(*p_mode=0*, *p_latency=None*, *p_max_torque=20*, *p_l1=1.0*, *p_l2=1.0*, *p_m1=1.0*, *p_m2=1.0*, *p_g=9.8*, *p_init_angles='random'*, *p_history_length=5*, *p_fct_strans: FctSTrans | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_fct_reward: FctReward | None = None*, *p_mujoco_file=None*, *p_frame_skip=None*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf=None*, *p_visualize: bool = False*, *p_plot_level: int = 2*, *p_rst_balancing='rst_002'*, *p_rst_swinging='rst_003'*, *p_rst_swinging_outer_pole='rst_004'*, *p_reward_weights: list | None = None*, *p_reward_trend: bool = False*, *p_reward_window: int = 0*, *p_random_range: list | None = None*, *p_balancing_range: list = (-0.2, 0.2)*, *p_swinging_outer_pole_range=(0.2, 0.5)*, *p_break_swinging: bool = False*, *p_logging=True*)

Bases: *DoublePendulumSystemRoot*, *Environment*

This is the root double pendulum environment class inherited from Environment class with four dimensional state space and underlying implementation of the Double Pendulum dynamics, default reward strategy.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_max_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p_init_angles** (*str, optional*) – C_ANGLES_UP starts the pendulum in an upright position C_ANGLES_DOWN starts the pendulum in a downward position

C_ANGLES_RND starts the pendulum from a random position.

- **p_history_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p_fct_strans** (FctSTrans, *optional*) – The custom State transition function.
- **p_fct_success** (FctSuccess, *optional*) – The custom Success Function.
- **p_fct_broken** (FctBroken, *optional*) – The custom Broken Function.
- **p_fct_reward** (FctReward, *optional*) – The custom Reward Function.
- **p_mujoco_file** (*optional*) – The corresponding mujoco file
- **p_frame_skip** (*optional*) – Number of frames to be skipped for visualization.
- **p_state_mapping** (*optional*) – State mapping configurations.
- **p_action_mapping** (*optional*) – Action mapping configurations.
- **p_camera_conf** (*optional*) – Camera configurations for mujoco specific visualization.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_plot_level** (*int*) – Types and number of plots to be plotted. Default = ALL C_PLOT_DEPTH_ENV only plots the environment C_PLOT_DEPTH_REWARD only plots the reward C_PLOT_ALL plots both reward and the environment
- **p_rst_balancing** – Reward strategy to be used for the balancing region of the environment
- **p_rst_swinging** – Reward strategy to be used for the swinging region of the environment
- **p_rst_swinging_outer_pole** – Reward Strategy to be used for swinging up outer pole
- **p_reward_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p_reward_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p_reward_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p_random_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p_balancing*) – The boundaries for state space of environment in balancing region
- **p_swinging_outer_pole_range** – The boundaries for state space of environment in swinging of outer pole region
- **p_break_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_REWARD_TYPE = 0

C_PLOT_DEPTH_ENV = 0

C_PLOT_DEPTH_REWARD = 1

C_PLOT_DEPTH_ALL = 2

C_VALID_DEPTH = [0, 1, 2]

C_RST_BALANCING_001 = 'rst_001'

C_RST_BALANCING_002 = 'rst_002'

C_RST_SWINGING_001 = 'rst_003'

C_RST_SWINGING_OUTER_POLE_001 = 'rst_004'

C_VALID_RST_BALANCING = ['rst_001', 'rst_002']

C_VALID_RST_SWINGING = ['rst_003']

C_VALID_RST_SWINGING_OUTER_POLE = ['rst_004']

compute_reward_001(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*)

Reward strategy with only new normalized state

Parameters

- **p_state_old** (State) – Normalized old state.
- **p_state_new** (State) – Normalized new state.

Returns

current_reward – current calculated Reward values.

Return type

Reward

compute_reward_002(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*)

Reward strategy with both new and old normalized state based on euclidean distance from the goal state. Designed the balancing zone.

Parameters

- **p_state_old** (State) – Normalized old state.
- **p_state_new** (State) – Normalized new state.

Returns

current_reward – current calculated Reward values.

Return type

Reward

compute_reward_003(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*)

Reward strategy with both new and old normalized state based on euclidean distance from the goal state, designed for the swinging of outer pole. Both angles and velocity and acceleration of the outer pole are considered for the reward computation.

Parameters

- **p_state_old** (State) – Normalized old state.
- **p_state_new** (State) – Normalized new state.

Returns

current_reward – current calculated Reward values.

Return type

Reward

compute_reward_004(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*)

Reward strategy with both new and old normalized state based on euclidean distance from the goal state, designed for the swinging up region. Both angles and velocity and acceleration of the outer pole are considered for the reward computation.

The reward strategy is as follows:

reward = (**|old_theta1n|** - **|new_theta1n|**) + (**|new_omega1n + new_alpha1n|** - **|old_omega1n + old_alpha1n|**)

Parameters

- **p_state_old** (State) – Normalized old state.
- **p_state_new** (State) – Normalized new state.

Returns

current_reward – current calculated Reward values.

Return type

Reward

class mlpro.rl.pool.envs.doublependulum.**DoublePendulumS4**(*p_mode=0*, *p_latency=None*, *p_max_torque=20*, *p_l1=1.0*, *p_l2=1.0*, *p_m1=1.0*, *p_m2=1.0*, *p_g=9.8*, *p_init_angles='random'*, *p_history_length=5*, *p_fct_strans:* FctSTrans | *None* = *None*, *p_fct_success:* FctSuccess | *None* = *None*, *p_fct_broken:* FctBroken | *None* = *None*, *p_fct_reward:* FctReward | *None* = *None*, *p_mujoco_file=None*, *p_frame_skip=None*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf=None*, *p_visualize: bool* = *False*, *p_plot_level: int* = *2*, *p_rst_balancing='rst_002'*, *p_rst_swinging='rst_003'*, *p_rst_swinging_outer_pole='rst_004'*, *p_reward_weights: list* | *None* = *None*, *p_reward_trend: bool* = *False*, *p_reward_window: int* = *0*, *p_random_range: list* | *None* = *None*, *p_balancing_range: list* = *(-0.2, 0.2)*, *p_swinging_outer_pole_range=(0.2, 0.5)*, *p_break_swinging: bool* = *False*, *p_logging=True*)

Bases: *DoublePendulumSystemS4*, *DoublePendulumRoot*

This is the Double Pendulum Static 4 dimensional environment that inherits from the double pendulum root class, inheriting the dynamics and default reward strategy.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.

- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_max_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p_init_angles** (*str, optional*) – C_ANGLES_UP starts the pendulum in an upright position C_ANGLES_DOWN starts the pendulum in a downward position C_ANGLES_RND starts the pendulum from a random position.
- **p_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p_history_length** (*int, optional*) – Historical trajectory points to display. The default is 5.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_plot_level** (*int*) – Types and number of plots to be plotted. Default = ALL C_PLOT_DEPTH_ENV only plots the environment C_PLOT_DEPTH_REWARD only plots the reward C_PLOT_ALL plots both reward and the environment
- **p_rst_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p_rst_swinging** – Reward strategy to be used for the swinging region of the environment
- **p_reward_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p_reward_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p_reward_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p_random_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p_balancing*) – The boundaries for state space of environment in balancing region
- **p_break_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_NAME = 'DoublePendulumS4'

class mlpro.rl.pool.envs.doublependulum.**DoublePendulumS7**(*p_mode=0*, *p_latency=None*, *p_max_torque=20*, *p_l1=1.0*, *p_l2=1.0*, *p_m1=1.0*, *p_m2=1.0*, *p_g=9.8*, *p_init_angles='random'*, *p_history_length=5*, *p_fct_strans: FctSTrans | None = None*, *p_fct_success: FctSuccess | None = None*, *p_fct_broken: FctBroken | None = None*, *p_fct_reward: FctReward | None = None*, *p_mujoco_file=None*, *p_frame_skip=None*, *p_state_mapping=None*, *p_action_mapping=None*, *p_camera_conf=None*, *p_visualize: bool = False*, *p_plot_level: int = 2*, *p_rst_balancing='rst_002'*, *p_rst_swinging='rst_003'*, *p_rst_swinging_outer_pole='rst_004'*, *p_reward_weights: list | None = None*, *p_reward_trend: bool = False*, *p_reward_window: int = 0*, *p_random_range: list | None = None*, *p_balancing_range: list = (-0.2, 0.2)*, *p_swinging_outer_pole_range=(0.2, 0.5)*, *p_break_swinging: bool = False*, *p_logging=True*)

Bases: *DoublePendulumSystemS7*, *DoublePendulumS4*

This is the classic implementation of Double Pendulum with 7 dimensional state space including derived accelerations of both the poles and the input torque. The dynamics of the system are inherited from the Double Pendulum Root class.

Parameters

- **p_mode** – Mode of environment. Possible values are Mode.C_MODE_SIM(default) or Mode.C_MODE_REAL.
- **p_latency** (*timedelta*) – Optional latency of environment. If not provided, the internal value of constant C_LATENCY is used by default.
- **p_max_torque** (*float, optional*) – Maximum torque applied to pendulum. The default is 20.
- **p_l1** (*float, optional*) – Length of pendulum 1 in m. The default is 0.5
- **p_l2** (*float, optional*) – Length of pendulum 2 in m. The default is 0.25
- **p_m1** (*float, optional*) – Mass of pendulum 1 in kg. The default is 0.5
- **p_m2** (*float, optional*) – Mass of pendulum 2 in kg. The default is 0.25
- **p_init_angles** (*str, optional*) – C_ANGLES_UP starts the pendulum in an upright position C_ANGLES_DOWN starts the pendulum in a downward position C_ANGLES_RND starts the pendulum from a random position.
- **p_g** (*float, optional*) – Gravitational acceleration. The default is 9.8
- **p_history_length** (*int, optional*) – Historical trajectory points to display. The default is 5.

- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_plot_level** (*int*) – Types and number of plots to be plotted. Default = ALL C_PLOT_DEPTH_ENV only plots the environment C_PLOT_DEPTH_REWARD only plots the reward C_PLOT_ALL plots both reward and the environment
- **p_rst_balancingL** – Reward strategy to be used for the balancing region of the environment
- **p_rst_swinging** – Reward strategy to be used for the swinging region of the environment
- **p_reward_weights** (*list*) – List of weights to be added to the dimensions of the state space for reward computation
- **p_reward_trend** (*bool*) – Boolean value stating whether to plot reward trend
- **p_reward_window** (*int*) – The number of latest rewards to be shown in the plot. Default is 0
- **p_random_range** (*list*) – The boundaries for state space for initialization of environment randomly
- **range** (*p_balancing*) – The boundaries for state space of environment in balancing region
- **p_break_swinging** (*bool*) – Boolean value stating whether the environment shall be broken outside the balancing region
- **p_logging** – Log level (see constants of class mlpro.bf.various.Log). Default = Log.C_LOG_WE.

C_NAME = 'DoublePendulumS7'

Grid World

Ver. 2.0.3 (2022-11-29)

This module provides an environment of customizable Gridworld.

class mlpro.rl.pool.envs.gridworld.**GridWorld**(*p_logging: bool = True*, *p_grid_size=(8, 8)*, *p_random_start_position: bool = True*, *p_random_goal_position: bool = True*, *p_max_step: int = 50*, *p_action_type: int = 0*, *p_start_position=None*, *p_goal_position=None*)

Bases: *Environment*

Custom environment of an n-D grid world where the agent has to go to a random or defined target.

Parameters

- **p_logging** (*bool*) – Subspace of an environment that is observed by the policy. Default = Log.C_LOG_ALL.
- **p_grid_size** (*dimension*) – Dimension of the grid world (n-D grid world), e.g. (8,8) for 2-D or (8,8,8) for 3-D. Default = (8,8)
- **p_random_start_position** (*bool*) – Randomize start position. Default = True.
- **p_random_goal_position** (*bool*) – Randomize goal position. Default = True.
- **p_max_step** (*int*) – Maximum step per episode. Default = 50.

- **p_action_type** (*int*) – Type of actions, which is either continuous action or discrete action. To be noted, discrete action is now limited to 2-d grid world. Default = C_ACTION_TYPE_C.
- **p_start_position** (*dimension*) – To define the starting position, if p_random_start_position is False, e.g. (3,2). Default = None.
- **p_goal_position** (*dimension*) – To define the goal positoin, if p_random_goal_position is False, e.g. (5,5). Default = None.

C_NAME = 'Grid World'

C_LATENCY = datetime.timedelta(seconds=1)

C_INFINITY = 3.4028235e+38

C_REWARD_TYPE = 0

C_ACTION_TYPE_CONT = 0

C_ACTION_TYPE_DISC_2D = 1

static setup_spaces()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

get_all_states()

init_plot(*p_figure=None*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot()

Updates the plot.

Parameters

****p_kwargs** – Implementation-specific plot data and/or parameters.

Multi-Cartpole

Ver. 1.3.3 (2023-02-22)

This module provides an environment with multivariate state and action spaces based on the OpenAI Gym environment 'CartPole-v1'.

class mlpro.rl.pool.envs.multicartpole.**MultiCartPole**(*p_num_envs=2*, *p_reward_type=0*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *Environment*

This environment multivariate space and action spaces by duplicating the OpenAI Gym environment 'CartPole-v1'. The number of internal CartPole sub-enironments can be parameterized.

C_NAME = 'MultiCartPole'

C_LATENCY = datetime.timedelta(seconds=1)

C_INFINITY = 3.4028235e+38

C_PLOT_ACTIVE: bool = True

static load(*p_path*, *p_filename*)

Loads content from the given path and file name. If file does not exist, it returns None.

