Accelerated exploration of multinary systems

Release 1.1

Elise Garel, Jean-Luc Parouty

Feb 23, 2022

CONTENTS

This documentation aims at helping use the codes developped for "Accelerated exploration of multinary systems using high-throughput experiments and machine learning" project.

This project combines material Science and AI, and consists in producing experimental data using combinatorial high-throuput methods and mixture design, and then to analyse them using Machine Learning tools to extract predictive models linking compositions, structures and properties.

Experiments Planification

This Matlab GUI interface allow to prepare the experimental work. It gives a set of 1D linear gradients or 2D planar gradients for 3 to 7 element systems. It can optimize the number of samples or the experiments price.

Requirements: Matlab2019 or more

PyTerK

These Python modules and notebooks perform iterative k-fold crossvalidation for Scikit-learn or Keras models, on any datasets, with parallelization of works. Settings are written in yml files, trainings are performed in "run" notebooks and results are visualized in "report" notebooks.

Requirements:

- **Install following libraries, via pip or conda**
  - tensorflow - keras
  - pandas
  - scikit-learn
- **Create environment variables:**
  - path to folder that contains the datasets DATASETS_DIR=$path/to/datasets/
  - path to folder that will contain the training results : RUN_DIR=$path/to/run/

Multiple Regression

These Python modules and notebooks perform iterative k-fold crossvalidation for statsmodel multilinear regression, on any datasets. Training and report are performed in one notebook.

Requirements:

- **Install following libraries, via pip or conda**
  - statsmodels.formula.api
  - pandas
  - scikit-learn
- **Create environment variables in bash_profile:**
  - path to folder that contains the datasets: export DATASETS_DIR=$path/to/datasets/
  - path to folder that will contain the training results : export RUN_DIR=$path/to/run/

Galanov Model

This module allows to compute elastic-plastic zone
$fracb_sc$, the constrain factor C and ductility characteristic
$delta_H$ by solving equations proposed by Galanov *et al* (Galanov, Ivanov, et Kartuzov, *Improved Core Model of the Indentation for the Experimental Determination of Mechanical Properties of Elastic-Plastic Materials and Its Application.*)

Requirements:

- **Install following libraries, via pip or conda**

    - pandas

- **Create environment variables:**
    - path to folder that will contain the training results : DATASETS_DIR=$path/to/datasets/

Pareto Front

This module allows to detect Pareto optimal composition, for two antagonistic criteria/properties.

Requirements:

- **Install following libraries, via pip or conda**
    - pandas
- **Create environment variables:**
    - path to folder that will contain the training results : DATASETS_DIR=$path/to/datasets/

Plot Predictions

These Matlab modules plot Machine Learning prediction. It can scatter composition point of plot alpha shape of compositions points. User can add color or point size to plot along the predicted properties of the composition points.

Requirements:

- Matlab2019 or later
- **Create environment variables:**
    - path to folder that will contain the training results : DATASETS_DIR=$path/to/datasets/

CHAPTER

ONE

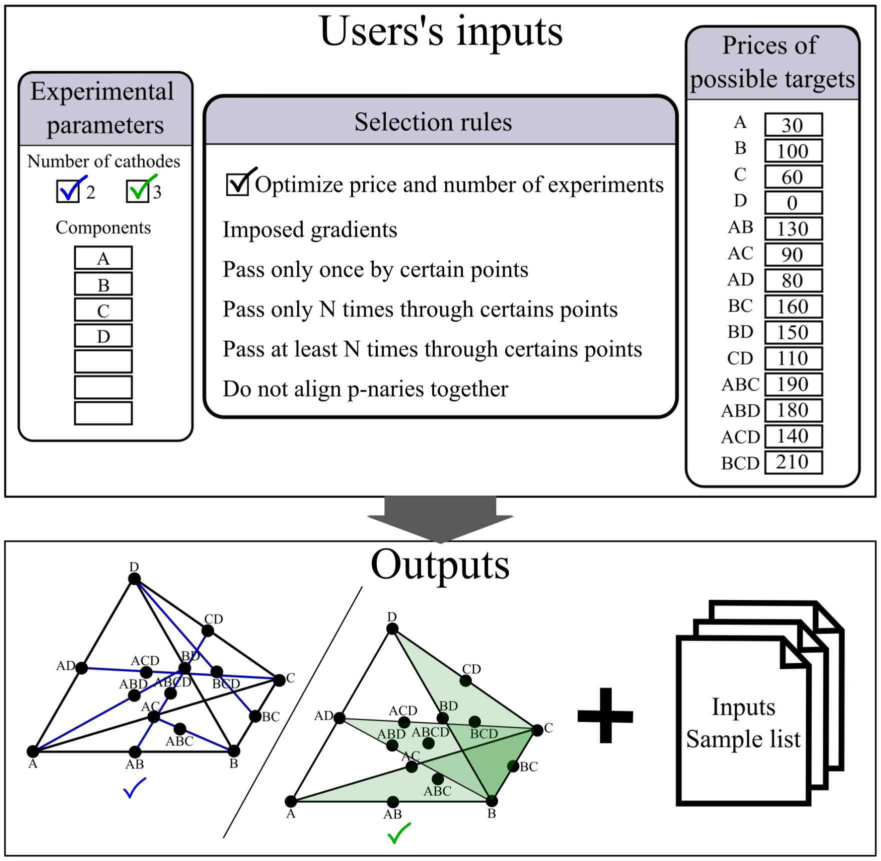
EXPERIMENTS PLANNIFICATION

1.1 Principle

A Matlab GUI interface was developed in order to automatically generate a set of experiments to screen a N-element composition space using a combinatorial approach and a 2- to 3-cathodes magnetron sputtering. The starting point of this method is based on simplex centroid mixture design, in order to screen the space as uniformly as possible. From the composition points given by the mixture design, all linear/planar gradients passing by 3/7 of them are computed. Then a set of gradients/planes is chosen in order to pass at least once by each point and to respect the user inputs. The choice is done with a random exploration of all possible gradients, that starts all over again if the set does not meet the requirements.

