GeoStatTools Documentation

Release 1.0.1

Lennart Schueler, Sebastian Mueller

May 20, 2021

CONTENTS

CHAPTER 1

GSTOOLS QUICKSTART

GeoStatTools provides geostatistical tools for random field generation and variogram estimation based on many readily provided and even user-defined covariance models.

1.1 Installation

The package can be installed via pip. On Windows you can install WinPython to get Python and pip running.

```
pip install gstools
```

1.2 Spatial Random Field Generation

The core of this library is the generation of spatial random fields. These fields are generated using the randomisation method, described by Heße et al. 2014.

Examples

Gaussian Covariance Model

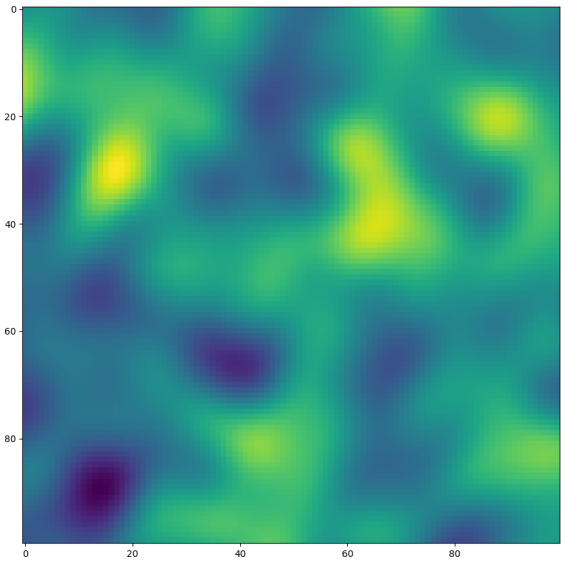
This is an example of how to generate a 2 dimensional spatial random field (*SRF*) with a *Gaussian* covariance model.

```
from gstools import SRF, Gaussian
import matplotlib.pyplot as plt
# structured field with a size 100x100 and a grid-size of 1x1
```

(continues on next page)

(continued from previous page)

```
x = y = range(100)
model = Gaussian(dim=2, var=1, len_scale=10)
srf = SRF(model)
field = srf((x, y), mesh_type='structured')
plt.imshow(field)
plt.show()
```



Truncated Power Law Model

GSTools also implements truncated power law variograms, which can be represented as a superposition of scale dependant modes in form of standard variograms, which are truncated by a lower- ℓ_{low} and an upper length-scale ℓ_{up}.

This example shows the truncated power law (*TPLStable*) based on the *Stable* covariance model and is given by

$$\gamma_{\ell_{\text{low}},\ell_{\text{up}}}(r) = \int_{\ell_{\text{low}}}^{\ell_{\text{up}}} \gamma(r,\lambda)\frac{\mathrm{d}\lambda}{\lambda}$$

with *Stable* modes on each scale:

$$\gamma(r,\lambda) = \sigma^2(\lambda)\cdot\left(1-\exp\left[-\left(\frac{r}{\lambda}\right)^{\alpha}\right]\right)$$
$$\sigma^2(\lambda) = C\cdot\lambda^{2H}$$

which gives Gaussian modes for `alpha=2` or Exponential modes for `alpha=1`.

For $\ell_{\text{low}} = 0$ this results in:

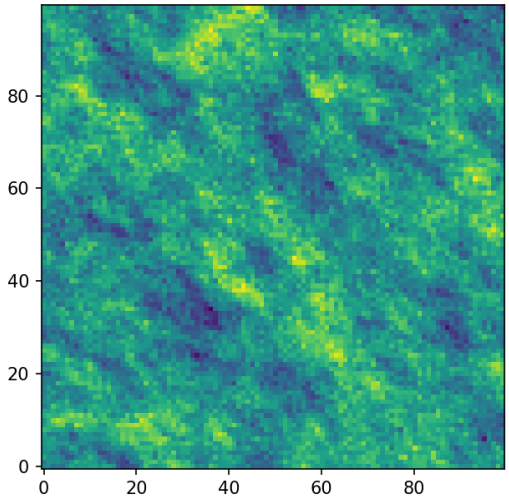
$$\gamma_{\ell_{\text{up}}}(r) = \sigma^2_{\ell_{\text{up}}}\cdot\left(1-\frac{2H}{\alpha}\cdot E_{1+\frac{2H}{\alpha}}\left[\left(\frac{r}{\ell_{\text{up}}}\right)^{\alpha}\right]\right)$$
$$\sigma^2_{\ell_{\text{up}}} = C\cdot\frac{\ell_{\text{up}}^{2H}}{2H}$$

```
import numpy as np
import matplotlib.pyplot as plt
from gstools import SRF, TPLStable
x = y = np.linspace(0, 100, 100)
```

(continues on next page)

(continued from previous page)

```
model = TPLStable(
    dim=2,            # spatial dimension
    var=1,            # variance (C calculated internally, so that `var` is 1)
    len_low=0,        # lower truncation of the power law
    len_scale=10,     # length scale (a.k.a. range), len_up = len_low + len_scale
    nugget=0.1,       # nugget
    anis=0.5,         # anisotropy between main direction and transversal ones
    angles=np.pi/4,   # rotation angles
    alpha=1.5,        # shape parameter from the stable model
    hurst=0.7,        # hurst coefficient from the power law
)
srf = SRF(model, mean=1, mode_no=1000, seed=19970221, verbose=True)
field = srf((x, y), mesh_type='structured', force_moments=True)
# show the field in correct xy coordinates
plt.imshow(field.T, origin="lower")
plt.show()
```



1.3 Estimating and fitting variograms

The spatial structure of a field can be analyzed with the variogram, which contains the same information as the covariance function.

All covariance models can be used to fit given variogram data by a simple interface.

Examples

This is an example of how to estimate the variogram of a 2 dimensional unstructured field and estimate the parameters of the covariance model again.

```
import numpy as np
from gstools import SRF, Exponential, Stable, vario_estimate_unstructured
from gstools.covmodel.plot import plot_variogram
import matplotlib.pyplot as plt
# generate a synthetic field with an exponential model
x = np.random.RandomState(19970221).rand(1000) * 100.
y = np.random.RandomState(20011012).rand(1000) * 100.
```

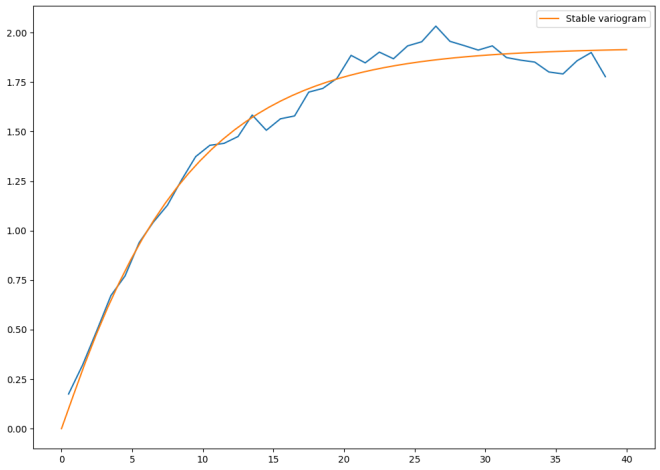
(continues on next page)

(continued from previous page)

```
model = Exponential(dim=2, var=2, len_scale=8)
srf = SRF(model, mean=0, seed=19970221)
field = srf((x, y))
# estimate the variogram of the field with 40 bins
bins = np.arange(40)
bin_center, gamma = vario_estimate_unstructured((x, y), field, bins)
plt.plot(bin_center, gamma)
# fit the variogram with a stable model. (no nugget fitted)
fit_model = Stable(dim=2)
fit_model.fit_variogram(bin_center, gamma, nugget=False)
plot_variogram(fit_model, x_max=40)
# output
print(fit_model)
plt.show()
```

Which gives:

```
Stable(dim=2, var=1.92, len_scale=8.15, nugget=0.0, anis=[1.], angles=[0.],␣
↪alpha=1.05)
```



1.4 User defined covariance models

One of the core-features of GSTools is the powerfull *CovModel* class, which allows to easy define covariance models by the user.

Examples

Here we reimplement the Gaussian covariance model by defining just the correlation function:

```
from gstools import CovModel
import numpy as np
# use CovModel as the base-class
class Gau(CovModel):
    def correlation(self, r):
        return np.exp(-(r/self.len_scale)**2)
```

And that's it! With `Gau` you now have a fully working covariance model, which you could use for field generation or variogram fitting as shown above.

1.5 VTK Export

After you have created a field, you may want to save it to file, so we provide a handy VTK export routine (*vtk_export*):

```
from gstools import SRF, Gaussian, vtk_export
x = y = range(100)
model = Gaussian(dim=2, var=1, len_scale=10)
srf = SRF(model)
field = srf((x, y), mesh_type='structured')
vtk_export("field", (x, y), field, mesh_type='structured')
```

Which gives a RectilinearGrid VTK file `field.vtr`.

1.6 Requirements

- Numpy >= 1.8.2
- SciPy >= 0.19.1
- hankel >= 0.3.6
- emcee
- pyevtk
- six

1.7 License

CHAPTER 2

GSTOOLS TUTORIALS

In the following you will find several Tutorials on how to use GSTools to explore its whole beauty and power.

2.1 Tutorial 1: Random Field Generation

The main feature of GSTools is the spatial random field generator *SRF*, which can generate random fields following a given covariance model. The generator provides a lot of nice features, which will be explained in the following

Theoretical Background

GSTools generates spatial random fields with a given covariance model or semi-variogram. This is done by using the so-called randomization method. The spatial random field is represented by a stochastic Fourier integral and its discretised modes are evaluated at random frequencies.

GSTools supports arbitrary and non-isotropic covariance models.

A very Simple Example

We are going to start with a very simple example of a spatial random field with an isotropic Gaussian covariance model and following parameters:

- variance $\sigma^2 = 1$
- correlation length $\lambda = 10$

First, we set things up and create the axes for the field. We are going to need the *SRF* class for the actual generation of the spatial random field. But *SRF* also needs a covariance model and we will simply take the *Gaussian* model.

```
import numpy as np
import matplotlib.pyplot as pt
from gstools import SRF, Gaussian

x = y = np.arange(100)
```

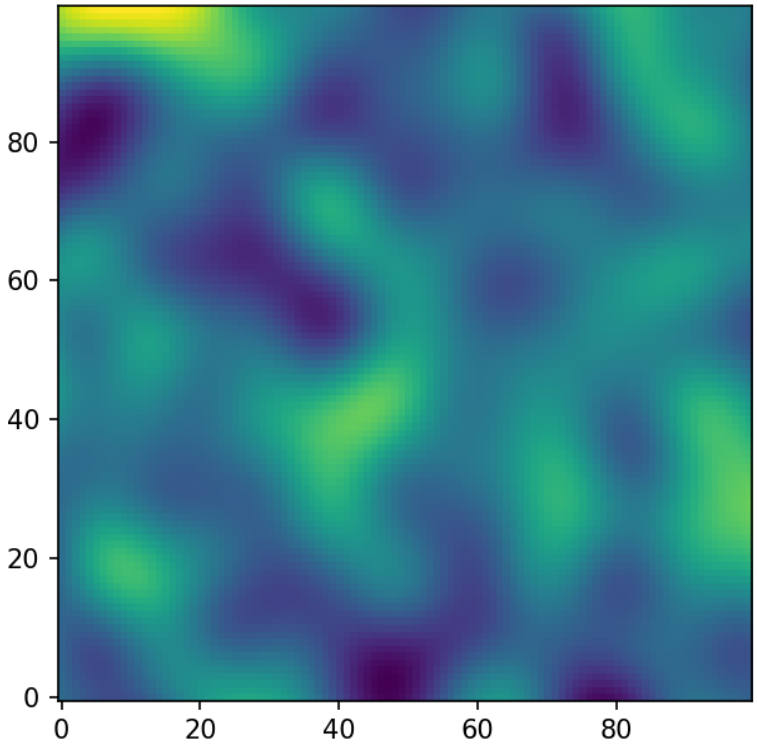
Now we create the covariance model with the parameters σ^2 and λ and hand it over to *SRF*. By specifying a seed, we make sure to create reproducible results:

```
model = Gaussian(dim=2, var=1, len_scale=10)
srf = SRF(model, seed=20170519)
```

With these simple steps, everything is ready to create our first random field. We will create the field on a structured grid (as you might have guessed from the x and y), which makes it easier to plot.

```
field = srf((x, y), mesh_type='structured')
pt.imshow(field.T, origin='lower')
pt.show()
```

Yielding



Wow, that was pretty easy!

The script can be found in `gstools/examples/00_gaussian.py`

Creating an Ensemble of Fields

Creating an ensemble of random fields would also be a great idea. Let's reuse most of the previous code.

```
import numpy as np
import matplotlib.pyplot as pt
from gstools import SRF, Gaussian

x = y = np.arange(100)

model = Gaussian(dim=2, var=1, len_scale=10)
srf = SRF(model)
```

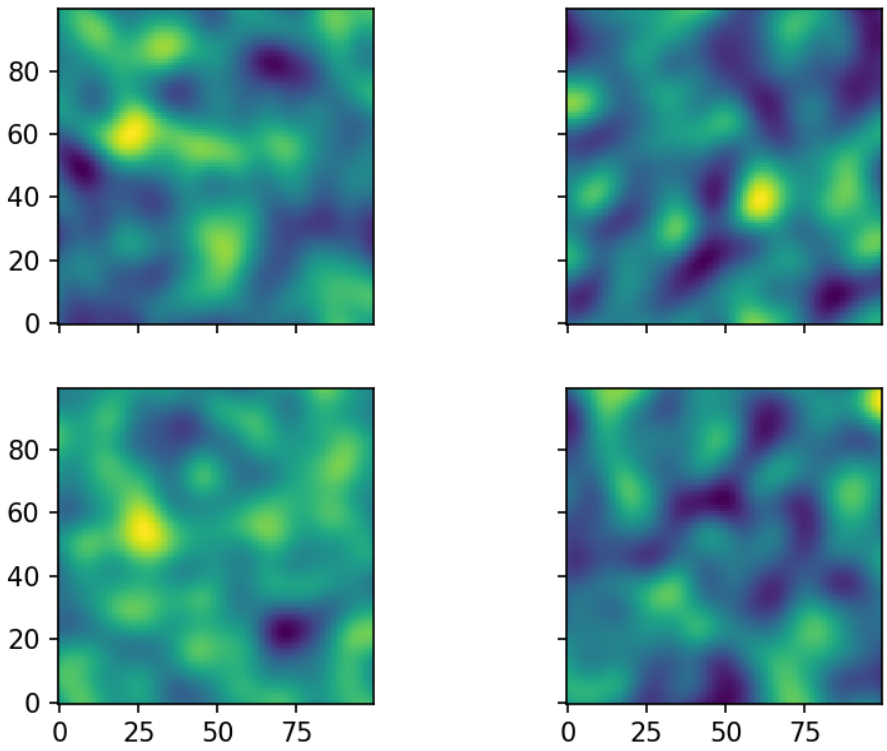
This time, we did not provide a seed to *SRF*, as the seeds will used during the actual computation of the fields. We will create four ensemble members, for better visualisation and save them in a list and in a first step, we will be using the loop counter as the seeds.

```
ens_no = 4
field = []
for i in range(ens_no):
    field.append(srf((x, y), seed=i, mesh_type='structured'))
```

Now let's have a look at the results:

```
fig, ax = pt.subplots(2, 2, sharex=True, sharey=True)
ax = ax.flatten()
for i in range(ens_no):
    ax[i].imshow(field[i].T, origin='lower')
pt.show()
```

Yielding



The script can be found in `gstools/examples/05_srf_ensemble.py`

Using better Seeds

It is not always a good idea to use incrementing seeds. Therefore GSTools provides a seed generator *MasterRNG*. The loop, in which the fields are generated would then look like

```
from gstools.random import MasterRNG
seed = MasterRNG(20170519)
for i in range(ens_no):
    field.append(srf((x, y), seed=seed(), mesh_type='structured'))
```

Creating Fancier Fields

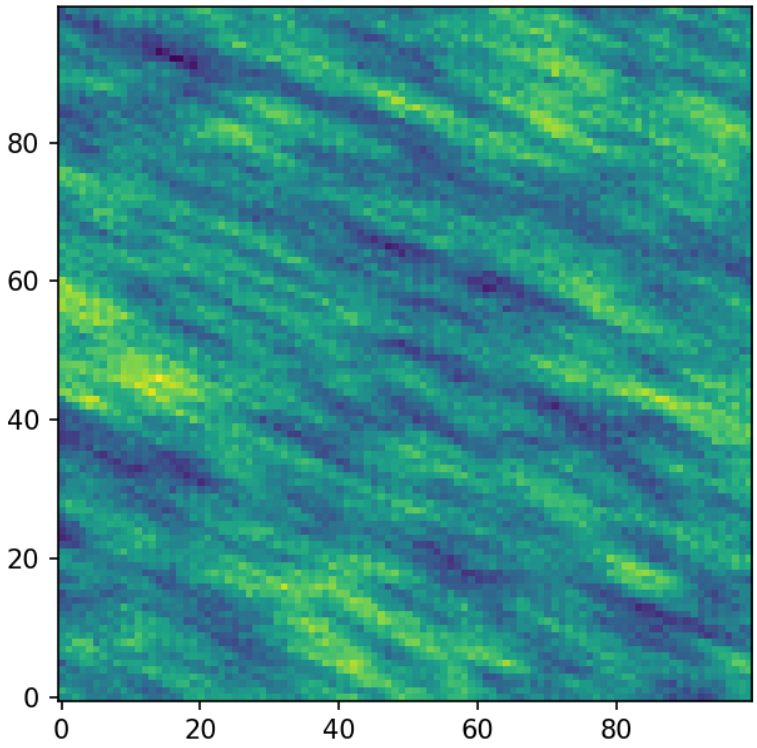
Only using Gaussian covariance fields gets boring. Now we are going to create much rougher random fields by using an exponential covariance model and we are going to make them anisotropic.