Parameters

- **file** (*p_path Path that contains the*) –
- **name** (*p_filename File*) –

Returns

A loaded object, if file content was loaded successfully. None otherwise.

switch_logging(*p_logging*)

Sets new log level.

Parameters

p_logging – Log level (constant C_LOG_LEVELS contains valid values)

static setup_spaces()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

collect_substates() → *State*

get_cycle_limit()

Returns limit of cycles per training episode.

simulate_reaction(*p_state:* State, *p_action:* Action) → *State*

Simulates a state transition based on a state and an action. The simulation step itself is carried out either by an internal custom implementation in method _simulate_reaction() or by an embedded external function.

Parameters

- **p_state** (State) – Current state.
- **p_action** (Action) – Action.

Returns

Subsequent state after transition

Return type

State

compute_success(*p_state:* State) → bool

Assesses the given state whether it is a 'success' state. Assessment is carried out either by a custom implementation in method _compute_success() or by an embedded external function.

Parameters
p_state (State) – State to be assessed.

Returns
success – True, if the given state is a 'success' state. False otherwise.

Return type
bool

compute_broken(*p_state:* State) → bool

Assesses the given state whether it is a 'broken' state. Assessment is carried out either by a custom implementation in method _compute_broken() or by an embedded external function.

Parameters
p_state (State) – State to be assessed.

Returns
broken – True, if the given state is a 'broken' state. False otherwise.

Return type
bool

init_plot(*p_figure: Figure | None = None*, *p_plot_settings: list = Ellipsis*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_step_rate: int = 0*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

Robot Manipulator

Ver. 1.1.8 (2022-11-09)

This module provides an environment of a robot manipulator based on Homogeneous Matrix

class mlpro.rl.pool.envs.robotinhtm.**RobotArm3D**

Bases: object

Auxiliary class for the implementation of robotinhtm. Generate the Kinematic of a pre-defined robot in Homogeneous Matrix.

add_link_joint(*jointAxis=None*, *lvector=None*, *thetaInit=None*, *adjustRot=None*, *adjustTheta=None*)

get_transformation_matrix(*theta*, *lvector*, *rotAxis*, *adjustRots=None*, *adjustThetas=None*)

update_joint_coords()

get_joint()

get_num_joint()

set_theta(*theta*)

get_homogeneous()

get_homogeneous_eef()

get_orientation()

update_theta(*deltaTheta*)

convert_to_quaternion()

class mlpro.rl.pool.envs.robotinhtm.**RobotHTM**(*p_num_joints=4*, *p_reset_seed=True*, *p_seed=None*, *p_target_mode: str = 'random'*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *Environment*

Custom environment for an arm robot represented as Homogeneous Matrix. The goal is to reach a certain point.

Parameters

- **p_num_joints** (*int*) – Number of joints. Default = 4.
- **p_reset_seed** (*bool*) – If True, random generator is reset. Default = True.
- **p_seed** – Seeding value for the random generator. Default = None.
- **p_target_mode** (*str*) – Target mode. Possible values are "random" (default) or "fixed".
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_NAME = 'RobotHTM'

C_REWARD_TYPE = 0

C_LATENCY = datetime.timedelta(seconds=1)

C_INFINITY = 3.4028235e+38

C_CYCLE_LIMIT = 100

static setup_spaces()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

set_theta(*theta*)

Environment Models

RobotInHTM

MLP Gridworld

MLP for RobotInHTM

Policies

SARS-Buffers

Prioritized Buffer

Ver. 1.0.1 (2021-09-26)

This module provides the Prioritized Buffer based on the reference.

class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**PrioritizedBufferElement**(*p_state:* State, *p_action:* Action, *p_reward:* Reward, *p_state_new:* State)

Bases: *SARSElement*

Element of a State-Action-Reward-Buffer.

class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**PrioritizedBuffer**(*p_size=1*, *alpha: float = 0.3*, *beta: float = 1*)

Bases: *SARSBuffer*

Prioritized Sampling State-Action-Reward-Buffer in dictionary.

add_element(*p_elem:* PrioritizedBufferElement)

Add element to the buffer.

Parameters
p_elem (BufferElement) – Element of Buffer

get_latest()

Returns latest buffered element.

get_all()

Return all buffered elements.

update_priorities(*p_list_idx: list*, *priorities: ndarray*)

Updates the priority tree. Needs to be called during each training step, utilising the element-wise calculated loss.

class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**SegmentTree**(*capacity: int*, *operation: Callable*, *init_value: float*)

Bases: object

Reference: https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py .. attribute:: capacity

type
int

tree

Type
list

operation

Type
function

operate(*start: int = 0*, *end: int = 0*) → float
Returns result of applying 'self.operation'.

class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**SumSegmentTree**(*capacity: int*)
Bases: *SegmentTree*

Reference: https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py

sum(*start: int = 0*, *end: int = 0*) → float
Returns arr[start] + … + arr[end].

retrieve(*upperbound: float*) → int
Find the highest index *i* about upper bound in the tree

class mlpro.rl.pool.sarsbuffer.PrioritizedBuffer.**MinSegmentTree**(*capacity: int*)
Bases: *SegmentTree*

Reference: https://github.com/openai/baselines/blob/master/baselines/common/segment_tree.py

min(*start: int = 0*, *end: int = 0*) → float
Returns min(arr[start], …, arr[end]).

Random SARS Buffer

Ver. 1.0.0 (2021-09-25)

This module provides implementation of SARBuffer with random sampling.

class mlpro.rl.pool.sarsbuffer.RandomSARSBuffer.**RandomSARSBuffer**(*p_size=1*)
Bases: *BufferRnd*

Random Sampling SARBuffer. This is just renaming class from BufferRnd

9.2.4 MLPro-GT - Game Theory

Game Boards

Bulk Goods Plant

Ver. 1.0.4 (2021-11-29)

This module provides an environment of Bulk Good Laboratory Plant (BGLP) following GT interface. This module provides game board classed based on BGLP environment of the reinforcement learning pool.

class mlpro.gt.pool.boards.bglp.**BGLP_GT**(*p_logging=True*, *t_step=0.5*, *t_set=10.0*, *demand=0.1*, *lr_margin=1.0*, *lr_demand=4.0*, *lr_energy=0.001*, *margin_p=[0.2, 0.8, 4]*, *prod_target=10000*, *prod_scenario='continuous'*)

Bases: *BGLP*, *GameBoard*

Game theoretical pendant for the reinforcement learning environment class BGLP.

C_NAME = 'BGLP_GT'

Multi-Cartpole

Ver. 1.1.0 (2022-11-07)

This module provides game board classes based on the Multi-CartPole environment of the reinforcement learning pool.

class mlpro.gt.pool.boards.multicartpole.**MultiCartPoleGT**(*p_num_envs=2*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *MultiCartPole*, *GameBoard*

Game theoretical pendant for the reinforcement learning environment class MultiCartPole.

C_NAME = 'MultiCartPole(GT)'

class mlpro.gt.pool.boards.multicartpole.**MultiCartPolePGT**(*p_num_envs=2*, *p_visualize: bool = True*, *p_logging=True*)

Bases: *MultiCartPole*, *PGameBoard*

Potential Game theoretical pendant for the reinforcement learning environment class MultiCartPole.

