SciKit GStat Documentation

Release 0.2.2

Mirko Mälicke

Oct 24, 2018

CONTENTS:

CHAPTER

ONE

WELCOME TO SCIKIT GSTAT'S DOCUMENTATION!

SciKit-Gstat is a scipy-styled analysis module for geostatistics. It includes two base classes `Variogram` and `DirectionalVariogram`. Both have a very similar interface and can compute experimental variograms and model variograms. The module makes use of a rich selection of semi-variance estimators and variogram model functions, while being extensible at the same time.

Note: Scikit-gstat was rewritten in major parts. Most of the changes are internal, but the attributes and behaviour of the *Variogram* has also changed substantially. A detailed description of the new versions usage will follow. The last version of the old Variogram class, 0.1.8, is kept in the *version-0.1.8* branch on GitHub, but not developed any further. It is not compatible to the current version.

CHAPTER

TWO

HOW TO CITE

In case you use SciKit-GStat in other software or scientific publications, please reference this module. It is published and has a DOI. It can be cited as:

> Mälicke, Mirko, & Schneider, Helge David. (2018). mmaelicke/scikit-gstat: Geostatistical variogram toolbox (Version v0.2.2). Zenodo. http://doi.org/10.5281/zenodo.1345584

2.1 Installation

The package can be installed directly from the Python Package Index or GitHub. The version on GitHub might be more recent, as only stable versions are uploaded to the Python Package Index.

2.1.1 PyPI

The version from PyPI can directly be installed using pip

```
pip install scikit-gstat
```

2.1.2 GitHub

The most recent version from GitHub can be installed like:

```
git clone https://github.com/mmaelicke/scikit-gstat.git
cd scikit-gstat
pip install -r requirements.txt
python setup.py install
```

2.1.3 Note

Depending on you OS, you might run into problems installing all requirements in a clean Python environment. These problems are usually caused by the scipy and numba package, which might need to be compiled. From our experience, no problems should occur, when an environment manager like anaconda is used. Then, the requirements can be installed like:

```
conda install numpy, scipy, numba
```

2.2 Getting Started

2.2.1 Load the class and data

The main class of scikit-gstat is the Variogram. It can directly be imported from the module, called skgstat. The main class can easily be demonstrated on random data.

```
In [1]: from skgstat import Variogram

In [2]: import numpy as np

In [3]: import matplotlib.pyplot as plt

In [4]: plt.style.use('ggplot')

In [5]: np.random.seed(42)

In [6]: coordinates = np.random.gamma(20, 5, (50,2))

In [7]: np.random.seed(42)

In [8]: values = np.random.normal(20, 5, 50)
```

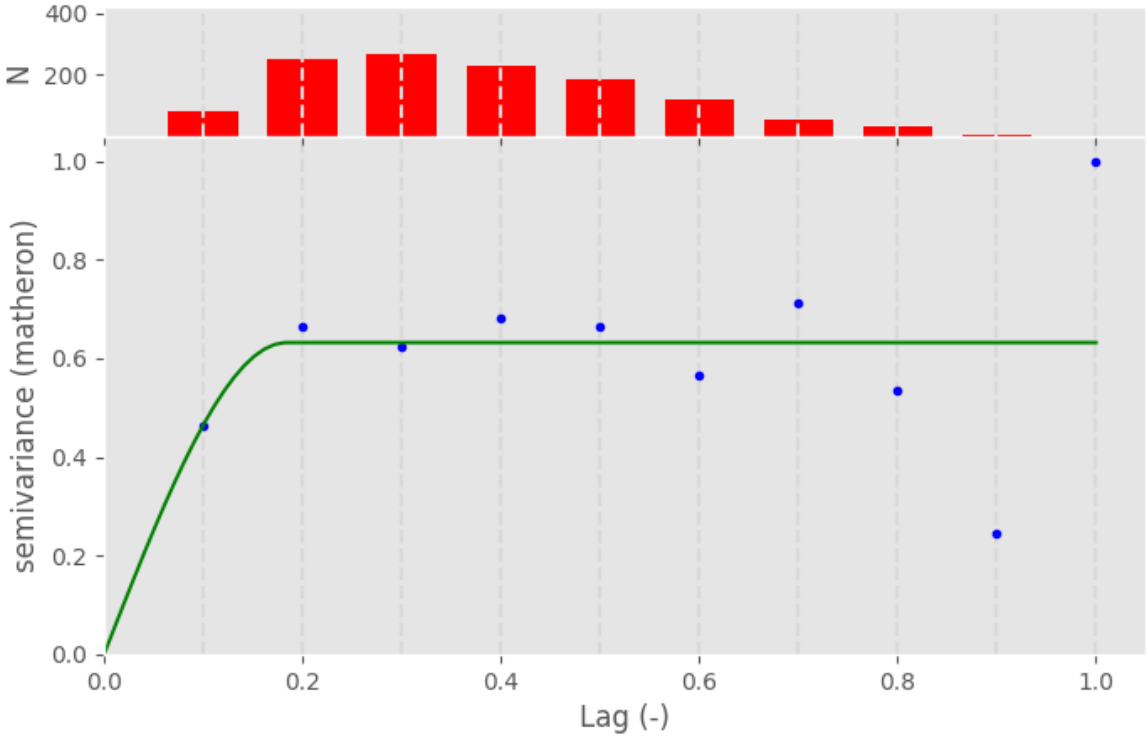
The Variogram needs at least an array of coordinates and an array of values on instantiation.

```
In [9]: V = Variogram(coordinates=coordinates, values=values)

In [10]: print(V)
spherical Variogram
--------------------
Estimator:          matheron

Effective Range:    1914.57

Sill:               749.20

Nugget:             0.00
```

2.2.2 Plot

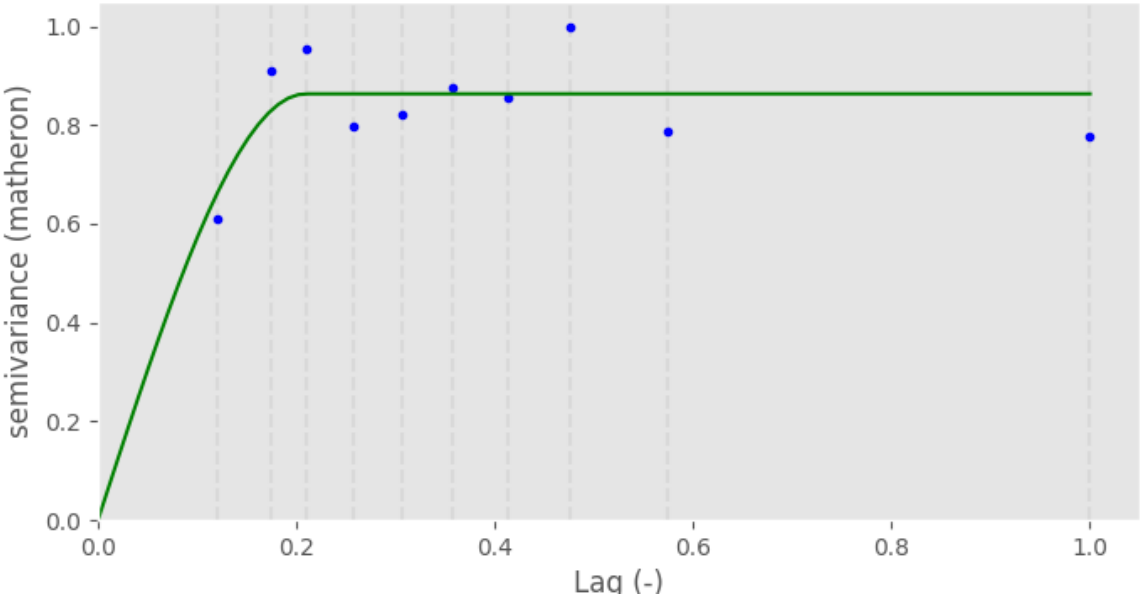
The Variogram class has its own plotting method.

```
In [11]: V.plot()
Out[11]: <Figure size 800x500 with 2 Axes>
```



With version 0.2, the histogram plot can also be disabled. This is most useful, when the binning method for the lag classes is changed from *'even'* step classes to *'uniform'* distribution in the lag classes.

```
In [12]: V.set_bin_func('uniform')

In [13]: V.plot(hist=False)
Out[13]: <Figure size 800x400 with 1 Axes>
```



2.3 Technical Notes

This chapter collects a number of technical advises for using scikit-gstat. These examples and information shall shall either explain the implementation or guide you to a correct usage of the packages. Not all features implemented make sense in every situation.

2.3.1 Fitting a theoretical model

General

The fit function of Variogram relies as of this writing on the `scipy.optimize.curve_fit()` function. That function can be used by ust passing a function and a set of x and y values and hoping for the best. However, this will not always yield the best parameters. Especially not for fitting a theoretical variogram function. There are a few assumptions and simplifications, that we can state in order to utilize the function in a more meaningful way.