The code is very similar to the previous examples, but with a different covariance model class *Exponential*. As model parameters we a using following

- variance $\sigma^2 = 1$
- correlation length $\lambda = (12, 3)^T$
- rotation angle $\theta = \pi/8$

```
from gstools import SRF, Exponential
import numpy as np
import matplotlib.pyplot as pt

x = y = np.arange(100)

model = Exponential(dim=2, var=1, len_scale=[12., 3.], angles=np.pi/8.)
srf = SRF(model, seed=20170519)

field = srf((x, y), mesh_type='structured')

pt.imshow(field.T, origin='lower')
pt.show()
```

Yielding



The anisotropy ratio could also have been set with

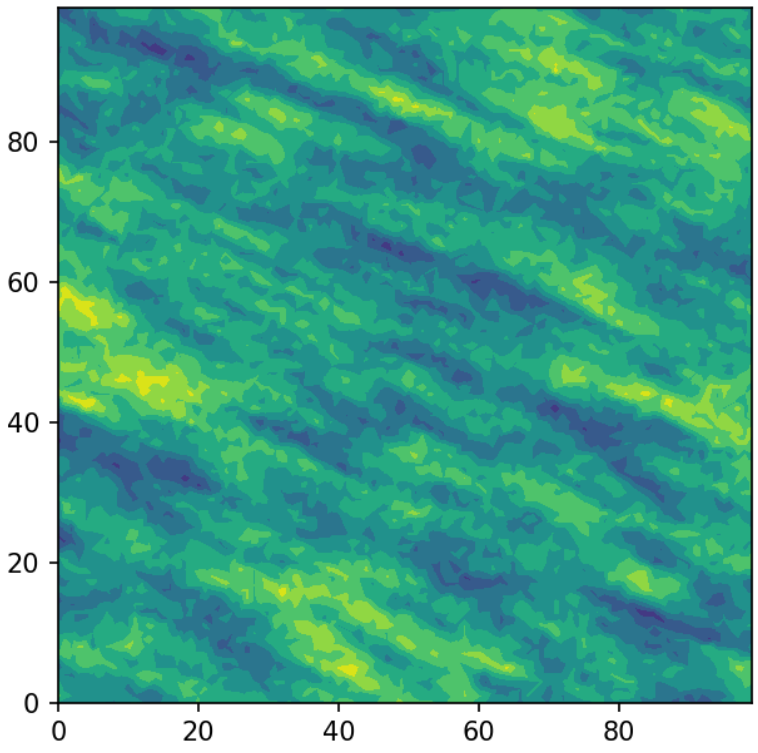
```
model = Exponential(dim=2, var=1, len_scale=12., anis=3./12., angles=np.pi/8.)
```

Using an Unstructured Grid

For many applications, the random fields are needed on an unstructured grid. Normally, such a grid would be read in, but we can simply generate one and then create a random field at those coordinates.

```
import numpy as np
import matplotlib.pyplot as pt
from gstools import SRF, Exponential
from gstools.random import MasterRNG

seed = MasterRNG(19970221)
rng = np.random.RandomState(seed())
x = rng.randint(0, 100, size=10000)
y = rng.randint(0, 100, size=10000)

model = Exponential(dim=2, var=1, len_scale=[12., 3.], angles=np.pi/8.)

srf = SRF(model, seed=20170519)

field = srf((x, y))

pt.tricontourf(x, y, field.T)
pt.axes().set_aspect('equal')
pt.show()
```

Yielding



Comparing this image to the previous one, you can see that be using the same seed, the same field can be computed on different grids.

The script can be found in `gstools/examples/06_unstr_srf_export.py`

Exporting a Field

Using the field from *previous example*, it can simply be exported to the file `field.vtu` and viewed by e.g. paraview with following lines of code

```
from gstools import vtk_export
vtk_export('field', (x, y), field)
```

The script can be found in `gstools/examples/04_export.py` and in `gstools/examples/06_unstr_srf_export.py`

Merging two Fields

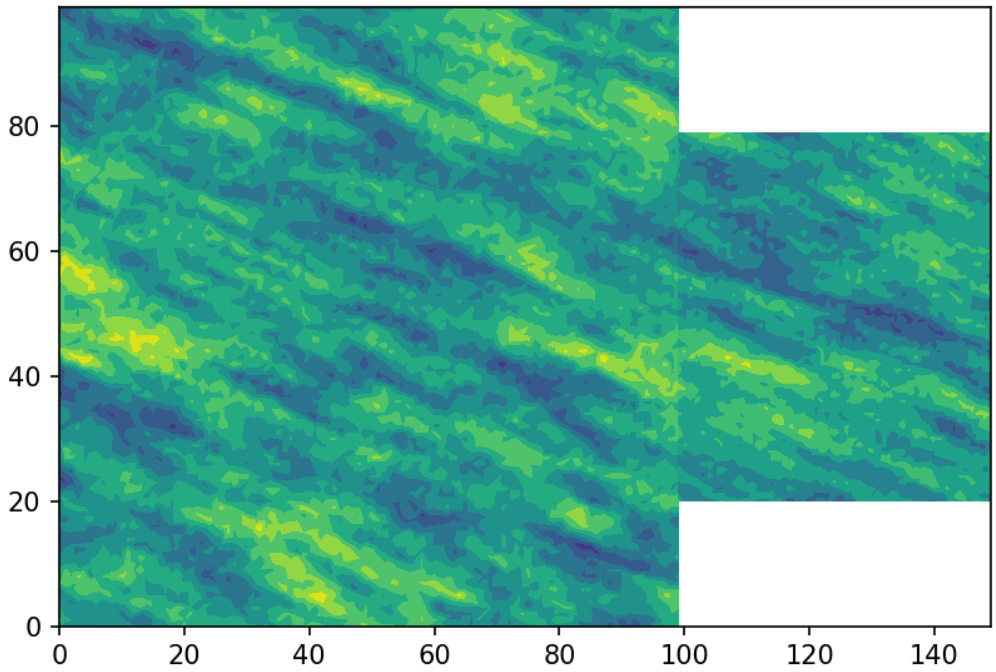
We can even generate the same field realisation on different grids. Let's try to merge two unstructured rectangular fields. The first field will be generated exactly like in example *Using an Unstructured Grid*:

```
import numpy as np
import matplotlib.pyplot as pt
from gstools import SRF, Exponential
from gstools.random import MasterRNG

seed = MasterRNG(19970221)
rng = np.random.RandomState(seed())
x = rng.randint(0, 100, size=10000)
y = rng.randint(0, 100, size=10000)

model = Exponential(dim=2, var=1, len_scale=[12., 3.], angles=np.pi/8.)

srf = SRF(model, seed=20170519)

field = srf((x, y))
```

But now we extend the field on the right hand side by creating a new unstructured grid and calculating a field with the same parameters and the same seed on it:

```
# new grid
seed = MasterRNG(20011012)
rng = np.random.RandomState(seed())
x2 = rng.randint(99, 150, size=10000)
y2 = rng.randint(20, 80, size=10000)

field2 = srf((x2, y2))

pt.tricontourf(x, y, field.T)
pt.tricontourf(x2, y2, field2.T)
pt.axes().set_aspect('equal')
pt.show()
```

Yielding



The slight mismatch where the two fields were merged is merely due to interpolation problems of the plotting routine. You can convince yourself be increasing the resolution of the grids by a factor of 10.

Of course, this merging could also have been done by appending the grid point `(x2, y2)` to the original grid `(x, y)` before generating the field. But one application scenario would be to generate hugh fields, which would not fit into memory anymore.

The script can be found in `gstools/examples/07_srf_merge.py`

2.2 Tutorial 2: The Covariance Model

One of the core-features of GSTools is the powerful *CovModel* class, which allows you to easily define arbitrary covariance models by yourself. The resulting models provide a bunch of nice features to explore the covariance models.

Theoretical Backgound

A covariance model is used to characterize the semi-variogram, denoted by γ, of a spatial random field. In GSTools, we use the following form for an isotropic and stationary field:

$$\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$$

Where:

- $\mathrm{cor}(r)$ is the so called correlation function depending on the distance r
- σ^2 is the variance
- n is the nugget (subscale variance)

Note: We are not limited to isotropic models. We support anisotropy ratios for length scales in orthogonal transversal directions like:

- x (main direction)
- y (1. transversal direction)
- z (2. transversal direction)

These main directions can also be rotated, but we will come to that later.

Example

Let us start with a short example of a self defined model (Of course, we provide a lot of predefined models [See: *gstools.covmodel*], but they all work the same way). Therefore we reimplement the Gaussian covariance model by defining just the correlation function:

```
from gstools import CovModel
import numpy as np
# use CovModel as the base-class
class Gau(CovModel):
    def correlation(self, r):
        return np.exp(-(r/self.len_scale)**2)
```

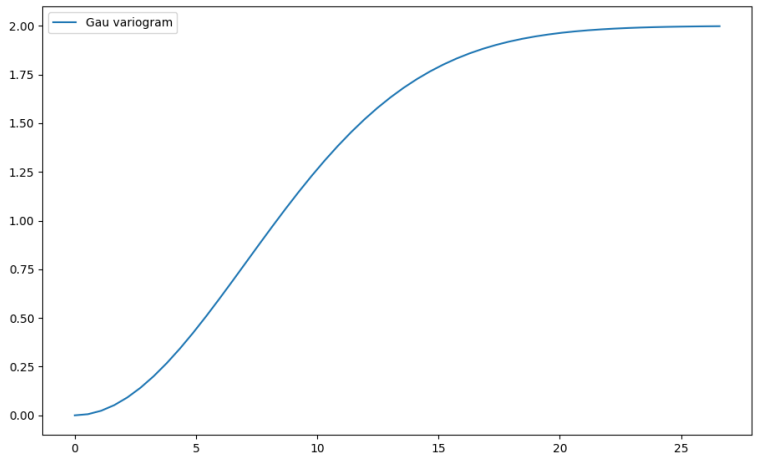
Now we can instantiate this model:

```
model = Gau(dim=2, var=2., len_scale=10)
```

To have a look at the variogram, let's plot it:

```
from gstools.covmodel.plot import plot_variogram
plot_variogram(model)
```

Which gives:



Parameters

We already used some parameters, which every covariance models has. The basic ones are:

- **dim** : dimension of the model
- **var** : variance of the model (on top of the subscale variance)
- **len_scale** : length scale of the model
- **nugget** : nugget (subscale variance) of the model

These are the common parameters used to characterize a covariance model and are therefore used by every model in GSTools. You can also access and reset them:

```
print(model.dim, model.var, model.len_scale, model.nugget, model.sill)
model.dim = 3
model.var = 1
model.len_scale = 15
model.nugget = 0.1
print(model.dim, model.var, model.len_scale, model.nugget, model.sill)
```

Which gives:

```
2 2.0 10 0.0 2.0
3 1.0 15 0.1 1.1
```

Note:

- The sill of the variogram is calculated by `sill = variance + nugget` So we treat the variance as everything **above** the nugget, which is sometimes called **partial sill**.
- A covariance model can also have additional parameters.

Anisotropy

The internally used (semi-) variogram represents the isotropic case for the model. Nevertheless, you can provide anisotropy ratios by:

```
model = Gau(dim=3, var=2., len_scale=10, anis=0.5)
print(model.anis)
print(model.len_scale_vec)
```

Which gives:

```
[0.5 1. ]
[10.  5. 10.]
```

As you can see, we defined just one anisotropy-ratio and the second transversal direction was filled up with `1.` and you can get the length-scales in each direction by the attribute *`len_scale_vec`*. For full control you can set a list of anistropy ratios: `anis=[0.5, 0.4]`.

Alternatively you can provide a list of length-scales:

```
model = Gau(dim=3, var=2., len_scale=[10, 5, 4])
print(model.anis)
print(model.len_scale)
print(model.len_scale_vec)
```

Which gives:

```
[0.5 0.4]
10
[10.  5.  4.]
```

Rotation Angles

The main directions of the field don't have to coincide with the spatial directions x, y and z. Therefore you can provide rotation angles for the model:

```
model = Gau(dim=3, var=2., len_scale=10, angles=2.5)
print(model.angles)
```

Which gives:

```
[2.5 0.  0. ]
```

Again, the angles were filled up with `0.` to match the dimension and you could also provide a list of angles. The number of angles depends on the given dimension:

- in 1D: no rotation performable
- in 2D: given as rotation around z-axis
- in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

Methods

The covariance model class *`CovModel`* of GSTools provides a set of handy methods.

Basics

One of the following functions defines the main characterization of the variogram:

- `variogram` : The variogram of the model given by

$$\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$$

- `variogram_normed` : The normalized variogram of the model given by

$$\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$$

- `covariance` : The (auto-)covariance of the model given by

$$C(r) = \sigma^2 \cdot \mathrm{cor}(r)$$

- `correlation` : The (auto-)correlation (or normalized covariance) of the model given by

$$\mathrm{cor}(r)$$

As you can see, it is the easiest way to define a covariance model by giving a correlation function as demonstrated by the above model `Gau`. If one of the above functions is given, the others will be determined:

```
model = Gau(dim=3, var=2., len_scale=10, nugget=0.5)
print(model.variogram(10.))
print(model.variogram_normed(10.))
print(model.covariance(10.))
print(model.correlation(10.))
```

Which gives:

```
1.7642411176571153
0.6321205588285577
0.7357588823428847
0.36787944117144233
```

Spectral methods

The spectrum of a covariance model is given by:

$$S(\mathbf{k}) = \left(\frac{1}{2\pi}\right)^n \int C(\|\mathbf{r}\|) e^{ib\mathbf{k}\cdot\mathbf{r}} d^n\mathbf{r}$$

Since the covariance function $C(r)$ is radially symmetric, we can calculate this by the hankel-transformation:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \cdot \frac{(2\pi)^{n/2}}{(bk)^{n/2-1}} \int_0^\infty r^{n/2-1} C(r) J_{n/2-1}(bkr) r dr$$

Where $k = \|\mathbf{k}\|$.