Features :

- **Adaptability to the user's needs**
  - The user enters the elements of the composition space they want to explore (from 3 to 7 elements).
  - Chooses if they are using two or three cathodes
  - Indicates if they want to preferentially explore some point of the mixture design.
- Representation of the composition space and of the gradients/planes that are explored.
- Give the list of targets that allow to perform the experiments


Users's inputs
Experimental parameters
Number of cathodes
2
3
Components
A
B
C
D
Selection rules
Optimize price and number of experiments
Imposed gradients
Pass only once by certain points
Pass only N times through certains points
Pass at least N times through certains points
Do not align p-naries together
Prices of possible targets
A 30
B 100
C 60
D 0
AB 130
AC 90
AD 80
BC 160
BD 150
CD 110
ABC 190
ABD 180
ACD 140
BCD 210
Outputs
Inputs Sample list

1.2 Modules

ExperimentsPlannification.modules_ep.**check_do_not_align**(*name_alignments*, *index*, *do_not_align*)

Check user condition to not align certain mixtures in the same gradient

Parameters

- **name_alignments** (array(str)) – name of points through which the gradients passes
- **index** (int) – index of alignments in the list of all alignments
- **do_not_align** (cell(list(str))) – list of mixtures that must not be aligned

Returns indicator: "ok" if alignement respects the user condition; else return "not ok".

Return type str

ExperimentsPlannification.modules_ep.**check_not_repeat**(*name_alignments*, *index*, *name_alignement_opt*, *not_repeat*)

Check user condition to not repeat certain mixtures in gradients set

Parameters

- **name_alignments** (array(str)) – name of points through which the gradients passes
- **index** (int) – index of alignments in the list of all alignments
- **name_alignement_opt** (array(str)) – gradients set that are already selected.
- **not_repeat** (array(str)) – mixtures to no repeat in the gradients set

Returns indicator: "ok" if alignement respects the user condition; else return "not ok".

Return type str

`ExperimentsPlannification.modules_ep.`**`check_repeat_only`**(*name_alignments*, *index*, *name_alignement_opt*, *repeat_only*)

Check user condition to not repeat certain mixtures in gradients set

Parameters

- **`name_alignments`** (`array(str)`) – name of points through which the gradients passes
- **`index`** (`int`) – index of alignments in the list of all alignments
- **`name_alignement_opt`** (`array(str)`) – gradients set that are allready selected.
- **`repeat_only`** (`array(str,int)`) – name of mixtures that must be repeated a limited number of time and this limited number of time

Returns indicator: "ok" if alignement respects the user condition; else return "not ok".

Return type str

`ExperimentsPlannification.modules_ep.`**`compute_alignments`**(*mixture*, *name_mixture*, *nb_type_mixture*)

For a reference mixture, the function calculates the vector coefficient between this reference mixture and all the other mixtures with same or higher order. Then it looks for equals vector coefficients for segments with a common point to determine which two other mixture points are aligned with the reference mixture

Parameters

- **`mixture`** (`cell{array}`) – coordinates of mixtures, cell index being the mixture order *eg:mixtures{2} contains the binaries coordinates*
- **`name_mixture`** (`cell{str}`) – name of mixture, cell index being the mixture order
- **`nb_type_mixture`** (`int`) – number of type/order of mixtures to explore

Returns

- alignments : coordinates of the mixtures through which the gradient pass (3x3 columns)
- name_alignments: mixture names through which the gradient pass

Return type array(float),array(str)

`ExperimentsPlannification.modules_ep.`**`compute_planes`**(*name_alignment*, *alignments*, *nb_type_mixture*)

From the gradients, planes are defines in the composition space made by 3 gradients with common points, encompassing 7 points of the mixtrure design. This means that the plane is centered on one of the point of the mixture design

eg: Nb-NbTi-Ti, Ti-TiZr-Zr and Nb-NbZr-Zr are forming a plane in a compositional space center on the ternary NbTiZr wich is a point of the mixture design: the plane is valid

Parameters

- **`name_alignment`** (`array(str)`) – points through which the gradient go
- **`alignments`** (`array(float)`) – coordinates of the points through which the gradient go (3x3 columns)
- **`nb_type_mixture`** (`int`) – number of type/order of mixtures to explore

Returns plane_points: mixture names encompassed by the planes

Returns plane_coord: coordinates of the mixtures encompassed by the planes (7x3 columns)

`ExperimentsPlannification.modules_ep.`**`coordinates_name_centroid_points`**`(`*nb_elements*, *name_elements*`)`

From the number and the name of the system elements, the function calculates the coordinates of the pure elements (standard uniform distribution in space) and of the equiolar mixtures of the Simplex Centroide mixture Design (all binaries, ternairies. . .).

Parameters **`nb_elements`** (`int`) – number of components

Name_element list(str) namer of components

Returns mixture: cell coordinates of all equimolar mixture

Return type cell

Returns name_mixture: containing the names of the equimolar mixtures.

Return type cell

`ExperimentsPlannification.modules_ep.`**`count_occur`**`(`*element*, *list*`)`

Count the numer of occurence of an element in a list

Parameters

- **`element`** – counted number or string
- **`list`** (`list`) – list in which the element is counted

Returns count: number of repetition of the element in list

Return type int

`ExperimentsPlannification.modules_ep.`**`fix_nb_repetition`**`(`*repeat_list*, *fig*, *position*`)`

This function is a callbacks of push buttons associated to listboxes When the buttons are pushed, the function identifies which mixture should be repeated Then it display in the interface the names of the mixtures that should be repeated and an edit box in which the user can enter the number of repetitions.