Default fit

The example below shows the performance of the fully unconstrained fit, performed by the Levenberg-Marquardt algorithm. In scikit-gstat, this can be used by setting the *fit_method* parameter to *lm*. However, this is not recommended.

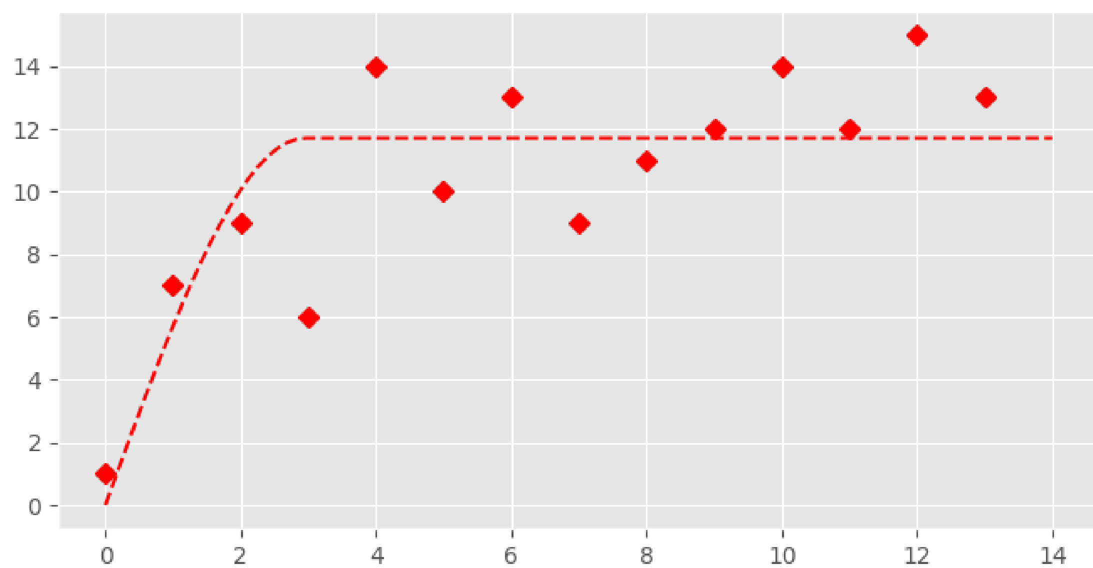
```
In [1]: from scipy.optimize import curve_fit

In [2]: import matplotlib.pyplot as plt

In [3]: plt.style.use('ggplot')

In [4]: import numpy as np

In [5]: from skgstat.models import spherical
```

The fit of a spherical model will be illustrated with some made-up data representing an experimental variogram:

```
In [6]: y = [1,7,9,6,14,10,13,9,11,12,14,12,15,13]

In [7]: x = list(range(len(y)))

In [8]: xi = np.linspace(0, len(y), 100)
```

As the `spherical` function is compiled using numba, we wrap the function in order to let `curve_fit` correctly infer the parameters. Then, fitting is a straightforward task.

```
In [9]: def f(h, a, b):
   ...:     return spherical(h, a, b)
   ...:

In [10]: cof_u, cov = curve_fit(f, x, y)

In [11]: yi = list(map(lambda x: spherical(x, *cof_u), xi))

In [12]: plt.plot(x, y, 'rD')
Out[12]: [<matplotlib.lines.Line2D at 0x2b20d1621a20>]

In [13]: plt.plot(xi, yi, '--r')
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[13]: [<matplotlib.lines.
↪Line2D at 0x2b20d1621710>]
```



In fact this looks quite good. But Levenberg-Marquardt is an unconstrained fitting algorithm and it could likely fail on finding a parameter set. The `fit` method can therefore also run a box constrained fitting algorithm. It is the Trust Region Reflective algorithm, that will find parameters within a given range (box). It is set by the *fit_method='tfr'* parameter and also the default setting.

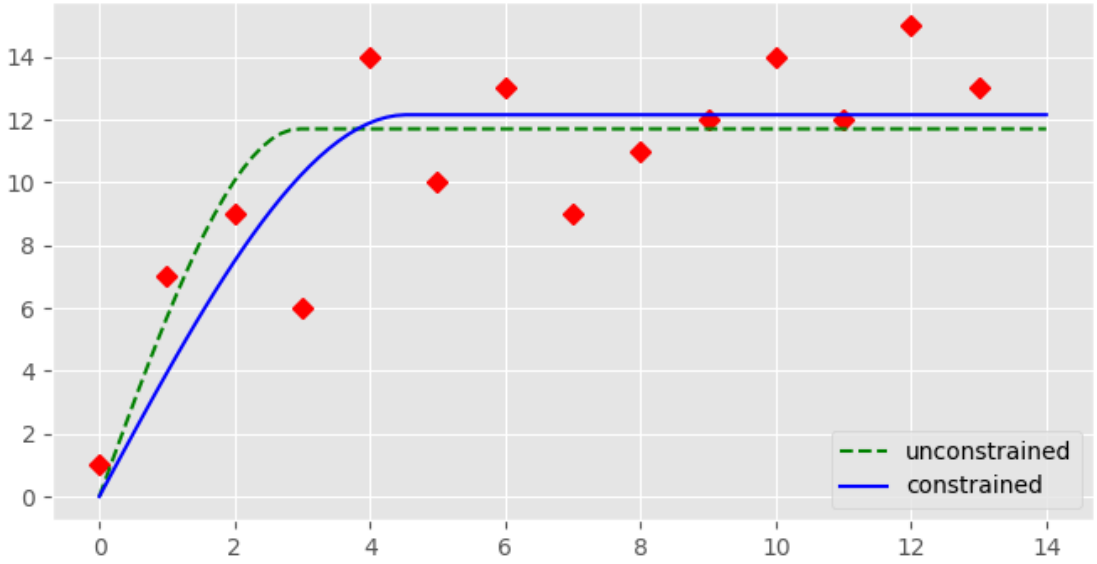
Constrained fit

The constrained fitting case was chosen to be the default method in skgstat as the region can easily be specified. Furthermore it is possible to make a good guess on initial values. As we fit actual variogram parameters, namely the effective range, sill, nugget and in case of a stable or Matérn model an additional shape parameter, we know that these parameters cannot be zero. The semi-variance is defined to be always positive. Thus the lower bound of the region will be zero in any case. The upper limit can easily be inferred from the experimental variogram. There are some simple rules, that all theoretical functions follow:

- the sill, nugget and their sum cannot be larger than the maximum empirical semi-variance
- the range cannot be larger than maxlag, or if maxlag is None the maximum value in the distances

The *Variogram* class will set the bounds to exactly these values as default behaviour. As an initial guess, it will use the mean value of semi-variances for the sill, the mean separating distance as range and 0 for the nugget. In the presented empirical variogram, difference between Levenberg-Marquardt and Trust Region Reflective is illustrated in the example below.

```
# default plot
In [14]: plt.plot(x, y, 'rD')
Out[14]: [<matplotlib.lines.Line2D at 0x2b20d1650080>]

In [15]: plt.plot(xi, yi, '--g', label='unconstrained')
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[15]: [<matplotlib.lines.
↪Line2D at 0x2b20d1650ba8>]

In [16]: cof, cov = curve_fit(f, x, y, p0=[3., 14.], bounds=(0, (np.max(x), np.
↪max(y))))

In [17]: yi = list(map(lambda x: spherical(x, *cof), xi))

In [18]: plt.plot(xi, yi, '-b', label='constrained')
Out[18]: [<matplotlib.lines.Line2D at 0x2b20d1662358>]

In [19]: plt.legend(loc='lower right')
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[19]: <matplotlib.legend.
↪Legend at 0x2b20d1662ef0>
```



The constrained fit, represented by the solid blue line is significantly different from the unconstrained fit (dashed, green line). The fit is overall better as a quick RMSE calculation shows:

```
In [20]: rmse_u = np.sqrt(np.sum([(spherical(_, *cof_u) - _)**2 for _ in x]))

In [21]: rmse_c = np.sqrt(np.sum([(spherical(_, *cof) - _)**2 for _ in x]))

In [22]: print('RMSE unconstrained: %.2f' % rmse_u)
RMSE unconstrained: 18.65

In [23]: print('RMSE constrained:   %.2f' % rmse_c)
\\\\\\\\\\\\\\\\\\\\\\\\\\RMSE constrained:   17.42
```

The last note about fitting a theoretical function, is that both methods assume all lag classes to be equally important for the fit. In the specific case of a variogram this is not true.