Depending on the spectrum, the spectral-density is defined by:

$$\tilde{S}(k) = \frac{S(k)}{\sigma^2}$$

You can access these methods by:

```
model = Gau(dim=3, var=2., len_scale=10)
print(model.spectrum(0.1))
print(model.spectral_density(0.1))
```

Which gives:

```
34.96564773852395
17.482823869261974
```

Note: The spectral-density is given by the radius of the input phase. But it is **not** a probability density function for the radius of the phase. To obtain the pdf for the phase-radius, you can use the methods *spectral_rad_pdf* or *ln_spectral_rad_pdf* for the logarithm.

The user can also provide a cdf (cumulative distribution function) by defining a method called `spectral_rad_cdf` and/or a ppf (percent-point function) by `spectral_rad_ppf`.

The attributes *has_cdf* and *has_ppf* will check for that.

Different scales

Besides the length-scale, there are many other ways of characterizing a certain scale of a covariance model. We provide two common scales with the covariance model.

Integral scale

The integral scale of a covariance model is calculated by:

$$I = \int_0^\infty \mathrm{cor}(r) dr$$

You can access it by:

```
model = Gau(dim=3, var=2., len_scale=10)
print(model.integral_scale)
print(model.integral_scale_vec)
```

Which gives:

```
8.862269254527579
[8.86226925 8.86226925 8.86226925]
```

You can also specify integral length scales like the ordinary length scale, and len_scale/anis will be recalculated:

```
model = Gau(dim=3, var=2., integral_scale=[10, 4, 2])
print(model.anis)
print(model.len_scale)
print(model.len_scale_vec)
print(model.integral_scale)
print(model.integral_scale_vec)
```

Which gives:

```
[0.4 0.2]
11.283791670955127
[11.28379167  4.51351667  2.25675833]
10.000000000000002
[10.  4.  2.]
```

Percentile scale

Another scale characterizing the covariance model, is the percentile scale. It is the distance, where the normalized variogram reaches a certain percentage of its sill.

```
model = Gau(dim=3, var=2., len_scale=10)
print(model.percentile_scale(0.9))
```

Which gives:

```
15.174271293851463
```

Note: The nugget is neglected by this percentile_scale.

Additional Parameters

Let's pimp our self-defined model `Gau` by setting the exponent as an additional parameter:

$$\mathrm{cor}(r) := \exp\left(-\left(\frac{r}{\ell}\right)^{\alpha}\right)$$

This leads to the so called **stable** covariance model and we can define it by

```
class Stab(CovModel):
    def default_opt_arg(self):
        return {"alpha": 1.5}
    def correlation(self, r):
        return np.exp(-(r/self.len_scale)**self.alpha)
```

As you can see, we override the method *CovModel.default_opt_arg* to provide a standard value for the optional argument `alpha` and we can access it in the correlation function by `self.alpha`

Now we can instantiate this model:

```
model1 = Stab(dim=2, var=2., len_scale=10)
model2 = Stab(dim=2, var=2., len_scale=10, alpha=0.5)
print(model1)
print(model2)
```

Which gives:

```
Stab(dim=2, var=2.0, len_scale=10, nugget=0.0, anis=[1.], angles=[0.], alpha=1.5)
Stab(dim=2, var=2.0, len_scale=10, nugget=0.0, anis=[1.], angles=[0.], alpha=0.5)
```

Note: You don't have to overrid the *CovModel.default_opt_arg*, but you will get a ValueError if you don't set it on creation.

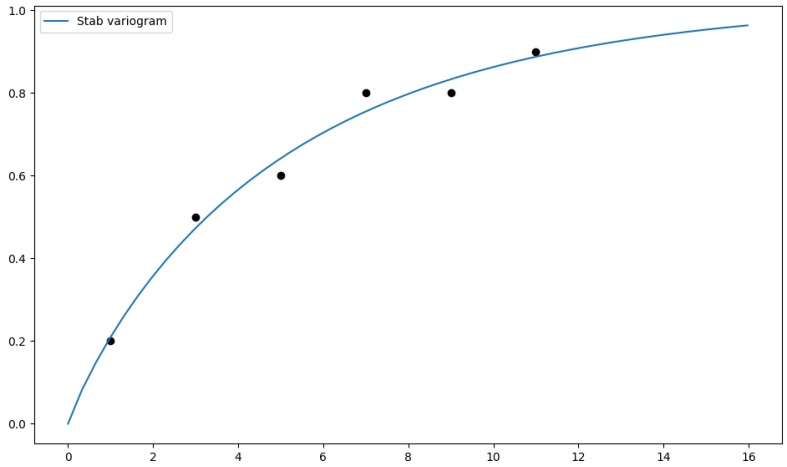
Fitting variogram data

The model class comes with a routine to fit the model-parameters to given variogram data. Have a look at the following:

```
# data
x = [1.0, 3.0, 5.0, 7.0, 9.0, 11.0]
y = [0.2, 0.5, 0.6, 0.8, 0.8, 0.9]
# fitting model
model = Stab(dim=2)
# we have to provide boundaries for the parameters
model.set_arg_bounds(alpha=[0, 3])
# fit the model to given data, deselect nugget
results, pcov = model.fit_variogram(x, y, nugget=False)
print(results)
# show the fitting
from matplotlib import pyplot as plt
from gstools.covmodel.plot import plot_variogram
plt.scatter(x, y, color="k")
plot_variogram(model)
plt.show()
```

Which gives:

```
{'var': 1.024575782651677,
 'len_scale': 5.081620691462197,
 'nugget': 0.0,
 'alpha': 0.906705123369987}
```



As you can see, we have to provide boundaries for the parameters. As a default, the following bounds are set:

- additional parameters: `[0.0, 1000.0]`
- variance: `[0.0, 100.0]`
- len_scale: `[0.0, 1000.0]`
- nugget: `[0.0, 100.0]`

Also, you can deselect parameters from fitting, so their predefined values will be kept. In our case, we fixed a `nugget` of `0.0`, which was set by default. You can deselect any standard or optional argument of the covariance model. The second return value `pcov` is the estimated covariance of `popt` from the used scipy routine `scipy.optimize.curve_fit`.

You can use the following methods to manipulate the used bounds:

CovModel.default_opt_arg_bounds()	Here you can provide a dictionary with default boundaries for the optional arguments.
CovModel.default_arg_bounds()	Here you can provide a dictionary with default boundaries for the standard arguments.
CovModel.set_arg_bounds(**kwargs)	Set bounds for the parameters of the model
CovModel.check_arg_bounds()	Here the arguments are checked to be within the given bounds

You can override the *CovModel.default_opt_arg_bounds* to provide standard bounds for your additional parameters.

To access the bounds you can use:

CovModel.var_bounds	Bounds for the variance
CovModel.len_scale_bounds	Bounds for the lenght scale
CovModel.nugget_bounds	Bounds for the nugget
CovModel.opt_arg_bounds	Bounds for the optional arguments
CovModel.arg_bounds	Bounds for all parameters

Provided Covariance Models

The following standard covariance models are provided by GSTools

Gaussian([dim, var, len_scale, nugget, ...])	The Gaussian covariance model
Exponential([dim, var, len_scale, nugget, ...])	The Exponential covariance model

Continued on next page

Table 3 – continued from previous page

Matern([dim, var, len_scale, nugget, anis, ...])	The Matérn covariance model
Rational([dim, var, len_scale, nugget, ...])	The rational quadratic covariance model
Stable([dim, var, len_scale, nugget, anis, ...])	The stable covariance model
Spherical([dim, var, len_scale, nugget, ...])	The Spherical covariance model
Linear([dim, var, len_scale, nugget, anis, ...])	The bounded linear covariance model
MaternRescal([dim, var, len_scale, nugget, ...])	The rescaled Matérn covariance model
SphericalRescal([dim, var, len_scale, ...])	The rescaled Spherical covariance model

As a special feature, we also provide truncated power law (TPL) covariance models

TPLGaussian([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Gaussian modes
TPLExponential([dim, var, len_scale, ...])	Truncated-Power-Law with Exponential modes
TPLStable([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Stable modes

2.3 Tutorial 3: Variogram Estimation

Estimating the spatial correlations is an important part of geostatistics. These spatial correlations can be expressed by the variogram, which can be estimated with the subpackage *gstools.variogram*. The variograms can be estimated on structured and unstructured grids.

Theoretical Background

The same (semi-)variogram as the Covariance Model is being used by this subpackage.

An Example with Actual Data

This example is going to be a bit more extensive and we are going to do some basic data preprocessing for the actual variogram estimation. But this example will be self-contained and all data gathering and processing will be done in this example script.

The complete script can be found in `gstools/examples/08_variogram_estimation.py`

This example will only work with Python 3.

The Data

We are going to analyse the Herten aquifer, which is situated in Southern Germany. Multiple outcrop faces where surveyed and interpolated to a 3D dataset. In these publications, you can find more information about the data:

Bayer, Peter; Comunian, Alessandro; Höyng, Dominik; Mariethoz, Gregoire (2015): Physicochemical properties and 3D geostatistical simulations of the Herten and the Descalvado aquifer analogs. PANGAEA, https://doi.org/10.1594/PANGAEA.844167,
Supplement to: Bayer, P et al. (2015): Three-dimensional multi-facies realizations of sedimentary reservoir and aquifer analogs. Scientific Data, 2, 150033, https://doi.org/10.1038/sdata.2015.33

Retrieving the Data

To begin with, we need to download and extract the data. Therefore, we are going to use some built-in Python libraries. For simplicity, many values and strings will be hardcoded.

```
import os
import urllib.request
import zipfile
import numpy as np
import matplotlib.pyplot as pt

def download_herten():
    # download the data, warning: its about 250MB
    print('Downloading Herten data')
    data_filename = 'data.zip'
    data_url = 'http://store.pangaea.de/Publications/Bayer_et_al_2015/Herten-
↪analog.zip'
    urllib.request.urlretrieve(data_url, 'data.zip')

    # extract the data
    with zipfile.ZipFile(data_filename, 'r') as zf:
        zf.extract(os.path.join('Herten-analog', 'sim-big_1000x1000x140',
                                'sim.vtk'))
```

That was that. But we also need a script to convert the data into a format we can use. This script is also kindly provided by the authors. We can download this script in a very similar manner as the data:

```
def download_scripts():
    # download a script for file conversion
    print('Downloading scripts')
    tools_filename = 'scripts.zip'
    tool_url = 'http://store.pangaea.de/Publications/Bayer_et_al_2015/tools.zip'
    urllib.request.urlretrieve(tool_url, tools_filename)

    # only extract the script we need
    with zipfile.ZipFile(tools_filename, 'r') as zf:
        zf.extract(os.path.join('tools', 'vtk2gslib.py'))
```

These two functions can now be called:

```
download_herten()
download_scripts()
```

Preprocessing the Data

First of all, we have to convert the data with the script we just downloaded

```
# import the downloaded conversion script
from tools.vtk2gslib import vtk2numpy

# load the Herten aquifer with the downloaded vtk2numpy routine
print('Loading data')
herten, grid = vtk2numpy(os.path.join('Herten-analog', 'sim-big_1000x1000x140',
↪'sim.vtk'))
```

The data only contains facies, but from the supplementary data, we know the hydraulic conductivity values of each facies, which we will simply paste here and assign them to the correct facies

```
# conductivity values per fazies from the supplementary data
cond = np.array([2.50E-04, 2.30E-04, 6.10E-05, 2.60E-02, 1.30E-01,
                 9.50E-02, 4.30E-05, 6.00E-07, 2.30E-03, 1.40E-04,])

# asign the conductivities to the facies
herten_cond = cond[herten]
```

Next, we are going to calculate the transmissivity, by integrating over the vertical axis

```
# integrate over the vertical axis, calculate transmissivity
herten_log_trans = np.log(np.sum(herten_cond, axis=2) * grid['dz'])
```

The Herten data provides information about the grid, which was already used in the previous code block. From this information, we can create our own grid on which we can estimate the variogram. As a first step, we are going to estimate an isotropic variogram, meaning that we will take point pairs from all directions into account. An unstructured grid is a natural choice for this. Therefore, we are going to create an unstructured grid from the given, structured one. For this, we are going to write another small function

```
def create_unstructured_grid(x_s, y_s):
    x_u, y_u = np.meshgrid(x_s, y_s)
    len_unstruct = len(x_s) * len(y_s)
    x_u = np.reshape(x_u, len_unstruct)
    y_u = np.reshape(y_u, len_unstruct)
    return x_u, y_u

# create a structured grid on which the data is defined
```

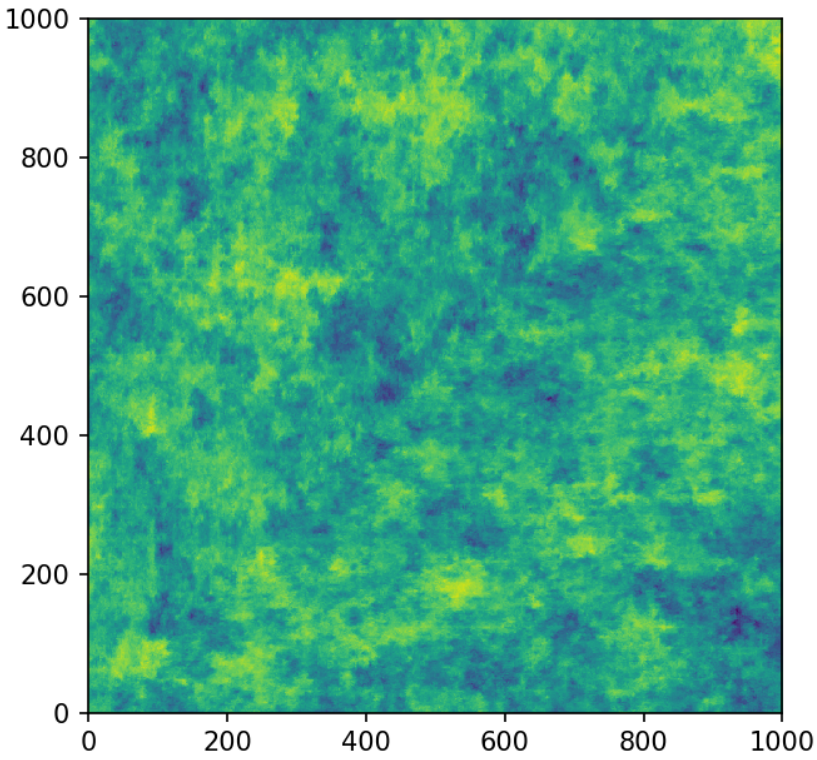
(continues on next page)

(continued from previous page)

```
x_s = np.arange(grid['ox'], grid['nx']*grid['dx'], grid['dx'])
y_s = np.arange(grid['oy'], grid['ny']*grid['dy'], grid['dy'])

# create an unstructured grid for the variogram estimation
x_u, y_u = create_unstructured_grid(x_s, y_s)
```

Let's have a look at the transmissivity field of the Herten aquifer

```
pt.imshow(herten_log_trans.T, origin='lower', aspect='equal')
pt.show()
```



Estimating the Variogram

Finally, everything is ready for the variogram estimation. For the unstructured method, we have to define the bins on which the variogram will be estimated. Through expert knowledge (i.e. fiddling around), we assume that the main features of the variogram will be below 10 metres distance. And because the data has a high spatial resolution, the resolution of the bins can also be high. The transmissivity data is still defined on a structured grid, but we can simply flatten it with `numpy.ndarray.flatten`, in order to bring it into the right shape. It might be more memory efficient to use `herten_log_trans.reshape(-1)`, but for better readability, we will stick to `numpy.ndarray.flatten`. Taking all data points into account would take a very long time (expert knowledge *wink*), thus we will only take 2000 datapoints into account, which are sampled randomly. In order to make the exact results reproducible, we can also set a seed.

```
from gstools import vario_estimate_unstructured

bins = np.linspace(0, 10, 50)
print('Estimating unstructured variogram')
```

(continues on next page)

(continued from previous page)

```
bin_center, gamma = vario_estimate_unstructured(
    (x_u, y_u),
    herten_log_trans.flatten(),
    bins,
    sampling_size=2000,
    sampling_seed=19920516,
    )
```

The estimated variogram is calculated on the centre of the given bins, therefore, the `bin_center` array is also returned.