Parameters

- **`repeat_list`** (`UIcontrol`) – contains the mixture that should be repeated
- **`fig`** (`figure`) – working interface / window
- **`position`** (`list(float)`) – position features of repeated list

Returns nb_repet: edit boxes in which the user will enter the number of repetition of each mixture

`ExperimentsPlannification.modules_ep.`**`get_elements`**`(`*elements*, *fig1*`)`

Acquire the components name entered by the user

Parameters

- **`elements`** (`UIcontrol`) – edit boxes in which the user has entered the elements names
- **`fig1`** (`figure`) – interface window

Returns name_elements: name of elements

Return type list(str)

`ExperimentsPlannification.modules_ep.`**`gradients_set`**`(`*name_mixture*, *mixture*, *alignments*, *name_alignement*`)`

Selection of a gradients set that pass at least once through each point of the mixture design and that respect user condition inputs

Parameters

- **name_mixture** (cell(str)) – name of mixture, cell index being the mixture order
- **mixture** (cell(float)) – coordinates of mixtures, cell index being the mixture order *eg:mixtures{2} contains the binaries coordinates*
- **alignments** (cell(float)) – coordinates of the points through which the gradient pass (3x3 columns)
- **name_alignement** (cell(str)) – points through which the gradient pass

Returns

- name_alignement_opt: name of mixture trhough whch the set of gradients pass
- alignement_opt: coordinates of mixture trhough whch the set of gradients pass

Return type array(str),array(float)

`ExperimentsPlannification.modules_ep.`**`index_alignments`**`(`*cell_coeff_dir*`)`

Called in *compute_alignments* [we get cell structure with vector coefficient between one reference mixture and all the mixtures with the same or higher orders .] This function compares all the coefficients one by one to find equal ones

Parameters **cell_coeff_dir** (cell) – contains director coefficient of vectors between one reference mixtures and all the others with same or higher order.

Returns cell indice_cell, indice_list: indices of the cell and list where two coefficients are equals. Allow to identify pair of equal coefficient to identify aligned points.

`ExperimentsPlannification.modules_ep.`**`kill_program`**`()`

Kill the programis the user pushed STOP button

Returns display the message “kill” to indicate state

`ExperimentsPlannification.modules_ep.`**`lineIntersect3D`**`(`*PA*, *PB*`)`

Find intersection point of lines in 3D space, in the least squares sense.

Parameters

- **PA** – Nx3-matrix containing starting point of N lines
- **PB** – Nx3-matrix containing end point of N lines

Returns

- P_Intersect: Best intersection point of the N lines, in least squares sense.
- distances: Distances from intersection point to the input lines

Anders Eikenes (2022). Intersection point of lines in 3D space (https://www.mathworks.com/matlabcentral/fileexchange/37192-intersection-point-of-lines-in-3d-space), MATLAB Central File Exchange. Retrieved February 10, 2022.

`ExperimentsPlannification.modules_ep.`**`listing_targets`**`(`*name_alignement_opt*`)`

Lists the targets to use from the selected optimized set of gradients

Parameters **name_alignement_opt** (array(str)) – name of mixtures through which pass the gradients

Returns list(str) list_target: list the target compositions to use to deposit these gradients

`ExperimentsPlannification.modules_ep.`**`listing_targets_3cath`**`(`*name_planes_opt*`)`

Lists the targets to use from the selected optimized set of planar gradients

Parameters **`name_planes_opt`** (`array(str)`) – name of mixtures encompassed by planar gradients

Returns list(str) list_target: list the target compositions to use to deposit these gradients

`ExperimentsPlannification.modules_ep.`**`plot_compo_space_gradients`**`(`*nb_elements*, *mixture*, *name_mixture*, *name_elements*, *gradients*, *gradients_color*`)`

Plot the composition space with all the simplexe centroid points and linear gradients

Parameters

- **`nb_elements`** (`int`) – number of components
- **`mixture`** (`cell(float)`) – mixture points coordinates
- **`name_mixture`** (`cell(str)`) – mixture names
- **`name_elements`** (`list(str)`) – name of the components
- **`gradients`** (`array(str)`) – coordinates of the gradients points
- **`gradients_color`** – color of the gradients for plot

Returns fig: plot the compositions space dans gradients

`ExperimentsPlannification.modules_ep.`**`plot_compo_space_planes`**`(`*nb_elements*, *mixture*, *name_mixture*, *name_elements*, *plane_coord*, *plane_color*`)`

Plot the composition space with all the simplexe centroid points and planar gradients

Parameters

- **`nb_elements`** (`int`) – number of components
- **`mixture`** (`cell(float)`) – mixture points coordinates
- **`name_mixture`** (`cell(str)`) – mixture names
- **`name_elements`** (`list(str)`) – name of the components
- **`plane_coord`** (`array(str)`) – coordinates of the planes points
- **`plane_color`** – color of the plane for plot

Returns fig: plot the compositions space dans gradients

`ExperimentsPlannification.modules_ep.`**`parameters_file`**`()`

Write the users inputs and chosen parameters for one run of the interface in text file.

`ExperimentsPlannification.modules_ep.`**`planes_set`**`(`*name_mixture*, *mixture*, *planes*, *name_planes*`)`

Selection of a planes set that encompass at least once each point of the mixture design and that respect user condition inputs

Parameters

- **`name_mixture`** (`cell(str)`) – name of mixture, cell index being the mixture order
- **`mixture`** (`cell(float)`) – coordinates of mixtures, cell index being the mixture order *eg:mixtures{2} contains the binaries coordinates*
- **`plane`** (`cell(float)`) – coordinates of the points through which the planes pass (3x3 columns)
- **`name_planes`** (`cell(str)`) – points through which the planes pass

Returns array(str) name_planes_opt: name of mixture trhough whch the set of planes pass

Returns array(float) planes_opt: coordinates of mixture trhough whch the set of planes pass

`ExperimentsPlannification.modules_ep.`**`price_calculation`**`(`*prices_list*, *target_list*`)`

Calculate the price of a set of experiment

Parameters

- **`prices_list`** (`list(str,float)`) – list of possible targets and associated price
- **`targets`** (`list(str)`) – list of targets associated to one set of linear gradients or planar gradients

Returns price: total price of the targets required for a set of linear gradients or planar gradients

Return type float

`ExperimentsPlannification.modules_ep.`**`vector_coeff`**`(`*A*, *B*`)`

Compute normed vector coefficients between two points.