C_NAME = 'MultiCartPole(PGT)'

9.2.5 MLPro-OA - Online Adaptivity

Coming soon...

9.3 Wrappers

9.3.1 Hyperopt

Ver. 1.1.1 (2022-10-17)

This module provides a wrapper class for hyperparameter tuning by reusing the Hyperopt framework.

See also: https://pypi.org/project/hyperopt/

class mlpro.wrappers.hyperopt.**WrHPTHyperopt**(*p_logging=True*, *p_algo='RND'*, *p_ids=None*)

Bases: *HyperParamTuner*, Wrapper, *ScientificObject*

This class is a ready to use wrapper class for Hyperopt framework. Objects of this type can be treated as a hyperparameter tuner object.

Parameters

- **p_logging** – Log level (see constants for log levels)

- **p_algo** (*str, optional*) – Selection of a hyperparameter tuning algorithm, default: C_ALGO_RAND
- **p_ids** (*list of str, optional*) – List of hyperparameter ids to be tuned, otherwise all hyperparameters, default: None

C_NAME

Name of the class.

Type
str

C_ALGO_TPE

Refer to Tree of Parzen Estimators (TPE) algorithm.

Type
str

C_ALGO_RAND

Refer to Random Grid Search algorithm.

Type
str

```
C_NAME = 'Hyperopt'

C_WRAPPED_PACKAGE = 'hyperopt'

C_ALGO_TPE = 'TPE'

C_ALGO_RAND = 'RND'

C_SCIREF_TYPE = 'Proceedings'

C_SCIREF_AUTHOR = 'James Bergstra, Dan Yamins, David D. Cox'

C_SCIREF_TITLE = 'Hyperopt:  A Python Library for Otimizing the Hyperparameters of
Machine Learning Algorithms'

C_SCIREF_CONFERENCE = 'Proceedings of the 12th Python in Science Conference'

C_SCIREF_YEAR = '2013'

C_SCIREF_PAGES = '13-19'

C_SCIREF_DOI = '10.25080/Majora-8b375195-003'

C_SCIREF_EDITOR = 'Stefan van der Walt, Jarrod Millman, Katy Huff'

C_LOG_SEPARATOR =
'------------------------------------------------------------------------------'
```

setup_spaces()

This method is used to setup the hyperparameter spaces, including the tuning boundaries and a suitable discrete value. The hyperparameter should be bounded both above and below. We are using the "quantized" continuous distributions for natural and integer numbers. Meanwhile the real numbers are not quantized. For different parameter expressions, please redefined this method and check http://hyperopt.github.io/hyperopt/getting-started/search_spaces/! For big data handling, please redifined this method!

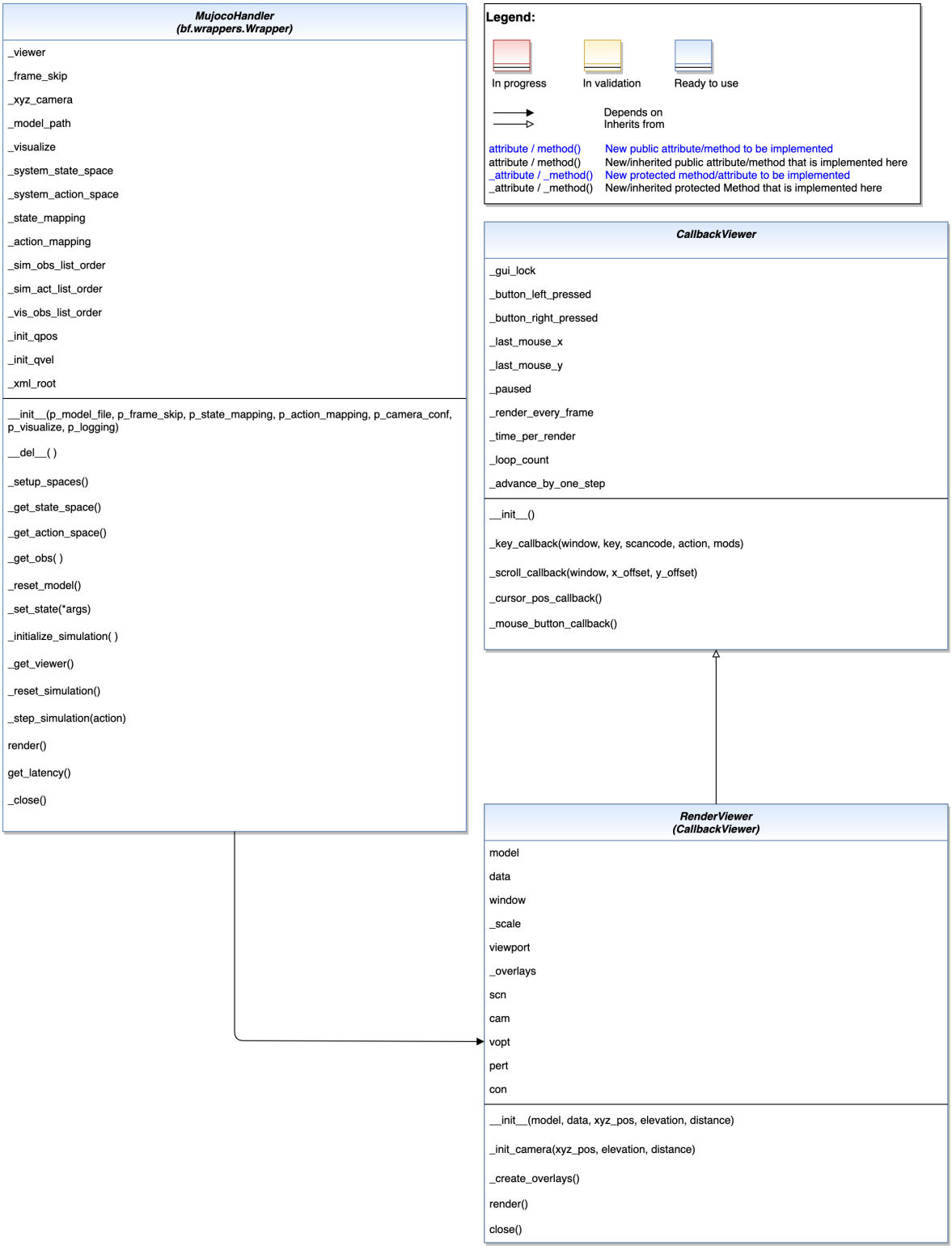
Returns
spaces – List of parameter expressions.

Return type
list

Cross References

- *Howto RL-HT-001: Hyperparameter Tuning using Hyperopt*

9.3.2 MuJoCo



Cross References

9.3.3 OpenAI Gym

Ver. 1.6.0 (2023-02-20)

This module provides wrapper classes for OpenAI Gym environments.