Distance weighted fit

While the standard Levenberg-Marquardt and Trust Region Reflective algorithms are both based on the idea of least squares, they assume all observations to be equally important. In the specific case of a theoretical variogram function, this is not the case. The variogram describes a dependency of covariance in value on the separation distances of the observations. This model already implies that the dependency is stronger on small distances. Considering a kriging interpolation as the main application of the variogram model, points on close distances will get higher weights for the interpolated value of an unobserved location. The weight on large distances will be neglected anyway. Hence, a good fit on small separating distances is way more important. The `curve_fit` function does not have an option for weighting the squares of specific observations. At least it does not call it 'weights'. In terms of scipy, you can define a 'sigma', which is the uncertainty of the respective point. The uncertainty σ influences the least squares calculation as described by the equation:

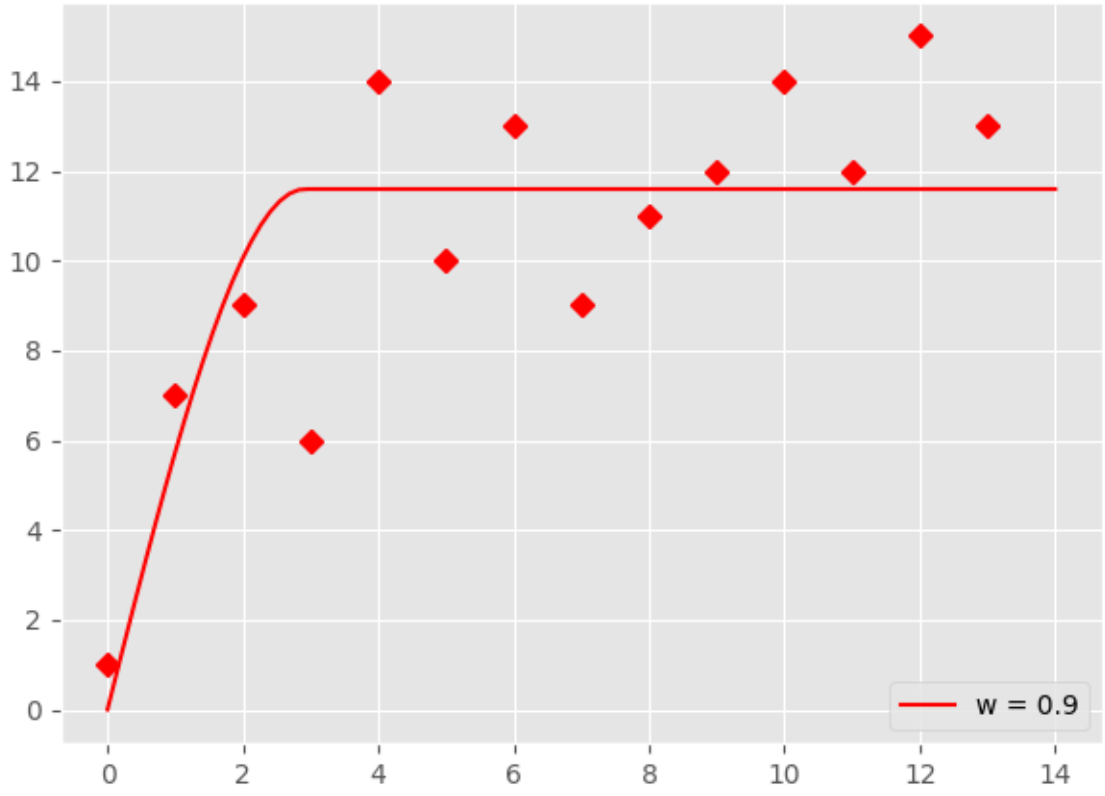
$$\chi_{sq} = \sum \left(\frac{r}{\sigma}\right)^2$$

That means, the larger σ is, the *less* weight it will receive. That also means, we can almost ignore points, by assigning a ridiculous high σ to them. The following example should illustrate the effect. This time, the first 7 points will be

weighted by a weight $\sigma = [0.1, 0.2, \dots 0.9]$ and the remaining points will receive a $\sigma = 1$. In the case of $\sigma = 0.1$, this would change the least squares cost function to:

$$\chi_{sq;x_{1:7}} = \sum (10r)^2$$

```
In [24]: cm = plt.get_cmap('autumn_r')

In [25]: sigma = np.ones(len(x))

In [26]: fig, ax = plt.subplots(1, 1, figsize=(7, 5))

In [27]: ax.plot(x, y, 'rD')
Out[27]: [<matplotlib.lines.Line2D at 0x2b20d16094a8>]

In [28]: for w in np.arange(0.1, 1., 0.1):
   ....:     s = sigma.copy()
   ....:     s[:6] *= w
   ....:     cof, cov = curve_fit(f, x, y, sigma=s)
   ....:     yi = list(map(lambda x: spherical(x, *cof), xi))
   ....:     ax.plot(xi, yi, linestyle='-', color=cm(w + 0.1), label='w = %.1f' % w)
   ....:
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[32]: [<matplotlib.lines.
↪Line2D at 0x2b20d18152b0>]

In [33]: ax.legend(loc='lower right')
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪<matplotlib.legend.Legend at 0x2b20d1815630>
```



In the figure above, you can see how the last points get more and more ignored by the fitting. A smaller w value means more weight on the first 7 points. The more yellow lines have a smaller sill and range.

The *Variogram* class accepts lists like sigma from the code example above as *Variogram.fit_sigma* property. This way, the example from above could be implemented. However, *Variogram.fit_sigma* can also apply a function of distance to the lag classes to derive the σ values. There are several predefined functions. These are:

- sigma='linear': The residuals get weighted by the lag distance normalized to the maximum lag distance, denoted as w_n
- sigma='exp': The residuals get weighted by the function: $w = e^{1/w_n}$
- sigma='sqrt': The residuals get weighted by the function: $w = \sqrt{(w_n)}$
- sigma='sq': The residuals get weighted by the function: $w = w_n^2$

The example below illustrates their effect on the sample experimental variograms used so far.

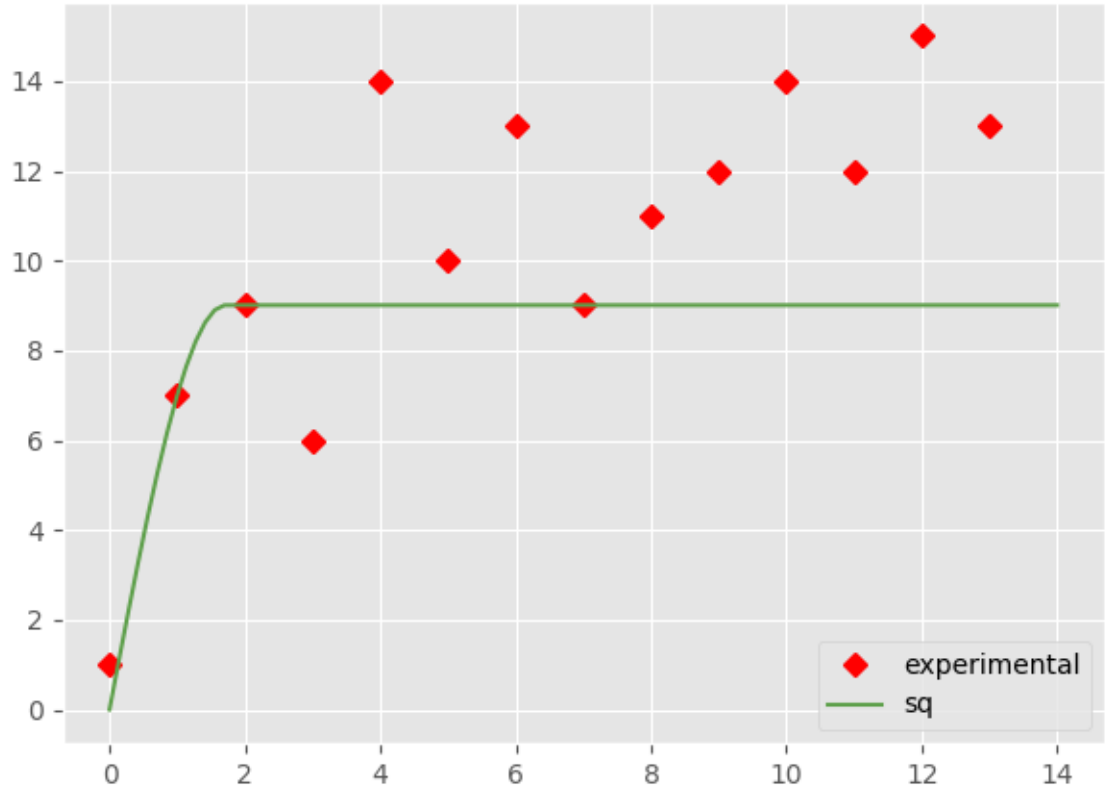
```
In [34]: cm = plt.get_cmap('gist_earth')

# increase the distance by one, to aviod zeros
In [35]: X = np.asarray([(_ + 1) for _ in x])

In [36]: s1 = X / np.max(X)

In [37]: s2 = np.exp(1. / X)
```