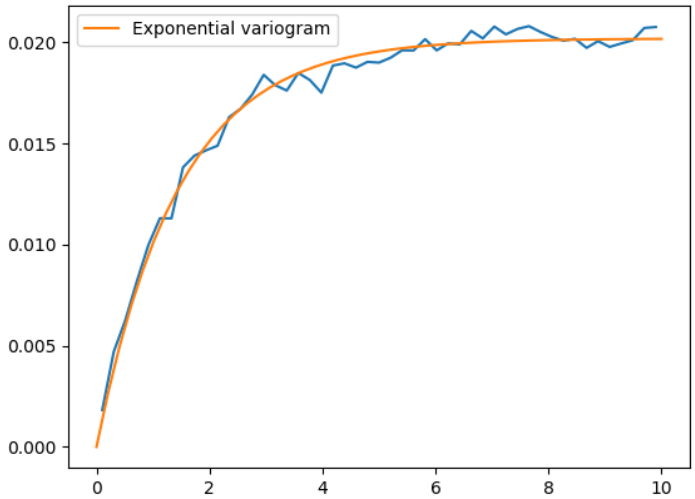
Fitting the Variogram

Now, we can see, if the estimated variogram can be modelled by a common variogram model. Let's try the *Exponential* model.

```
from gstools import Exponential

# fit an exponential model
fit_model = Exponential(dim=2)
fit_model.fit_variogram(bin_center, gamma, nugget=False)
```

Finally, we can visualise some results. For quickly plotting a covariance model, GSTools provides some helper functions.

```
from gstools.covmodel.plot import plot_variogram
pt.plot(bin_center, gamma)
plot_variogram(fit_model, x_max=bins[-1])
pt.show()
```



That looks like a pretty good fit! By printing the model, we can directly see the fitted parameters

```
print(fit_model)
```

which gives

```
Exponential(dim=2, var=0.020193095802479327, len_scale=1.4480057557321007,␣
↪nugget=0.0, anis=[1.], angles=[0.])
```

With this data, we could start generating new ensembles of the Herten aquifer with the *SRF* class.

Estimating the Variogram in Specific Directions

Estimating a variogram on a structured grid gives us the possibility to only consider values in a specific direction. This could be a first test, to see if the data is anisotropic. In order to speed up the calculations, we are going to only use every 10th datapoint and for a comparison with the isotropic variogram calculated earlier, we only need the first 21 array items.

```
x_s = x_s[::10][:21]
y_s = y_s[::10][:21]
herten_trans_log = herten_log_trans[::10,::10]
```

With this much smaller data set, we can immediately estimate the variogram in the x- and y-axis

```
from gstools import vario_estimate_structured
print('Estimating structured variograms')
gamma_x = vario_estimate_structured(herten_trans_log, direction='x')[:21]
gamma_y = vario_estimate_structured(herten_trans_log, direction='y')[:21]
```

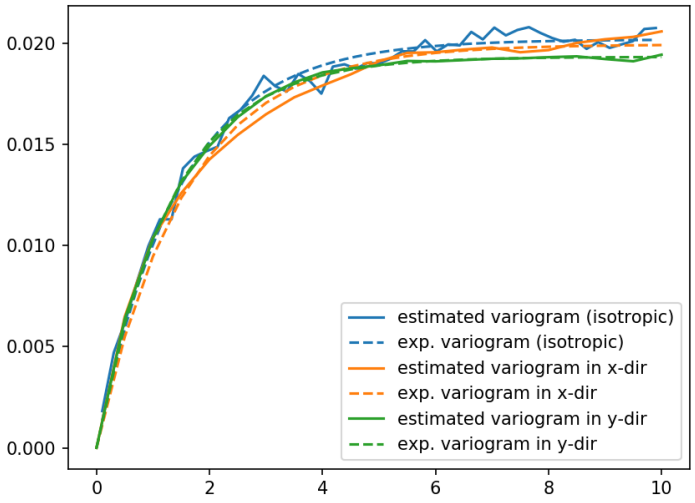
With these two estimated variograms, we can start fitting *Exponential* covariance models

```
fit_model_x = Exponential(dim=2)
fit_model_x.fit_variogram(x_s, gamma_x, nugget=False)
fit_model_y = Exponential(dim=2)
fit_model_y.fit_variogram(y_s, gamma_y, nugget=False)
```

Now, the isotropic variogram and the two variograms in x- and y-direction can be plotted together with their respective models, which will be plotted with dashed lines.

```
line, = pt.plot(bin_center, gamma, label='estimated variogram (isotropic)')
pt.plot(bin_center, fit_model.variogram(bin_center), color=line.get_color(),
        linestyle='--', label='exp. variogram (isotropic)')

line, = pt.plot(x_s, gamma_x, label='estimated variogram in x-dir')
pt.plot(x_s, fit_model_x.variogram(x_s), color=line.get_color(),
        linestyle='--', label='exp. variogram in x-dir')

line, = pt.plot(y_s, gamma_y, label='estimated variogram in y-dir')
pt.plot(y_s, fit_model_y.variogram(y_s),
    color=line.get_color(), linestyle='--', label='exp. variogram in y-dir')

pt.legend()
pt.show()
```

Giving



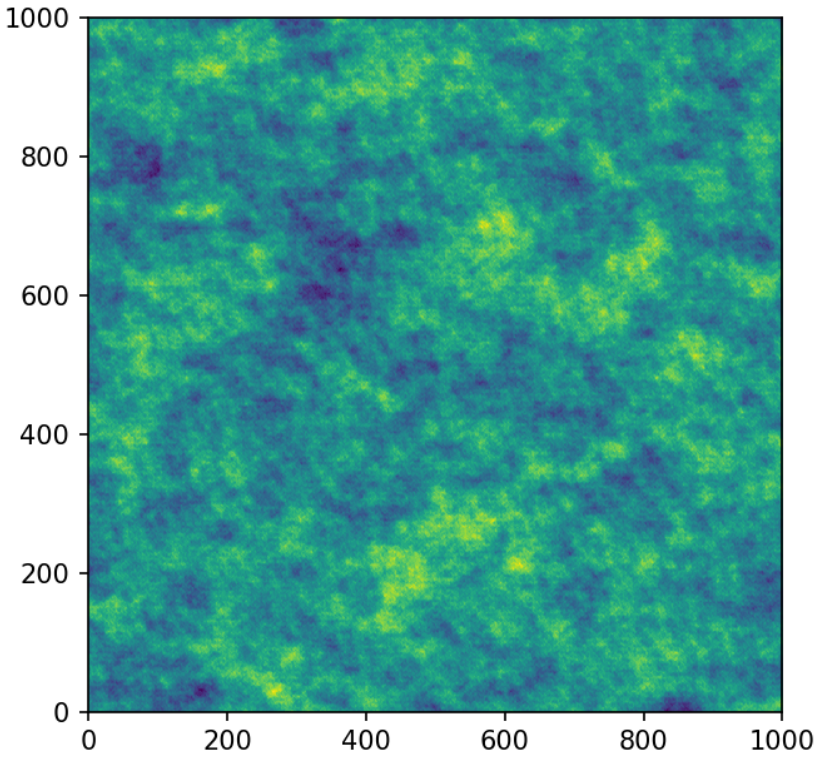
The plot might be a bit cluttered, but at least it is pretty obvious that the Herten aquifer has no apparent anisotropies in its spatial structure.

Creating a Spatial Random Field from the Herten Parameters

With all the hard work done, it's straight forward now, to generate new *Herten realisations*

```
from gstools import SRF

srf = SRF(fit_model, seed=19770928)
new_herten = srf((x_s, y_s), mesh_type='structured')

pt.imshow(new_herten.T, origin='lower')
pt.show()
```

Yielding



That's pretty neat! Executing the code given on this site, will result in a lower resolution of the field, because we overwrote *x_s* and *y_s* for the directional variogram estimation. In the example script, this is not the case and you will get a high resolution field.

And Now for Some Cleanup

In case you want all the downloaded data and scripts to be deleted, use following commands

```
os.remove('data.zip')
os.remove('scripts.zip')
rmtree('Herten-analog')
rmtree('tools')
```

And in case you want to play around a little bit more with the data, you can comment out the function calls `download_herten()` and `download_scripts()`, after they where called at least once and also comment out the cleanup. This way, the data will not be downloaded with every script execution.

CHAPTER 3

GSTOOLS API

3.1 Purpose

GeoStatTools is a library providing geostatistical tools for random field generation and variogram estimation based on a list of provided or even user-defined covariance models.

The following functionalities are directly provided on module-level.

3.2 Subpackages

covmodel	GStools subpackage providing a set of handy covariance models.
field	GStools subpackage providing tools for spatial random fields.
variogram	GStools subpackage providing tools for estimating and fitting variograms.
random	GStools subpackage for random number generation.
tools	GStools subpackage providing miscellaneous tools.

3.3 Classes

Spatial Random Field

Class for random field generation

SRF(model[, mean, upscaling, generator])	A class to generate spatial random fields (SRF).

Covariance Base-Class

Class to construct user defined covariance models

CovModel([dim, var, len_scale, nugget, ...])	Base class for the GSTools covariance models

Covariance Models

Standard Covariance Models

Gaussian([dim, var, len_scale, nugget, ...])	The Gaussian covariance model
Exponential([dim, var, len_scale, nugget, ...])	The Exponential covariance model
Matern([dim, var, len_scale, nugget, anis, ...])	The Matérn covariance model
Rational([dim, var, len_scale, nugget, ...])	The rational quadratic covariance model
Stable([dim, var, len_scale, nugget, anis, ...])	The stable covariance model
Spherical([dim, var, len_scale, nugget, ...])	The Spherical covariance model
Linear([dim, var, len_scale, nugget, anis, ...])	The bounded linear covariance model
MaternRescal([dim, var, len_scale, nugget, ...])	The rescaled Matérn covariance model
SphericalRescal([dim, var, len_scale, ...])	The rescaled Spherical covariance model

Truncated Power Law Covariance Models

TPLGaussian([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Gaussian modes
TPLExponential([dim, var, len_scale, ...])	Truncated-Power-Law with Exponential modes
TPLStable([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Stable modes

3.4 Functions

VTK-Export

Routines to export fields to the vtk format

vtk_export(filename, pos, field[, ...])	Export a field to vtk
vtk_export_structured(filename, pos, field)	Export a field to vtk structured rectilinear grid file
vtk_export_unstructured(filename, pos, field)	Export a field to vtk structured rectilinear grid file

variogram estimation

Estimate the variogram of a given field

vario_estimate_structured(field[, direction])	Estimates the variogram on a regular grid.
vario_estimate_unstructured(pos, field, ...)	Estimates the variogram on a unstructured grid.

3.5 gstools.covmodel

GStools subpackage providing a set of handy covariance models.

Subpackages

base	GStools subpackage providing the base class for covariance models.
models	GStools subpackage providing different covariance models.
tpl_models	GStools subpackage providing truncated power law covariance models.
plot	GStools subpackage providing plotting routines for the covariance models.

Covariance Base-Class

Class to construct user defined covariance models

CovModel([dim, var, len_scale, nugget, ...])	Base class for the GSTools covariance models

Covariance Models

Standard Covariance Models

Gaussian([dim, var, len_scale, nugget, ...])	The Gaussian covariance model
Exponential([dim, var, len_scale, nugget, ...])	The Exponential covariance model
Matern([dim, var, len_scale, nugget, anis, ...])	The Matérn covariance model
Rational([dim, var, len_scale, nugget, ...])	The rational quadratic covariance model
Stable([dim, var, len_scale, nugget, anis, ...])	The stable covariance model
Spherical([dim, var, len_scale, nugget, ...])	The Spherical covariance model
Linear([dim, var, len_scale, nugget, anis, ...])	The bounded linear covariance model
MaternRescal([dim, var, len_scale, nugget, ...])	The rescaled Matérn covariance model
SphericalRescal([dim, var, len_scale, ...])	The rescaled Spherical covariance model

Truncated Power Law Covariance Models

TPLGaussian([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Gaussian modes
TPLExponential([dim, var, len_scale, ...])	Truncated-Power-Law with Exponential modes
TPLStable([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Stable modes

gstools.covmodel.base

GStools subpackage providing the base class for covariance models.

The following classes are provided

CovModel([dim, var, len_scale, nugget, ...])	Base class for the GSTools covariance models

class gstools.covmodel.base.**CovModel**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: object

Base class for the GSTools covariance models

Parameters

- **dim** (int, optional) – dimension of the model. Default: 3
- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: 0.0
- **integral_scale** (float or list or None, optional) – If given, len_scale will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: None
- **var_raw** (float or None, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, var will be ignored. (This is just for models that override *CovModel.var_factor*) Default: None
- **hankel_kw** (dict or None, optional) – Modify the init-arguments of hankel.SymmetricFourierTransform used for the spectrum calculation. Use with caution (Better: Don't!). None is equivalent to {"a": -1, "b": 1, "N": 1000, "h": 0.001}. Default: None

Examples

```
>>> from gstools import CovModel
>>> import numpy as np
>>> class Gau(CovModel):
...     def correlation(self, r):
...         return np.exp(-(r/self.len_scale)**2)
...
>>> model = Gau()
>>> model.spectrum(2)
0.00825830126008459
```

Attributes

angles numpy.ndarray: The rotation angles (in rad) of the model.

anis numpy.ndarray: The anisotropy factors of the model.

arg list of str: Names of all arguments

arg_bounds dict: Bounds for all parameters

dim int: The dimension of the model.

dist_func tuple of callable:

hankel_kw dict: Keywords for hankel.SymmetricFourierTransform

has_cdf bool: State if a cdf is defined by the user

has_ppf bool: State if a ppf is defined by the user

integral_scale float: The main integral scale of the model.

integral_scale_vec numpy.ndarray: The integral scales in each direction.

len_scale float: The main length scale of the model.

len_scale_bounds list: Bounds for the lenght scale

len_scale_vec numpy.ndarray: The length scales in each direction.

name str: The name of the CovModel class

nugget float: The nugget of the model.

nugget_bounds list: Bounds for the nugget

opt_arg list of str: Names of the optional arguments

opt_arg_bounds dict: Bounds for the optional arguments

sill float: The sill of the variogram.

var float: The variance of the model.

var_bounds list: Bounds for the variance

var_raw float: The raw variance of the model without factor

Methods

check_arg_bounds()	Here the arguments are checked to be within the given bounds
check_opt_arg()	Here you can run checks for the optional arguments
default_arg_bounds()	Here you can provide a dictionary with default boundaries for the standard arguments.
default_opt_arg()	Here you can provide a dictionary with default values for the optional arguments.
default_opt_arg_bounds()	Here you can provide a dictionary with default boundaries for the optional arguments.
fit_variogram(x_data, y_data, **para_deselect)	fit the isotropic variogram-model to given data
fix_dim()	Set a fix dimension for the model
ln_spectral_rad_pdf(r)	The log radial spectral density of the model depending on the dimension
percentile_scale([per])	calculate the distance, where the given percentile of the variance is reached by the variogram

Continued on next page

Table 13 – continued from previous page

set_arg_bounds(**kwargs)	Set bounds for the parameters of the model
spectral_density(k)	The spectral density of the covariance model.
spectral_rad_pdf(r)	The radial spectral density of the model depending on the dimension
spectrum(k)	The spectrum of the covariance model.
var_factor()	Optional factor for the variance

check_arg_bounds()
Here the arguments are checked to be within the given bounds

check_opt_arg()
Here you can run checks for the optional arguments

This is in addition to the bound-checks

Notes

- You can use this to raise a ValueError/warning
- Any return value will be ignored
- This method will only be run once, when the class is initialized

default_arg_bounds()
Here you can provide a dictionary with default boundaries for the standard arguments.

default_opt_arg()
Here you can provide a dictionary with default values for the optional arguments.

default_opt_arg_bounds()
Here you can provide a dictionary with default boundaries for the optional arguments.

fit_variogram(*x_data*, *y_data*, ***para_deselect*)
fit the isotropic variogram-model to given data

Parameters

- **x_data** (numpy.ndarray) – The radii of the meassured variogram.
- **y_data** (numpy.ndarray) – The messured variogram
- ****para_deselect** – You can deselect the parameters to be fitted, by setting them "False" as keywords. By default, all parameters are fitted.

