## Contents

1  Installation  3  
2  pyblock API  5  
3  pyblock tutorial  11  
4  References  17  
5  Indices and tables  19  
Bibliography  21  
Python Module Index  23  

pyblock implements the reblocking analysis (see, for example, the description given by Flyvbjerg and Petersen\textsuperscript{1}), to remove serial correlation from a data set and hence obtain an improved estimate of the standard error. Functions for additional analysis, interpretation and manipulation of the resultant mean and standard error estimates are also provided.

A command-line interface is currently not provided but the \textit{API} is simple to use from either a Python/IPython shell or to create an application-specific script.

CHAPTER 1

Installation

1.1 Dependencies

- numpy
- pandas (0.13 and later)
- matplotlib

Pandas is only required for `pyblock.pd_utils` and `pyblock.error` and matplotlib for `pyblock`.

`pd_utils`. Hence pandas and/or matplotlib need not be installed if those submodules are not required, in which case `pyblock/__init__.py` must be modified to stop the `pyblock.pd_utils` and `pyblock.error` from being automatically imported.

1.2 Installation instructions

`pyblock` can be installed from PyPI:

```
$ pip install pyblock
```

Or from the source package:

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

Both `pip` and `setup.py` have options for installing in non-default locations, such as home directories. Add `--help` to the above commands for details.

Alternatively, `pyblock` can be used directly from source by adding the location of the `pyblock` directory to the `PYTHONPATH` environment variable.
pyblock is a python module for analysis of correlated data.

## 2.1 pyblock.blocking

Tools for reblocking of data to remove serial correlation from data sets.

**pyblock.blocking.reblock** *(data, rowvar=1, ddof=None, weights=None)*

Blocking analysis of correlated data.

Repeatedly average neighbouring data points in order to remove the effect of serial correlation on the estimate of the standard error of a data set, as described by Flyvbjerg and Petersen [*Flyvbjerg*]. The standard error is constant (within error bars) once the correlation has been removed.

If a weighting is provided then the weighted variance and standard error of each variable is calculated, as described in [*Pozzi*]. Bessel correction is obtained using the “effective sample size” from [*Madansky*].

- **data** [numpy.ndarray] 1D or 2D array containing multiple variables and data points. See rowvar.
- **rowvar** [int] If rowvar is non-zero (default) then each row represents a variable and each column a data point per variable. Otherwise the relationship is swapped. Only used if data is a 2D array.
- **ddof** [int] If not None, then the standard error and covariance are normalised by \( (N - ddof) \), where \( N \) is the number of data points per variable. Otherwise, the numpy default is used (i.e. \( (N - 1) \)).
- **weights** [numpy.array] A 1D weighting of the data to be reblocked. For multidimensional data an identical weighting is applied to the data for each variable.

**block_info** [list of collections.namedtuple()] Statistics from each reblocking iteration. Each tuple contains:

- **block** [int] blocking iteration. Each iteration successively averages neighbouring pairs of data points. The final data point is discarded if the number of data points is odd.
- **ndata**: int number of data points in the blocking iteration.
mean [numpy.ndarray] mean of each variable in the data set.

cov [numpy.ndarray] covariance matrix.

std_err [numpy.ndarray] standard error of each variable.

std_err_err [numpy.ndarray] an estimate of the error in the standard error, assuming a Gaussian distribution.

**pyblock.blocking.find_optimal_block (ndata, stats)**

Find the optimal block length from a reblocking calculation.

Inspect a reblocking calculation and find the block length which minimises the stochastic error and removes the effect of correlation from the data set. This follows the procedures detailed by [Wolff] and [Lee] et al.

ndata [int] number of data points (‘observations’) in the data set.

stats [list of tuples] statistics in the format as returned by `pyblock.blocking.reblock()`.

list of int the optimal block index for each variable (i.e. the first block index in which the correlation has been removed). If NaN, then the statistics provided were not sufficient to estimate the correlation length and more data should be collected.