See also: https://pypi.org/project/gym

class mlpro.wrappers.openai_gym.**WrEnvGYM2MLPro**(*p_gym_env*, *p_state_space:* MSpace | *None* = *None*, *p_action_space:* MSpace | *None* = *None*, *p_visualize: bool* = *True*, *p_logging=True*)

Bases: `Wrapper`, *Environment*

This class is a ready to use wrapper class for OpenAI Gym environments. Objects of this type can be treated as an environment object. Encapsulated gym environment must be compatible to class gym.Env.

Parameters

- **p_gym_env** (*Env*) – Gym environment object
- **p_state_space** (MSpace) – Optional external state space object that meets the state space of the Gym environment
- **p_action_space** (MSpace) – Optional external action space object that meets the action space of the Gym environment
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Wrapper Gym2MLPro'

C_WRAPPED_PACKAGE = 'gym'

C_MINIMUM_VERSION = '0.21.0'

C_PLOT_ACTIVE: bool = True

static load(*p_path*, *p_filename*)

Loads content from the given path and file name. If file does not exist, it returns None.

Parameters

- **file** (*p_path Path that contains the*) –
- **name** (*p_filename File*) –

Returns

A loaded object, if file content was loaded successfully. None otherwise.

static recognize_space(*p_gym_space*) → *ESpace*

Detecting a gym space and transform it to MLPro space. Hence, the transformed space can be directly compatible in MLPro.

Parameters

p_gym_space (container spaces (`Tuple` and `Dict`)) – Spaces are crucially used in Gym to define the format of valid actions and observations.

Returns

space – MLPro compatible space.

Return type

ESpace

static setup_spaces()

To setup spaces. To be optionally defined by the users.

simulate_reaction(*p_state:* State, *p_action:* Action) → *State*

Simulates a state transition based on a state and an action. The simulation step itself is carried out either by an internal custom implementation in method _simulate_reaction() or by an embedded adaptive function.

Parameters

- **p_state** (State) – Current state.
- **p_action** (Action) – Action.

Returns
state – Subsequent state after transition

Return type
State

compute_reward(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*) → *Reward*

Computes a reward for the state transition, given by two successive states. The reward computation itself is carried out either by a custom implementation in method _compute_reward() or by an embedded adaptive function.

Parameters

- **p_state_old** (State) – Optional state before transition. If None the internal previous state of the environment is used.
- **p_state_new** (State) – Optional tate after transition. If None the internal current state of the environment is used.

Returns
Reward object.

Return type
Reward

compute_success(*p_state:* State) → bool

Assesses the given state whether it is a 'success' state. Assessment is carried out either by a custom implementation in method _compute_success() or by an embedded adaptive function.

Parameters
p_state (State) – State to be assessed.

Returns
True, if the given state is a 'success' state. False otherwise.

Return type
bool

compute_broken(*p_state:* State) → bool

Assesses the given state whether it is a 'broken' state. Assessment is carried out either by a custom implementation in method _compute_broken() or by an embedded adaptive function.

Parameters
p_state (State) – State to be assessed.

Returns
True, if the given state is a 'broken' state. False otherwise.

Return type
bool

init_plot(*p_figure: Figure | None = None*, *p_plot_settings: list = Ellipsis*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_step_rate: int = 0*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(***p_kwargs*)

Updating the actual plot, deployed by render functionality from OpenAI Gym.

get_cycle_limit()

To obtain the information regarding the cycle limit from the environment.

Returns
the number of the cycle limit.

Return type
float

class mlpro.wrappers.openai_gym.**WrEnvMLPro2GYM**(*p_mlpro_env: Environment*, *p_state_space: MSpace | None = None*, *p_action_space: MSpace | None = None*, *p_new_step_api: bool = False*, *p_render_mode: str | None = None*, *p_logging=True*)

Bases: Wrapper, Env

This class is a ready to use wrapper class for MLPro to OpenAI Gym environments. Objects of this type can be treated as an gym.Env object. Encapsulated MLPro environment must be compatible to class Environment.

Parameters

- **p_mlpro_env** (Environment) – MLPro's Environment object
- **p_state_space** (MSpace) – Optional external state space object that meets the state space of the MLPro environment
- **p_action_space** (MSpace) – Optional external action space object that meets the state space of the MLPro environment
- **p_new_step_api** (*bool*) – If true, the user assures that the environment compatible to Gym version 0.25.0 or above. Otherwise, it is false. Default = False.
- **p_render_mde** (*str*) – To allow the user to specify render_mode handled by the environment, for instance, 'human', 'rgb_array', and 'single_rgb_array'. Default = None.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Wrapper MLPro2Gym'

C_WRAPPED_PACKAGE = 'gym'

C_MINIMUM_VERSION = '0.21.0'

metadata = {'render.modes': ['human']}

static recognize_space(*p_mlpro_space*)

Detecting a MLPro space and transform it to gym space. Hence, the transformed space can be directly compatible in gym.

Parameters
p_mlpro_space (ESpace) – MLPro compatible space.

Returns
space – Spaces are crucially used in Gym to define the format of valid actions and observations.

Return type
container spaces (Tuple and Dict)

step(*action*)

To execute one time step within the environment.

Parameters
action (*ActType*) – an action provided by the agent.

Returns

- **obs** (*object*) – This will be an element of the environment's observation_space. This may, for instance, be a numpy array containing the positions and velocities of certain objects.
- **reward.get_overall_reward()** (*float*) – The amount of reward returned as a result of taking the action.
- **terminated** (*bool*) – whether a *terminal state* (as defined under the MDP of the task) is reached. In this case further step() calls could return undefined results.
- **truncated** (*bool*) – whether a truncation condition outside the scope of the MDP is satisfied. Typically a timelimit, but could also be used to indicate agent physically going out of bounds. Can be used to end the episode prematurely before a *terminal state* is reached.
- **info** (*dict*) – It contains auxiliary diagnostic information (helpful for debugging, learning, and logging). This might, for instance, contain: metrics that describe the agent's performance state, variables that are hidden from observations, or individual reward terms that are combined to produce the total reward. It also can contain information that distinguishes truncation and termination, however this is deprecated in favour of returning two booleans, and will be removed in a future version.

reset_new(*seed=None*, *return_info=False*, *options=None*)

Resets the environment to an initial state and returns the initial observation. This is for new gym version.

Parameters

- **seed** (*int, optional*) – The seed that is used to initialize the environment's PRNG. If the environment does not already have a PRNG and seed=None (the default option) is passed, a seed will be chosen from some source of entropy (e.g. timestamp or /dev/urandom). However, if the environment already has a PRNG and seed=None is passed, the PRNG will *not* be reset. If you pass an integer, the PRNG will be reset even if it already exists. Usually, you want to pass an integer *right after the environment has been initialized and then never again*. Please refer to the minimal example above to see this paradigm in action. The default is None.
- **return_info** (*bool*) – If true, return additional information along with initial observation. This info should be analogous to the info returned in *step()*. The default is False.
- **options** (*dict, optional*) – Additional information to specify how the environment is reset (optional, depending on the specific environment). The default is None.

