pypec Documentation

Release 0.1.1

Milan Skocic

Nov 08, 2023

CONTENTS

CHAPTER

ONE

GETTING STARTED

1.1 Description

pypec aims at fitting PEC data with an graphical interface.

In terminal enter the following command:

```
python -m pypec
```

A pdf version of the documentation can be found here `pypec`. The source code can be viewed on GitHub.

1.2 How to install

```
$ python setup.py install

or

$ pip install pypec
```

1.3 Dependencies

```
numpy>=1.17
scipy>=1.5
matplotlib>=3.0.0
```

1.4 License

```
GNU General Public License v3 (GPLv3)
```

CHAPTER

TWO

USER GUIDE

2.1 PhotoElectrochemistry

2.1.1 Basics

Photo-electrochemistry characterizations are used to study films at macroscopic, mesoscopic, and microscopic scales. The latter advances were used to support (photo-)electrochemical studies of the electronic and optical properties of passive films and oxidized metals, and of their interfaces with electrolytes, providing informations on the nature and structure of these materials and to use properties such as the oxidation behaviour of a metallic substrate.

Basically, two kinds of curves are recorded in the course of photoelectrochemical characterization experiments, photocurrent voltammograms and photocurrent energy spectra. In photocurrent voltammograms, photocurrents are measured as a function of the potential, V, applied to the semiconducting electrode, at a given photon energy, $E = h\nu$. In photocurrent energy spectra, photocurrents are recorded, at a given applied potential, V, as a function of the photon energy, E. The analysis of the shapes of photocurrent voltammograms may allow to obtain informations such as the semiconducting type of the material, the energy of the surface band levels, the presence of macroscopic defects inducing photogenerated electron–hole pairs recombinations.

However, despite attempts to refine the Gartner-Butler model by taking into account surface or volume recombination, a complete description of the photocurrent voltammograms remains difficult, for the latter developments make use of a high number of adjustable parameters, most of them being very difficult to assess. The analysis of the photocurrent energy spectra is intended to identify the chemical nature of the material constituting the semiconducting electrode, through the value of their bandgap energies, E_g as, on the one hand, bandgap energy values have been reported in the literature for numerous compounds, and as, on the other hand, bandgap values may be estimated from thermodynamic extensive atomic data. Practically, photocurrent energy spectra are usually analyzed by means of linear transforms to take benefit of the fact that, using the simplified form of the Gartner–Butler model, the quantum yield, η, of the photocurrent is proportional to the light absorption coefficient.

In such conditions, η, obeys to the following relationship:

$$(\eta * E)^{1/n} = K(E - E_g)$$

where C is a constant (things other than E being equal), E_g is the bandgap energy of the semiconductor, and n depends on the band to band transition type, $n = 1/2$ for an allowed direct transition, and $n = 2$ for an allowed indirect transition. Direct transitions are rarely observed in more or less disordered thin oxide films.

2.1.2 Fitting

Linear transformations were successfully performed for oxides made of one or two constituents. However, for complex oxide scales formed of several p-type and n-type phases, the complete description of the photocurrent energy spectra could not be achieved, and only semi-quantitative and/or partial informations could be obtained on the oxides present in the scales.

As I^*_{PH} is measured under modulated light conditions and thus actually is a complex number, the real and the imaginary parts of the photocurrent should be considered simultaneously when analyzing and fitting the photocurrent energy spectra, rather than their modulus [1].

$$\begin{aligned} I^*_{PH} &= |I^*_{PH}| \cos\theta + j|I^*_{PH}| \sin\theta \\ I^*_{PH} &= \sum_{i}^{i=N} J_{PH,i} \cos\theta_i + j \sum_{i}^{i=N} J_{PH,i} \sin\theta_i \end{aligned} \tag{2.1}$$

where $J_{PH,i}$ and θ_i represent the modulus and phase shift, respectively, of the photocurrent issued from the ith semiconducting constituent of the oxide layer. For thin semiconducting films, the space charge regions are low compared to penetration depth of the light. $J_{PH,i}$ may thus be expected, at a given applied potential, to follow the simplified form of the Gartner–Butler model.

$$(J_{PH,i} * E)^{1/n} = K_i(E - E_{g,i})$$

where $E_{g,i}$ and K_i represent the energy gap and a value proportional to C (I^*_{PH} * is proportional to but not equal to η) for the ith semiconducting constituent.