[Wolff] (Eq 47) and [Lee] et al. (Eq 14) give the optimal block size to be

$$B^3 = 2n^2_{cor}$$

where $n$ is the number of data points in the data set, $B$ is the number of data points in each ‘block’ (ie the data set has been divided into $n/B$ contiguous blocks) and $n_{cor}$. [todo] - describe $n_{cor}$. Following the scheme proposed by [Lee] et al., we hence look for the largest block size which satisfies

$$B^3 >= 2n^2_{cor}$$

From Eq 13 in [Lee] et al. (which they cast in terms of the variance):

$$n_{err}SE = SE_{true}$$

where the ‘error factor’, $n_{err}$, is the square root of the estimated correlation length, $SE$ is the standard error of the data set and $SE_{true}$ is the true standard error once the correlation length has been taken into account. Hence the condition becomes:

$$B^3 >= 2n(SE(B)/SE(0))^4$$

where $SE(B)$ is the estimate of the standard error of the data divided in blocks of size $B$.

I am grateful to Will Vigor for discussions and the initial implementation.

### 2.2 pyblock.error

Simple error propogation.

**Note:** We only implement the functions as we need them...

**pyblock.error.ratio (stats_A, stats_B, cov_AB, data_len)**

Calculate the mean and standard error of $f(A,B) = A/B$. 
**stats_A** [pandas.Series or pandas.DataFrame] Statistics (containing at least the ‘mean’ and ‘standard error’ fields) for variable $A$. The rows contain different values of these statistics (e.g. from a reblocking analysis) if pandas.DataFrame are passed.

**stats_B** [pandas.Series or pandas.DataFrame] Similarly for variable $B$.

**cov_AB** [float or pandas.Series] Covariance between variables $A$ and $B$. If stats_A and stats_B are pandas.DataFrame, then this must be a pandas.Series, with the same index as stats_A and stats_B.

**data_len** [int or pandas.Series] Number of data points (‘observations’) used to obtain the statistics given in stats_A and stats_B. If stats_A and stats_B are pandas.DataFrame, then this must be a pandas.Series, with the same index as stats_A and stats_B.

**stats** [pandas.Series or pandas.DataFrame] Mean and standard error (and, if possible/relevant, optimal reblock iteration) for $f(A, B)$. If stats_A, stats_B are pandas.DataFrame, this is a pandas.DataFrame with the same index, otherwise a pandas.Series is returned.

**pyblock.error.product** (stats_A, stats_B, cov_AB, data_len)
Calculate the mean and standard error of $f(A, B) = A \times B$.

See ratio().

**pyblock.error.substraction** (stats_A, stats_B, cov_AB, data_len)
Calculate the mean and standard error of $f(A, B) = A - B$.

See ratio().

**pyblock.error.addition** (stats_A, stats_B, cov_AB, data_len)
Calculate the mean and standard error of $f(A, B) = AB$.

See ratio().

**pyblock.error.pretty_fmt_err** (val, err)
Pretty formatting of a value and associated error.

val [number] a (noisy) value.

err: number error associated with the value.

val_str [str] Value to the number of significant digits known, with the error in the last digit in brackets.

```python
>>> pretty_fmt_err(1.2345, 0.01)
'1.23(1)'
>>> pretty_fmt_err(12331, 40)
'12330(40)'
```

Rounding is handled with Python’s `round` function, which handles rounding numbers at the midpoint in a range (eg 5 if round to the nearest 10) in a slightly odd way. As we’re normally dealing with noisy data and rounding to remove more than just one significant figure, this is unlikely to impact us.
2.3 pyblock.pd_utils

Pandas-based wrapper around pyblock.blocking.

**pyblock.pd_utils.reblock**(data, axis=0, weights=None)

Blocking analysis of correlated data.

- **data** [pandas.Series or pandas.DataFrame] Data to be blocked. See axis for order.
- **axis** [int] If non-zero, variables in data are in rows with the columns corresponding to the observation values. Blocking is then performed along the rows. Otherwise each column is a variable, the observations are in the columns and blocking is performed down the columns. Only used if data is a pandas.DataFrame.
- **weights** [pandas.Series or pandas.DataFrame] A 1D weighting of the data to be reblocked. For multidimensional data an identical weighting is applied to the data for each variable.
- **data_len** [pandas.Series] Number of data points used in each reblocking iteration. Note some reblocking iterations discard a data point if there were an odd number of data points in the previous iteration.
- **block_info** [pandas.DataFrame] Mean, standard error and estimated standard error for each variable at each reblock step.
- **covariance** [pandas.DataFrame] Covariance matrix at each reblock step.