Returns

- **obs** (*object*) – This will be an element of the environment's `observation_space`. This may, for instance, be a numpy array containing the positions and velocities of certain objects.
- **info** (*dict*) – It contains auxiliary diagnostic information (helpful for debugging, learning, and logging). This might, for instance, contain: metrics that describe the agent's performance state, variables that are hidden from observations, or individual reward terms that are combined to produce the total reward. It also can contain information that distinguishes truncation and termination, however this is deprecated in favour of returning two booleans, and will be removed in a future version.

`reset_old()`

Resets the environment to an initial state and returns the initial observation. This is for old gym version.

Parameters

- `seed` (*int, optional*) – The seed that is used to initialize the environment's PRNG. If the environment does not already have a PRNG and `seed=None` (the default option) is passed, a seed will be chosen from some source of entropy (e.g. timestamp or /dev/urandom). However, if the environment already has a PRNG and `seed=None` is passed, the PRNG will *not* be reset. If you pass an integer, the PRNG will be reset even if it already exists. Usually, you want to pass an integer *right after the environment has been initialized and then never again*. Please refer to the minimal example above to see this paradigm in action. The default is None.
- `return_info` (*bool*) – If true, return additional information along with initial observation. This info should be analogous to the info returned in *step()*. The default is False.
- `options` (*dict, optional*) – Additional information to specify how the environment is reset (optional, depending on the specific environment). The default is None.

Returns

- **obs** (*object*) – This will be an element of the environment's `observation_space`. This may, for instance, be a numpy array containing the positions and velocities of certain objects.
- **info** (*dict*) – It contains auxiliary diagnostic information (helpful for debugging, learning, and logging). This might, for instance, contain: metrics that describe the agent's performance state, variables that are hidden from observations, or individual reward terms that are combined to produce the total reward. It also can contain information that distinguishes truncation and termination, however this is deprecated in favour of returning two booleans, and will be removed in a future version.

`render(mode='human')`

Compute the render frames as specified by render_mode attribute during initialization of the environment.

Parameters

`mode` (*str, optional*) – To allow the user to specify render_mode handled by the environment, for instance, 'human', 'rgb_array', and 'single_rgb_array'. The default is 'human'.

Returns

Rendering is successful or not.

Return type

bool

`close()`

Override close in your subclass to perform any necessary cleanup. Environments will automatically *close()* themselves when garbage collected or when the program exits.

Cross References

- *Howto RL-WP-001: MLPro to OpenAI Gym*
- *Howto RL-AGENT-001: Run an Agent with Own Policy*

9.3.4 OpenML

Cross References ..

- *Howto BF-STREAMS-051: Accessing Data from OpenML*

9.3.5 Optuna

Ver. 1.1.1 (2022-10-17)

This module provides a wrapper class for hyperparameter tuning by reusing Optuna framework.

See also: https://pypi.org/project/optuna/

class mlpro.wrappers.optuna.**WrHPTOptuna**(*p_logging=True*, *p_ids=None*, *p_visualization=False*)

Bases: Wrapper, *HyperParamTuner*, *ScientificObject*

This class is a ready to use wrapper class for Optuna framework. Objects of this type can be treated as a hyperparameter tuner object.

Parameters

- **p_logging** (Log) – Log level (see constants for log levels)
- **p_ids** (*list of str, optional*) – List of hyperparameter ids to be tuned, otherwise all hyperparameters, default: None
- **p_visualization** (*boolean*) – enable visualization at the end of the tuning, default: False

C_NAME

Name of the class.

Type
str

```
C_NAME = 'Optuna'

C_WRAPPED_PACKAGE = 'optuna'

C_SCIREF_TYPE = 'Proceedings'

C_SCIREF_AUTHOR = 'Akiba, Takuya and Sano, Shotaro and Yanase, Toshihiko and Ohta,
Takeru and Koyama, Masanori'

C_SCIREF_TITLE = 'Optuna:  A Next-Generation Hyperparameter Optimization Framework'

C_SCIREF_YEAR = '2019'

C_SCIREF_ISBN = '9781450362016'

C_SCIREF_PUBLISHER = 'Association for Computing Machinery'

C_SCIREF_CITY = 'New York'
```

```
C_SCIREF_COUNTRY = 'USA'

C_SCIREF_URL = 'https://doi.org/10.1145/3292500.3330701'

C_SCIREF_DOI = '10.1145/3292500.3330701'

C_SCIREF_PAGES = '2623-2631'

C_SCIREF_BOOKTITLE = 'Proceedings of the 25th ACM SIGKDD International Conference on
Knowledge Discovery & Data Mining'

C_LOG_SEPARATOR =
'------------------------------------------------------------------------------'
```

get_parameters(*trial*)

This method is used to get parameters within boundaries. The hyperparameter should be bounded both above and below. For different parameter expressions, please redefined this method and check https://optuna.readthedocs.io/en/stable/reference/generated/optuna.trial.Trial.html! For big data handling, please redifined this method!

Parameters
trial (*object*) – Suggest hyperparameters using a trial object.

Returns
parameters – List of parameter expressions.

Return type
list

Cross Reference

- *Howto RL HT 002: Hyperparameter Tuning using Optuna*

9.3.6 PettingZoo

Ver. 2.1.0 (2023-02-21)

This module provides wrapper classes for PettingZoo multi-agent environments.

See also: https://pypi.org/project/PettingZoo/

class mlpro.wrappers.pettingzoo.**WrEnvPZOO2MLPro**(*p_zoo_env*, *p_state_space:* MSpace | *None* = *None*, *p_action_space:* MSpace | *None* = *None*, *p_visualize: bool* = *True*, *p_logging=True*)

Bases: Wrapper, *Environment*

This class is a ready to use wrapper class for Petting Zoo environments. Objects of this type can be treated as an environment object. Encapsulated petting zoo environment must be compatible to class pettingzoo.env.

Parameters

- **p_pzoo_env** – Petting Zoo environment object
- **p_state_space** (MSpace) – Optional external state space object that meets the state space of the gym environment
- **p_action_space** (MSpace) – Optional external action space object that meets the action space of the gym environment
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.

- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Wrapper PettingZoo2MLPro'

C_WRAPPED_PACKAGE = 'pettingzoo'

C_MINIMUM_VERSION = '1.20.0'

C_PLOT_ACTIVE: bool = True

C_SUPPORTED_MODULES = ['pettingzoo.classic', 'pettingzoo.butterfly', 'pettingzoo.atari', 'pettingzoo.mpe', 'pettingzoo.sisl']

static load(*p_path*, *p_filename*)

Loads content from the given path and file name. If file does not exist, it returns None.

Parameters

- **file** (*p_path Path that contains the*) –
- **name** (*p_filename File*) –

Returns

A loaded object, if file content was loaded successfully. None otherwise.

static setup_spaces()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

simulate_reaction(*p_state:* State, *p_action:* Action) → *State*

Simulates a state transition based on a state and an action. The simulation step itself is carried out either by an internal custom implementation in method _simulate_reaction() or by an embedded external function.