For a given vector of m (K_i, θ_i, $E_{g,i}$) triplets, m representing the supposed number of semiconducting phases contributing to the photocurrent, the scalar function to be minimized by the Nelder-Mead function was defined as the product of the square roots of two quantities:

$$D_{Re} = \sqrt{\sum_{E}(ReI^*_{PH,exp} - ReI^*_{PH,calc})^2}$$

$$D_{Im} = \sqrt{\sum_{E}(ImI^*_{PH,exp} - ImI^*_{PH,calc})^2}$$

$$D = D_{Re}.D_{Im}$$

The 3 m variables can be locked or not by the user. Initial estimates can be provided by the user or can be randomly generated. Several successive calls of the Nelder-Mead procedure are necessary to reach the minimum of the scalar function and a stable set of the output parameters. The user is free to set the number of successive calls of the Nelder-Mead procedure. Constraints on the 3 m variables can be set by the user.

2.2 GUI

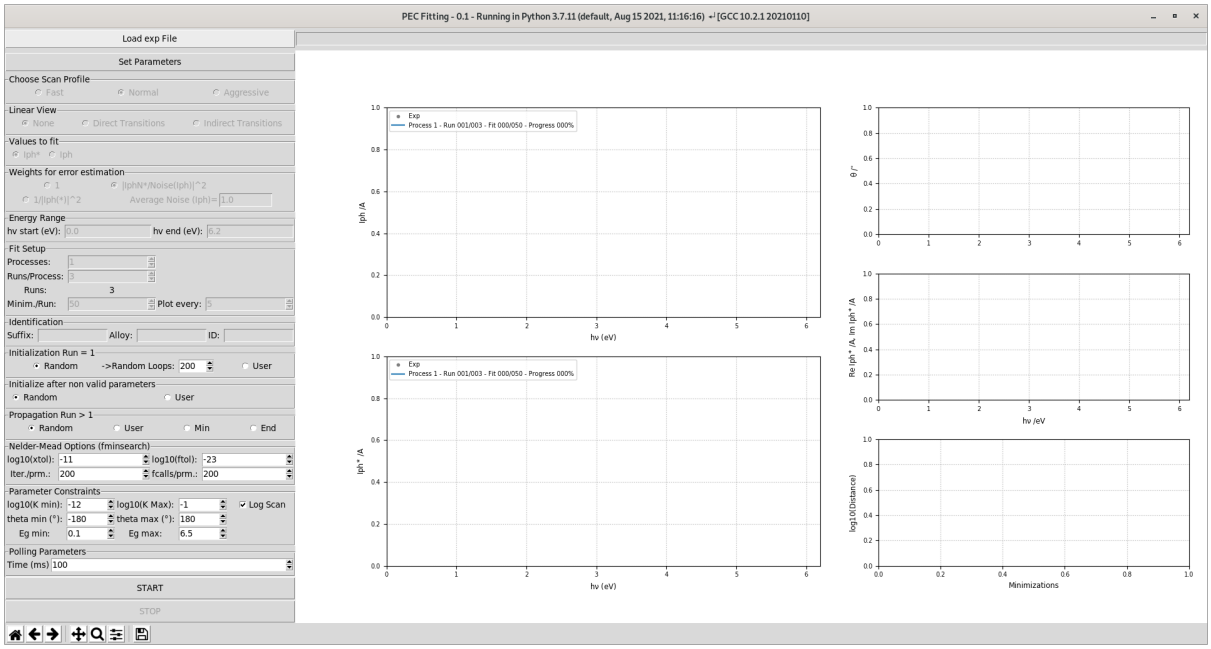
The main window contains all the elements necessary to run the fit. The design is cluttered but it allows exposing directly all the fitting settings to the user without deep menus.

The different steps, presented in the left pane, for performing the fit are:

- **load data: the accepted formats are:**
  - **.dot files* files which are ascii files developped in the SiMaP Lab
  - **.data files* files which are generic ascii files
- set the number of semiconductive contributions (*Parameter Table*)

At this point the fit can be ran.

If needed select custom choice for all the fitting parameter in the left pane.