Returns

- **fit_para** (dict) – Dictonary with the fitted parameter values
- **pcov** (numpy.ndarray) – The estimated covariance of *popt* from scipy.optimize.curve_fit

Notes

You can set the bounds for each parameter by accessing *CovModel.set_arg_bounds*.

The fitted parameters will be instantly set in the model.

fix_dim()
Set a fix dimension for the model

ln_spectral_rad_pdf(*r*)
The log radial spectral density of the model depending on the dimension

percentile_scale(*per=0.9*)
calculate the distance, where the given percentile of the variance is reached by the variogram

set_arg_bounds(***kwargs*)
Set bounds for the parameters of the model

Parameters ****kwargs** – Parameter name as keyword ("var", "len_scale", "nugget", <opt_arg>) and a list of 2 or 3 values as value:

- `[a, b]` or
- `[a, b, <type>]`

<type> is one of `"oo"`, `"cc"`, `"oc"` or `"co"` to define if the bounds are open ("o") or closed ("c").

spectral_density(k)
The spectral density of the covariance model.

This is given by:

$$\tilde{S}(k) = \frac{S(k)}{\sigma^2}$$

Where $S(k)$ is the spectrum of the covariance model.

Parameters **k** (`float`) – Radius of the phase: $k = \|\mathbf{k}\|$

spectral_rad_pdf(r)
The radial spectral density of the model depending on the dimension

spectrum(k)
The spectrum of the covariance model.

This is given by:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \int C(r) e^{ib\mathbf{k}\cdot\mathbf{r}} d^n\mathbf{r}$$

Internally, this is calculated by the hankel transformation:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \cdot \frac{(2\pi)^{n/2}}{(bk)^{n/2-1}} \int_0^\infty r^{n/2-1} f(r) J_{n/2-1}(bkr) r dr$$

Where $C(r)$ is the covariance function of the model.

Parameters **k** (`float`) – Radius of the phase: $k = \|\mathbf{k}\|$

var_factor()
Optional factor for the variance

angles
The rotation angles (in rad) of the model.

Type `numpy.ndarray`

anis
The anisotropy factors of the model.

Type `numpy.ndarray`

arg
Names of all arguments

Type `list` of `str`

arg_bounds
Bounds for all parameters

Notes

Keys are the opt-arg names and values are lists of 2 or 3 values:

- `[a, b]` or
- `[a, b, <type>]`

<type> is one of `"oo"`, `"cc"`, `"oc"` or `"co"` to define if the bounds are open ("o") or closed ("c").

Type `dict`

dim
The dimension of the model.

Type `int`

dist_func
pdf, cdf and ppf from the radial spectral density

Type `tuple` of `callable`

hankel_kw
Keywords for `hankel.SymmetricFourierTransform`

Type `dict`

has_cdf
State if a cdf is defined by the user

Type `bool`

has_ppf
State if a ppf is defined by the user

Type `bool`

integral_scale
The main integral scale of the model.

Raises `ValueError` – If integral scale is not setable.

Type `float`

integral_scale_vec
The integral scales in each direction.

Notes

This is calculated by:

- `integral_scale_vec[0] = integral_scale`
- `integral_scale_vec[1] = integral_scale*anis[0]`
- `integral_scale_vec[2] = integral_scale*anis[1]`

Type `numpy.ndarray`

len_scale
The main length scale of the model.

Type `float`

len_scale_bounds
Bounds for the lenght scale

Notes

Is a list of 2 or 3 values:

- `[a, b]` or
- `[a, b, <type>]`

<type> is one of `"oo"`, `"cc"`, `"oc"` or `"co"` to define if the bounds are open ("o") or closed ("c").

Type `list`

len_scale_vec
The length scales in each direction.

Notes

This is calculated by:

- `len_scale_vec[0] = len_scale`
- `len_scale_vec[1] = len_scale*anis[0]`
- `len_scale_vec[2] = len_scale*anis[1]`

Type `numpy.ndarray`

name
The name of the CovModel class

Type `str`

nugget
The nugget of the model.

Type `float`

nugget_bounds
Bounds for the nugget

Notes

Is a list of 2 or 3 values:

- `[a, b]` or
- `[a, b, <type>]`

<type> is one of `"oo"`, `"cc"`, `"oc"` or `"co"` to define if the bounds are open ("o") or closed ("c").

Type `list`

opt_arg
Names of the optional arguments

Type `list` of `str`

opt_arg_bounds
Bounds for the optional arguments

Notes

Keys are the opt-arg names and values are lists of 2 or 3 values:

- `[a, b]` or

- `[a, b, <type>]`

<type> is one of `"oo"`, `"cc"`, `"oc"` or `"co"` to define if the bounds are open ("o") or closed ("c").

Type `dict`

sill
The sill of the variogram.

Notes

This is calculated by:

- `sill = variance + nugget`

Type `float`

var
The variance of the model.

Type `float`

var_bounds
Bounds for the variance

Notes

Is a list of 2 or 3 values:

- `[a, b]` or
- `[a, b, <type>]`

<type> is one of `"oo"`, `"cc"`, `"oc"` or `"co"` to define if the bounds are open ("o") or closed ("c").

Type `list`

var_raw
The raw variance of the model without factor

(See. CovModel.var_factor)

Type `float`

gstools.covmodel.models

GStools subpackage providing different covariance models.

The following classes and functions are provided

Gaussian([dim, var, len_scale, nugget, ...])	The Gaussian covariance model
Exponential([dim, var, len_scale, nugget, ...])	The Exponential covariance model
Matern([dim, var, len_scale, nugget, anis, ...])	The Matérn covariance model
Rational([dim, var, len_scale, nugget, ...])	The rational quadratic covariance model
Stable([dim, var, len_scale, nugget, anis, ...])	The stable covariance model
Spherical([dim, var, len_scale, nugget, ...])	The Spherical covariance model
Linear([dim, var, len_scale, nugget, anis, ...])	The bounded linear covariance model
MaternRescal([dim, var, len_scale, nugget, ...])	The rescaled Matérn covariance model
SphericalRescal([dim, var, len_scale, ...])	The rescaled Spherical covariance model

class gstools.covmodel.models.**Gaussian**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

The Gaussian covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \exp\left(-\frac{\pi}{4}\cdot\left(\frac{r}{\ell}\right)^2\right)$$

Parameters

- **dim** (int, optional) – dimension of the model. Default: 3
- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: 0.0
- **integral_scale** (float or list or None, optional) – If given, len_scale will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: None
- **var_raw** (float or None, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, var will be ignored. (This is just for models that override *CovModel.var_factor*) Default: None

- **hankel_kw** (dict or None, optional) – Modify the init-arguments of hankel.SymmetricFourierTransform used for the spectrum calculation. Use with caution (Better: Don't!). None is equivalent to {"a": -1, "b": 1, "N": 1000, "h": 0.001}. Default: None

Methods

calc_integral_scale()	The integral scale of the gaussian model is the length scale
correlation(r)	Gaussian correlation function
covariance(r)	Covariance of the model
spectral_rad_cdf(r)	The cdf of the radial spectral density
spectral_rad_ppf(u)	The ppf of the radial spectral density
spectrum(k)	The spectrum of the covariance model.
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

calc_integral_scale()
The integral scale of the gaussian model is the length scale

correlation(*r*)
Gaussian correlation function

$$\mathrm{cor}(r) = \exp\left(-\frac{\pi}{4} \cdot \left(\frac{r}{\ell}\right)^2\right)$$

covariance(*r*)
Covariance of the model

Given by: $C\left(r\right) = \sigma^2 \cdot \mathrm{cor}\left(r\right)$

Where $\mathrm{cor}(r)$ is the correlation function.

spectral_rad_cdf(*r*)
The cdf of the radial spectral density

spectral_rad_ppf(*u*)
The ppf of the radial spectral density

spectrum(*k*)
The spectrum of the covariance model.

This is given by:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \int C(r) e^{ib\mathbf{k}\cdot\mathbf{r}} d^n\mathbf{r}$$

Internally, this is calculated by the hankel transformation:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \cdot \frac{(2\pi)^{n/2}}{(bk)^{n/2-1}} \int_0^\infty r^{n/2-1} f(r) J_{n/2-1}(bkr) r dr$$

Where $C(r)$ is the covariance function of the model.

Parameters **k** (float) – Radius of the phase: $k = \|\mathbf{k}\|$

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma\left(r\right) = \sigma^2 \cdot \left(1 - \mathrm{cor}\left(r\right)\right) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**Exponential**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

The Exponential covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \exp\left(-\frac{r}{\ell}\right)$$

Parameters

- **dim** (int, optional) – dimension of the model. Default: 3
- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: 0.0
- **integral_scale** (float or list or None, optional) – If given, len_scale will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: None
- **var_raw** (float or None, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, var will be ignored. (This is just for models that override *CovModel.var_factor*) Default: None
- **hankel_kw** (dict or None, optional) – Modify the init-arguments of hankel.SymmetricFourierTransform used for the spectrum calculation. Use with caution (Better: Don't!). None is equivalent to {"a": -1, "b": 1, "N": 1000, "h": 0.001}. Default: None

Methods

calc_integral_scale()	The integral scale of the exponential model is the length scale
correlation(r)	Exponential correlation function
covariance(r)	Covariance of the model
spectral_rad_cdf(r)	The cdf of the radial spectral density
spectral_rad_ppf(u)	The ppf of the radial spectral density
spectrum(k)	The spectrum of the covariance model.
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

calc_integral_scale()
The integral scale of the exponential model is the length scale

correlation(*r*)
Exponential correlation function

$$\mathrm{cor}(r) = \exp\left(-\frac{r}{\ell}\right)$$

covariance(*r*)
Covariance of the model

Given by: $C\left(r\right) = \sigma^2 \cdot \mathrm{cor}\left(r\right)$

Where $\mathrm{cor}(r)$ is the correlation function.

spectral_rad_cdf(*r*)
The cdf of the radial spectral density

spectral_rad_ppf(*u*)
The ppf of the radial spectral density

spectrum(*k*)
The spectrum of the covariance model.

This is given by:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \int C(r) e^{ib\mathbf{k}\cdot\mathbf{r}} d^n\mathbf{r}$$

Internally, this is calculated by the hankel transformation:

$$S(k) = \left(\frac{1}{2\pi}\right)^n \cdot \frac{(2\pi)^{n/2}}{(bk)^{n/2-1}} \int_0^\infty r^{n/2-1} f(r) J_{n/2-1}(bkr) r dr$$

Where $C(r)$ is the covariance function of the model.

Parameters **k** (float) – Radius of the phase: $k = \|\mathbf{k}\|$

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma\left(r\right) = \sigma^2 \cdot \left(1 - \mathrm{cor}\left(r\right)\right) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}\left(r\right) = 1 - \mathrm{cor}\left(r\right)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**Spherical**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

The Spherical covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \begin{cases} 1 - \frac{3}{2} \cdot \frac{r}{\ell} + \frac{1}{2} \cdot \left(\frac{r}{\ell}\right)^3 & r < \ell \\ 0 & r \geq \ell \end{cases}$$

Parameters

- **dim** (int, optional) – dimension of the model. Default: 3
- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: 0.0
- **integral_scale** (float or list or None, optional) – If given, len_scale will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: None
- **var_raw** (float or None, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, var will be ignored. (This is just for models that override *CovModel.var_factor*) Default: None
- **hankel_kw** (dict or None, optional) – Modify the init-arguments of hankel.SymmetricFourierTransform used for the spectrum calculation. Use with caution (Better: Don't!). None is equivalent to {"a": -1, "b": 1, "N": 1000, "h": 0.001}. Default: None

Methods

correlation(r)	Spherical correlation function
covariance(r)	Covariance of the model
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

correlation(*r*)
Spherical correlation function

$$\mathrm{cor}(r) = \begin{cases} 1 - \frac{3}{2}\cdot\frac{r}{\ell} + \frac{1}{2}\cdot\left(\frac{r}{\ell}\right)^3 & r < \ell \\ 0 & r \geq \ell \end{cases}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**SphericalRescal**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

The rescaled Spherical covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \begin{cases} 1 - \frac{9}{16}\cdot\frac{r}{\ell} + \frac{27}{1024}\cdot\left(\frac{r}{\ell}\right)^3 & r < \frac{8}{3}\ell \\ 0 & r \geq \frac{8}{3}\ell \end{cases}$$

Parameters

- **dim** (int, optional) – dimension of the model. Default: 3
- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis

  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`

- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *CovModel.var_factor*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Methods

calc_integral_scale()	The integral scale of this spherical model is the length scale
correlation(r)	Rescaled Spherical correlation function
covariance(r)	Covariance of the model
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

calc_integral_scale()
The integral scale of this spherical model is the length scale

correlation(*r*)
Rescaled Spherical correlation function

$$\mathrm{cor}(r) = \begin{cases} 1 - \frac{9}{16} \cdot \frac{r}{\ell} + \frac{27}{1024} \cdot \left(\frac{r}{\ell}\right)^3 & r < \frac{8}{3}\ell \\ 0 & r \geq \frac{8}{3}\ell \end{cases}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**Rational**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)
Bases: *gstools.covmodel.base.CovModel*

The rational quadratic covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \left(1 + \frac{1}{2\alpha} \cdot \left(\frac{r}{\ell}\right)^2\right)^{-\alpha}$$

α is a shape parameter and should be > 0.5.

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **alpha** (`float`, optional) – Shape parameter. Standard range: `(0, inf)` Default: `1.0`

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`
- **var** (`float`, optional) – variance of the model (the nugget is not included in "this" variance) Default: `1.0`
- **len_scale** (`float` or `list`, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: `1.0`
- **nugget** (`float`, optional) – nugget of the model. Default: `0.0`
- **anis** (`float` or `list`, optional) – anisotropy ratios in the transversal directions [y, z]. Default: `1.0`
- **angles** (`float` or `list`, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`
- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *`CovModel.var_factor`* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *`CovModel.var_factor`*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Methods

`correlation`(r)	Rational correlation function
`covariance`(r)	Covariance of the model
`default_opt_arg`()	The defaults for the optional arguments:
`default_opt_arg_bounds`()	The defaults boundaries for the optional arguments:

Continued on next page

Table 19 – continued from previous page

variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

correlation(*r*)
Rational correlation function

$$\mathrm{cor}(r) = \left(1 + \frac{1}{2\alpha} \cdot \left(\frac{r}{\ell}\right)^2\right)^{-\alpha}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"alpha": 1.0}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"alpha": [0.5, inf]}`

Returns Boundaries for optional arguments

Return type `dict`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**Stable**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)
Bases: *gstools.covmodel.base.CovModel*

The stable covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \exp\left(-\left(\frac{r}{\ell}\right)^{\alpha}\right)$$

α is a shape parameter with $\alpha \in (0, 2]$

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **alpha** (`float`, optional) – Shape parameter. Standard range: `(0, 2]` Default: `1.5`

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`
- **var** (`float`, optional) – variance of the model (the nugget is not included in "this" variance) Default: `1.0`
- **len_scale** (`float` or `list`, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: `1.0`
- **nugget** (`float`, optional) – nugget of the model. Default: `0.0`
- **anis** (`float` or `list`, optional) – anisotropy ratios in the transversal directions [y, z]. Default: `1.0`
- **angles** (`float` or `list`, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`
- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *`CovModel.var_factor`* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *`CovModel.var_factor`*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Methods

`check_opt_arg`()	Checks for the optional arguments
`correlation`(r)	Stable correlation function
`covariance`(r)	Covariance of the model
`default_opt_arg`()	The defaults for the optional arguments:
`default_opt_arg_bounds`()	The defaults boundaries for the optional arguments:
`variogram`(r)	Isotropic variogram of the model
`variogram_normed`(r)	Normalized variogram of the model

check_opt_arg()
Checks for the optional arguments

Warns alpha – If alpha is < 0.3, the model tends to a nugget model and gets numerically unstable.

correlation(*r*)

Stable correlation function

$$\mathrm{cor}(r) = \exp\left(-\left(\frac{r}{\ell}\right)^{\alpha}\right)$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"alpha": 1.5}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"alpha": [0, 2, "oc"]}`

Returns Boundaries for optional arguments

Return type `dict`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**Matern**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)
Bases: *gstools.covmodel.base.CovModel*

The Matérn covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \cdot \left(\sqrt{2\nu} \cdot \frac{r}{\ell}\right)^{\nu} \cdot \mathrm{K}_{\nu}\left(\sqrt{2\nu} \cdot \frac{r}{\ell}\right)$$

Where Γ is the gamma function and K_{ν} is the modified Bessel function of the second kind.