Parameters

- **p_state** (State) – Current state.
- **p_action** (Action) – Action.

Returns

Subsequent state after transition

Return type

State

compute_reward(*p_state_old:* State | *None* = *None*, *p_state_new:* State | *None* = *None*) → *Reward*

Computes a reward for the state transition, given by two successive states. The reward computation itself is carried out either by a custom implementation in method _compute_reward() or by an embedded adaptive function.

Parameters

- **p_state_old** (State) – Optional state before transition. If None the internal previous state of the environment is used.
- **p_state_new** (State) – Optional tate after transition. If None the internal current state of the environment is used.

Returns
Reward object.

Return type
Reward

`compute_success`(*p_state:* State) → bool

Assesses the given state whether it is a 'success' state. Assessment is carried out either by a custom implementation in method _compute_success() or by an embedded external function.

Parameters
p_state (State) – State to be assessed.

Returns
success – True, if the given state is a 'success' state. False otherwise.

Return type
bool

`compute_broken`(*p_state:* State) → bool

Assesses the given state whether it is a 'broken' state. Assessment is carried out either by a custom implementation in method _compute_broken() or by an embedded external function.

Parameters
p_state (State) – State to be assessed.

Returns
broken – True, if the given state is a 'broken' state. False otherwise.

Return type
bool

`init_plot`(*p_figure: Figure | None = None*, *p_plot_settings: list = Ellipsis*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_step_rate: int = 0*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (PlotSettings) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

`update_plot`(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

`get_cycle_limit`()

Returns limit of cycles per training episode.

`class mlpro.wrappers.pettingzoo.`**`WrEnvMLPro2PZoo`**(*p_mlpro_env:* Environment, *p_num_agents*, *p_state_space:* MSpace | *None = None*, *p_action_space:* MSpace | *None = None*, *p_logging=True*)

Bases: `Wrapper`

This class is a ready to use wrapper class for MLPro to PettingZoo environments. Objects of this type can be treated as an AECEnv object. Encapsulated MLPro environment must be compatible to class Environment. To be noted, this wrapper is not capable for parallel environment yet.

Parameters

- **p_mlpro_env** (Environment) – MLPro's Environment object
- **p_num_agents** (*int*) – Number of Agents
- **p_state_space** (MSpace) – Optional external state space object that meets the state space of the MLPro environment
- **p_action_space** (MSpace) – Optional external action space object that meets the action space of the MLPro environment

C_TYPE = 'Wrapper MLPro2PettingZoo'

C_WRAPPED_PACKAGE = 'pettingzoo'

C_MINIMUM_VERSION = '1.20.0'

class raw_env(*p_mlpro_env*, *p_num_agents*, *p_state_space: MSpace | None = None*, *p_action_space: MSpace | None = None*, *p_render_mode='human'*)

Bases: AECEnv

metadata: Dict[str, Any] = {'name': 'pzoo_custom', 'render_modes': ['human', 'ansi']}

possible_agents: List[AgentID]

observation_spaces: Dict[AgentID, gymnasium.spaces.Space]

action_spaces: Dict[AgentID, gymnasium.spaces.Space]

agents: List[AgentID]

terminations: Dict[AgentID, bool]

truncations: Dict[AgentID, bool]

rewards: Dict[AgentID, float]

infos: Dict[AgentID, Dict[str, Any]]

agent_selection: AgentID

step(*action*)

Accepts and executes the action of the current agent_selection in the environment.

Automatically switches control to the next agent.

observe(*agent_id*)

Returns the observation an agent currently can make.

last() calls this function.

reset(*seed*, *options*)

Resets the environment to a starting state.

render(*mode='human'*)

Renders the environment as specified by self.render_mode.

Render mode can be *human* to display a window. Other render modes in the default environments are *'rgb_array'* which returns a numpy array and is supported by all environments outside of classic, and *'ansi'* which returns the strings printed (specific to classic environments).

close()

Closes any resources that should be released.

Closes the rendering window, subprocesses, network connections, or any other resources that should be released.

Cross References

- Howto RL-006: Run own multi-agent with Petting Zoo environment
- Howto RL-009: Wrap native MLPro environment class to PettingZoo environment

9.3.7 River

Ver. 1.4.1 (2022-12-09)

This module provides wrapper functionalities to incorporate public data sets of the River ecosystem.

Learn more: https://www.riverml.xyz/

class mlpro.wrappers.river.**WrStreamProviderRiver**(*p_logging=True*)

Bases: Wrapper, *StreamProvider*

Wrapper class for River as StreamProvider.

Parameters

p_logging – Log level of stream objects (see constants of class Log). Default: Log.C_LOG_ALL.

C_NAME = 'River'

C_WRAPPED_PACKAGE = 'river'

C_SCIREF_TYPE = 'Online'

C_SCIREF_AUTHOR = 'River'

C_SCIREF_URL = 'riverml.xyz'

class mlpro.wrappers.river.**WrStreamRiver**(*p_id=0*, *p_name: str = ''*, *p_num_instances: int = 0*, *p_version: str = ''*, *p_feature_space:* MSpace | *None = None*, *p_label_space:* MSpace | *None = None*, *p_mode=0*, *p_logging=True*, ***p_kwargs*)

Bases: *Stream*

Wrapper class for Streams from River.

Parameters

- **p_id** – Optional id of the stream. Default = 0.
- **p_name** (*str*) – Optional name of the stream. Default = ''.
- **p_num_instances** (*int*) – Optional number of instances in the stream. Default = 0.
- **p_version** (*str*) – Optional version of the stream. Default = ''.
- **p_feature_space** (MSpace) – Optional feature space. Default = None.
- **p_label_space** (MSpace) – Optional label space. Default = None.
- **p_mode** – Operation mode. Default: Mode.C_MODE_SIM.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.

- **p_kwargs** (*dict*) – Further stream specific parameters.

C_NAME = 'River stream'

C_SCIREF_TYPE = 'Online'

Cross References

- *Howto BF-STREAMS-053: Accessing Data from River*

9.3.8 Stable Baslines3

Ver. 1.2.6 (2023-02-16)

This module provides wrapper classes for integrating stable baselines3 policy algorithms.

See also: https://pypi.org/project/stable-baselines3/

class mlpro.wrappers.sb3.**DummyEnv**(*p_observation_space=None*, *p_action_space=None*)

Bases: Env

Dummy class for Environment. This is required due to some of the SB3 Policy Algorithm requires to have an Environment. As for now, it only needs the observation space and the action space.

observation_space = None

action_space = None

compute_reward(*achieved_goal: int | ndarray*, *desired_goal: int | ndarray*, *_info: Dict[str, Any] | None*) → float32

class mlpro.wrappers.sb3.**VecExtractDictObs**(*venv: VecEnv*, *observation_space: Space | None = None*, *action_space: Space | None = None*)

Bases: VecEnvWrapper

A vectorized wrapper for filtering a specific key from dictionary observations. This is used for HER incorporation on off-policy algorithms. Similar to Gym's FilterObservation wrapper:

https://github.com/openai/gym/blob/master/gym/wrappers/filter_observation.py

reset() → ndarray

Reset all the environments and return an array of observations, or a tuple of observation arrays.