2.2.1 *.dot files

They have a specific formating and they are provided by the PEC setup in the SiMaP Lab.

2.2.2 *.data files

The *.data files are generic ascii files where:

- the first column is the energy of the incident light in eV.
- the second column is the modulus of the photocurrent in A.
- the thrid column is the phase shift of the photocurrent in degrees.

2.2.3 Parameter Table

The parameter table allows for fitting the *3m* variables. The table is structured as shown below:

Ki	Fit Kgi	theta i	Fit Phase i	Egi	Fit Egi
K 1	0 or 1	Phase 1	0 or 1	Eg 1	0 or 1
K 2	0 or 1	Phase 2	0 or 1	Eg 2	0 or 1
...	...	...	...	...	...
K n	0 or 1	Phase n	0 or 1	Eg n	0 or 1

Each parameter K_i, θ_i and Eg_i can be locked by setting the **Fit X** column to 0.

CHAPTER

THREE

RELEASE NOTES

3.1 pypec 0.1.1 Release Note

3.1.1 Summary

- Switch to pyproject.toml.

3.1.2 Download

pypec

3.1.3 Contributors

Milan Skocic

3.1.4 Commits

Full Changelog: https://github.com/MilanSkocic/pypec/compare/0.1.0...0.1.1

3.2 pypec 0.1 Release Notes

3.2.1 Highlights

pypec aims at fitting PEC data with an graphical interface. The program was initially developped in Python 2.7 during my PhD from 2012 to 2015.

In 2016, the code was rewritten in Python 3 at the end of my PhD with slight modifications of the original code. The code remained untouched for almost 3 years and is now again under development in this repository.

This is the first release after migrating to github which will serve as a backup for the version developed during my PhD with completed documentation. No modification to the original code.

3.2.2 New Features

- Load data: *.dot files (data format developped in Grenoble Lab SiMaP)
- multi-process fitting: N processes x M fits x L loops
- automatic creation of result folder based on sample name and fitting parameters
- results saved in a folder

3.2.3 Download

pypec

3.2.4 Contributors

Milan Skocic

3.2.5 Commits

Full Changelog: https://github.com/MilanSkocic/pypec/compare/....0.1.0

CHAPTER

FOUR

AUTOGENERATED DOCUMENTATION

4.1 Graphical FrontEnd

Graphical frontend for fitting photo-current spectra.

class pypec.Analyse_PEC.**Analyse_PEC**(*master=None*)

Construct a frame widget with the parent MASTER.

Valid resource names: background, bd, bg, borderwidth, class, colormap, container, cursor, height, highlightbackground, highlightcolor, highlightthickness, relief, takefocus, visual, width.

Methods

update_nb_runs()	Update the number of cpu and run per process on the graphical interface.

AddFiles_cb
Fit_cb
ask_quit
autoscale
create_fit_lines
get_progress
on_Run_Fit
on_hv_limits
on_start_workers
on_stop_button
plot_Graph
plot_Re_Im
plot_fit_lines
plot_ligne_V
prm_binary
process_queue
remove_fit_folder
remove_fit_lines
start
update_figure
update_legend
update_nb_fit_in_run

AddFiles_cb()

Fit_cb()

ask_quit()

autoscale(*chart*)

create_fit_lines()

get_progress(*run=1*, *fit=0*)

on_Run_Fit()

on_hv_limits(**args*)

on_start_workers()

on_stop_button()

plot_Graph()

plot_Re_Im()

plot_fit_lines(**args*)

plot_ligne_V()

prm_binary()

process_queue()

remove_fit_folder()

remove_fit_lines()

start()

update_figure()

update_legend(*axes*, *run=1*, *fit=0*, *location='upper left'*)

update_nb_fit_in_run()

update_nb_runs()

Update the number of cpu and run per process on the graphical interface.

class pypec.Analyse_PEC.**ParameterTable**(*master*, *prm*, *last_prm_folder*, ***kwargs*)

Construct a frame widget with the parent MASTER.

Valid resource names: background, bd, bg, borderwidth, class, colormap, container, cursor, height, highlightbackground, highlightcolor, highlightthickness, relief, takefocus, visual, width.

class pypec.Analyse_PEC.**ParameterWindow**(*master*, *prm_init*, *last_prm_folder*)

Construct a toplevel widget with the parent MASTER.