(continues on next page)

(continued from previous page)

```
In [38]: s3 = np.sqrt(s1)

In [39]: s4 = np.power(s1, 2)

In [40]: s = (s1, s2, s3, s4)

In [41]: labels = ('linear', 'exp', 'sqrt', 'sq')
```

```
In [42]: plt.plot(x, y, 'rD', label='experimental')
Out[42]: [<matplotlib.lines.Line2D at 0x2b20d3f740b8>]

In [43]: for i in range(4):
   ....:     cof, cov = curve_fit(f, x, y, sigma=s[i], p0=(6.,14.), bounds=(0,(14,
↪14)))
   ....:     yi = list(map(lambda x: spherical(x, *cof), xi))
   ....:     plt.plot(xi, yi, linestyle='-', color=cm((i/6)), label=labels[i])
   ....:
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\Out[45]: [<matplotlib.lines.
↪Line2D at 0x2b20d3f74710>]

In [46]: plt.legend(loc='lower right')
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪<matplotlib.legend.Legend at 0x2b20d3f744e0>
```



That's it.

2.4 Code Reference

2.4.1 Variogram Class

class skgstat.**Variogram**(*coordinates=None*, *values=None*, *estimator='matheron'*, *model='spherical'*, *dist_func='euclidean'*, *bin_func='even'*, *normalize=True*, *fit_method='trf'*, *fit_sigma=None*, *use_nugget=False*, *maxlag=None*, *n_lags=10*, *verbose=False*, *harmonize=False*)

Variogram Class

Calculates a variogram of the separating distances in the given coordinates and relates them to one of the semi-variance measures of the given dependent values.

__init__(*coordinates=None*, *values=None*, *estimator='matheron'*, *model='spherical'*, *dist_func='euclidean'*, *bin_func='even'*, *normalize=True*, *fit_method='trf'*, *fit_sigma=None*, *use_nugget=False*, *maxlag=None*, *n_lags=10*, *verbose=False*, *harmonize=False*)

Variogram Class

Note: The directional variogram estimation is not re-implemented yet. Therefore the parameters is-directional, azimuth and tolerance will be ignored at the moment and can be subject to changes.

Parameters

- **coordinates** (*numpy.ndarray*) – Array of shape (m, n). Will be used as m observation points of n-dimensions. This variogram can be calculated on 1 - n dimensional coordinates. In case a 1-dimensional array is passed, a second array of same length containing only zeros will be stacked to the passed one.
- **values** (*numpy.ndarray*) – Array of values observed at the given coordinates. The length of the values array has to match the m dimension of the coordinates array. Will be used to calculate the dependent variable of the variogram.
- **estimator** (*str, callable*) – String identifying the semi-variance estimator to be used. Defaults to the Matheron estimator. Possible values are:
  - matheron [Matheron, default]
  - cressie [Cressie-Hawkins]
  - dowd [Dowd-Estimator]
  - genton [Genton]
  - minmax [MinMax Scaler]
  - entropy [Shannon Entropy]

  If a callable is passed, it has to accept an array of absoulte differences, aligned to the 1D distance matrix (flattened upper triangle) and return a scalar, that converges towards small values for similarity (high covariance).
- **model** (*str*) – String identifying the theoretical variogram function to be used to describe the experimental variogram. Can be one of:
  - spherical [Spherical, default]
  - exponential [Exponential]
  - gaussian [Gaussian]
  - cubic [Cubic]
  - stable [Stable model]
  - matern [Matérn model]
  - nugget [nugget effect variogram]
- **dist_func** (*str*) – String identifying the distance function. Defaults to 'euclidean'. Can be any metric accepted by scipy.spatial.distance.pdist. Additional parameters are not (yet) passed through to pdist. These are accepted by pdist for some of the metrics. In these cases the default values are used.
- **bin_func** (*str*) – String identifying the binning function used to find lag class edges. At the moment there are two possible values: 'even' (default) or 'uniform'. Even will find n_lags bins of same width in the interval [0,maxlag[. 'uniform' will identfy n_lags bins on the same interval, but with varying edges so that all bins count the same amount of observations.
- **normalize** (*bool*) – Defaults to False. If True, the independent and dependent variable will be normalized to the range [0,1].
- **fit_method** (*str*) – String identifying the method to be used for fitting the theoretical variogram function to the experimental. More info is given in the Variogram.fit docs. Can be one of:

- 'lm': Levenberg-Marquardt algorithm for unconstrained problems. This is the faster algorithm, yet is the fitting of a variogram not unconstrianed.
- 'trf': Trust Region Reflective function for non-linear constrained problems. The class will set the boundaries itself. This is the default function.

- **fit_sigma** (*numpy.ndarray, str*) – Defaults to None. The sigma is used as measure of uncertainty during variogram fit. If fit_sigma is an array, it has to hold n_lags elements, giving the uncertainty for all lags classes. If fit_sigma is None (default), it will give no weight to any lag. Higher values indicate higher uncertainty and will lower the influcence of the corresponding lag class for the fit. If fit_sigma is a string, a pre-defined function of separating distance will be used to fill the array. Can be one of:
  - 'linear': Linear loss with distance. Small bins will have higher impact.
  - 'exp': The weights decrease by a e-function of distance
  - 'sqrt': The weights decrease by the squareroot of distance
  - 'sq': The weights decrease by the squared distance.

  More info is given in the Variogram.fit_sigma documentation.
- **use_nugget** (*bool*) – Defaults to False. If True, a nugget effet will be added to all Variogram.models as a third (or fourth) fitting parameter. A nugget is essentially the y-axis interception of the theoretical variogram function.
- **maxlag** (*float, str*) – Can specify the maximum lag distance directly by giving a value larger than 1. The binning function will not find any lag class with an edge larger than maxlag. If 0 < maxlag < 1, then maxlag is relative and maxlag * max(Variogram.distance) will be used. In case maxlag is a string it has to be one of 'median', 'mean'. Then the median or mean of all Variogram.distance will be used. Note maxlag=0.5 will use half the maximum separating distance, this is not the same as 'median', which is the median of all separating distances
- **n_lags** (*int*) – Specify the number of lag classes to be defined by the binning function.
- **verbose** (*bool*) – Set the Verbosity of the class. Not Implemented yet.
- **harmonize** (*bool*) – this kind of works so far, but will be rewritten (and documented)

NS

Nash Sutcliffe efficiency of the fitted Variogram

Returns

bin_func

Binning function

Returns an instance of the function used for binning the separating distances into the given amount of bins. Both functions use the same signature of func(distances, n, maxlag).

The setter of this property utilizes the Variogram.set_bin_func to set a new function.

Returns **binning_function**

Return type function

See also:

Variogram.set_bin_func

clone()

Deep copy of self

Return a deep copy of self.

Returns

Return type *Variogram*

compiled_model
Compiled theoretical variogram model

Compile the model using the actual fitting parameters to return a function implementing them.

Returns

Return type callable

data(*n=100*, *force=False*)
Theoretical variogram function

Calculate the experimental variogram and apply the binning. On success, the variogram model will be fitted and applied to n lag values. Returns the lags and the calculated semi-variance values. If force is True, a clean preprocessing and fitting run will be executed.

Parameters

- **n** (*integer*) – length of the lags array to be used for fitting. Defaults to 100, which will be fine for most plots
- **force** (*boolean*) – If True, the preprocessing and fitting will be executed as a clean run. This will force all intermediate results to be recalculated. Defaults to False

Returns **variogram** – first element is the created lags array second element are the calculated semi-variance values

Return type tuple

describe()
Variogram parameters

Return a dictionary of the variogram parameters.

Returns

Return type dict

distance_difference_plot(*ax=None*, *plot_bins=True*, *show=True*)
Raw distance plot

Plots all absoulte value differences of all point pair combinations over their separating distance, without sorting them into a lag.

Parameters

- **ax** (*None, AxesSubplot*) – If None, a new matplotlib.Figure will be created. In case a Figure was already created, pass the Subplot to use as ax argument.
- **plot_bins** (*bool*) – If True (default) the bin edges will be included into the plot.
- **show** (*bool*) – If True (default), the show method of the Figure will be called before returning the Figure. Can be set to False, to avoid doubled figure rendering in Jupyter notebooks.

Returns

Return type matplotlib.pyplot.Figure

experimental
Experimental Variogram

Array of experimental (empirical) semivariance values. The array length will be aligned to Variogram.bins. The current Variogram.estimator has been used to calculate the values. Depending on the setting of Variogram.harmonize (True | False), either Variogram._experimental or Variogram.isotonic will be returned.