**pyblock.blocking.reblock()**: numpy-based implementation; see for documentation and notes on the reblocking procedure. pyblock.pd_utils.reblock() is a simple wrapper around this.

**pyblock.pd_utils.optimal_block**(block_sub_info)

Get the optimal block value from the reblocking data.

- **block_sub_info**: pandas.DataFrame or pandas.Series Reblocking data (i.e. the first item of the tuple returned by reblock), or a subset thereof containing the statistics columns for one or more data items.
- **index** [int] Reblocking index corresponding to the reblocking iteration at which serial correlation has been removed (as estimated by the procedure in pyblock.blocking.find_optimal_block). If multiple data sets are passed in block_sub_info, this is the maximum index out of all data sets. Set to inf if an optimal block is not found for a data set.

ValueError block_sub_info contains no Series or column in DataFrame named ‘optimal block’.

**pyblock.pd_utils.reblock_summary**(block_sub_info)

Get the data corresponding to the optimal block from the reblocking data.

- **block_sub_info**: pandas.DataFrame or pandas.Series Reblocking data (i.e. the first item of the tuple returned by reblock), or a subset thereof containing the statistics columns for one or more data items.
- **summary** [pandas.DataFrame] Mean, standard error and estimate of the error in the standard error corresponding to the optimal block size in the reblocking data (or largest optimal size if multiple data sets are given. The index is labelled with the data name, if known. An empty DataFrame is returned if no optimal block size was found.

2.4 pyblock.plot

Helper for plotting reblocking plots.
pyblock.plot.plot_reblocking(block_info, plotfile=None, plotshow=True)

Plot the reblocking data.

block_info [pandas.DataFrame] Reblocking data (i.e. the first item of the tuple returned by reblock).

plotfile [string] If not null, save the plot to the given filename. If '-', then show the plot interactively. See also plotshow.

plotshow [bool] If plotfile is not given or is '-', then show the plot interactively.

fig [matplotlib.figure.Figure] plot of the reblocking data.

pyblock.blocking implements the reblocking algorithm\(^1\) and an algorithm\(^2,3\) for suggesting the most appropriate block size (and thus estimate of the standard error in the data set) for data contained within numpy arrays. pyblock.pd_utils provides a nice wrapper around this using pandas, and it is highly recommended to use this if possible.

pyblock.error contains functions for simple error propagation and formatting of output of a value and it’s associated error.

## 2.5 References


The estimate of the standard error of a set of data assumes that the data points are completely independent. If this is not true, then naively calculating the standard error of the entire data set can give a substantial underestimate of the true error. This arises in, for example, Monte Carlo simulations where the state at one step depends upon the state at the previous step. Data calculated from the stochastic state hence has serial correlations.

A simple way to remove these correlations is to repeatedly average neighbouring pairs of data points and calculate the standard error on the new data set. As no data is discarded in this process (assuming the data set contains $2^n$ values), the error estimate should remain approximately constant if the data is truly independent.

pyblock is a python module for performing this reblocking analysis.

Normally correlated data comes from an experiment or simulation but we’ll use randomly generated data which is serially correlated in order to show how pyblock works.

```python
import numpy

def corr_data(N, L):
    '''Generate random correlated data containing $2^N$ data points. Random data is convolved over a $2^L/10$ length to give the correlated signal.'''
    return numpy.convolve(numpy.random.randn(2**N), numpy.ones(2**L)/10, 'same')

rand_data = corr_data(16, 6)

plot(rand_data);
```
If we zoom in, we can clearly see that neighbouring data points do not immediately appear to be independent:

```python
plot(rand_data[:1000]);
plot(rand_data[40000:41000]);
```

`pyblock` can perform a reblocking analysis to get a better estimate of the standard error of the data set:

```python
import pyblock
reblock_data = pyblock.blocking.reblock(rand_data)
for reblock_iter in reblock_data:
    print(reblock_iter)
```
The standard error of the original data set is clearly around 8 times too small. Note that the standard error of the last few reblock iterations fluctuates substantially—this is simply because of the small number of data points at those iterations.