ν is a shape parameter and should be >= 0.5.

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **nu** (`float`, optional) – Shape parameter. Standard range: `[0.5, 60]` Default: `1.0`

Parameters

- **dim** (int, optional) – dimension of the model. Default: 3
- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: 0.0
- **integral_scale** (float or list or None, optional) – If given, len_scale will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: None
- **var_raw** (float or None, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, var will be ignored. (This is just for models that override *CovModel.var_factor*) Default: None
- **hankel_kw** (dict or None, optional) – Modify the init-arguments of hankel.SymmetricFourierTransform used for the spectrum calculation. Use with caution (Better: Don't!). None is equivalent to {"a": -1, "b": 1, "N": 1000, "h": 0.001}. Default: None

Methods

check_opt_arg()	Checks for the optional arguments
correlation(r)	Matérn correlation function
covariance(r)	Covariance of the model
default_opt_arg()	The defaults for the optional arguments:
default_opt_arg_bounds()	The defaults boundaries for the optional arguments:
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

check_opt_arg()
Checks for the optional arguments

Warns **nu** – If nu is > 50, the model gets numerically unstable.

correlation(*r*)
Matérn correlation function

$$\mathrm{cor}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \cdot \left(\sqrt{2\nu} \cdot \frac{r}{\ell}\right)^{\nu} \cdot \mathrm{K}_{\nu}\left(\sqrt{2\nu} \cdot \frac{r}{\ell}\right)$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"nu": 1.0}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"nu": [0.5, 60.0, "cc"]}`

Returns Boundaries for optional arguments

Return type `dict`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**MaternRescal**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

The rescaled Matérn covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \cdot \left(\frac{\pi}{B\left(\nu, \frac{1}{2}\right)} \cdot \frac{r}{\ell} \right)^{\nu} \cdot \mathrm{K}_{\nu} \left(\frac{\pi}{B\left(\nu, \frac{1}{2}\right)} \cdot \frac{r}{\ell} \right)$$

Where Γ is the gamma function, K_{ν} is the modified Bessel function of the second kind and B is the Euler beta function.

ν is a shape parameter and should be > 0.5.

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **nu** (`float`, optional) – Shape parameter. Standard range: `[0.5, 60]` Default: `1.0`

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`

- **var** (float, optional) – variance of the model (the nugget is not included in "this" variance) Default: 1.0
- **len_scale** (float or list, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: 1.0
- **nugget** (float, optional) – nugget of the model. Default: 0.0
- **anis** (float or list, optional) – anisotropy ratios in the transversal directions [y, z]. Default: 1.0
- **angles** (float or list, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: 0.0
- **integral_scale** (float or list or None, optional) – If given, len_scale will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: None
- **var_raw** (float or None, optional) – raw variance of the model which will be multiplied with *CovModel.var_factor* to result in the actual variance. If given, var will be ignored. (This is just for models that override *CovModel.var_factor*) Default: None
- **hankel_kw** (dict or None, optional) – Modify the init-arguments of hankel.SymmetricFourierTransform used for the spectrum calculation. Use with caution (Better: Don't!). None is equivalent to {"a": -1, "b": 1, "N": 1000, "h": 0.001}. Default: None

Methods

check_opt_arg()	Checks for the optional arguments
correlation(r)	Rescaled Matérn correlation function
covariance(r)	Covariance of the model
default_opt_arg()	The defaults for the optional arguments:
default_opt_arg_bounds()	The defaults boundaries for the optional arguments:
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

check_opt_arg()
Checks for the optional arguments

Warns **nu** – If nu is > 50, the model gets numerically unstable.

correlation(*r*)
Rescaled Matérn correlation function

$$\mathrm{cor}(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \cdot \left(\frac{\pi}{B\left(\nu, \frac{1}{2}\right)} \cdot \frac{r}{\ell} \right)^{\nu} \cdot \mathrm{K}_{\nu} \left(\frac{\pi}{B\left(\nu, \frac{1}{2}\right)} \cdot \frac{r}{\ell} \right)$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"nu": 1.0}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"nu": [0.5, 60.0, "cc"]}`

Returns Boundaries for optional arguments

Return type `dict`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

class gstools.covmodel.models.**Linear**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)
Bases: *gstools.covmodel.base.CovModel*

The bounded linear covariance model

Notes

This model is given by the following correlation function:

$$\mathrm{cor}(r) = \begin{cases} 1 - \frac{r}{\ell} & r < \ell \\ 0 & r \geq \ell \end{cases}$$

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`
- **var** (`float`, optional) – variance of the model (the nugget is not included in "this" variance) Default: `1.0`
- **len_scale** (`float` or `list`, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: `1.0`
- **nugget** (`float`, optional) – nugget of the model. Default: `0.0`
- **anis** (`float` or `list`, optional) – anisotropy ratios in the transversal directions [y, z]. Default: `1.0`
- **angles** (`float` or `list`, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis

  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`

- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *`CovModel.var_factor`* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *`CovModel.var_factor`*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Methods

`correlation`(r)	Linear correlation function
`covariance`(r)	Covariance of the model
`variogram`(r)	Isotropic variogram of the model
`variogram_normed`(r)	Normalized variogram of the model

correlation(*r*)
Linear correlation function

$$\mathrm{cor}(r) = \begin{cases} 1 - \frac{r}{\ell} & r < \ell \\ 0 & r \geq \ell \end{cases}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

gstools.covmodel.tpl_models

GStools subpackage providing truncated power law covariance models.

The following classes and functions are provided

TPLGaussian([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Gaussian modes
TPLExponential([dim, var, len_scale, ...])	Truncated-Power-Law with Exponential modes
TPLStable([dim, var, len_scale, nugget, ...])	Truncated-Power-Law with Stable modes

class gstools.covmodel.tpl_models.**TPLGaussian**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

Truncated-Power-Law with Gaussian modes

Notes

The truncated power law is given by a superposition of scale-dependent variograms:

$$\gamma_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}(r) = \int_{\ell_{\mathrm{low}}}^{\ell_{\mathrm{up}}} \gamma(r,\lambda)\frac{\mathrm{d}\lambda}{\lambda}$$

with *Gaussian* modes on each scale:

$$\gamma(r,\lambda) = \sigma^2(\lambda)\cdot\left(1-\exp\left[-\left(\frac{r}{\lambda}\right)^2\right]\right)$$
$$\sigma^2(\lambda) = C\cdot\lambda^{2H}$$

This results in:

$$\gamma_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}(r) = \sigma^2_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}\cdot\left(1-H\cdot\frac{\ell_{\mathrm{up}}^{2H}\cdot E_{1+H}\left[\left(\frac{r}{\ell_{\mathrm{up}}}\right)^2\right]-\ell_{\mathrm{low}}^{2H}\cdot E_{1+H}\left[\left(\frac{r}{\ell_{\mathrm{low}}}\right)^2\right]}{\ell_{\mathrm{up}}^{2H}-\ell_{\mathrm{low}}^{2H}}\right)$$
$$\sigma^2_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}} = \frac{C\cdot\left(\ell_{\mathrm{up}}^{2H}-\ell_{\mathrm{low}}^{2H}\right)}{2H}$$

The "length scale" of this model is equivalent by the integration range:

$$\ell = \ell_{\mathrm{up}} - \ell_{\mathrm{low}}$$

If you want to define an upper scale truncation, you should set `len_low` and `len_scale` accordingly.

The following Parameters occure:

- $C > 0$: The scaling factor from the Power-Law. This parameter will be calculated internally by the given variance. You can access C directly by `model.var_raw`
- $0 < H < 1$: The hurst coefficient (`model.hurst`)
- $\ell_{\mathrm{low}} \geq 0$: The lower length scale truncation of the model (`model.len_low`)
- $\ell_{\mathrm{up}} \geq 0$: The upper length scale truncation of the model (`model.len_up`)

  This will be calculated internally by:

  - `len_up = len_low + len_scale`

That means, that the `len_scale` in this model actually represents the integration range for the truncated power law.

- $E_s(x)$ is the exponential integral.

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **hurst** (`float`, optional) – Hurst coefficient of the power law. Standard range: `(0, 1)`. Default: `0.5`
- **len_low** (`float`, optional) – The lower length scale truncation of the model. Standard range: `[0, 1000]`. Default: `0.0`

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`
- **var** (`float`, optional) – variance of the model (the nugget is not included in "this" variance) Default: `1.0`
- **len_scale** (`float` or `list`, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: `1.0`
- **nugget** (`float`, optional) – nugget of the model. Default: `0.0`
- **anis** (`float` or `list`, optional) – anisotropy ratios in the transversal directions [y, z]. Default: `1.0`
- **angles** (`float` or `list`, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`
- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *`CovModel.var_factor`* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *`CovModel.var_factor`*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Attributes

`len_up` `float`: The upper length scale truncation of the model.

Methods

`correlation`(r)	Truncated-Power-Law with Gaussian modes - correlation function
`covariance`(r)	Covariance of the model
`default_opt_arg`()	The defaults for the optional arguments:

Continued on next page

Table 25 – continued from previous page

default_opt_arg_bounds()	The defaults boundaries for the optional arguments:
var_factor()	Factor for C (Power-Law factor) to result in variance
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

correlation(*r*)
Truncated-Power-Law with Gaussian modes - correlation function

If `len_low=0` we have a simple representation:

$$\mathrm{cor}(r) = H \cdot E_{1+H}\left[\left(\frac{r}{\ell}\right)^2\right]$$

The general case:

$$\mathrm{cor}(r) = H \cdot \frac{\ell_{\mathrm{up}}^{2H} \cdot E_{1+H}\left[\left(\frac{r}{\ell_{\mathrm{up}}}\right)^2\right] - \ell_{\mathrm{low}}^{2H} \cdot E_{1+H}\left[\left(\frac{r}{\ell_{\mathrm{low}}}\right)^2\right]}{\ell_{\mathrm{up}}^{2H} - \ell_{\mathrm{low}}^{2H}}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"hurst": 0.5, "len_low": 0.0}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"hurst": [0, 1, "oo"], "len_low": [0, 1000, "cc"]}`

Returns Boundaries for optional arguments

Return type `dict`

var_factor()
Factor for C (Power-Law factor) to result in variance

This is used to result in the right variance, which is depending on the hurst coefficient and the length-scale extents

$$\frac{\ell_{\mathrm{up}}^{2H} - \ell_{\mathrm{low}}^{2H}}{2H}$$

Returns factor

Return type `float`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

len_up
The upper length scale truncation of the model.

- `len_up = len_low + len_scale`

Type `float`

class gstools.covmodel.tpl_models.**TPLExponential**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

Truncated-Power-Law with Exponential modes

Notes

The truncated power law is given by a superposition of scale-dependent variograms:

$$\gamma_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}(r) = \int_{\ell_{\mathrm{low}}}^{\ell_{\mathrm{up}}} \gamma(r,\lambda)\frac{\mathrm{d}\lambda}{\lambda}$$

with *Exponential* modes on each scale:

$$\gamma(r,\lambda) = \sigma^2(\lambda)\cdot\left(1-\exp\left[-\frac{r}{\lambda}\right]\right)$$
$$\sigma^2(\lambda) = C\cdot\lambda^{2H}$$

This results in:

$$\gamma_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}(r) = \sigma^2_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}\cdot\left(1-2H\cdot\frac{\ell_{\mathrm{up}}^{2H}\cdot E_{1+2H}\left[\frac{r}{\ell_{\mathrm{up}}}\right]-\ell_{\mathrm{low}}^{2H}\cdot E_{1+2H}\left[\frac{r}{\ell_{\mathrm{low}}}\right]}{\ell_{\mathrm{up}}^{2H}-\ell_{\mathrm{low}}^{2H}}\right)$$
$$\sigma^2_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}} = \frac{C\cdot\left(\ell_{\mathrm{up}}^{2H}-\ell_{\mathrm{low}}^{2H}\right)}{2H}$$

The "length scale" of this model is equivalent by the integration range:

$$\ell = \ell_{\mathrm{up}} - \ell_{\mathrm{low}}$$

If you want to define an upper scale truncation, you should set `len_low` and `len_scale` accordingly.

The following Parameters occure:

- $C > 0$: The scaling factor from the Power-Law. This parameter will be calculated internally by the given variance. You can access C directly by `model.var_raw`
- $0 < H < 1$: The hurst coefficient (`model.hurst`)
- $\ell_{\mathrm{low}} \geq 0$: The lower length scale truncation of the model (`model.len_low`)
- $\ell_{\mathrm{up}} \geq 0$: The upper length scale truncation of the model (`model.len_up`)

  This will be calculated internally by:

  - `len_up = len_low + len_scale`

That means, that the `len_scale` in this model actually represents the integration range for the truncated power law.

- $E_s(x)$ is the exponential integral.

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **hurst** (`float`, optional) – Hurst coefficient of the power law. Standard range: `(0, 1)`. Default: `0.5`
- **len_low** (`float`, optional) – The lower length scale truncation of the model. Standard range: `[0, 1000]`. Default: `0.0`

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`
- **var** (`float`, optional) – variance of the model (the nugget is not included in "this" variance) Default: `1.0`
- **len_scale** (`float` or `list`, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: `1.0`
- **nugget** (`float`, optional) – nugget of the model. Default: `0.0`
- **anis** (`float` or `list`, optional) – anisotropy ratios in the transversal directions [y, z]. Default: `1.0`
- **angles** (`float` or `list`, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`
- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *`CovModel.var_factor`* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *`CovModel.var_factor`*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Attributes

`len_up` `float`: The upper length scale truncation of the model.

Methods

`correlation`(r)	Truncated-Power-Law with Exponential modes - correlation function
`covariance`(r)	Covariance of the model
`default_opt_arg`()	The defaults for the optional arguments:

Continued on next page

Table 26 – continued from previous page

default_opt_arg_bounds()	The defaults boundaries for the optional arguments:
var_factor()	Factor for C (Power-Law factor) to result in variance
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

correlation(*r*)
Truncated-Power-Law with Exponential modes - correlation function

If `len_low=0` we have a simple representation:

$$\mathrm{cor}(r) = H \cdot E_{1+H}\left[\frac{r}{\ell}\right]$$

The general case:

$$\mathrm{cor}(r) = 2H \cdot \frac{\ell_{\mathrm{up}}^{2H} \cdot E_{1+2H}\left[\frac{r}{\ell_{\mathrm{up}}}\right] - \ell_{\mathrm{low}}^{2H} \cdot E_{1+2H}\left[\frac{r}{\ell_{\mathrm{low}}}\right]}{\ell_{\mathrm{up}}^{2H} - \ell_{\mathrm{low}}^{2H}}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"hurst": 0.5, "len_low": 0.0}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"hurst": [0, 1, "oo"], "len_low": [0, 1000, "cc"]}`

Returns Boundaries for optional arguments

Return type `dict`

var_factor()
Factor for C (Power-Law factor) to result in variance

This is used to result in the right variance, which is depending on the hurst coefficient and the length-scale extents

$$\frac{\ell_{\mathrm{up}}^{2H} - \ell_{\mathrm{low}}^{2H}}{2H}$$

Returns factor

Return type `float`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

len_up
The upper length scale truncation of the model.