Returns **vario** – Array of the experimental semi-variance values aligned to Variogram.bins.

Return type numpy.ndarray

See also:

`Variogram._experimental`, *`Variogram.isotonic`*

fit (*force=False*, *method=None*, *sigma=None*, ***kwargs*)
Fit the variogram

The fit function will fit the theoretical variogram function to the experimental. The preprocessed distance matrix, pairwise differences and binning will not be recalculated, if already done. This could be forced by setting the force parameter to true.

In case you call fit function directly, with method or sigma, the parameters set on Variogram object instantiation will get overwritten. All other keyword arguments will be passed to scipy.optimize.curve_fit function.

Parameters

- **force** (*bool*) – If set to True, a clean preprocessing of the distance matrix, pairwise differences and the binning will be forced. Default is False.
- **method** (*string*) – A string identifying one of the implemented fitting procedures. Can be one of ['lm', 'trf']:
  - lm: Levenberg-Marquardt algorithms implemented in scipy.optimize.leastsq function.
  - trf: Trust Region Reflective algorithm implemented in scipy.optimize.least_squares(method='trf')
- **sigma** (*string, array*) – Uncertainty array for the bins. Has to have the same dimension as self.bins. Refer to Variogram.fit_sigma for more information.

Returns

Return type void

See also:

`scipy.optimize()`, `scipy.optimize.curve_fit()`, `scipy.optimize.leastsq()`, `scipy.optimize.least_squares()`

fit_sigma
Fitting Uncertainty

Set or calculate an array of observation uncertainties aligned to the Variogram.bins. These will be used to weight the observations in the cost function, which divides the residuals by their uncertainty.

When setting fit_sigma, the array of uncertainties itself can be given, or one of the strings: ['linear', 'exp', 'sqrt', 'sq']. The parameters described below refer to the setter of this property.

Parameters **sigma** (*string, array*) – Sigma can either be an array of discrete uncertainty values, which have to align to the Variogram.bins, or of type string. Then, the weights for fitting are calculated as a function of (lag) distance.

- **sigma='linear'**: The residuals get weighted by the lag distance normalized to the maximum lag distance, denoted as w_n
- **sigma='exp'**: The residuals get weighted by the function: $w = e^{1/w_n}$
- **sigma='sqrt'**: The residuals get weighted by the function: $w = \sqrt{(w_n)}$
- **sigma='sq'**: The residuals get weighted by the function: $w = w_n^2$

Returns

Return type void

Notes

The cost function is defined as:

$$chisq = \sum \frac{r}{\sigma}^2$$

where r are the residuals between the experimental variogram and the modeled values for the same lag. Following this function, small values will increase the influence of that residual, while a very large sigma will cause the observation to be ignored.

See also:

`scipy.optimize.curve_fit`

isotonic

Return the isotonic harmonized experimental variogram. This means, the experimental variogram is monotonic after harmonization.

The harmonization is done using PAVA algorithm:

Barlow, R., D. Bartholomew, et al. (1972): Statistical Interference Under Order Restrictions. John Wiley and Sons, New York.

Hiterding, A. (2003): Entwicklung hybrider Interpolationsverfahren für den automatisierten Betrieb am Beispiel meteorologischer Größen. Dissertation, Institut für Geoinformatik, Westphälische Wilhelms-Universität Münster, IfGIprints, Münster. ISBN: 3-936616-12-4

TODO: solve the import

Returns np.ndarray, monotonized experimental variogram

lag_classes()

Iterate over the lag classes

Generates an iterator over all lag classes. Can be zipped with Variogram.bins to identify the lag.

Returns

Return type iterable

lag_groups()

Lag class groups

Retuns a mask array with as many elements as self._diff has, identifying the lag class group for each pairwise difference. Can be used to extract all pairwise values within the same lag bin.

Returns

Return type numpy.ndarray

See also:

Variogram.lag_classes()

location_trend(*axes=None*)
Location Trend plot

Plots the values over each dimension of the coordinates in a scatter plot. This will visually show correlations between the values and any of the coordinate dimension. If there is a value dependence on the location, this would violate the intrinsic hypothesis. This is a weaker form of stationarity of second order.

Parameters **axes** (*list*) – Can be None (default) or a list of matplotlib.AxesSubplots. If a list is passed, the location trend plots will be plotted on the given instances. Note that then length of the list has to match the dimeonsionality of the coordinates array. In case 3D coordinates are used, three subplots have to be given.

Returns

Return type matplotlib.Figure

mean_residual
Mean Model residuals

Calculates the mean, absoulte deviations between the experimental variogram and theretical model values.

Returns

Return type float

model_deviations()
Model Deviations

Calculate the deviations between the experimental variogram and the recalculated values for the same bins using the fitted theoretical variogram function. Can be utilized to calculate a quality measure for the variogram fit.

Returns **deviations** – first element is the experimental variogram second element are the corresponding values of the theoretical model.

Return type tuple

nrmse
NRMSE

Calculate the normalized root mean squared error between the experimental variogram and the theoretical model values at corresponding lags. Can be used as a fitting quality measure

Returns

Return type float

See also:

Variogram.residuals, *Variogram.rmse*

Notes

The NRMSE is implemented as:

$$NRMSE = \frac{RMSE}{mean(y)}$$

where RMSE is Variogram.rmse and y is Variogram.experimental

nrmse_r
NRMSE

Alternative normalized root mean squared error between the experimental variogram and the theoretical model values at corresponding lags. Can be used as a fitting quality measure.

Returns

Return type float

See also:

Variogram.rmse, *Variogram.nrmse*

Notes

Unlike Variogram.nrmse, nrmse_r is not normalized to the mean of y, but the differece of the maximum y to its mean:

$$NRMSE_r = \frac{RMSE}{max(y) - mean(y)}$$

parameters
Extract just the variogram parameters range, sill and nugget from the self.describe return

Returns

plot (*axes=None*, *grid=True*, *show=True*, *hist=True*)
Variogram Plot

Plot the experimental variogram, the fitted theoretical function and an histogram for the lag classes. The axes attribute can be used to pass a list of AxesSubplots or a single instance to the plot function. Then these Subplots will be used. If only a single instance is passed, the hist attribute will be ignored as only the variogram will be plotted anyway.

Parameters

- **axes** (*list, tuple, array, AxesSubplot or None*) – If None, the plot function will create a new matplotlib figure. Otherwise a single instance or a list of AxesSubplots can be passed to be used. If a single instance is passed, the hist attribute will be ignored.
- **grid** (*bool*) – Defaults to True. If True a custom grid will be drawn through the lag class centers
- **show** (*bool*) – Defaults to True. If True, the show method of the passed or created matplotlib Figure will be called before returning the Figure. This should be set to False, when used in a Notebook, as a returned Figure object will be plotted anyway.
- **hist** (*bool*) – Defaults to True. If False, the creation of a histogram for the lag classes will be suppressed.

Returns

Return type matplotlib.Figure

preprocessing (*force=False*)
Preprocessing function

Prepares all input data for the fit and transform functions. Namely, the distances are calculated and the value differences. Then the binning is set up and bin edges are calculated. If any of the listed subsets are already prepared, their processing is skipped. This behaviour can be changed by the force parameter. This will cause a clean preprocessing.

Parameters **force** (*bool*) – If set to True, all preprocessing data sets will be deleted. Use it in case you need a clean preprocessing.

Returns

Return type void

r

Pearson correlation of the fitted Variogram

Returns

residuals

Model residuals

Calculate the model residuals defined as the differences between the experimental variogram and the theoretical model values at corresponding lag values

Returns

Return type numpy.ndarray

rmse

RMSE

Calculate the Root Mean squared error between the experimental variogram and the theoretical model values at corresponding lags. Can be used as a fitting quality measure.

Returns

Return type float

See also:

Variogram.residuals

Notes

The RMSE is implemented like:

$$RMSE = \sqrt{\frac{\sum_{i=0}^{i=N(x)}(x-y)^2}{N(x)}}$$

set_bin_func (*bin_func*)

Set binning function

Sets a new binning function to be used. The new binning method is set by a string identifying the new function to be used. Can be one of: [‘even’, ‘uniform’].

Parameters **bin_func** (*str*) – Can be one of:

- **’even’**: Use skgstat.binning.even_width_lags for using n_lags lags of equal width up to maxlag.
- **’uniform’**: Use skgstat.binning.uniform_count_lags for using n_lags lags up to maxlag in which the pairwise differences follow a uniform distribution.