Valid resource names: background, bd, bg, borderwidth, class, colormap, container, cursor, height, highlightbackground, highlightcolor, highlightthickness, menu, relief, screen, takefocus, use, visual, width.

Methods

get_paths
get_prm

get_paths()

get_prm()

class pypec.Analyse_PEC.**ScrolledFrame**(*master*, ***kwargs*)

Construct a frame widget with the parent MASTER.

Valid resource names: background, bd, bg, borderwidth, class, colormap, container, cursor, height, highlightbackground, highlightcolor, highlightthickness, relief, takefocus, visual, width.

4.2 Iph Functions

Modules - Iph Functions

This module contains functions requiered for computing and optimizing the photo-current values from input parameters i.e. triplets (K_i, θ_i, Eg_i) and experimental data for each semi-conductive phase [1].

pypec.iph_functions.**get_Iph_calc**(*hv*, *prm_array*, *phi_N*)

Compute the complex values of Iph based on the values and states of the triplets (K_i, θ_i, Eg_i) [1].

$$Iph^* = \frac{Iph}{\Phi_N}$$

Parameters

hv: 1d array
Vector of energies for which the complex Iph has to be computed.

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

phi_N: 1d array
Represents the values of the normalized photon flux to the maximum value. If nphf is a unity vector, the true photo-current is returned otherwise the as-measured photo-current is returned.

Returns

iph_calc_complex: 1d array
Vector of the computed complex values of Iph.

pypec.iph_functions.**get_LCC**(*iph_exp_complex*, *iph_calc_complex*)

pypec.iph_functions.**get_distance**(*iph_exp_complex*, *iph_calc_complex*)

Compute the distance D between Iph_{exp} and Iph_{calc}. The distance is computed by multiplying the distances on real and imaginary parts of Iph:

$$\begin{aligned}
\Delta Re &= Re\,Iph_{exp} - Re\,Iph_{calc} \\
\Delta Im &= Im\,Iph_{exp} - Im\,Iph_{calc} \\
D_{Re} &= \sqrt{\sum \Delta Re^2} \\
D_{Im} &= \sqrt{\sum \Delta Im^2} \\
D &= D_{Re} \cdot D_{Im}
\end{aligned}$$

Parameters

iph_exp_complex: 1d numpy array
Contains the complex values of the Iph_{exp}.

iph_calc_complex: 1d numpy array
Contains the complex values of the Iph_{calc}.

Returns

D: float
The computed distance on real and imaginary parts of Iph:.

pypec.iph_functions.**get_exp_data**(*filepath*)

Get the data array of data files according to their extension.

Supported files are .dot files recorded by PECLab software and .data files were the first three columns represent $h\nu$, $|Iph^*|$, θ.

Parameters

filepath: string
Path to the data file.

Returns

data_array: 2d array
Experimental data.

pypec.iph_functions.**get_header_footer_dot_file**(*filepath*)

Find the number of lines in header and footer in .dot files.

Parameters

filepath: path to the dot file

Returns

skip_header: int
number of lines in header

skip_footer: int
number of lines in footer

nbpoints: int
number of data lines

pypec.iph_functions.**get_random_prm_values**(*prm_array*, *K_bound=(1e-12, 0.1)*, *theta_bound=(-180.0, 180.0)*, *Eg_bound=(0.1, 6.2)*, *phase_flag=True*)

Generates random values for the triplets (K_i, θ_i, Eg_i) to be fitted based on the states given by the *prm_array*.

By default, the limits are:

- K_i: $[10^{-12}, 10^{-1}]$
- θ_i: $[-\pi, +\pi]$
- Eg_i: $[0.1, 6.0]$

Parameters

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

K_bound:tuple
Contains the lower and upper limits for the K_i values.

theta_bound: tuple
Contains the lower and upper limits for the θ_i values.

Eg_bound:tuple
Contains the lower and upper limits for the Eg_i values.

phase_flag: bool
Indicates if the values of θ_i have to be randomized.