- `len_up = len_low + len_scale`

Type `float`

class gstools.covmodel.tpl_models.**TPLStable**(*dim=3*, *var=1.0*, *len_scale=1.0*, *nugget=0.0*, *anis=1.0*, *angles=0.0*, *integral_scale=None*, *var_raw=None*, *hankel_kw=None*, ***opt_arg*)

Bases: *gstools.covmodel.base.CovModel*

Truncated-Power-Law with Stable modes

Notes

The truncated power law is given by a superposition of scale-dependent variograms:

$$\gamma_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}(r) = \int_{\ell_{\mathrm{low}}}^{\ell_{\mathrm{up}}} \gamma(r,\lambda)\frac{\mathrm{d}\lambda}{\lambda}$$

with *Stable* modes on each scale:

$$\gamma(r,\lambda) = \sigma^2(\lambda)\cdot\left(1-\exp\left[-\left(\frac{r}{\lambda}\right)^{\alpha}\right]\right)$$
$$\sigma^2(\lambda) = C\cdot\lambda^{2H}$$

This results in:

$$\gamma_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}(r) = \sigma^2_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}}\cdot\left(1-\frac{2H}{\alpha}\cdot\frac{\ell_{\mathrm{up}}^{2H}\cdot E_{1+\frac{2H}{\alpha}}\left[\left(\frac{r}{\ell_{\mathrm{up}}}\right)^{\alpha}\right]-\ell_{\mathrm{low}}^{2H}\cdot E_{1+\frac{2H}{\alpha}}\left[\left(\frac{r}{\ell_{\mathrm{low}}}\right)^{\alpha}\right]}{\ell_{\mathrm{up}}^{2H}-\ell_{\mathrm{low}}^{2H}}\right)$$
$$\sigma^2_{\ell_{\mathrm{low}},\ell_{\mathrm{up}}} = \frac{C\cdot\left(\ell_{\mathrm{up}}^{2H}-\ell_{\mathrm{low}}^{2H}\right)}{2H}$$

The "length scale" of this model is equivalent by the integration range:

$$\ell = \ell_{\mathrm{up}} - \ell_{\mathrm{low}}$$

If you want to define an upper scale truncation, you should set `len_low` and `len_scale` accordingly.

The following Parameters occure:

- $0 < \alpha \leq 2$: The shape parameter of the Stable model.
  - $\alpha = 1$: Exponential modes
  - $\alpha = 2$: Gaussian modes
- $C > 0$: The scaling factor from the Power-Law. This parameter will be calculated internally by the given variance. You can access C directly by `model.var_raw`
- $0 < H < 1$: The hurst coefficient (`model.hurst`)
- $\ell_{\mathrm{low}} \geq 0$: The lower length scale truncation of the model (`model.len_low`)
- $\ell_{\mathrm{up}} \geq 0$: The upper length scale truncation of the model (`model.len_up`)

  This will be calculated internally by:

  - `len_up = len_low + len_scale`

That means, that the `len_scale` in this model actually represents the integration range for the truncated power law.

- $E_s(x)$ is the exponential integral.

Other Parameters

- ****opt_arg** – The following parameters are covered by these keyword arguments
- **hurst** (`float`, optional) – Hurst coefficient of the power law. Standard range: `(0, 1)`. Default: `0.5`
- **alpha** (`float`, optional) – Shape parameter of the stable model. Standard range: `(0, 2]`. Default: `1.5`
- **len_low** (`float`, optional) – The lower length scale truncation of the model. Standard range: `[0, 1000]`. Default: `0.0`

Parameters

- **dim** (`int`, optional) – dimension of the model. Default: `3`
- **var** (`float`, optional) – variance of the model (the nugget is not included in "this" variance) Default: `1.0`
- **len_scale** (`float` or `list`, optional) – length scale of the model. If a single value is given, the same length-scale will be used for every direction. If multiple values (for main and transversal directions) are given, *anis* will be recalculated accordingly. Default: `1.0`
- **nugget** (`float`, optional) – nugget of the model. Default: `0.0`
- **anis** (`float` or `list`, optional) – anisotropy ratios in the transversal directions [y, z]. Default: `1.0`
- **angles** (`float` or `list`, optional) – angles of rotation:
  - in 2D: given as rotation around z-axis
  - in 3D: given by yaw, pitch, and roll (known as Tait–Bryan angles)

  Default: `0.0`
- **integral_scale** (`float` or `list` or `None`, optional) – If given, `len_scale` will be ignored and recalculated, so that the integral scale of the model matches the given one. Default: `None`
- **var_raw** (`float` or `None`, optional) – raw variance of the model which will be multiplied with *`CovModel.var_factor`* to result in the actual variance. If given, `var` will be ignored. (This is just for models that override *`CovModel.var_factor`*) Default: `None`
- **hankel_kw** (`dict` or `None`, optional) – Modify the init-arguments of `hankel.SymmetricFourierTransform` used for the spectrum calculation. Use with caution (Better: Don't!). `None` is equivalent to `{"a": -1, "b": 1, "N": 1000, "h": 0.001}`. Default: `None`

Attributes

`len_up` `float`: The upper length scale truncation of the model.

Methods

check_opt_arg()	Checks for the optional arguments
correlation(r)	Truncated-Power-Law with Stable modes - correlation function
covariance(r)	Covariance of the model
default_opt_arg()	The defaults for the optional arguments:
default_opt_arg_bounds()	The defaults boundaries for the optional arguments:
var_factor()	Factor for C (Power-Law factor) to result in variance
variogram(r)	Isotropic variogram of the model
variogram_normed(r)	Normalized variogram of the model

check_opt_arg()
Checks for the optional arguments

Warns **alpha** – If alpha is < 0.3, the model tends to a nugget model and gets numerically unstable.

correlation(*r*)
Truncated-Power-Law with Stable modes - correlation function

If `len_low=0` we have a simple representation:

$$\mathrm{cor}(r) = \frac{2H}{\alpha} \cdot E_{1+\frac{2H}{\alpha}}\left[\left(\frac{r}{\ell}\right)^{\alpha}\right]$$

The general case:

$$\mathrm{cor}(r) = \frac{2H}{\alpha} \cdot \frac{\ell_{\mathrm{up}}^{2H} \cdot E_{1+\frac{2H}{\alpha}}\left[\left(\frac{r}{\ell_{\mathrm{up}}}\right)^{\alpha}\right] - \ell_{\mathrm{low}}^{2H} \cdot E_{1+\frac{2H}{\alpha}}\left[\left(\frac{r}{\ell_{\mathrm{low}}}\right)^{\alpha}\right]}{\ell_{\mathrm{up}}^{2H} - \ell_{\mathrm{low}}^{2H}}$$

covariance(*r*)
Covariance of the model

Given by: $C(r) = \sigma^2 \cdot \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

default_opt_arg()
The defaults for the optional arguments:

- `{"hurst": 0.5, "alpha": 1.5, "len_low": 0.0}`

Returns Defaults for optional arguments

Return type `dict`

default_opt_arg_bounds()
The defaults boundaries for the optional arguments:

- `{"hurst": [0, 1, "oo"], "alpha": [0, 2, "oc"], "len_low": [0, 1000, "cc"]}`

Returns Boundaries for optional arguments

Return type `dict`

var_factor()
Factor for C (Power-Law factor) to result in variance

This is used to result in the right variance, which is depending on the hurst coefficient and the length-scale extents

$$\frac{\ell_{\mathrm{up}}^{2H} - \ell_{\mathrm{low}}^{2H}}{2H}$$

Returns factor

Return type `float`

variogram(*r*)
Isotropic variogram of the model

Given by: $\gamma(r) = \sigma^2 \cdot (1 - \mathrm{cor}(r)) + n$

Where $\mathrm{cor}(r)$ is the correlation function.

variogram_normed(*r*)
Normalized variogram of the model

Given by: $\tilde{\gamma}(r) = 1 - \mathrm{cor}(r)$

Where $\mathrm{cor}(r)$ is the correlation function.

len_up
The upper length scale truncation of the model.

- `len_up = len_low + len_scale`

Type `float`

gstools.covmodel.plot

GStools subpackage providing plotting routines for the covariance models.

The following classes and functions are provided

plot_variogram(model[, x_min, x_max])	plot variogram of a given CovModel
plot_variogram_normed(model[, x_min, x_max])	plot normalized variogram of a given CovModel
plot_covariance(model[, x_min, x_max])	plot covariance of a given CovModel
plot_correlation(model[, x_min, x_max])	plot correlation function of a given CovModel
plot_spectrum(model[, x_min, x_max])	plot specturm of a given CovModel
plot_spectral_density(model[, x_min, x_max])	plot spectral density of a given CovModel
plot_spectral_rad_pdf(model[, x_min, x_max])	plot radial spectral propability density function of a given CovModel

gstools.covmodel.plot.**plot_variogram**(*model*, *x_min=0.0*, *x_max=None*)
plot variogram of a given CovModel

gstools.covmodel.plot.**plot_variogram_normed**(*model*, *x_min=0.0*, *x_max=None*)
plot normalized variogram of a given CovModel

gstools.covmodel.plot.**plot_covariance**(*model*, *x_min=0.0*, *x_max=None*)
plot covariance of a given CovModel

gstools.covmodel.plot.**plot_correlation**(*model*, *x_min=0.0*, *x_max=None*)
plot correlation function of a given CovModel

gstools.covmodel.plot.**plot_spectrum**(*model*, *x_min=0.0*, *x_max=None*)
plot specturm of a given CovModel

gstools.covmodel.plot.**plot_spectral_density**(*model*, *x_min=0.0*, *x_max=None*)
plot spectral density of a given CovModel

gstools.covmodel.plot.**plot_spectral_rad_pdf**(*model*, *x_min=0.0*, *x_max=None*)
plot radial spectral propability density function of a given CovModel

3.6 gstools.field

GStools subpackage providing tools for spatial random fields.

Subpackages

generator	GStools subpackage providing generators for spatial random fields.
upscaling	GStools subpackage providing upscaling routines for the spatial random field.

Spatial Random Field

SRF(model[, mean, upscaling, generator])	A class to generate spatial random fields (SRF).

class gstools.field.**SRF**(*model*, *mean=0.0*, *upscaling='no_scaling'*, *generator='RandMeth'*, ***generator_kwargs*)

A class to generate spatial random fields (SRF).

Parameters

- **model** (*CovModel*) – Covariance Model to use for the field.
- **mean** (float, optional) – mean value of the SRF
- **var_upscaling** (str, optional) – Method to be used for upscaling the variance at each point depending on the related element volume. See the point_volumes keyword in the *SRF.__call__* routine. At the moment, the following upscaling methods are provided:
  - "no_scaling" : No upscaling is applied to the variance. See: *var_no_scaling*
  - "coarse_graining" : A volume depended variance is calculated by the upscaling technique coarse graining. See: *var_coarse_graining*

  Default: "no_scaling"
- **generator** (str, optional) – Name of the generator to use for field generation. At the moment, the following generators are provided:
  - "RandMeth" : The Randomization Methode. See: *RandMeth*

  Default: "RandMeth"
- ****generator_kwargs** – keyword arguments that are forwarded to the generator in use. Have a look at the provided generators for further information.

Attributes

do_rotation bool: State if a rotation should be performed

generator callable: The generator of the field.

model *CovModel*: The covariance model of the spatial random field.

upscaling str: Name of the upscaling method for the variance at each

Methods

__call__(pos[, seed, force_moments, ...])	Generate the spatial random field.
set_generator(generator, **generator_kwargs)	Set the generator for the field
structured(*args, **kwargs)	Generate an SRF on a structured mesh
unstructured(*args, **kwargs)	Generate an SRF on an unstructured mesh
upscaling_func(*args, **kwargs)	The upscaling method applied to the field variance

__call__(*pos*, *seed=nan*, *force_moments=False*, *point_volumes=0.0*, *mesh_type='unstructured'*)
Generate the spatial random field.

Parameters

- **pos** (list) – the position tuple, containing main direction and transversal directions
- **seed** (int, optional) – seed for RNG for reseting. Default: keep seed from generator
- **force_moments** (bool) – Force the generator to exactly match mean and variance. Default: False
- **point_volumes** (float or numpy.ndarray) – If your evaluation points for the field are coming from a mesh, they are probably representing a certain element volume. This volumes can be passed by *point_volumes* to apply the given variance upscaling. If *point_volumes* is 0 nothing is changed. Default: 0
- **mesh_type** (str) – 'structured' / 'unstructured'

Returns **field** – the SRF

Return type numpy.ndarray

set_generator(*generator*, ***generator_kwargs*)
Set the generator for the field

Parameters

- **generator** (str, optional) – Name of the generator to use for field generation. Default: "RandMeth"
- ****generator_kwargs** – keyword arguments that are forwarded to the generator in use.

structured(**args*, ***kwargs*)
Generate an SRF on a structured mesh

See *SRF.__call__*

unstructured(**args*, ***kwargs*)
Generate an SRF on an unstructured mesh

See *SRF.__call__*

upscaling_func(**args*, ***kwargs*)
The upscaling method applied to the field variance

do_rotation
State if a rotation should be performed depending on the model.

Type bool

generator
The generator of the field.

Default: *RandMeth*

Type callable

model
The covariance model of the spatial random field.

Type *CovModel*

upscaling
Name of the upscaling method for the variance at each point depending on the related element volume.

See the `point_volumes` keyword in the *SRF.__call__* routine. Default: "no_scaling"

Type `str`

gstools.field.generator

GStools subpackage providing generators for spatial random fields.

The following classes are provided

RandMeth(model[, mode_no, seed, ...])	Randomization method for calculating isotropic spatial random fields.

class gstools.field.generator.**RandMeth**(*model*, *mode_no=1000*, *seed=None*, *chunk_tmp_size=10000000.0*, *verbose=False*, ***kwargs*)

Randomization method for calculating isotropic spatial random fields.

Parameters

- **model** (*CovModel*) – covariance model
- **mode_no** (int, optional) – number of Fourier modes. Default: 1000
- **seed** (int or None, optional) – the seed of the random number generator. If "None", a random seed is used. Default: None
- **chunk_tmp_size** (int, optional) – Number of points (number of coordinates * mode_no) to be handled by one chunk while creating the fild. This is used to prevent memory overflows while generating the field. Default: 1e7
- **verbose** (bool, optional) – State if there should be output during the generation. Default: False
- ****kwargs** – Placeholder for keyword-args

Notes

The Randomization method is used to generate isotropic spatial random fields characterized by a given covariance model. The calculation looks like:

$$u(x) = \sqrt{\frac{\sigma^2}{N}} \cdot \sum_{i=1}^{N} \left(Z_{1,i} \cdot \cos\left(\langle k_i, x\rangle\right) + Z_{2,i} \cdot \sin\left(\langle k_i, x\rangle\right)\right)$$

where:

- N : fourier mode number
- $Z_{j,i}$: random samples from a normal distribution
- k_i : samples from the spectral density distribution of the covariance model

Attributes

chunk_tmp_size int: temporary chunk size

mode_no int: The number of modes in the randomization method.

model *CovModel*: The covariance model of the spatial random field.

name str: The name of the generator

seed int: the seed of the master RNG

verbose bool: verbosity of the generator

Methods

__call__(x[, y, z])	Calculates the random modes for the randomization method.
reset_seed([seed])	Recalculate the random amplitudes and wave numbers with the given seed.
update([model, seed])	Update the model and the seed.

__call__ (*x*, *y=None*, *z=None*)
Calculates the random modes for the randomization method.

Parameters

- **x** (float, numpy.ndarray) – the x components of the position tuple, the shape has to be (len(x), 1, 1) for 3d and accordingly shorter for lower dimensions
- **y** (float, numpy.ndarray, optional) – the y components of the pos. tupls
- **z** (float, numpy.ndarray, optional) – the z components of the pos. tuple

Returns the random modes

Return type numpy.ndarray

reset_seed (*seed=nan*)
Recalculate the random amplitudes and wave numbers with the given seed.