Parameters **`A,B`** (`list(float)`) – coordinates of two points

Returns coordinates of the normed vector corresponding to (AB) line

CHAPTER TWO

PYTERK PACKAGE

2.1 Module contents

PyTerK - A Python Iterated K-fold cross validation with shuffling

By E Garel / JL Parouty - SIMaP 2021

This package allows you to perform a **statistical evaluation** of different learning strategies (Keras/sklearn) by varying different (hyper)parameters.

2.1.1 Description :

It is possible to combine the following (hyper)parameters :

- datasets
- models (with their characteristics…)
- batch size
- epochs
- iterations
- k fold
- seed (to control pseudo random generator)

It is possible, for example, to combine 3 datasets, with 3 models and to perform for each combination, 5 iterations of a cross validation of KFold type, with k=10. In this case, the total number of models to test would be 3x3x5x10=450 training sessions… So, be careful, the number of model.fit can quickly be very important !

The tasks will be run in **parallel** on the different CPUs/cores available.

2.1.2 Documentation and examples :

Here is a basic example, detailled in a notebook :

```python
import pyterk.config       as config
import pyterk.reporter     as reporter
import pyterk.task_manager as task_manager

settings = config.load('settings_example.yml')

```

(continues on next page)

(continued from previous page)

```
task_manager.add_combinational_iterative_manyfold(settings, run_key= 'Example-03.1')
task_manager.run()

reporter.show_run_reports(settings)
```

This will retrieve all settings from *settings_example.yml*, prepare the different tasks and execute them. The last call, intended to be used from a Jupyter lab notebook, displays a complete execution report.

You can find **3 full example notebooks**, with a setting file :

- settings_example.yml
- 01-Example-01.ipynb
- 02-Example-02.ipynb
- 03-Example-03.ipynb

`pyterk.`**`VERSION = 2.14`**
pyterk version

2.2 Submodules

2.3 pyterk.config module

Configuration management.

The settings files allow to specify datasets and models.

2.3.1 Utilisation:

Loading a settings file:

```
settings = config.load('settings_example.yml')
```

or:

```
settings = config.load('settings_example.yml',
                       datasets_dir_env='DATASETS_DIR')
```

where DATASETS_DIR is an environment variable that will override *datasets_dir* directive in settings file.

`pyterk.config.`**`datasets = None`**
datasets profiles

`pyterk.config.`**`load`**(*filename*, *datasets_dir_env='DATASETS_DIR'*, *run_dir_env='RUN_DIR'*, *verbose=0*)
Load a setting file and dfined datasets. If given, environment variable can be use to overide *datasets_dir* directive from setting file. Usefull for portability between several sites.

Parameters

- **filename** (*string*) – Filename of the yaml setting file
- **datasets_dir_env** (*string*) – Name of the overiding environment variable
- **verbose** (*int*) – verbose mode for loaded datasets (0).

Returns A dict from setting file, completed by datasets and more.

`pyterk.config.models = None`
models profiles

`pyterk.config.run_dir = None`
run_dir, the place to put all output directories

`pyterk.config.runs = None`
dict of runs section

`pyterk.config.settings = None`
Dict of settings

2.4 pyterk.models module

This module is for internal use only - You do not have to interact with.

`pyterk.models.get_model(`*profile*`)`
Get a model from a model profile. The profile contains the module and function name of the model, and the arguments. The model will be retrieved by calling the function with the arguments.

Parameters `profile` (*dict*) – a model profile

Returns keras or scikitlearn model as defined in the profile.

Return type model (keras/scikitlearn model)

2.5 pyterk.reporter module

Module to generate execution reports.

During the run of the tasks, the bestmodel and results are saved in h5 and json files:

- *about.json* : information and description of the task
- *history.json* : history from model.fit()
- *evaluation.json* : evaluation from model.evaluate()
- *bestmodel.h5* : best model

2.5.1 Example :

```
reporter.show_run_reports(settings,
                          args       = ['dataset_id','model_id','batch_size'],
                          evaluation = [2])
```

This module will retrieve information from json files and generate a report.

`pyterk.reporter.plot_confusion(`*run_dir*, *predict_type='softmax'*, *normalize='pred'*, *figsize=(5, 5)*, *savefig=True*, *mplstyle='pyterk'*`)`
Plot a confusion matrix

Parameters

- `iterations_dir` – a directory with iterations subdirs (iter-000, iter-001, …)

- `predict_type` – sigmoid, softmax or classes
- `normalize` – true, pred, all or None (pred)
- `figsize` – figure size
- `savefig` – save fig (True) or not (False)

Returns Just plot the matrix and print report and hamming loss

`pyterk.reporter.`**`plot_distribution`**(*run_dir*, *metric_id=0*, *bins=10*, *min=None*, *max=None*, *figsize=(10, 8)*, *savefig=False*, *mplstyle='pyterk'*)

Plot distribution of a given metric from an evaluation.json saved file. For a kfold or an iterative kfold, all evaluation data are concatened in an evaluation.json file in main run_dir.

Parameters

- **`run_dir`** (*`string`*) – directory path of json evaluation file
- **`metricid`** (*`int`*) – number of metric to plot. Example : 2
- **`min`** (*`int`*) – min value
- **`max`** (*`int`*) – max value
- **`bins`** (*`int`*) – number of bins
- **`figsize`** (*`tuple`*) – figure size, default is (10,8)
- **`savefig`** (*`boolean`*) – if True, figure will be save in run_dir.
- **`mplstyle`** (*`string`*) – name of matplotlib style. default is 'pyterk', but all matplotlib are ok (default, bmh, . . .)

Returns Nothing, but display a beautifull distribution plot !

