impax Documentation

Release 0.1.2

ClimateImpactLab

Dec 01, 2019

Contents

This package contains tools for projecting impacts at the Climate Impact Lab

Contents:

CHAPTER 1

impax

Impact Forecasting for the Climate Impact Lab

- Free software: MIT license
- Documentation: https://impax.readthedocs.io.

1.1 Features

- Compute impacts from a variance/covariance matrix
- Read and sample from CSVV files
- Produce diagnostic plots helpful for impact assessment

1.2 Credits

This package was created by Justin Simcock and Michael Delgado at the Climate Impact Lab.

CHAPTER 2

Installation

2.1 Stable release

To install impax, run this command in your terminal:

```
$ pip install impax
```

This is the preferred method to install impax, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for impax can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/]/impax
```

Or download the tarball:

```
$ curl  -OL https://github.com/]/impax/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use impax in a project:

```
import impax
```

CHAPTER 4

impax

4.1 impax package

4.1.1 Submodules

4.1.2 impax.cli module

Console script for impax.

4.1.3 impax.estimate module

class impax.estimate.**MultivariateNormalEstimator**(*coefficients*, *vcv*, *index*)

Bases: object

Stores a median and residual VCV matrix for multidimensional variables with named indices and provides multivariate sampling and statistical analysis functions

Parameters

- **coefficients** (*array*) – length $(m_1 * m_2 * \cdots * m_n)$ 1-d numpy.ndarray with regression coefficients
- **vcv** (*array*) – $(m_1 * m_2 * \cdots * m_n)x(m_1 * m_2 * \cdots * m_n)$ numpy.ndarray with variance-covariance matrix for multivariate distribution
- **index** (*Index*) – Index or $(m_1 * m_2 * \cdots * m_n)$ 1-d MultiIndex describing the multivariate space

median()

Returns the median values (regression coefficients)

Returns median – DataArray of coefficients

Return type DataArray

sample(*seed=None*)
Sample from the multivariate normal distribution

Takes a draw from a multivariate distribution and returns an `xarray.DataArray` of parameter estimates.

Returns **draw** – `DataArray` of parameter estimates drawn from the multivariate normal

Return type DataArray

impax.estimate.**get_gammas**(**args*, ***kwargs*)

impax.estimate.**read_csvv**(*csvv_path*)
Returns the estimator object from a CSVV file

Parameters **path** (*str_or_buffer*) – path to csvv file

Returns **estimator** – `Gamma` object with median and VCV matrix indexed by prednames, covarnames, and outcomes

Return type *MultivariateNormalEstimator*

4.1.4 impax.impax module

class impax.impax.**Impact**
Bases: `object`

Base class for computing an impact as specified by the Climate Impact Lab

compute(*weather*, *betas*, *clip_flat_curve=True*, *t_star=None*)
Computes an impact for a unique set of gdp, climate, weather and gamma coefficient inputs. For each set of these, we take the analytic minimum value between two points, save t_star to disk and compute analytical min for function m_star for a given covariate set.

This operation is called for every adaptation scenario specified in the run script.

Parameters

- **weather** (*DataArray*) – weather `DataArray`
- **betas** (*DataArray*) – covarname by outcome `DataArray`
- **clip_flat_curve** (*bool*) – flag indicating that flat-curve clipping should be performed on the result
- **t_star** (*DataArray*) – `xarray.DataArray` with minimum temperatures used for clipping

Returns

Return type py:class *~xarray.Dataset* of impacts by hierid by outcome group

compute_t_star(*betas*, *bounds=None*)

get_t_star(*betas*, *bounds*, *t_star_path=None*)
Read precomputed t_star

Parameters

- **betas** (*DataArray*) – `DataArray` of betas as prednames by hierid
- **bounds** (*list*) – values between which to evaluate function
- **path** (*str*) – place to load t-star from

impact_function(*betas*, *weather*)
computes the dot product of betas and annual weather by outcome group

Parameters

- **betas** (*DataArray*) – DataArray of hierid by predname by outcome
- **weather** (*DataArray*) – DataArray of hierid by predname by outcome

Returns

- *DataArray* – DataArray of impact by outcome by hierid
- *.. note::* – overrides *impact_function* method in Impact base class

min_function
alias of exceptions.NotImplementedError

postprocess_annual(*impact*)

postprocess_daily(*impact*)

class impax.impax.**PolynomialImpact**
Bases: *impax.impax.Impact*

Polynomial-specific Impact spec, with ln(gdppc) and climtas for covariates

static min_function(***kwargs*)
helper function to call minimization function for given mortality polynomial spec mortality_polynomial implements findpolymin through *np.apply_along_axis*