Returns

Return type void

See also:

Variogram.bin_func(), skgstat.binning.uniform_count_lags(), skgstat.binning.even_width_lags()

set_dist_function(*func*)
Set distance function

Set the function used for distance calculation. func can either be a callable or a string. The ranked distance function is not implemented yet. strings will be forwarded to the scipy.spatial.distance.pdist function as the metric argument. If func is a callable, it has to return the upper triangle of the distance matrix as a flat array (Like the pdist function).

Parameters **func** (*string, callable*) –

Returns

Return type numpy.array

set_model(*model_name*)
Set model as the new theoretical variogram function.

set_values(*values*)
Set new values

Will set the passed array as new value array. This array has to be of same length as the first axis of the coordinates array. The Variogram class does only accept one dimensional arrays. On success all fitting parameters are deleted and the pairwise differences are recalculated.

Parameters **values** (*numpy.ndarray*) –

Returns

Return type void

See also:

Variogram.values()

to_DataFrame(*n=100*, *force=False*)
Variogram DataFrame

Returns the fitted theoretical variogram as a pandas.DataFrame instance. The n and force parameter control the calaculation, refer to the data funciton for more info.

Parameters

- **n** (*integer*) – length of the lags array to be used for fitting. Defaults to 100, which will be fine for most plots
- **force** (*boolean*) – If True, the preprocessing and fitting will be executed as a clean run. This will force all intermediate results to be recalculated. Defaults to False

Returns

Return type pandas.DataFrame

See also:

Variogram.data()

transform(*x*)
Transform

Transform a given set of lag values to the theoretical variogram function using the actual fitting and preprocessing parameters in this Variogram instance

Parameters **x** (*numpy.array*) – Array of lag values to be used as model input for the fitted theoretical variogram model

Returns

Return type numpy.array

value_matrix

Value matrix

Returns a matrix of pairwise differences in absolute values. The matrix will have the shape (m, m) with m = len(Variogram.values). Note that Variogram.values holds the values themselves, while the value_matrix consists of their pairwise differences.

Returns **values** – Matrix of pairwise absolute differences of the values.

Return type numpy.matrix

See also:

`Variogram._diff`

values

Values property

Array of observations, the variogram is build for. The setter of this property utilizes the Variogram.set_values function for setting new arrays.

Returns **values**

Return type numpy.ndarray

See also:

Variogram.set_values

2.4.2 DirectionalVariogram Class

class skgstat.**DirectionalVariogram**(*coordinates=None*, *values=None*, *estimator='matheron'*, *model='spherical'*, *dist_func='euclidean'*, *bin_func='even'*, *normalize=True*, *fit_method='trf'*, *fit_sigma=None*, *directional_model='triangle'*, *azimuth=0*, *tolerance=45.0*, *bandwidth='q33'*, *use_nugget=False*, *maxlag=None*, *n_lags=10*, *verbose=False*, *harmonize=False*)

DirectionalVariogram Class

Calculates a variogram of the separating distances in the given coordinates and relates them to one of the semivariance measures of the given dependent values.

The direcitonal version of a Variogram will only form paris of points that share a specified spatial relationship.

__init__(*coordinates=None*, *values=None*, *estimator='matheron'*, *model='spherical'*, *dist_func='euclidean'*, *bin_func='even'*, *normalize=True*, *fit_method='trf'*, *fit_sigma=None*, *directional_model='triangle'*, *azimuth=0*, *tolerance=45.0*, *bandwidth='q33'*, *use_nugget=False*, *maxlag=None*, *n_lags=10*, *verbose=False*, *harmonize=False*)

Variogram Class

Directional Variogram. The calculation is not performant and not tested yet.

Parameters

- **coordinates** (*numpy.ndarray*) – Array of shape (m, n). Will be used as m observation points of n-dimensions. This variogram can be calculated on 1 - n dimensional coordinates. In case a 1-dimensional array is passed, a second array of same length containing only zeros will be stacked to the passed one.
- **values** (*numpy.ndarray*) – Array of values observed at the given coordinates. The length of the values array has to match the m dimension of the coordinates array. Will be used to calculate the dependent variable of the variogram.
- **estimator** (*str, callable*) – String identifying the semi-variance estimator to be used. Defaults to the Matheron estimator. Possible values are:
  - matheron [Matheron, default]
  - cressie [Cressie-Hawkins]
  - dowd [Dowd-Estimator]
  - genton [Genton]
  - minmax [MinMax Scaler]
  - entropy [Shannon Entropy]

  If a callable is passed, it has to accept an array of absoulte differences, aligned to the 1D distance matrix (flattened upper triangle) and return a scalar, that converges towards small values for similarity (high covariance).
- **model** (*str*) – String identifying the theoretical variogram function to be used to describe the experimental variogram. Can be one of:
  - spherical [Spherical, default]
  - exponential [Exponential]
  - gaussian [Gaussian]
  - cubic [Cubic]
  - stable [Stable model]
  - matern [Matérn model]
  - nugget [nugget effect variogram]
- **dist_func** (*str*) – String identifying the distance function. Defaults to 'euclidean'. Can be any metric accepted by scipy.spatial.distance.pdist. Additional parameters are not (yet) passed through to pdist. These are accepted by pdist for some of the metrics. In these cases the default values are used.
- **bin_func** (*str*) – String identifying the binning function used to find lag class edges. At the moment there are two possible values: 'even' (default) or 'uniform'. Even will find n_lags bins of same width in the interval [0,maxlag[. 'uniform' will identfy n_lags bins on the same interval, but with varying edges so that all bins count the same amount of observations.
- **normalize** (*bool*) – Defaults to False. If True, the independent and dependent variable will be normalized to the range [0,1].
- **fit_method** (*str*) – String identifying the method to be used for fitting the theoretical variogram function to the experimental. More info is given in the Variogram.fit docs. Can be one of:
  - 'lm': Levenberg-Marquardt algorithm for unconstrained problems. This is the faster algorithm, yet is the fitting of a variogram not unconstrianed.

  - 'trf': Trust Region Reflective function for non-linear constrained problems. The class will set the boundaries itself. This is the default function.

- **fit_sigma** (*numpy.ndarray, str*) – Defaults to None. The sigma is used as measure of uncertainty during variogram fit. If fit_sigma is an array, it has to hold n_lags elements, giving the uncertainty for all lags classes. If fit_sigma is None (default), it will give no weight to any lag. Higher values indicate higher uncertainty and will lower the influcence of the corresponding lag class for the fit. If fit_sigma is a string, a pre-defined function of separating distance will be used to fill the array. Can be one of:
  - 'linear': Linear loss with distance. Small bins will have higher impact.
  - 'exp': The weights decrease by a e-function of distance
  - 'sqrt': The weights decrease by the squareroot of distance
  - 'sq': The weights decrease by the squared distance.

  More info is given in the Variogram.fit_sigma documentation.

- **directional_model** (*string, Polygon*) – The model used for selecting all points fulfilling the directional constraint of the Variogram. A predefined model can be selected by passing the model name as string. Optionally a callable accepting the current local coordinate system and returning a Polygon representing the search area itself can be passed. In this case, the tolerance and bandwidth has to be incorporated by hand into the model. The azimuth is handled by the class. The predefined options are:
  - 'compass': includes points in the direction of the azimuth at given tolerance. The bandwidth parameter will be ignored.
  - 'triangle': constructs a triangle with an angle of tolerance at the point of interest and union an rectangle parallel to azimuth, once the hypotenuse length reaches bandwidth.
  - 'circle': constructs a half circle touching the point of interest, dislocating the center at the distance of bandwidth in the direction of azimuth. The half circle is union with an rectangle parallel to azimuth.

  Visual representations, usage hints and implementation specifics are given in the documentation.

- **azimuth** (*float*) – The azimuth of the directional dependence for this Variogram, given as an angle in **degree**. The East of the coordinate plane is set to be at 0° and is counted clockwise to 180° and counter-clockwise to -180°. Only Points lying in the azimuth of a specific point will be used for forming point pairs.

- **tolerance** (*float*) – The tolerance is given as an angle in **degree**- Points being dislocated from the exact azimuth by half the tolerance will be accepted as well. It's half the tolerance as the point may be dislocated in the positive and negative direction from the azimuth.

- **bandwidth** (*float*) – Maximum tolerance acceptable in **coordinate units**, which is usually meter. Points at higher distances may be far dislocated from the azimuth in terms of coordinate distance, as the tolerance is defined as an angle. THe bandwidth defines a maximum width for the search window. It will be perpendicular to and bisected by the azimuth.

- **use_nugget** (*bool*) – Defaults to False. If True, a nugget effet will be added to all Variogram.models as a third (or fourth) fitting parameter. A nugget is essentially the y-axis interception of the theoretical variogram function.