If step_async is still doing work, that work will be cancelled and step_wait() should not be called until step_async() is invoked again.

Returns
observation

step_async(*actions: ndarray*) → None

Tell all the environments to start taking a step with the given actions. Call step_wait() to get the results of the step.

You should not call this if a step_async run is already pending.

step_wait() → Tuple[ndarray | Dict[str, ndarray] | Tuple[ndarray, ...], ndarray, ndarray, List[Dict]]

Wait for the step taken with step_async().

Returns
observation, reward, done, information

class mlpro.wrappers.sb3.**WrPolicySB32MLPro**(*p_sb3_policy*, *p_cycle_limit*, *p_observation_space: MSpace*, *p_action_space: MSpace*, *p_ada: bool = True*, *p_visualize: bool = False*, *p_logging=True*, *p_num_envs: int = 1*, *p_desired_goals=None*)

Bases: Wrapper, *Policy*

This class provides a policy wrapper from Standard Baselines 3 (SB3). Especially On-Policy Algorithm

Parameters

- **p_sb3_policy** – SB3 Policy
- **p_cycle_limit** – Maximum number of cycles
- **p_observation_space** (MSpace) – Observation Space
- **p_action_space** (MSpace) – Environment Action Space
- **p_ada** (*bool*) – Adaptability. Defaults to True.
- **p_visualize** (*bool*) – Boolean switch for visualisation. Default = False.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.
- **p_num_envs** (*int*) – Number of environments, specifically for vectorized environment.
- **p_desired_goals** (*list, Optional*) – Desired state goals for Hindsight Experience Replay (HER).

C_TYPE = 'Wrapper SB3 -> MLPro'

C_WRAPPED_PACKAGE = 'stable_baselines3'

C_NAME = 'Policy DQN'

Cross References

- Howto RL-WP-004: Train an Agent with SB3

9.3.9 Scikit-learn

Ver. 1.3.2 (2022-12-13)

This module provides wrapper functionalities to incorporate public data sets of the Scikit-learn ecosystem.

Learn more: https://scikit-learn.org

class mlpro.wrappers.sklearn.**WrStreamProviderSklearn**(*p_logging=True*)

Bases: Wrapper, *StreamProvider*

Wrapper class for Sklearn as StreamProvider

C_NAME = 'Stream Provider Scikit-learn'

C_WRAPPED_PACKAGE = 'scikit-learn'

C_SCIREF_TYPE = 'Online'

C_SCIREF_AUTHOR = 'Scikit-learn'

C_SCIREF_URL = 'https://scikit-learn.org'

class mlpro.wrappers.sklearn.**WrStreamSklearn**(*p_id*, *p_name*, *p_num_instances: int = 0*, *p_version: str = ''*, *p_logging=True*, *p_mode=0*, ***p_kwargs*)

Bases: *Stream*

Wrapper class for Streams from Sklearn

Parameters

- **p_id** – Id of the stream.
- **p_name** (*str*) – Name of the stream.
- **p_num_instances** (*int*) – Number of instances in the stream.
- **p_version** (*str*) – Version of the stream. Default = ''.
- **p_feature_space** (MSpace) – Optional feature space. Default = None.
- **p_label_space** (MSpace) – Optional label space. Default = None.
- **p_mode** – Operation mode. Valid values are stored in constant C_VALID_MODES.
- **p_logging** – Log level (see constants of class Log). Default: Log.C_LOG_ALL.
- **p_kwargs** (*dict*) – Further stream specific parameters.

C_NAME = 'Sklearn stream'

C_SCIREF_TYPE = 'Online'

Cross References

- *Howto BF-STREAMS-052: Accessing Data from Scikit-Learn*

CHAPTER

TEN

A3 - PROJECT MLPRO

10.1 Release Notes

Release Notes on GitHub:

- Latest release
- Release history

Upcoming:

- Upcoming release

10.2 Publications

10.2.1 Papers

- Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya, Andreas Schwung: "MLPro 1.0 - Standardized reinforcement learning and game theory in Python", Machine Learning with Applications, 2022, ISSN 2666-8270, doi: 10.1016/j.mlwa.2022.100341
- Detlef Arend, Mochammad Rizky Diprasetya, Steve Yuwono, Andreas Schwung: "MLPro–An integrative middleware framework for standardized machine learning tasks in Python", Software Impacts, 2022, doi: 10.1016/j.simpa.2022.100421

10.2.2 Code Ocean Capsules

- Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya, Andreas Schwung: "MLPro - Reinforcement Learning - Demo of Advanced Training with Stagnation Detection", Code Ocean, 2022, doi: 10.24433/CO.9151382.v1
- Mochammad Rizky Diprasetya, Detlef Arend, Steve Yuwono, Andreas Schwung: "MLPro - Reinforcement Learning - Demo of Training UR5 ROS Environment", Code Ocean, 2022, doi: 10.24433/CO.6279818.v1

10.2.3 GitHub Repository

- Detlef Arend, Steve Yuwono, Mochammad Rizky Diprasetya, Andreas Schwung: "MLPro - A Synoptic Framework for Standardized Machine Learning Tasks in Python", Zenodo, 2021, doi: 10.5281/zenodo.6653485

10.3 Contribution

We look forward to your contributions to MLPro! Found a problem or have a good idea for an improvement? Just let us know directly on GitHub:

- Problem Report
- Proposal for an improvement

If you are interested in actively shaping MLPro, then simply contact us at: mlpro@listen.fh-swf.de

Our templates will help you create additional content in MLPro style. Of course, we will assist you in placing your contribution correctly.

We thank all contributors for their active support!

10.4 Disclaimer

If you require any more information or have any questions about our project's disclaimer, please feel free to contact us by email at mlpro@listen.fh-swf.de.

10.4.1 Disclaimers for MLPro Usage

All the information on this website is published in good faith and for general information purpose only. We do not make any warranties about the completeness, reliability and accuracy of this information. Any action you take upon the information you find on this project is strictly at your own risk. The South Westphalia University of Applied Sciences, Germany will not be liable for any losses and/or damages in connection with the use of MLPro.

From our website, you can visit other websites by following hyperlinks to such external sites. While we strive to provide only quality links to useful and ethical websites, we have no control over the content and nature of these sites. These links to other websites do not imply a recommendation for all the content found on these sites. Site owners and content may change without notice and may occur before we have the opportunity to remove a link which may have gone 'bad'.

Please be also aware that when you leave our website, other sites may have different privacy policies and terms which are beyond our control. Please be sure to check the Privacy Policies of these sites as well as their "Terms of Service" before engaging in any business or uploading any information.

10.4.2 Consent

By using our website, you hereby consent to our disclaimer and agree to its terms.

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

S

T

U

V

W