`pyterk.reporter.`**`plot_history`**(*run_dir*, *metric='val_mae'*, *min=None*, *max=None*, *figsize=(10, 8)*, *savefig=False*, *mplstyle='pyterk'*)

Plot history evolution from history.json saved file. For a kfold or an iterative kfold, all history data are concatened in history.json file in main run_dir. This will plot a curve for each one in a common plot.

Parameters

- **`run_dir`** (*`string`*) – directory path of json history file
- **`metric`** (*`string`*) – metric name to plot. Example : 'val_mae'
- **`figsize`** (*`tuple`*) – figure size, default is (10,8)
- **`savefig`** (*`boolean`*) – if True, figure will be save in run_dir.
- **`mplstyle`** (*`string`*) – name of matplotlib style. default is 'pyterk', but all matplotlib are ok (default, bmh, . . .)

Returns Nothing, but display a beautifull plot !

`pyterk.reporter.`**`plot_kfold_correlation`**(*run_dir*, *channel=0*, *figsize=(8, 6)*, *axes_min='auto'*, *axes_max='auto'*, *yy_deltamax=None*, *marker='o'*, *markersize=8*, *alpha=0.7*, *color='auto'*, *savefig=True*, *mplstyle='pyterk'*)

Plot a correlation for a (y_test, y_pred) saved json file.

Parameters

- **`run_file`** – a manyfold directory where kfold subdirectories are
- **`channel`** – composant of y to plot
- **`figsize`** (*`tuple`*) – figure size, default is (10,8)

- **axes_min** – min value for x and y axe. 'auto' or float
- **axes_max** – max value for x and y axe. 'auto' or float
- **mplstyle** (*string*) – name of matplotlib style. default is 'pyterk', but all matplotlib are ok (default, bmh, ...)
- **marker** – marker, default is '.'
- **markersize** – marker size
- **alpha** – marker alpha
- **color** – plot color or 'auto'
- **savefig** – if True, save fig in run_dir

Returns Nothing, but display a beautifull correlation plot

pyterk.reporter.**show_report**(*run_dir*, *padding=''*, *sections=['title', 'context', 'args', 'settings', 'evaluation', 'monitoring', 'history', 'distribution', 'correlation']*, *context=['function', 'version', 'date', 'description', 'seed']*, *args=['run_dir', 'dataset_id', 'model_id', 'n_iter', 'k_fold', 'epochs', 'batch_size']*, *settings=['file', 'version', 'description', 'datasets_dir', 'run_dir']*, *evaluation=['all']*, *monitoring=['duration', 'used_data']*, *history=[{'metric': 'val_mae', 'min': None, 'max': None, 'figsize': (8, 6), 'savefig': True, 'mplstyle': 'pyterk'}]*, *distribution=[{'metric_id': 2, 'bins': 4, 'min': None, 'max': None, 'figsize': (8, 6), 'savefig': True, 'mplstyle': 'pyterk'}]*, *correlation=[{'axes_min': 'auto', 'axes_max': 'auto', 'figsize': (8, 6), 'marker': '.', 'markersize': 8, 'alpha': 0.7, 'color': 'auto', 'savefig': True, 'mplstyle': 'pyterk'}]*, *confusion=[{'normalize': 'pred', 'predict_type': 'softmax', 'figsize': (5, 5), 'savefig': True, 'mplstyle': 'pyterk'}]*)

Builds and displays a report from the json data of a given run_dir.

Parameters

- **run_dir** (*string*) – directory path of json report file
- **sections** (*list*) – list of sections to include in the report
- **context** (*list*) – informations to include in context section
- **args** (*list*) – informations to include in args section
- **settings** (*list*) – informations to include in settings section
- **evaluation** (*list*) – #metrics to include in evaluation section. 'all' mean all. Example : [0,1,2]
- **history** (*dict*) – parameters for history plot - see *plot_history*
- **distribution** (*dict*) – parameters for metrics distribution plot
- **correlation** (*dict*) – parameters for correlation plot
- **confusion** (*dict*) – parameters for confusion matrix (need yytest files)

`pyterk.reporter.`**`show_run_reports`**(*run_config, run_filter='.*', sections=['title', 'context', 'args', 'settings', 'evaluation', 'monitoring', 'history', 'distribution', 'correlation', 'confusion'], context=['function', 'version', 'date', 'description', 'seed'], args=['run_dir', 'dataset_id', 'model_id', 'n_iter', 'k_fold', 'epochs', 'batch_size'], settings=['file', 'version', 'description', 'datasets_dir', 'run_dir'], evaluation=['all'], monitoring=['duration', 'used_data'], history=[{'metric': 'val_mae', 'min': None, 'max': None, 'figsize': (8, 6), 'savefig': True, 'mplstyle': 'pyterk'}], distribution=[{'metric_id': 2, 'bins': 4, 'min': None, 'max': None, 'figsize': (8, 6), 'savefig': True, 'mplstyle': 'pyterk'}], correlation=[{'axes_min': 'auto', 'axes_max': 'auto', 'figsize': (8, 6), 'marker': '.', 'markersize': 8, 'alpha': 0.7, 'color': 'auto', 'savefig': True, 'mplstyle': 'pyterk'}], confusion=[{'normalize': 'pred', 'predict_type': 'softmax', 'figsize': (5, 5), 'savefig': True, 'mplstyle': 'pyterk'}]*)

Displays a full report in two parts, short and long, for all runs defined in the settings. Very simple to use…

Parameters

- **`run_config`** (*`dict`*) – settings, issued from config.load()
- **`run_filter`** (*`regx`*) – regex to filter run entries from yml settings file (.*)
- **`sections`** (*`list`*) – list of sections to include in the report
- **`context`** (*`list`*) – informations to include in context section
- **`args`** (*`list`*) – informations to include in args section
- **`settings`** (*`list`*) – informations to include in settings section
- **`evaluation`** (*`list`*) – #metrics to include in evaluation section. 'all' mean all. Example : [0,1,2]
- **`history`** (*`dict`*) – parameters for history plot - see *plot_history*
- **`distribution`** (*`dict`*) – parameters for metrics distribution plot
- **`correlation`** (*`dict`*) – parameters for correlation plot
- **`confusion`** (*`dict`*) – parameters for confusion matrix (need yytest files)

Returns Nothing, but display a short and long report, with index.