Parameters

- **betas** (*DataArray*) – DataArray of hierid by predname by outcome
- **dim** (*str*) – dimension to apply minimization to
- **bounds** (*list*) – values between which to search for t_star

Returns

Return type DataArray of hierid by predname by outcome

Note: overides *min_function* in Impact base class

---

impax.impax.**construct_covars**(*add_constant=True*, ***covars*)
Helper function to construct the covariates dataarray

Parameters

- **add_constant** (*bool*) – flag indicating whether a constant term should be added. The constant term will have the same shape as the other covariate DataArrays
- **covars** (*dict*) – dictionary of covariate name, covariate (str path or xarray.DataArray) pairs

Returns **combined** – Combined DataArray of covariate variables, with variables concatenated along the new *covarnames* dimension

Return type DataArray

impax.impax.**construct_weather**(***weather*)
Helper function to build out weather dataarray

Parameters **weather** (*dict*) – dictionary of prednames and weather (either `str` file paths or `xarray.DataArray` objects) for each predname

Returns **combined** – Combined `DataArray` of weather variables, with variables concatenated along the new *prednames* dimension

Return type DataArray

4.1.5 impax.mins module

`impax.mins.`**minimize_polynomial**(*da*, *dim='prednames'*, *bounds=(-inf, inf)*)

Finds the minimizing values of polynomials given an array of coefficients

---

Note: The coefficients along the dimension `dim` must be in _ascending_ power order and must not contain the zeroth-order term.

Parameters

- **da** (*DataArray*) – `DataArray` of coefficients of a `(da.size[dim])`-order polynomial in ascending power order along the dimension `dim`. The coefficients must not contain the zeroth-order term.
- **dim** (*str, optional*) – dimension along which to evaluate the coefficients (default `prednames`)
- **bounds** (*list, optional*) – domain on the polynomial within which to search for the minimum value, default `(-inf, inf)`

Returns `DataArray` in the same shape as da, with the minimizing value of the polynomial raised to the appropriate power in place of each coefficient

Return type DataArray

Examples

Create an array with two functions:

..math:

```
egin{array}{rcl}
    f_1 & = & x^2 \
    f_2 & = & -x^2 + 2x
\end{array}
```

This is specified as a 2-dimensional `xarray.DataArray`:

```
>>> da = xr.DataArray(
...      [[0, 1],   # x^2
...       [2, -1]], # -x^2 + 2x
...      dims=('spec', 'x'),
...      coords={'spec': ['f1', 'f2'], 'x': ['x1', 'x2']})
...
```

These functions can be minimized using *impax.mins.minimize_polynomial()*:

```
>>> minimize_polynomial(
...     da, dim='x')
...
<xarray.DataArray (spec: 2, x: 2)>
array([[  0.,   0.],
       [-inf,  inf]])
Coordinates:
  * x        (x) ... 'x1' 'x2'
  * spec     (spec) ... 'f1' 'f2'
```

Use the same function, but impose the domain limit $[2, 4]$:

```
>>> minimize_polynomial(
...     da, dim='x', bounds=[2, 4])
...
<xarray.DataArray (spec: 2, x: 2)>
array([[  2.,   4.],
       [  4.,  16.]])
Coordinates:
  * x        (x) ... 'x1' 'x2'
  * spec     (spec) ... 'f1' 'f2'
```

4.1.6 Module contents

impax.**minimize_polynomial**(*da*, *dim='prednames'*, *bounds=(-inf, inf)*)

Finds the minimizing values of polynomials given an array of coefficients

Note: The coefficients along the dimension `dim` must be in _ascending_ power order and must not contain the zeroth-order term.

Parameters

- **da** (*DataArray*) – `DataArray` of coefficients of a `(da.size[dim])`-order polynomial in ascending power order along the dimension `dim`. The coefficients must not contain the zeroth-order term.
- **dim** (*str, optional*) – dimension along which to evaluate the coefficients (default `prednames`)
- **bounds** (*list, optional*) – domain on the polynomial within which to search for the minimum value, default `(-inf, inf)`

Returns `DataArray` in the same shape as da, with the minimizing value of the polynomial raised to the appropriate power in place of each coefficient

Return type DataArray

Examples

Create an array with two functions:

..math:

```
egin{array}{rcl}
    f_1 & = & x^2 \
    f_2 & = & -x^2 + 2x
\end{array}
```