Returns

random_prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

pypec.iph_functions.**get_results_array**(*hv*, *iph_exp_complex*, *iph_calc_complex*)

Build the data array of the experimental and calculated data: $h\nu$, $|Iph_{exp}|$, θ_{exp}, $|Iph_{calc}|$ and θ_{calc}

Parameters

hv: 1d numpy array
Contains the energy vector.

iph_exp_complex: 1d array
Contains the complex values of Iph_{exp}.

iph_calc_complex: 1d array
Contains the complex values of Iph_{calc}.

Returns

data_array: 2d array
Array containing the .

pypec.iph_functions.**get_summary**(*fit_folder*)

List result files for the triplets (K_i, θ_i, Eg_i) at the end and the minimum of each run.

Compute the distance, the LCCs for the energy interval that was used for minimizing the the triplets (K_i, θ_i, Eg_i).

The results are saved in 4 files: .SumEnd, .SumEndEg, *.SumMin, *.SumMinEg.

Parameters

fit_folder: string
Path of the fit folder.

pypec.iph_functions.**import_prm_file**(*filepath*)

Import the triplets (K_i, θ_i, Eg_i) from text file where each line represents a contributing semi-conductive phase.

Parameters

filepath: string
Absolute or relative file path to the text file.

Returns

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

pypec.iph_functions.**minimize**(*hv*, *iph_exp_complex*, *phi_N*, *weights*, *prm_array*, *Ki_log_flag=True*, *maxiter=None*, *maxfun=None*, *xtol=1e-11*, *ftol=1e-23*, *full_output=True*, *retall=False*, *disp=False*, *callback=None*)

Execute the Nelder-Mead algorithm based on parameter values given by `prm_array` and energy vector $h\nu$.

First, the `prm_array` is flattened and the parameters (K_i, θ_i, Eg_i) to be fitted are extracted and sent to the `target_function()` through the Nelder-Mead algorithm.

Once the parameters were computed by the Nelder-Mead algorithm, `the prm_array` is updated with the new values.

Parameters

hv: 1d numpy array
Contains the energy vector.

iph_exp_complex: 1d numpy array
Contains the complex values of the experimental photo-current.

phi_N: 1d array
Contains the normalized photon spectrum.

weights: 1d array
Contains the weights of the data.

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

Ki_log_flag: bool
Indicates if the K_i values are in logarithmic space.

maxiter
[int, optional] Maximum number of iterations to perform.

maxfun
[number, optional] Maximum number of function evaluations to make.

xtol
[float, optional] Relative error in xopt acceptable for convergence.

ftol
[number, optional] Relative error in func(xopt) acceptable for convergence.

full_output
[bool, optional] Set to True if fopt and warnflag outputs are desired.

retall
[bool, optional] Set to True to return list of solutions at each iteration.

disp
[bool, optional] Set to True to print convergence messages.

callback
[callable, optional] Called after each iteration, as callback(xk), where xk is the current parameter vector.

Returns

prm_array: 2d array
Represents the updated values and states of the triplets (K_i, θ_i, Eg_i).

fopt
[float] Value of function at minimum: `fopt = func(xopt)`.

pypec.iph_functions.**plot_summary**(*fit_folder*)

Plot the result files that were created by the *get_summary()* for he triplets (K_i, θ_i, Eg_i) at the end and the minimum of each run.

The results are saved in 2 files: -0-End.pdf, -0-Min.pdf.

Parameters

fit_folder: string
Path of the fit folder.

pypec.iph_functions.**save_pdf**(*filepath*, *hv*, *iph_exp_complex*, *iph_calc_complex*, *mask*, *all_results*)

pypec.iph_functions.**save_results**(*run*, *process_id*, *fit_folder*, *datafilepath*, *suffix*, *hv*, *mask*, *iph_exp_complex*, *phi_N*, *prm_min_run*, *prm_end_run*, *distance_min_run*, *distance_end_run*, *minimization_results*, *header_minimization_results*)

pypec.iph_functions.**scatter_logpolar**(*ax*, *theta*, *r_*, *ticks=5*, *bullseye=0.0*, ***kwargs*)

pypec.iph_functions.**shift_phase**(*prm_array*, *theta_bound=(-180.0, 180.0)*)

Compute the modulo of θ_i values with 2π and then shift the values of θ_i by the amplitude of the boundaries in order to be in between the boundaries.

By default, the boundaries for θ_i are set to $[-\pi, +\pi]$.