2.6 pyterk.task_manager module

Allows to generate tasks and to execute them in a distributed way.

See example notebook : *03-Example-03.ipynb*

Example :

```
task_manager.add_combinational_iterative_manyfold(settings = settings,
                                                  run_key = 'Example-03.3')
```

`pyterk.task_manager.`**`add_combinational_iterative_manyfold`**(*settings=None, run_key=None, verbose=1*)

Add tasks for a combinational iterative manyfold - *see 03-Example-03*.ipynb Generates all the tasks of the combinatorial described in the run section of the settings file.

Parameters

- **settings** (*dict*) – settings
- **run_key** (*string*) – name of the config run section
- **verbose** (*int*) – verbosity of generated tasks

Returns Nothings. Task are added to the pending taks queue.

pyterk.task_manager.**add_iterative_manyfold**(*settings=None*, *run_dir=None*, *dataset_id=None*, *model_id=None*, *n_iter=2*, *k_fold=10*, *epochs=10*, *batch_size=10*, *description=None*, *save_xxtest=False*, *save_yytest=False*, *verbose=1*)

Add tasks for an iterative manyfold - see *02-Example-02.ipynb* Generate n_ter*k_fold tasks, each iteration will generate a subdirectory in run_dir.

Parameters

- **settings** (*dict*) – settings
- **run_dir** (*string*) – run directoty to output k results (json files and best model)
- **dataset_id** (*string*) – datasets id in settings file
- **model_id** (*string*) – model id in settings file
- **n_iter** (*int*) – number of iteration
- **k_fold** (*int*) – number of fold
- **epochs** (*int*) – number of epochs
- **batch_size** (*int*) – size of batch
- **description** (*string*) – description of the action
- **save_xxtest** (*Boolean*) – save x_test as json file, or not
- **save_yytest** (*Boolean*) – save y_test and y_pred as json file, or not
- **verbose** (*int*) – verbosity of generated tasks

Returns Nothings. Task are added to the pending taks queue.

pyterk.task_manager.**add_manyfold**(*settings=None*, *run_dir=None*, *dataset_id=None*, *model_id=None*, *k_fold=10*, *epochs=10*, *batch_size=10*, *description=None*, *save_xxtest=False*, *save_yytest=False*, *verbose=1*)

Add tasks for a manyfold - see *01-Example-01.ipynb* Generate k_fold tasks, each task will generate one subdirectory in run_dir.

Parameters

- **settings** (*dict*) – settings
- **run_dir** (*string*) – run directoty to output k results (json files and best model)
- **dataset_id** (*string*) – datasets id in settings file
- **model_id** (*string*) – model id in settings file
- **k_fold** (*int*) – number of fold
- **epochs** (*int*) – number of epochs
- **batch_size** (*int*) – size of batch
- **description** (*string*) – description of the action
- **save_xxtest** (*Boolean*) – save x_test as json file, or not

- `save_yytest` (*Boolean*) – save y_test and y_pred as json file, or not
- `verbose` (*int*) – verbosity of generated tasks

Returns Nothings. Task are added to the pending taks queue.

`pyterk.task_manager.`**`reset`**`()`
Reset pending tasks. Suppress all of them !

`pyterk.task_manager.`**`run`**`(`*processes=None*, *maxtasksperchild=10*, *verbose=1*`)`

`pyterk.task_manager.`**`seed`**`(`*seed=None*`)`
Init random generators with given seed

`pyterk.task_manager.`**`show_tasks_size`**`()`
Print pending tasks size

2.7 pyterk.worker module

This module is for internal use only - You do not have to interact with ;-).

`pyterk.worker.`**`get_model_family`**`(`*model*`)`
Should return the model family : 'tensorflow', 'keras' or 'sklearn'

`pyterk.worker.`**`init`**`(`*s*, *l*, *v*`)`

`pyterk.worker.`**`model_fit`**`(`*run_dir=None*, *dataset_id=None*, *train_index=None*, *test_index=None*, *model_id=None*, *epochs=None*, *batch_size=None*, *seed=None*, *description=None*, *save_xxtest=False*, *save_yytest=False*`)`

`pyterk.worker.`**`model_fit_sklearn`**`(`*model*, *run_dir=None*, *x_train=None*, *y_train=None*, *x_test=None*, *y_test=None*, *save_xxtest=False*, *save_yytest=False*`)`

`pyterk.worker.`**`model_fit_tensorflow`**`(`*model*, *run_dir=None*, *x_train=None*, *y_train=None*, *x_test=None*, *y_test=None*, *epochs=None*, *batch_size=None*, *save_xxtest=False*, *save_yytest=False*`)`

CHAPTER

THREE

MULTIPLE REGRESSION MODULE

3.1 Description

Module to train Multiple Linear Regression with Scheffe interaction terms, with iterative k-fold crossvalidation. The method that lies behind the model training is identical to PyTerK one, with iterative k-fold cross-validation.