- **maxlag** (*float, str*) – Can specify the maximum lag distance directly by giving a value larger than 1. The binning function will not find any lag class with an edge larger than

maxlag. If 0 < maxlag < 1, then maxlag is relative and maxlag * max(Variogram.distance) will be used. In case maxlag is a string it has to be one of 'median', 'mean'. Then the median or mean of all Variogram.distance will be used. Note maxlag=0.5 will use half the maximum separating distance, this is not the same as 'median', which is the median of all separating distances

- **n_lags** (*int*) – Specify the number of lag classes to be defined by the binning function.
- **verbose** (*bool*) – Set the Verbosity of the class. Not Implemented yet.
- **harmonize** (*bool*) – this kind of works so far, but will be rewritten (and documented)

local_reference_system(*poi*)
Calculate local coordinate system

The coordinates will be transformed into a local reference system that will simplify the directional dependence selection. The point of interest (poi) of the current iteration will be used as origin of the local reference system and the x-axis will be rotated onto the azimuth.

Parameters **poi** (*tuple*) – First two coordinate dimensions of the point of interest. will be used as the new origin

Returns **local_ref** – Array of dimension (m, 2) where m is the length of the coordinates array. Transformed coordinates in the same order as the original coordinates.

Return type numpy.array

search_area(*poi=0*, *ax=None*)
Plot Search Area

Parameters

- **poi** (*integer*) – Point of interest. Index of the coordinate that shall be used to visualize the search area.
- **ax** (*None, matplotlib.AxesSubplot*) – If not None, the Search Area will be plotted into the given Subplot object. If None, a new matplotlib Figure will be created and returned

Returns

Return type plot

set_directional_model(*model_name*)
Set new directional model

The model used for selecting all points fulfilling the directional constraint of the Variogram. A predefined model can be selected by passing the model name as string. Optionally a function can be passed that accepts the current local coordinate system and returns a Polygon representing the search area. In this case, the tolerance and bandwidth has to be incorporated by hand into the model. The azimuth is handled by the class. The predefined options are:

- **'compass': includes points in the direction of the azimuth at given** tolerance. The bandwidth parameter will be ignored.
- 'triangle': constructs a triangle with an angle of tolerance at the point of interest and union an rectangle parallel to azimuth, once the hypotenuse length reaches bandwidth.
- 'circle': constructs a half circle touching the point of interest, dislocating the center at the distance of bandwidth in the direction of azimuth. The half circle is union with an rectangle parallel to azimuth.

Visual representations, usage hints and implementation specifics are given in the documentation.

Parameters **model_name** (*string, callable*) – The name of the predefined model (string) or a function that accepts the current local coordinate system and returns a Polygon of the search area.

2.4.3 Estimator Functions

Scikit-GStat implements various semi-variance estimators. These fucntions can be found in the skgstat.estimators submodule. Each of these functions can be used independently from Variogram class. In this case the estimator is expecting an array of pairwise differences to calculate the semi-variance. Not the values themselves.

Matheron

skgstat.estimators.**matheron**()

Matheron Semi-Variance

Calculates the Matheron Semi-Variance from an array of pairwise differences. Returns the semi-variance for the whole array. In case a semi-variance is needed for multiple groups, this function has to be mapped on each group. That is the typical use case in geostatistics.

Parameters **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.

Returns

Return type numpy.float64

Notes

This implementation is done after the original publication[1] and the notes on their application[2]. Following[1], the semi-variance is calculated as:

$$\gamma(h) = \frac{1}{2N(h)} * \sum_{i=1}^{N(h)} (x)^2$$

with:

$$x = Z(x_i) - Z(x_{i+h})$$

where x is exactly the input array x.

References

Cressie

skgstat.estimators.**cressie**()

Cressie-Hawkins Semi-Variance

Calculates the Cressie-Hawkins Semi-Variance from an array of pairwise differences. Returns the semi-variance for the whole array. In case a semi-variance is needed for multiple groups, this function has to be mapped on each group. That is the typical use case in geostatistics.

[1] Matheron, G. (1962): Traité de Géostatistique Appliqué, Tonne 1. Memoires de Bureau de Recherches Géologiques et Miniéres, Paris.
[2] Matheron, G. (1965): Les variables regionalisées et leur estimation. Editions Masson et Cie, 212 S., Paris.

Parameters **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.

Returns

Return type numpy.float64

Notes

This implementation is done after the publication by Cressie and Hawkins from 1980[3]:

$$2\gamma(h) = \frac{(\frac{1}{N(h)} \sum_{i=1}^{N(h)} |x|^{0.5})^4}{0.457 + \frac{0.494}{N(h)} + \frac{0.045}{N^2(h)}}$$

with:

$$x = Z(x_i) - Z(x_{i+h})$$

where x is exactly the input array x.

References

Dowd

skgstat.estimators.**dowd**(*x*)

Dowd semi-variance

Calculates the Dowd semi-variance from an array of pairwise differences. Returns the semi-variance for the whole array. In case a semi-variance is needed for multiple groups, this function has to be mapped on each group. That is the typical use case in geostatistics.

Parameters **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.

Returns

Return type numpy.float64

Notes

The Dowd estimator is based on the median of all pairwise differences in each lag class and is therefore robust to exteme values at the cost of variability. This implementation is done after the publication _[4]:

$$2\gamma(h) = 2.198 * median(x)^2$$

with:

$$x = Z(x_i) - Z(x_{i+h})$$

where x is exactly the input array x.

[3] Cressie, N., and D. Hawkins (1980): Robust estimation of the variogram. Math. Geol., 12, 115-125.

References

Genton

skgstat.estimators.**genton**()
Genton robust semi-variance estimator

Return the Genton semi-variance of the given sample x. Genton is a highly robust varigram estimator, that is designed to be location free and robust on extreme values in x. Genton is based on calculating kth order statistics and will for large data sets be close or equal to the 25% quartile of all ordered point pairs in X.

Parameters **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.

Returns

Return type numpy.float64

Notes

The Genton estimator is described in great detail in the original publication[1] and befined as:

$$Q_{N_h} = 2.2191\{|V_i(h) - V_j(h)|; i < j\}_{(k)}$$

and

$$k = \binom{[N_h/2] + 1}{2}$$

and

$$q = \binom{N_h}{2}$$

where k is the kth quantile of all q point pairs. For large N (k/q) will be close to 0.25. For N >= 500, (k/q) is close to 0.25 by two decimals and will therefore be set to 0.5 and the two binomial coefficients k, q are not calculated.

References

Shannon Entropy

skgstat.estimators.**entropy**(*x*, *bins=None*)
Shannon Entropy estimator

Calculates the Shannon Entropy H as a variogram estimator. It is highly recommended to calculate the bins and explicitly set them as a list. In case this function is called for more than one lag class in a variogram, setting bins to None would result in different bin edges in each lag class. This would be very difficult to interpret.

Parameters

- **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.

- **bins** (*int, list, str*) – list of the bin edges used to calculate the empirical distribution of x. If bins is a list, these values are used directly. In case bins is a integer, as many even width bins will be calculated between the minimum and maximum value of x. In case bins is a string, it will be passed as bins argument to numpy.histograms function.

Returns **entropy** – Shannon entropy of the given pairwise differences.

Return type numpy.float64

Notes

MinMax

Warning: This is an experimental semi-variance estimator. It is heavily influenced by extreme values and outliers. That behaviour is usually not desired in geostatistics.

skgstat.estimators.**minmax**(*x*)

Minimum - Maximum Estimator

Returns a custom value. This estimator is the difference of maximum and minimum pairwise differences, normalized by the mean. MinMax will be very sensitive to extreme values.

Do only use this estimator, in case you know what you are doing. It is experimental and might change its behaviour in a future version.

Parameters **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.

Returns

Return type numpy.float64

Percentile

Warning: This is an experimental semi-variance estimator. It uses just a percentile of the given pairwise differences and does not bear any information about their variance.

skgstat.estimators.**percentile**(*x*, *p=50*)

Percentile estimator

Returns a given percentile as semi-variance. Do only use this estimator, in case you know what you are doing.

Do only use this estimator, in case you know what you are doing. It is experimental and might change its behaviour in a future version.

Parameters

- **x** (*numpy.ndarray*) – Array of pairwise differences. These values should be the distances between pairwise observations in value space. If xi and x[i+h] fall into the h separating distance class, x should contain abs(xi - x[i+h]) as an element.
- **p** (*int*) – Desired percentile. Should be given as whole numbers 0 < p < 100.

Returns

Return type np.float64

2.4.4 Variogram models

Scikit-GStat implements different theoretical variogram functions. These model functions expect a single lag value or an array of lag values as input data. Each function has at least a parameter *a* for the effective range and a parameter *c0* for the sill. The nugget parameter *b* is optinal and will be set to $b := 0$ if not given.

Spherical model

skgstat.models.**spherical**(*h*, *r*, *c0*, *b=0*)

Spherical Variogram function

Implementation of the spherical variogram function. Calculates the dependent variable for a given lag (h). The nugget (b) defaults to be 0.

