
elpy Documentation

Release unknown

Wai-Shing Luk

Dec 01, 2022

CONTENTS

1	Contents	3
1.1	License	3
1.2	Contributors	3
1.3	Changelog	3
1.4	ellpy	4
2	Indices and tables	11
	Python Module Index	13
	Index	15

This is the documentation of **elpy**.

CONTENTS

1.1 License

The MIT License (MIT)

Copyright (c) 2021 Wai-Shing Luk

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

1.2 Contributors

- Wai-Shing Luk <luk036@gmail.com>

1.3 Changelog

1.3.1 Version 0.1

- Feature A added
- FIX: nasty bug #1729 fixed
- add your changes here!

1.4 ellpy

1.4.1 ellpy package

Subpackages

ellpy.oracles package

Submodules

ellpy.oracles.chol_ext module

ellpy.oracles.cholutil module

ellpy.oracles.corr_bspline_oracle module

ellpy.oracles.corr_oracle module

ellpy.oracles.csdlowpass_oracle module

ellpy.oracles.cycle_ratio_oracle module

ellpy.oracles.gmi_oracle module

ellpy.oracles.lmi0_oracle module

ellpy.oracles.lmi_old_oracle module

ellpy.oracles.lmi_oracle module

ellpy.oracles.lowpass_oracle module

ellpy.oracles.lsq_corr_oracle module

ellpy.oracles.mle_corr_oracle module

ellpy.oracles.network_oracle module

ellpy.oracles.optscaling_oracle module

ellpy.oracles.profit_oracle module

ellpy.oracles.qmi_oracle module

ellpy.oracles.setup module

ellpy.oracles.spectral_fact module

Module contents

EllPy Oracles

Submodules

ellpy.cutting_plane module

class ellpy.cutting_plane.**CInfo**(feasible: *bool*, num_iters: *int*, status: *CUTStatus*)

Bases: *object*

class ellpy.cutting_plane.**CUTStatus**(value)

Bases: *Enum*

An enumeration.

noeffect = 3

nosoln = 1

smallenough = 2

success = 0

class ellpy.cutting_plane.**Options**

Bases: *object*

max_it: *int* = 2000

tol: *float* = 1e-08

ellpy.cutting_plane.**bsearch**(Omega: *~typing.Callable[[~typing.Any], bool]*, Interval: *~typing.Tuple*, options=<ellpy.cutting_plane.Options object>) → *Tuple*[*Any*, *CInfo*]

[summary]

Parameters

- **Omega** (*[type]*) – [description]
- **I** (*[type]*) – interval (initial search space)

Keyword Arguments

options (*[type]*) – [description] (default: {Options()})

Returns

[description]

Return type

[type]

class ellpy.cutting_plane.**bsearch_adaptor**(*P*, *S*, options=<ellpy.cutting_plane.Options object>)

Bases: *object*

property **x_best**

[summary]

Returns

[description]

Return type

[[type](#)]

`ellpy.cutting_plane.cutting_plane_dc`(*Omega*: `~typing.Callable[[~typing.Any, ~typing.Any], ~typing.Any]`, *S*, *t*, *options*=<`ellpy.cutting_plane.Options` object>) → `Tuple[Any, Any, CInfo]`

Cutting-plane method for solving convex optimization problem

Parameters

- **Omega** ([[type](#)]) – perform assessment on *x0*
- **S** ([[type](#)]) – Search Space containing *x**
- **t** ([float](#)) – initial best-so-far value

Keyword Arguments

options ([[type](#)]) – [description] (default: {`Options()`})

Returns

solution vector *t*: final best-so-far value ret {`CInfo`}

Return type

*x*_best (`Any`)

`ellpy.cutting_plane.cutting_plane_feas`(*Omega*: `~typing.Callable[[~typing.Any], ~typing.Any]`, *S*, *options*=<`ellpy.cutting_plane.Options` object>) → `CInfo`

Find a point in a convex set (defined through a cutting-plane oracle).