This is specified as a 2-dimensional `xarray.DataArray`:

```
>>> da = xr.DataArray(
...     [[0, 1],   # x^2
...      [2, -1]], # -x^2 + 2x
...     dims=('spec', 'x'),
...     coords={'spec': ['f1', 'f2'], 'x': ['x1', 'x2']})
...
```

These functions can be minimized using *impax.mins.minimize_polynomial()*:

```
>>> minimize_polynomial(
...     da, dim='x')
...
<xarray.DataArray (spec: 2, x: 2)>
array([[  0.,   0.],
       [-inf,  inf]])
Coordinates:
  * x        (x) ... 'x1' 'x2'
  * spec     (spec) ... 'f1' 'f2'
```

Use the same function, but impose the domain limit $[2, 4]$:

```
>>> minimize_polynomial(
...     da, dim='x', bounds=[2, 4])
...
<xarray.DataArray (spec: 2, x: 2)>
array([[  2.,   4.],
       [  4.,  16.]])
Coordinates:
  * x        (x) ... 'x1' 'x2'
  * spec     (spec) ... 'f1' 'f2'
```

impax.**construct_covars**(*add_constant=True*, ***covars*)

Helper function to construct the covariates dataarray

Parameters

- **add_constant** (*bool*) – flag indicating whether a constant term should be added. The constant term will have the same shape as the other covariate DataArrays
- **covars** (*dict*) – dictionary of covariate name, covariate (`str` path or `xarray.DataArray`) pairs

Returns **combined** – Combined `DataArray` of covariate variables, with variables concatenated along the new *covarnames* dimension

Return type DataArray

impax.**construct_weather**(***weather*)

Helper function to build out weather dataarray

Parameters **weather** (*dict*) – dictionary of prednames and weather (either `str` file paths or `xarray.DataArray` objects) for each predname

Returns **combined** – Combined `DataArray` of weather variables, with variables concatenated along the new *prednames* dimension

Return type DataArray

impax.**read_csvv**(*csvv_path*)

Returns the estimator object from a CSVV file

Parameters **path** (*str_or_buffer*) – path to csvv file

Returns **estimator** – `Gamma` object with median and VCV matrix indexed by prednames, covarnames, and outcomes

Return type *MultivariateNormalEstimator*

class impax.**MultivariateNormalEstimator**(*coefficients*, *vcv*, *index*)

Bases: `object`

Stores a median and residual VCV matrix for multidimensional variables with named indices and provides multivariate sampling and statistical analysis functions

Parameters

- **coefficients** (*array*) – length $(m_1 * m_2 * \cdots * m_n)$ 1-d `numpy.ndarray` with regression coefficients
- **vcv** (*array*) – $(m_1 * m_2 * \cdots * m_n)x(m_1 * m_2 * \cdots * m_n)$ `numpy.ndarray` with variance-covariance matrix for multivariate distribution
- **index** (*Index*) – `Index` or $(m_1 * m_2 * \cdots * m_n)$ 1-d `MultiIndex` describing the multivariate space

median()

Returns the median values (regression coefficients)

Returns **median** – `DataArray` of coefficients

Return type DataArray

sample(*seed=None*)

Sample from the multivariate normal distribution

Takes a draw from a multivariate distribution and returns an `xarray.DataArray` of parameter estimates.

Returns **draw** – `DataArray` of parameter estimates drawn from the multivariate normal

Return type DataArray

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/ClimateImpactLab/impax/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

impax could always use more documentation, whether as part of the official impax docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/ClimateImpactLab/impax/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *impax* for local development.

1. Fork the *impax* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/impax.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv impax
$ cd impax/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 impax tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/ClimateImpactLab/impax/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_impax
```

CHAPTER 6

Credits

This repository is a project of the Climate Impact Lab

6.1 Development Lead

- Justin Simcock <jsimcock@rhg.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.1.2 (Current version)

7.1.1 API changes

- `impax.csvv.get_gammas()` has been deprecated. Use `impax.read_csvv()` instead (GH #37)
- `_prep_gammas()` has been removed, and `sample()` now takes no arguments and returns a sample by default. Seeding the random number generator is now left up to the user (GH #36)

7.1.2 Bug fixes

- fix py3k compatability issues (GH #39)
- fix travis status icon in README
- add tests for `impax.mins._minimize_polynomial()`, fix major math errors causing a failure to find minima in *`impax.mins`* module, and clarify documentation (GH #58)

7.2 0.1.0 (2017-10-12)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index