Parameters

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

theta_bound: tuple
Contains the lower and upper limits for the :math`theta _i` values.

Returns

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i) where the θ_i values were shifted.

pypec.iph_functions.**sort_prm_Eg**(*prm_array*)

Sort the `prm_array` based on values of Eg_i.

Parameters

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

Returns

prm_array: 2d array
Represents the sorted values and states of the triplets (K_i, θ_i, Eg_i).

pypec.iph_functions.**target_func**(*p*, *hv*, *prm_array*, *iph_exp_complex*, *phi_N*, *weights*, *Ki_log_flag=True*)

Update the triplets (K_i, θ_i, Eg_i) from the flattened parameter vector `p` sent by the optimization algorithm. The `prm_array` will be flattened and the indexes of the parameters to be fitted will be updated with `p` vector.

The calculated complex values of Iph will be sent along the experimental values to the *get_distance()* function. The value of the distance between the experimental and calculated data will sent back to the optimization algorithm.

Parameters

p: 1d array
Parameter vector sent by the optimization algorithm which is always. flattened.

hv: 1d array
Vector of energies for which the complex values of Iph have to be calculated.

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

iph_exp_complex: 1d numpy array
Contains the complex values of the experimental Iph.

phi_N: 1d array
Represents the values of the normalized photon flux to the maximum value.

weights: 1d array
Contains the values of the data weights.

Ki_log_flag: bool
Indicates if the K_i values are in logarithmic space.

Returns

distance: float
Calculated distance between experimental and calculated data values. See the *get_distance()* function.

pypec.iph_functions.**validate_prm**(*prm_array*, *K_bound=(1e-12, 0.1)*, *Eg_bound=(0.1, 6.2)*)

Check if the values of K_i and Eg_i are within the boundaries.

Parameters

prm_array: 2d array
Represents the values and states of the triplets (K_i, θ_i, Eg_i).

K_bound:tuple
Contains the lower and upper limits for the K_i values.

Eg_bound:tuple
Contains the lower and upper limits for the Eg_i values.

Returns

valid: bool
Set to True if value of K_i or Eg_i is out of the boundaries.

4.3 Parallel Processes

Module for controlling the parallel processes running during the fitting procedure

class pypec.Parallel_Process.**MinimizationProcess**(*output_queue*, *name*, *prm_init*, *nb_run*, *nb_SC*, *init_type*, *random_loops*, *hv*, *iph_exp_complex*, *iph_exp_complex_CI*, *phi_N*, *phi_N_CI*, *weights*, *hv_limits*, *nb_fit_in_run*, *fit_folder*, *filepath*, *suffix*, *NelderMead_options=(1e-11, 1e-23, 200, 200)*, *ParameterConstraints=((1e-12, 0.1), (-180, 180), (0.1, 6.2), True)*, *update_every=5*)

Methods

run()	Method to be run in sub-process; can be overridden in sub-class

shutdown

run()
Method to be run in sub-process; can be overridden in sub-class

shutdown()

class pypec.Parallel_Process.**PlotSummaryProcess**(*fit_folder*)

Parameters

fit_folder: str
Path to the folder where the summary files will be saved.

Attributes

fit_folder: str
Path to the folder where the summary files will be saved.

Methods

run()	Method to be run in sub-process; can be overridden in sub-class

run()
Method to be run in sub-process; can be overridden in sub-class

pypec.Parallel_Process.**get_cpu_number**()
DocString

pypec.Parallel_Process.**get_queue**()
DocString

pypec.Parallel_Process.**initialize_processes**(*n*, *daemon=True*, ***kwargs*)
Create N processes as daemons and return them in a list.

Parameters

n: int
Number of processes to be created.

daemon: bool
Flag for creating daemon process.

kwargs: dict
See *MinimizationProcess*

pypec.Parallel_Process.**start_processes**(*workers*)

Parameters

workers: subprocess workers

Returns

0: return integer

CHAPTER

FIVE

BIBLIOGRAPHY

5.1 Bibliography

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Photoelectrochemistry of Oxidation Layers: A Novel Approach to Analyze Photocurrent Energy Spectra. Petit, jp., boichot, r., loucif, a. et al. *Oxidation of Metals*, 2013.

PYTHON MODULE INDEX

p

INDEX

R

S

T

U

V