Description:

A function *f(x)* is *convex* if there always exist a *g(x)* such that $f(z) \geq f(x) + g(x)' \cdot (z - x)$, for all *z*, *x* in dom *f*. Note that dom *f* does not need to be a convex set in our definition. The affine function $g'(x - x_c) + \beta$ is called a cutting-plane, or a “cut” for short. This algorithm solves the following feasibility problem:

find *x* s.t. $f(x) \leq 0$,

A *separation oracle* asserts that an evaluation point *x0* is feasible, or provide a cut that separates the feasible region and *x0*.

Parameters

- **Omega** ([[type](#)]) – perform assessment on *x0*
- **S** ([[type](#)]) – Initial search space known to contain *x**

Keyword Arguments

options ([[type](#)]) – [description] (default: {`Options()`})

Returns

solution vector *niter*: number of iterations performed

Return type

x

`ellpy.cutting_plane.cutting_plane_q`(*Omega*, *S*, *t*, *options*=<`ellpy.cutting_plane.Options` object>)

Cutting-plane method for solving convex discrete optimization problem

Parameters

- **Omega** ([[type](#)]) – perform assessment on *x0*
- **S** ([[type](#)]) – Search Space containing *x**
- **t** ([float](#)) – initial best-so-far value

Keyword Arguments

options (*[type]*) – [description] (default: {Options()})

Returns

solution vector *t* (float): best-so-far optimal value *niter* ([type]): number of iterations performed

Return type

x_best (float)

ellpy.ell module

ellpy.problem module

class ellpy.problem.**Problem**(*S, oracle, options=<ellpy.cutting_plane.Options object>*)

Bases: *object*

property optim_value

The optimal value from the last time the problem was solved.

Return type

float or None

property optim_var

The optimal value from the last time the problem was solved.

Return type

x_best or None

solve(*args, **kwargs)

Solves the problem using the specified method.

Parameters

- **method** (*function*) – The solve method to use.
- **solver** (*str, optional*) – The solver to use.
- **verbose** (*bool, optional*) – Overrides the default of hiding solver output.
- **solver_specific_opts** (*dict, optional*) – A dict of options that will be passed to the specific solver. In general, these options will override any default settings imposed by cvxpy.

Returns

The optimal value for the problem, or a string indicating why the problem could not be solved.

Return type

float

property solver_stats

Returns an object containing additional information returned by the solver.

property status

The status from the last time the problem was solved.

Return type

str

class `ellpy.problem.SolverStats`(*solver_name*)

Bases: `object`

Reports some of the miscellaneous information that is returned by the solver after solving but that is not captured directly by the Problem instance.

num_iters

The number of iterations the solver had to go through to find a solution.

Type

`int`

ellpy.skeleton module

This is a skeleton file that can serve as a starting point for a Python console script. To run this script uncomment the following lines in the `[options.entry_points]` section in `setup.cfg`:

```
console_scripts =
    fibonacci = ellpy.skeleton:run
```

Then run `pip install .` (or `pip install -e .` for editable mode) which will install the command `fibonacci` inside your current environment.

Besides console scripts, the header (i.e. until `_logger...`) of this file can also be used as template for Python modules.

Note: This skeleton file can be safely removed if not needed!

References

- https://setuptools.readthedocs.io/en/latest/userguide/entry_point.html
- https://pip.pypa.io/en/stable/reference/pip_install

`ellpy.skeleton.fib`(*n*)

Fibonacci example function

Parameters

n (`int`) – integer

Returns

n-th Fibonacci number

Return type

`int`

`ellpy.skeleton.main`(*args*)

Wrapper allowing `fib()` to be called with string arguments in a CLI fashion

Instead of returning the value from `fib()`, it prints the result to the `stdout` in a nicely formatted message.