In addition to the mean and standard error at each iteration, the covariance and an estimate of the error in the standard error are also calculated. Each tuple also contains the number of data points used at the given reblock iteration.

`pyblock.blocking` can also suggest the reblock iteration at which the standard error has converged (i.e. the iteration at
which the serial correlation has been removed and every data point is truly independent).

```python
opt = pyblock.blocking.find_optimal_block(len(rand_data), reblock_data)
print(opt)
print(reblock_data[opt[0]])
```

```output
[10]
BlockTuple(block=10, ndata=64, mean=array(0.02972941238888185), cov=array(0.
˓→-0455635621966979), std_err=array(0.026682028770755133), std_err_err=array(0.
˓→-002377024048671685))
```

Whilst the above uses just a single data set, pyblock is designed to work on multiple data sets at once (e.g. multiple outputs from the same simulation). In that case, different optimal reblock iterations might be found for each data set. The only assumption is that the original data sets are of the same length.

It is also possible to reblock weighted data sets. If the `pyblock.blocking` routine is supplied with an array of weights in addition to the data, the weighted variance and standard error of each data set are calculated.

### 3.1 pandas integration

The core pyblock functionality is built upon numpy. However, it is more convenient to use the pandas-based wrapper around pyblock.blocking, not least because it makes working with multiple data sets more pleasant.

```python
import pandas as pd
rand_data = pd.Series(rand_data)

rand_data.head()
```

```
0  -0.294901
1  -0.360847
2  -0.386010
3  -0.496183
4  -0.625507
dtype: float64
```

```python
(data_length, reblock_data, covariance) = pyblock.pd_utils.reblock(rand_data)
```

```
# number of data points at each reblock iteration
data_length
reblock
0    65536
1    32768
2    16384
3    8192
4    4096
5    2048
6    1024
7     512
8     256
9     128
10     64
11     32
12      16
```
# mean, standard error and estimate of the error in the standard error at each reblock iteration
# Note the suggested reblock iteration is already indicated.
# pyblock names the data series 'data' if no name is provided in the pandas.Series/pandas.DataFrame.
reblock_data

covariance

We can also plot the convergence of the standard error estimate and obtain a summary of the suggested data to quote:

```python
pyblock.plot.plot_reblocking(reblock_data);
```

The standard error clearly converges to ~0.022. The suggested reblock iteration (which uses a slightly conservative formula) is indicated by the arrow on the plot.

```python
pyblock.pd_utils.reblock_summary(reblock_data)
```

`pyblock.error` also contains simple error propagation functions for combining multiple noisy data sets and can handle multiple data sets at once (contained either within a `numpy` array using `pyblock.blocking` or within a `pandas.DataFrame`).

### 3.1. pandas integration
CHAPTER 4

References
CHAPTER 5

Indices and tables

- genindex
- modindex
- search
Bibliography


p
pyblock, 5
pyblock.blocking, 5
pyblock.error, 6
pyblock.pd_utils, 8
pyblock.plot, 8
Index

A
addition() (in module pyblock.error), 7

F
find_optimal_block() (in module pyblock.blocking), 6

O
optimal_block() (in module pyblock.pd_utils), 8

P
plot_reblocking() (in module pyblock.plot), 8
pretty_fmt_err() (in module pyblock.error), 7
product() (in module pyblock.error), 7
pyblock (module), 5
pyblock.blocking (module), 5
pyblock.error (module), 6
pyblock.pd_utils (module), 8
pyblock.plot (module), 8

R
ratio() (in module pyblock.error), 6
reblock() (in module pyblock.blocking), 5
reblock() (in module pyblock.pd_utils), 8
reblock_summary() (in module pyblock.pd_utils), 8

S
subtraction() (in module pyblock.error), 7