Parameters

- **h** (*float*) – Specifies the lag of separating distances that the dependent variable shall be calculated for. It has to be a positive real number.
- **r** (*float*) – The effective range. Note this is not the range parameter! However, for the spherical variogram the range and effective range are the same.
- **c0** (*float*) – The sill of the variogram, where it will flatten out. The function will not return a value higher than C0 + b.
- **b** (*float*) – The nugget of the variogram. This is the value of independent variable at the distance of zero. This is usually attributed to non-spatial variance.

Returns **gamma** – Unlike in most variogram function formulas, which define the function for $2 * \gamma$, this function will return γ only.

Return type numpy.float64

Notes

The implementation follows[6]:

$$\gamma = b + C_0 * \left(1.5 * \frac{h}{r} - 0.5 * \frac{h}{r}^3\right)$$

if $h < r$, and

$$\gamma = b + C_0$$

else. r is the effective range, which is in case of the spherical variogram just a.

References

Exponential model

skgstat.models.**exponential**(*h*, *r*, *c0*, *b=0*)

Exponential Variogram function

[6] Burgess, T. M., & Webster, R. (1980). Optimal interpolation and isarithmic mapping of soil properties. I.The semi-variogram and punctual kriging. Journal of Soil and Science, 31(2), 315–331, http://doi.org/10.1111/j.1365-2389.1980.tb02084.x

Implementation of the exponential variogram function. Calculates the dependent variable for a given lag (h). The nugget (b) defaults to be 0.

Parameters

- **h** (*float*) – Specifies the lag of separating distances that the dependent variable shall be calculated for. It has to be a positive real number.
- **r** (*float*) – The effective range. Note this is not the range parameter! For the exponential variogram function the range parameter a is defined to be $a = \frac{r}{3}$. The effective range is the lag where 95% of the sill are exceeded. This is needed as the sill is only approached asymptotically by an exponential function.
- **c0** (*float*) – The sill of the variogram, where it will flatten out. The function will not return a value higher than C0 + b.
- **b** (*float*) – The nugget of the variogram. This is the value of independent variable at the distance of zero. This is usually attributed to non-spatial variance.

Returns **gamma** – Unlike in most variogram function formulas, which define the function for $2 * \gamma$, this function will return γ only.

Return type numpy.float64

Notes

The implementation following[7] and[8] is as:

$$\gamma = b + C_0 * \left(1 - e^{-\frac{h}{a}}\right)$$

a is the range parameter, that can be calculated from the effective range r as: $a = \frac{r}{3}$.

References

Gaussian model

skgstat.models.**gaussian**(*h*, *r*, *c0*, *b=0*)

Gaussian Variogram function

Implementation of the Gaussian variogram function. Calculates the dependent variable for a given lag (h). The nugget (b) defaults to be 0.

Parameters

- **h** (*float*) – Specifies the lag of separating distances that the dependent variable shall be calculated for. It has to be a positive real number.
- **r** (*float*) – The effective range. Note this is not the range parameter! For the exponential variogram function the range parameter a is defined to be $a = \frac{r}{3}$. The effetive range is the lag where 95% of the sill are exceeded. This is needed as the sill is only approached asymptotically by an exponential function.
- **c0** (*float*) – The sill of the variogram, where it will flatten out. The function will not return a value higher than C0 + b.
- **b** (*float*) – The nugget of the variogram. This is the value of independent variable at the distance of zero. This is usually attributed to non-spatial variance.

[7] Cressie, N. (1993): Statistics for spatial data. Wiley Interscience.
[8] Chiles, J.P., Delfiner, P. (1999). Geostatistics. Modeling Spatial Uncertainty. Wiley Interscience.

Returns **gamma** – Unlike in most variogram function formulas, which define the function for $2 * \gamma$, this function will return γ only.

Return type numpy.float64

Notes

This implementation follows[9]:

$$\gamma = b + c_0 * \left(1 - e^{-\frac{h^2}{a^2}}\right)$$

a is the range parameter, that can be calculated from the effective range r as:

$$a = \frac{r}{2}$$

References

Cubic model

`skgstat.models.`**`cubic`**(*h*, *r*, *c0*, *b=0*)

Cubic Variogram function

Implementation of the Cubic variogram function. Calculates the dependent variable for a given lag (h). The nugget (b) defaults to be 0.

Parameters

- **h** (*float*) – Specifies the lag of separating distances that the dependent variable shall be calculated for. It has to be a positive real number.
- **r** (*float*) – The effective range. Note this is not the range parameter! However, for the cubic variogram the range and effective range are the same.
- **c0** (*float*) – The sill of the variogram, where it will flatten out. The function will not return a value higher than C0 + b.
- **b** (*float*) – The nugget of the variogram. This is the value of independent variable at the distance of zero. This is usually attributed to non-spatial variance.

Returns **gamma** – Unlike in most variogram function formulas, which define the function for $2 * \gamma$, this function will return γ only.

Return type numpy.float64

Notes

This implementation is like:

$$\gamma = b + c_0 * \left[7 * \left(\frac{h^2}{a^2}\right) - \frac{35}{4} * \left(\frac{h^3}{a^3}\right) + \frac{7}{2} * \left(\frac{h^5}{a^5}\right) - \frac{3}{4} * \left(\frac{h^7}{a^7}\right)\right]$$

a is the range parameter. For the cubic function, the effective range and range parameter are the same.

[9] Chiles, J.P., Delfiner, P. (1999). Geostatistics. Modeling Spatial Uncertainty. Wiley Interscience.

Stable model

skgstat.models.**stable**(*h*, *r*, *c0*, *s*, *b=0*)
Stable Variogram function

Implementation of the stable variogram function. Calculates the dependent variable for a given lag (h). The nugget (b) defaults to be 0.

Parameters

- **h** (*float*) – Specifies the lag of separating distances that the dependent variable shall be calculated for. It has to be a positive real number.
- **r** (*float*) – The effective range. Note this is not the range parameter! For the stable variogram function the range parameter a is defined to be $a = \frac{r}{3^{\frac{1}{s}}}$. The effective range is the lag where 95% of the sill are exceeded. This is needed as the sill is only approached asymptotically by the e-function part of the stable model.
- **c0** (*float*) – The sill of the variogram, where it will flatten out. The function will not return a value higher than C0 + b.
- **s** (*float*) – Shape parameter. For s <= 2 the model will be shaped more like a exponential or spherical model, for s > 2 it will be shaped most like a Gaussian function.
- **b** (*float*) – The nugget of the variogram. This is the value of independent variable at the distance of zero. This is usually attributed to non-spatial variance.

Returns **gamma** – Unlike in most variogram function formulas, which define the function for $2 * \gamma$, this function will return γ only.

Return type numpy.float64

Notes

The implementation is:

$$\gamma = b + C_0 * \left(1. - e^{-\frac{h}{a}^s}\right)$$

a is the range parameter and is calculated from the effective range r as:

$$a = \frac{r}{3^{\frac{1}{s}}}$$

Matérn model

skgstat.models.**matern**(*h*, *r*, *c0*, *s*, *b=0*)
Matérn Variogram function

Implementation of the Matérn variogram function. Calculates the dependent variable for a given lag (h). The nugget (b) defaults to be 0.

Parameters

- **h** (*float*) – Specifies the lag of separating distances that the dependent variable shall be calculated for. It has to be a positive real number.
- **r** (*float*) – The effective range. Note this is not the range parameter! For the Matérn variogram function the range parameter a is defined to be $a = \frac{r}{2}$. The effective range is the lag where 95% of the sill are exceeded. This is needed as the sill is only approached asymptotically by Matérn model.

- **c0** (*float*) – The sill of the variogram, where it will flatten out. The function will not return a value higher than C0 + b.
- **s** (*float*) – Smoothness parameter. The smoothness parameter can shape a smooth or rough variogram function. A value of 0.5 will yield the exponential function, while a smoothness of +inf is exactly the Gaussian model. Typically a value of 10 is close enough to Gaussian shape to simulate its behaviour. Low values are considered to be 'smooth', while larger values are considered to describe a 'rough' random field.
- **b** (*float*) – The nugget of the variogram. This is the value of independent variable at the distance of zero. This is usually attributed to non-spatial variance.

Returns **gamma** – Unlike in most variogram function formulas, which define the function for $2 * \gamma$, this function will return γ only.

Return type numpy.float64

Notes

The formula and references will follow.

2.5 Changelog

2.5.1 Version 0.2.2

- added DirectionalVariogram class for direction-dependent variograms
- [Variogram] changed default values for *estimator* and *model* from function to string

2.5.2 Version 0.2.1

- added various unittests

2.5.3 Version 0.2.0

- completely rewritten Variogram class compared to v0.1.8