Parameters

args (`List[str]`) – command line parameters as list of strings (for example `["--verbose", "42"]`).

`ellpy.skeleton.parse_args(args)`

Parse command line parameters

Parameters

args (`List[str]`) – command line parameters as list of strings (for example ["--help"]).

Returns

command line parameters namespace

Return type

`argparse.Namespace`

`ellpy.skeleton.run()`

Calls `main()` passing the CLI arguments extracted from `sys.argv`

This function can be used as entry point to create console scripts with `setuptools`.

`ellpy.skeleton.setup_logging(loglevel)`

Setup basic logging

Parameters

loglevel (`int`) – minimum loglevel for emitting messages

Module contents

Note: This is the main page of your project's [Sphinx](#) documentation. It is formatted in [reStructuredText](#). Add additional pages by creating `rst`-files in `docs` and adding them to the `toctree` below. Use then [references](#) in order to link them from this page, e.g. [Contributors](#) and [Changelog](#).

It is also possible to refer to the documentation of other Python packages with the [Python domain syntax](#). By default you can reference the documentation of [Sphinx](#), [Python](#), [NumPy](#), [SciPy](#), [matplotlib](#), [Pandas](#), [Scikit-Learn](#). You can add more by extending the `intersphinx_mapping` in your Sphinx's `conf.py`.

The pretty useful extension [autodoc](#) is activated by default and lets you include documentation from docstrings. Docstrings can be written in [Google style](#) (recommended!), [NumPy style](#) and [classical style](#).

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

e

`ellpy`, [9](#)

`ellpy.cutting_plane`, [5](#)

`ellpy.oracles`, [5](#)

`ellpy.problem`, [7](#)

`ellpy.skeleton`, [8](#)

B

bsearch() (in module *ellpy.cutting_plane*), 5
bsearch_adaptor (class in *ellpy.cutting_plane*), 5

C

CInfo (class in *ellpy.cutting_plane*), 5
CUTStatus (class in *ellpy.cutting_plane*), 5
cutting_plane_dc() (in module *ellpy.cutting_plane*), 6
cutting_plane_feas() (in module *ellpy.cutting_plane*), 6
cutting_plane_q() (in module *ellpy.cutting_plane*), 6

E

ellpy
 module, 9
ellpy.cutting_plane
 module, 5
ellpy.oracles
 module, 5
ellpy.problem
 module, 7
ellpy.skeleton
 module, 8

F

fib() (in module *ellpy.skeleton*), 8

M

main() (in module *ellpy.skeleton*), 8
max_it (*ellpy.cutting_plane.Options* attribute), 5
module
 ellpy, 9
 ellpy.cutting_plane, 5
 ellpy.oracles, 5
 ellpy.problem, 7
 ellpy.skeleton, 8

N

noeffect (*ellpy.cutting_plane.CUTStatus* attribute), 5
nosoln (*ellpy.cutting_plane.CUTStatus* attribute), 5
num_iters (*ellpy.problem.SolverStats* attribute), 8

O

optim_value (*ellpy.problem.Problem* property), 7
optim_var (*ellpy.problem.Problem* property), 7
Options (class in *ellpy.cutting_plane*), 5

P

parse_args() (in module *ellpy.skeleton*), 8
Problem (class in *ellpy.problem*), 7

R

run() (in module *ellpy.skeleton*), 9

S

setup_logging() (in module *ellpy.skeleton*), 9
smallemough (*ellpy.cutting_plane.CUTStatus* attribute), 5
solve() (*ellpy.problem.Problem* method), 7
solver_stats (*ellpy.problem.Problem* property), 7
SolverStats (class in *ellpy.problem*), 7
status (*ellpy.problem.Problem* property), 7
success (*ellpy.cutting_plane.CUTStatus* attribute), 5

T

tol (*ellpy.cutting_plane.Options* attribute), 5

X

x_best (*ellpy.cutting_plane.bsearch_adaptor* property), 5