Contains functions to :

- generate interactions
- train regression models
- plot iterative k-fold crossvalidation results

3.2 Functions

`MultipleRegression.`**`Scheffe_interactions_terms`**`(`*data*, *in_percent='True'*, *compo_columns=['Zr', 'Nb', 'Mo', 'Ti', 'Cr']*`)`

Shaping composition in percentage rate into percentage Compute interaction terms for all Scheffe interactions for quartic multiple regression and add it to dataframe data

Parameters **`panda.DataFrame`** – dataset that contains compositions in Zr, Nb, Mo, Ti, Cr in columns of the same name

Returns extended input dataset with interactions

Return type DataFrame

`MultipleRegression.`**`fit_outputs`**`(`*model_expression*, *k*, *nb_it*, *output*, *X*, *y*`)`

Takes an OLS model expression, and use it to perform regression between X and y . Model regression is performed using iterative k-fold crossvalidation Evaluation is performed through R2 and MAE computation

Parameters

- **`OLS-formula`** – contain OLS formula for regression
- **`k`** (*`int`*) – number of folds for iterative k-fold crossfvalidation
- **`nb_it`** (*`int`*) – number of iterations for iterative k-fold crossfvalidation
- **`output`** (*`str`*) – name of the Y output to fit
- **`X`** (*`panda.DtataFrame`*) – contains composition and interaction terms for regression input
- **`y`** (*`panda.DataFrame`*) – contains single column dataframe with regression output

Return model model coefficients and p-values

Return type statsmodels.regression.linear_model.RegressionResultsWrapper

Return MAE_list list of MAE for every run of iterative k-fold crossvalidation, between expected vs predicted value on test set

Return type list

Return R2_adj_list list of R2 adjusted for every run of iterative k-fold crossvalidation, between expected vs predicted value on test set

Return type list

Returns Y_pred : list of predicted values on test set

Return type list

Returns Y_test : list of expected values on test set

Return type list

`MultipleRegression.`**`plot_result`**(*metric*, *output*, *val_metric*, *Y_pred*, *Y_test*, *min_hist*, *max_hist*, *iter*, *kfold*, *save_distri*, *save_regression*)

Plot metric histogram and regression between predictions and test values and save graphs

Parameters

- **`metric`** (*`str`*) – name of the metric distribution to plot
- **`output`** (*`str`*) – name of the Y output to fit
- **`val_metric`** (*`list`*) – list of MAE for every run of iterative k-fold crossvalidation, between expected vs predicted value on test set
- **`Y_pred`** (*`list`*) – list of predicted values on test set
- **`Y_test`** (*`list`*) – list of expected values on test set
- **`min_hist`** (*`int`*) – minimun of abscissa for metric distribution histogram
- **`max_hist`** (*`int`*) – maximum of abscissa for metric distribution histogram
- **`iter`** (*`int`*) – plot regression over a certain number of iterations
- **`save_distri`** (*`str`*) – path to save metric distribution
- **`save_regression`** (*`std`*) – path to save regression

Parm int kfold plot regression over a certain number of k-fold for each iteration

CHAPTER

FOUR

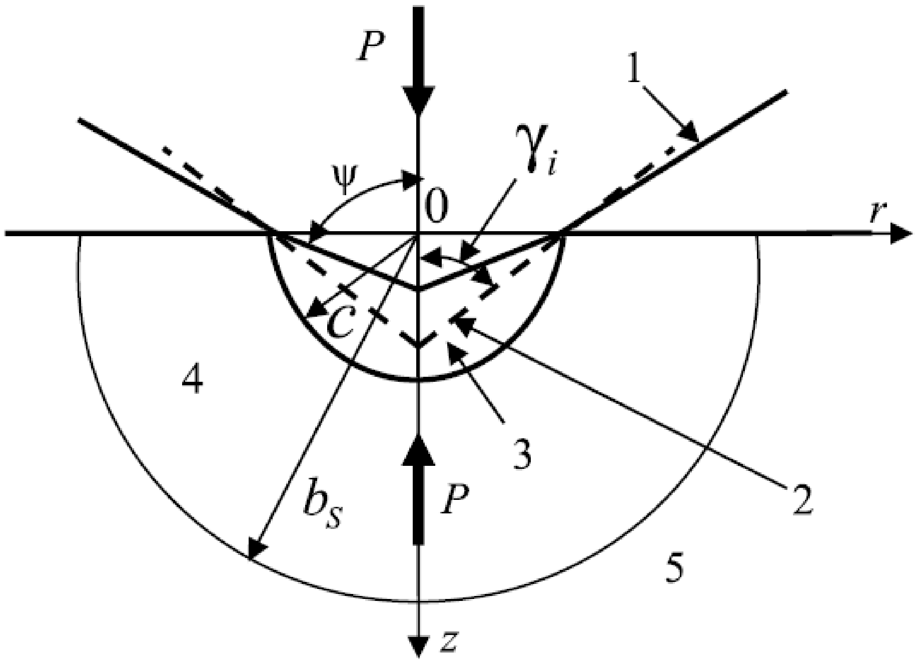
GALANOV MODEL MODULE

4.1 Galanov model

This code allows to compute elastic-plastic zone $\frac{b_s}{c}$, the constrain factor C and ductility characteristic δ_H by solving equations proposed by Galanov *et al* (Galanov, Ivanov, et Kartuzov, *Improved Core Model of the Indentation for the Experimental Determination of Mechanical Properties of Elastic-Plastic Materials and Its Application*.)

4.1.1 Model summary

Scheme of the interaction of a conical indenter and a sample under load P



We know:

- E_S : elastic modulus of the sample
- ν_S : Poisson coefficient of the sample
- E_i : elastic modulus of indenter

- ν_i : Poisson coefficient of indenter
- H : hardness of the sample
- $\cot\gamma_i = \sqrt[4]{\frac{\pi^2}{27}}\cot\gamma_B$ with $\gamma_B = 65$ for Brkovitch indenter

From:

$$E_i^* = \frac{E_i}{1-\nu_i**2}$$
$$Ks = \frac{E_s}{3(1-2\nu_s)}$$

We define:

$$\alpha_S = \frac{2(1-2\nu_s}{3(1-\nu_S)}$$
$$\beta_S = \frac{E_S}{6\,(1-\nu_S)\,H}$$
$$\theta_S = \frac{H}{K_S}$$

and dimensionless unknowns

$$x = \frac{b_S}{c}$$
$$y = \frac{Y_S}{HM}$$
$$z = \cot\psi$$

The system to solve is then:

$$\begin{cases} z = \cot\gamma_i - \frac{2HM}{E_i^*} \\ (1-\theta_S(1-2y\ln x))\left(x^3-\alpha_S\right) = \frac{\beta_S z}{y} \\ 1 = y\left(\frac{2}{3}+\frac{y\left(x^3-\alpha_S\right)}{\beta_S}\ln\left[1-\frac{y\left(x^3-\alpha_S\right)}{2\beta_S}\right]^{-1}+2\ln x\right) \end{cases}$$

were z is completely known. Moreover, $C = \frac{1}{y}$

`GalanovModel.`**`Galanov_math_values`**`(`*E_s*, *nu_s*, *E_i*, *nu_i*, *H_s*`)`