Parameters **seed** (int or None or numpy.nan, optional) – the seed of the random number generator. If None, a random seed is used. If numpy.nan, the actual seed will be kept. Default: numpy.nan

---

Notes

Even if the given seed is the present one, modes will be racalculated.

---

update (*model=None*, *seed=nan*)
Update the model and the seed.

If model and seed are not different, nothing will be done.

Parameters

- **model** (*CovModel* or None, optional) – covariance model. Default: None
- **seed** (int or None or numpy.nan, optional) – the seed of the random number generator. If None, a random seed is used. If numpy.nan, the actual seed will be kept. Default: numpy.nan

chunk_tmp_size
temporary chunk size

Type int

mode_no
The number of modes in the randomization method.

Type int

model
The covariance model of the spatial random field.

Type *CovModel*

name
The name of the generator

Type str

seed
the seed of the master RNG

Notes

If a new seed is given, the setter property not only saves the new seed, but also creates new random modes with the new seed.

Type int

verbose
verbosity of the generator

Type bool

gstools.field.upscaling

GStools subpackage providing upscaling routines for the spatial random field.

The following functions are provided

var_coarse_graining(model[, point_volumes])	Coarse Graning procedure to upscale the variance for uniform flow
var_no_scaling(model, *args, **kwargs)	Dummy function to bypass scaling

gstools.field.upscaling.**var_coarse_graining**(*model*, *point_volumes=0.0*)

Coarse Graning procedure to upscale the variance for uniform flow

Parameters

- **model** (*CovModel*) – Covariance Model used for the field.
- **point_volumes** (float or numpy.ndarray) – Volumes of the elements at the given points. Default: 0

Returns **scaled_var** – The upscaled variance

Return type float or numpy.ndarray

Notes

This procedure was presented in [Attinger03]. It applies the upscaling procedure 'Coarse Graining' to the Groundwater flow equation under uniform flow on a lognormal distributed conductivity field following a gaussian covariance function. A filter over a cube with a given edge-length λ is applied and an upscaled conductivity field is obtained. The upscaled field is again following a gaussian covariance function with scale dependent variance and length-scale:

$$\lambda = V^{\frac{1}{d}}$$

$$\sigma^2(\lambda) = \sigma^2 \cdot \left(\frac{\ell^2}{\ell^2 + \left(\frac{\lambda}{2}\right)^2} \right)^{\frac{d}{2}}$$

$$\ell(\lambda) = \left(\ell^2 + \left(\frac{\lambda}{2} \right)^2 \right)^{\frac{1}{2}}$$

Therby λ will be calculated from the given point_volumes V by assuming a cube with the given volume.

The upscaled length scale will be ignored by this routine.

References

gstools.field.upscaling.**var_no_scaling**(*model*, **args*, ***kwargs*)

Dummy function to bypass scaling

Parameters **model** (*CovModel*) – Covariance Model used for the field.

Returns **var** – The model variance.

Return type float

3.7 gstools.variogram

GStools subpackage providing tools for estimating and fitting variograms.

Variogram estimation

vario_estimate_unstructured(pos, field, ...)	Estimates the variogram on a unstructured grid.
vario_estimate_structured(field[, direction])	Estimates the variogram on a regular grid.

gstools.variogram.**vario_estimate_unstructured**(*pos*, *field*, *bin_edges*, *sampling_size=None*, *sampling_seed=None*)

Estimates the variogram on a unstructured grid.

The algorithm calculates following equation:

$$\gamma(r_k) = \frac{1}{2N}\sum_{i=1}^{N}(z(\mathbf{x}_i) - z(\mathbf{x}_i'))^2, \text{ with}$$
$$r_k \leq \|\mathbf{x}_i - \mathbf{x}_i'\| < r_{k+1}$$

Notes

Internally uses double precision and also returns doubles.

Parameters

- **pos** (list) – the position tuple, containing main direction and transversal directions
- **field** (numpy.ndarray) – the spatially distributed data
- **bin_edges** (numpy.ndarray) – the bins on which the variogram will be calculated
- **sampling_size** (int or None, optional) – for large input data, this method can take a long time to compute the variogram, therefore this argument specifies the number of data points to sample randomly Default: None
- **sampling_seed** (int or None, optional) – seed for samples if sampling_size is given. Default: None

Returns the estimated variogram and the bin centers

Return type tuple of numpy.ndarray

gstools.variogram.**vario_estimate_structured**(*field*, *direction='x'*)

Estimates the variogram on a regular grid.

The indices of the given direction are used for the bins. The algorithm calculates following equation:

$$\gamma(r_k) = \frac{1}{2N}\sum_{i=1}^{N}(z(\mathbf{x}_i) - z(\mathbf{x}_i'))^2, \text{ with}$$
$$r_k \leq \|\mathbf{x}_i - \mathbf{x}_i'\| < r_{k+1}$$

Warning: It is assumed that the field is defined on an equidistant Cartesian grid.

Notes

Internally uses double precision and also returns doubles.

Parameters

- **field** (numpy.ndarray) – the spatially distributed data
- **direction** (str) – the axis over which the variogram will be estimated (x, y, z)

Returns the estimated variogram along the given direction.

Return type numpy.ndarray

3.8 gstools.random

GStools subpackage for random number generation.

Random Number Generator

RNG([seed])	A random number generator for different distributions and multiple streams.

Seed Generator

MasterRNG(seed)	Master random number generator for generating seeds.

Distribution factory

dist_gen([pdf_in, cdf_in, ppf_in])	Distribution Factory

class gstools.random.**RNG**(*seed=None*)

A random number generator for different distributions and multiple streams.

Parameters **seed** (int or None, optional) – The seed of the master RNG, if None, a random seed is used. Default: None

Attributes

random numpy.random.RandomState:

seed int: the seed of the master RNG

Methods

sample_dist([pdf, cdf, ppf, size])	Sample from a distribution given by pdf, cdf and/or ppf
sample_ln_pdf(ln_pdf[, size, sample_around, ...])	Sample from a distribution given by ln(pdf)
sample_sphere(dim[, size])	Uniform sampling on a d-dimensional sphere

sample_dist(*pdf=None*, *cdf=None*, *ppf=None*, *size=None*, ***kwargs*)

Sample from a distribution given by pdf, cdf and/or ppf

Parameters

- **pdf** (callable or None, optional) – Probability density function of the given distribution, that takes a single argument Default: None
- **cdf** (callable or None, optional) – Cumulative distribution function of the given distribution, that takes a single argument Default: None
- **ppf** (callable or None, optional) – Percent point function of the given distribution, that takes a single argument Default: None
- **size** (int or None, optional) – sample size. Default: None

- ****kwargs** – Keyword-arguments that are forwarded to scipy.stats.rv_continuous.

Returns **samples** – the samples from the given distribution

Return type float or numpy.ndarray

Notes

At least pdf or cdf needs to be given.

sample_ln_pdf(*ln_pdf*, *size=None*, *sample_around=1.0*, *nwalkers=50*, *burn_in=20*, *oversampling_factor=10*)

Sample from a distribution given by ln(pdf)

This algorithm uses the emcee.EnsembleSampler

Parameters

- **ln_pdf** (callable) – The logarithm of the Probability density function of the given distribution, that takes a single argument
- **size** (int or None, optional) – sample size. Default: None
- **sample_around** (float, optional) – Starting point for initial guess Default: 1.
- **nwalkers** (int, optional) – The number of walkers in the mcmc sampler. Used for the emcee.EnsembleSampler class. Default: 100
- **burn_in** (int, optional) – Number of burn-in runs in the mcmc algorithm. Default: 100
- **oversampling_factor** (int, optional) – To guess the sample number needed for proper results, we use a factor for oversampling. The intern used sample-size is calculated by

  ```
  sample_size = max(burn_in, (size/nwalkers)*oversampling_factor)
  ```

  So at least, as much as the burn-in runs. Default: 10

sample_sphere(*dim*, *size=None*)

Uniform sampling on a d-dimensional sphere

Parameters

- **dim** (int) – Dimension of the sphere. Just 1, 2, and 3 supported.
- **size** (int, optional) – sample size

Returns **coord** – x[, y[, z]] coordinates on the sphere with shape (dim, size)

Return type numpy.ndarray

random

Get a stream to the numpy Random number generator

You can use this, to call any provided distribution from numpy.random.RandomState.

Type numpy.random.RandomState

seed

the seed of the master RNG

The setter property not only saves the new seed, but also creates a new master RNG function with the new seed.

Type int

class gstools.random.**MasterRNG**(*seed*)

Master random number generator for generating seeds.

Parameters **seed** (int or None, optional) – The seed of the master RNG, if None, a random seed is used. Default: None

Attributes

seed int: the seed of the master RNG

Methods

__call__()	Returns a random seed.

__call__()
Returns a random seed.

seed
the seed of the master RNG

The setter property not only saves the new seed, but also creates a new master RNG function with the new seed.

Type int

gstools.random.**dist_gen**(*pdf_in=None*, *cdf_in=None*, *ppf_in=None*, ***kwargs*)
Distribution Factory

Parameters

- **pdf_in** (callable or None, optional) – Proprobability distribution function of the given distribution, that takes a single argument Default: None
- **cdf_in** (callable or None, optional) – Cumulative distribution function of the given distribution, that takes a single argument Default: None
- **ppf_in** (callable or None, optional) – Percent point function of the given distribution, that takes a single argument Default: None
- ****kwargs** – Keyword-arguments forwarded to scipy.stats.rv_continuous.

Returns **dist** – The constructed distribution.

Return type scipy.stats.rv_continuous

Notes

At least pdf or cdf needs to be given.

3.9 gstools.tools

GStools subpackage providing miscellaneous tools.

Export

vtk_export(filename, pos, field[, ...])	Export a field to vtk
vtk_export_structured(filename, pos, field)	Export a field to vtk structured rectilinear grid file
vtk_export_unstructured(filename, pos, field)	Export a field to vtk structured rectilinear grid file

Special functions

inc_gamma(s, x)	The (upper) incomplete gamma function
exp_int(s, x)	The exponential integral $E_s(x)$
inc_beta(a, b, x)	The incomplete Beta function
stable_cov_norm(r, len_scale, hurst, alpha)	The normalized covariance function of the stable model

Geometric

xyz2pos(x[, y, z, dtype])	Convert postional arguments to x, y, z
pos2xyz(pos[, dtype, calc_dim])	Convert postional arguments to x, y, z
r3d_x(theta)	Rotation matrix about x axis.
r3d_y(theta)	Rotation matrix about y axis.
r3d_z(theta)	Rotation matrix about z axis.

gstools.tools.**vtk_export_structured**(*filename*, *pos*, *field*, *fieldname='field'*)
Export a field to vtk structured rectilinear grid file

Parameters

- **filename** (str) – Filename of the file to be saved, including the path. Note that an ending (*.vtr or *.vtu) will be added to the name.
- **pos** (list) – the position tuple, containing main direction and transversal directions
- **field** (numpy.ndarray) – Structured field to be saved. As returned by SRF.
- **fieldname** (str, optional) – Name of the field in the VTK file. Default: "field"

gstools.tools.**vtk_export_unstructured**(*filename*, *pos*, *field*, *fieldname='field'*)
Export a field to vtk structured rectilinear grid file

Parameters

- **filename** (str) – Filename of the file to be saved, including the path. Note that an ending (*.vtr or *.vtu) will be added to the name.
- **pos** (list) – the position tuple, containing main direction and transversal directions
- **field** (numpy.ndarray) – Unstructured field to be saved. As returned by SRF.
- **fieldname** (str, optional) – Name of the field in the VTK file. Default: "field"

gstools.tools.**vtk_export**(*filename*, *pos*, *field*, *fieldname='field'*, *mesh_type='unstructured'*)
Export a field to vtk

Parameters

- **filename** (str) – Filename of the file to be saved, including the path. Note that an ending (*.vtr or *.vtu) will be added to the name.
- **pos** (list) – the position tuple, containing main direction and transversal directions
- **field** (numpy.ndarray) – Unstructured field to be saved. As returned by SRF.
- **fieldname** (str, optional) – Name of the field in the VTK file. Default: "field"
- **mesh_type** (str, optional) – 'structured' / 'unstructured'. Default: structured

gstools.tools.**inc_gamma**(*s*, *x*)
The (upper) incomplete gamma function

Given by: $\Gamma(s,x) = \int_x^\infty t^{s-1}\, e^{-t}\, \mathrm{d}t$

Parameters

- **s** (float) – exponent in the integral
- **x** (numpy.ndarray) – input values

gstools.tools.**exp_int**(*s*, *x*)
The exponential integral $E_s(x)$

Given by: $E_s(x) = \int_1^\infty \frac{e^{-xt}}{t^s}\, \mathrm{d}t$

Parameters

- **s** (float) – exponent in the integral
- **x** (numpy.ndarray) – input values

gstools.tools.**inc_beta**(*a*, *b*, *x*)
The incomplete Beta function

Given by: $B(a,b;\, x) = \int_0^x t^{a-1}\, (1-t)^{b-1}\, dt$

Parameters

- **a** (float) – first exponent in the integral
- **b** (float) – second exponent in the integral
- **x** (numpy.ndarray) – input values

gstools.tools.**stable_cov_norm**(*r*, *len_scale*, *hurst*, *alpha*)
The normalized covariance function of the stable model

Given by

$$\tilde{C}(r) = \frac{2H}{\alpha} \cdot E_{1+\frac{2H}{\alpha}}\left(\left(\frac{r}{\ell}\right)^\alpha\right)$$

Parameters

- **r** (numpy.ndarray) – input values
- **len_scale** (float) – length-scale of the model.
- **hurst** (float) – Hurst coefficient of the power law.
- **alpha** (float, optional) – Shape parameter of the stable model.

gstools.tools.**xyz2pos**(*x*, *y=None*, *z=None*, *dtype=None*)
Convert postional arguments to x, y, z

Parameters

- **x** (numpy.ndarray) – grid axis in x-direction if structured, or first components of position vectors if unstructured

- **y** (numpy.ndarray, optional) – analog to x
- **z** (numpy.ndarray, optional) – analog to x
- **dtype** (*data-type, optional*) – The desired data-type for the array. If not given, then the type will be determined as the minimum type required to hold the objects in the sequence. Default: None

Returns **pos** – the position tuple

Return type numpy.ndarray

gstools.tools.**pos2xyz**(*pos*, *dtype=None*, *calc_dim=False*)
Convert postional arguments to x, y, z

Parameters

- **pos** (iterable) – the position tuple, containing main direction and transversal directions
- **dtype** (*data-type, optional*) – The desired data-type for the array. If not given, then the type will be determined as the minimum type required to hold the objects in the sequence. Default: None
- **calc_dim** (bool) – State if the dimension should be returned

Returns

- **x** (numpy.ndarray) – first components of position vectors
- **y** (numpy.ndarray or None) – analog to x
- **z** (numpy.ndarray or None) – analog to x
- **dim** (int, optional) – dimension (only if calc_dim is True)

Notes

If len(pos) > 3, everything after pos[2] will be ignored.

gstools.tools.**r3d_x**(*theta*)
Rotation matrix about x axis.

Parameters **theta** (float) – Rotation angle

Returns Rotation matrix.

Return type numpy.ndarray

gstools.tools.**r3d_y**(*theta*)
Rotation matrix about y axis.

Parameters **theta** (float) – Rotation angle

Returns Rotation matrix.

Return type numpy.ndarray

gstools.tools.**r3d_z**(*theta*)
Rotation matrix about z axis.

Parameters **theta** (float) – Rotation angle

Returns Rotation matrix.

Return type numpy.ndarray

BIBLIOGRAPHY

[Attinger03] Attinger, S. 2003, ''Generalized coarse graining procedures for flow in porous media", Computational Geosciences, 7(4), 253–273.

PYTHON MODULE INDEX

g

INDEX

Symbols

A

C

U

V

X