Compute the known values in Galanov models from machanical properties of sample and indenter

Parameters

- **E_s** (*float*) – elastic modulus of sample
- **nu_s** (*float*) – Poisson coefficient of sample
- **E_i** (*float*) – elastic modulus of indenter
- **nu_i** (*float*) – Poisson coefficient of indenter
- **H_s** (*float*) – Hardness of the sample

Returns

- Ei_star
- Ks
- alpha_s

- beta_s
- cot_gamma_i,z
- theta_s

Return type float

`GalanovModel.delta_H_value`(*E_s*, *H_s*, *nu_s*, *z*)

Compute ductility characteristics

Parameters

- **E_s** (*float*) – elastic modulus of sample
- **nu_s** (*float*) – Poisson coefficient of sample
- **H_s** (*float*) – Hardness of the sample
- **z** (*float*) – Galanov unknown

Returns delta_H the ductility characteristics

Return type float between 0 and 1

`GalanovModel.system_x_y`(*p*, **args*)

Solve Galanov sytem

Parameters

- **p** (*tuple*) – tuple of values of x and y
- **args** (*tuple*) – contains alpha_s,beta_s,theta_s,z

Returns result of equation 1 and 2 with x and y values

CHAPTER

FIVE

PARETO FRONT MODULE

5.1 Description

Module to train detect Pareto optimum points for two properties

Contains functions to :

- define Pareto front with a certain acceptation width
- plot Pareto optimal points among all points
- save Pareto optimal points

5.2 Functions

`ParetoFront.`**`pareto_frontier`**(*dataframe*, *objective1*, *objective2*, *acceptation_range_objective2*)

Pareto frontier selection process

Parameters

- **`dataframe`** (*`panda.DataFrame`*) – datasets that contains the two antagonistic properties values
- **`objective1`** (*`str`*) – column of the dataframe that contains the first objective/property to optimize
- **`objective2`** (*`str`*) – column of the dataframe that contains the second objective/property to optimize
- **`acceptation_range_objective2`** (*`float`*) – width of Pareto band, value is zero if we keep the strictly optimal values, higher than zero if we accept points close to the front.

Returns

- pareto_front: Pareto optimal points or near optimal points
- color: optimal points are associated to red, near optimal in orange.

Return type panda.DataFrame,str

`ParetoFront.`**`plot_pareto`**(*dataframe*, *pareto_front*, *objective1*, *objective2*, *txt_objective1*, *txt_objective2*, *colors*, *save*)

Plotting Pareto frontier

Parameters

- **dataframe** (*panda.DataFrame*) – datasets that contains the two antagonistic properties values
- **pareto_front** (*panda.DataFrame*) – Pareto optimal and near optimal points
- **objective1** (*str*) – column of the dataframe that contains the first objective/property to optimize
- **objective2** (*str*) – column of the dataframe that contains the second objective/property to optimize
- **txt_objective1** (*str*) – axis name corresponding to objective1
- **txt_objective2** (*str*) – axis name corresponding to objective2
- **colors** (*str*) – color associated with optimal (red) and near optimal(orange) points
- **save** (*str*) – path to save the Pareto front plot

Returns Nothing, just plot the Pareto front.

CHAPTER

SIX

PLOT PREDICTIONS

6.1 Principle

These Matlab modules plot Machine Learning prediction. It can scatter composition point of plot alpha shape of compositions points. User can add color or point size to plot along the predicted properties of the composition points.

6.2 Modules

`PlotPredictions.modules_pp.`**`plot_predictions`**(*DT*, *TR*, *name_elements*, *cell_coordinates*, *cell_type_plot*, *cell_colors*, *cell_size*)

Function that plot predicted properties of chosen compositions

Parameters

- **`DT,TR`** – Delaunay Traiangulation object for composition space plot
- **`name_elements`** (`str`) – contains the name of the alloy elements
- **`cell_coordinates`** (`cell(float)`) – contains the coordinates of the compositions to plot. Different groups can be plotted, these groups are in different cells
- **`cell_type_plot`** (`cell(str)`) – type of plot for the different groups of points. Can be simple scattering, convex hull or alpha shape.
- **`cell_colors`** (`cell`) – contains colors of the poiunts or hull.
- **`cell_size`** (`cell(float)`) – contains the size of the point if type is scatter. If other type, juts wright 0 in the cell

Returns show plot.

`PlotPredictions.modules_pp.`**`select_interp_compo`**(*database*, *compo_indices*, *output_column*, *isovalue*, *mask*, *connexions_table*, *interpolation*, *x*, *y*, *z*)

Select database lines thanks to a mask and return the associated coordinates. Can interpolate a property isolvalue and return the corresponding compositions.

Parameters

- **`database`** (`table`) – database as a table
- **`compo_indices`** (`list(int)`) – column indices that contains the compositions
- **`output_column`** (`str`) – output one wants to represent for a certain value
- **`isovalue`** (`float`) – value of output one want to lpot in the simplexe

- **mask** (list) – database selection. If isovalue is interpolated from data then the mask select a range of isovalue +/- delta. Else it select directly the isovalue
- **connexions_table** (array) – array of indices of nearest neighbours
- **interpolation** (bool) – boolean to indicate if isovalue is obtained from interpolation or not
- **x,y,z** (float) – coordinates of vertices (pure elements)

Returns

- selection_coord : coordinates of selected compositions

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX