A Python package for simulating random fields

Khaya Mpehle

We introduce the python package `GaussRF`, a package concerned primarily with simulating random fields. The package started off as an implementation of the Karhunen-Loeve function expansion to simulate Gaussian fields in one-dimension and two-dimensional rectangles. The package is slowly expanding, with support for Poisson processes in two-dimensions newly added, and plans for yet more types of random fields currently being pursued. The original desire to create this package came from the author's interest in their applications in oceanography and geosciences. For example, statistical fluctuations of sea surface temperature, a problem of oceanography, can be modelled by Gaussian random fields, and fractures in rock can be modelled by cluster point processes, a problem of geology. This package's goal is to collect in one directory stochastic and statistical algorithms for commonly used random fields.

1 Package Overview

The package works by representing simulation of certain types of random fields as Python object classes. For example, the class `GaussF_KL1D` simulates Gaussian processes using the Karhunen-Loeve function expansion. The input parameters for this class are the expansion order `N`, the number of points `M`, the Covaraince function `Cov`, a method `method`, and a sample size `samples`. The method defaults to `"KL_EOLE"`, utilising the KL expansion with EOLE weights, and `samples = 1` so that only one realization is returned. This produces a random field object, which can then acted on by its various methods to do various things, such as produce its values as an array or produce the grid of points used in the simulation. For example, Brownian motion is simulated and then outputted as an array using the method `Gfield()`:

```
# Define Brownian motion
def Bm_cov(s, t):
    return np.minimum(s, t)
# Generate a Brownian motion simulation object

X_sim = GaussF_KL1D(N = 20, M = 100, a = 0., b = 1.,
                Cov = Bm_cov, method = "KL_EOLE", samples = 1)

# Get the Brownian motion simulation as an array
X = X_sim.Gfield()
```

Or, omitting explicit references to the argument variables

```
def Bm_cov(s, t):
    return np.minimum(s, t)
```

```
X_sim = GaussF_KL1D(20, 100, 0., 1., Bm_cov)
X = X_sim.Gfield()
t = X_sim.grid()
```

So far, there are three primary modules. These are `GaussRF`, `point_process` and `std_funcs`. The first module holds the simulation procedures for Gaussian random fields in one-dimension and two-dimensions. The second module holds the simulation procedure for inhomogeneous Poisson processes in two-dimensions, with plans for the module to support other kinds of point processes. The third module holds standard functions such as common covariance functions for Gaussian random fields.

2 examples

Here we present some examples of random fields calculated using `GaussRF`. The examples include convergence tests with random fields whose Karhunen-Loève expansions are known analytically, and realisations of random fields.

2.1 Convergence of the Nyström method

As a first example, we test the convergence of the Karhunen-Loéve expansion. For a random field $H(\boldsymbol{x}, \omega)$, approximated by the truncated series $\hat{H}(\boldsymbol{x}, \omega)$, the error measure[1] is

$$\varepsilon(\boldsymbol{x}) = \frac{\mathrm{Var}\left(H(\boldsymbol{x}, \omega) - \hat{H}(\boldsymbol{x}, \omega)\right)}{\mathrm{Var}(H(\boldsymbol{x}, \omega))}. \tag{1}$$

Since a Gaussian RF is determined by its covariance and mean, and we are presuming the mean has been removed, this error should converge pointwise as the approximation $\hat{H}$ becomes better. A global measure of the error variance is

$$\varepsilon = \int_{\Omega} \varepsilon(\boldsymbol{x})\, dV \tag{2}$$

If the variance of the function is a constant, the global square integrated error variance is

$$\varepsilon = 1 - \frac{1}{\Omega\sigma^2} \sum_{i=1}^{N} \lambda_i \tag{3}$$

For $d = 1, 2$, consider the Gaussian RF on the domain $D = [0, 1]^d$ with covariance function

$$\exp\left(-||\boldsymbol{x} - \boldsymbol{y}||_1\right). \tag{4}$$

This is one of the rare cases where the eigenvalues and eigenfunctions of the Karhunen-Loève expansion are known analytically [**?**]. In one dimension, the eigenvalues are

$$\lambda_k = \frac{2}{w_k^2 + 1}, \tag{5}$$

where w_k are the positive, increasing roots of the function

$$f(w) = (w^2 - 1)\tan w - 2w. \tag{6}$$

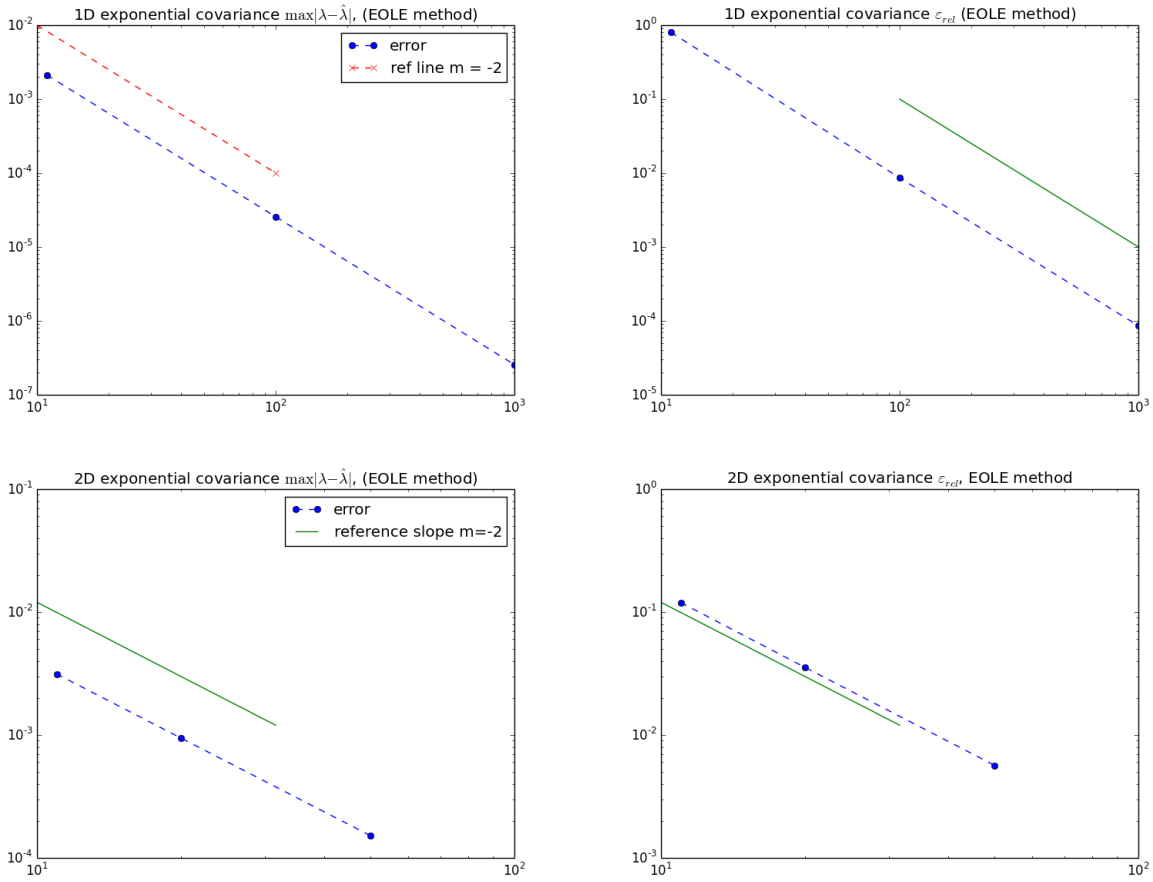

Figure 1: Top row: the maximum difference between the exact and numerical eigenvalues for the 1D exponential covariance on $[0, 1]$ (left) and the relative error variance between the approximate and exact truncated KL expansion (right). Bottom row: Similar to the top row, but in 2D rectangle $[0, 1]^2$.

In two dimensions, the eigenvalues and eigenfunctions are the simple tensor product of one-dimensional problem's spectral quantities. Thus the eigenvalues in two-dimensions can be indexed as

$$\lambda_{k_1\, k_2} = \lambda_{k_1}\lambda_{k_2} \tag{7}$$

for λ_{k_i} the k_ith eigenvalue of the one-dimensional exponential covariance. Using `GaussRF`, we can return the eigenvalues and -vectors from a simulation object using the method `eigens()`, and proceed to test convergence over a range of grid points. Supposing that we have an array `w_roots` of the roots $f(w)$, by using Newton's method for example, then a script for testing convergence would look like:

```
# 1- Dimensional convergence
def exp_cov(x, y):
    return np.exp(-np.abs(x - y))
N = 10
M_vals = [11, 100, 100] # 11 because M > N and N = 10
L_ref = 2. / ( 1 + w_roots**2)
errors = []
rel_evar = np.zeros(len(M_vals))
counter = 0
a, b = 0., 1.
err_var_ref = 1 - np.sum(L_ref) / (b - a)
for M in M_vals:
    phi, L = GaussF_KL1D(N, M, a, b, exp_cov).eigens()
    err_var_comp = 1 - np.sum(L_comp) / (b - a)
    rel_evar[counter] = np.abs( evar_comp - evar_ref) / evar_ref
    error.append(np.abs(L_ref - L_comp).max())
    counter += 1
```

Figure 1 shows log-log plots of the maximum eigenvalue error

$$\max |\lambda - \hat{\lambda}| \tag{8}$$

and the relative error variance, defined by

$$\varepsilon_{\text{rel}} = \frac{|\varepsilon - \varepsilon_{\text{ref}}|}{\varepsilon_{\text{ref}}}, \tag{9}$$

where ε_{ref} is (3) as calculated using the exact eigenvalues (5) and (7), against the mesh size h. The plots suggest that the convergence of the numerically computed eigenvalues $\hat{\lambda}$ is quadratic in the mesh size, i.e

$$\max |\lambda - \hat{\lambda}| \leq Ch^2 \tag{10}$$

where h is the mesh size. This quadratic convergence is consistent with [6], giving us confidence that the implemented Nyström method works.

2.2 2D Exponential Covariance

Here we simulate the 2D Gaussian random field with the covariance

$$R(s,t) = \exp\left(-\frac{||s-t||}{\rho}\right),$$

on the domain $[0,1] \times [0,1]$ with $\rho = 0.1$ in an $N = 100$ term Karhunen-Loève expansion. We use 50×50 simulation points as this is all the ram on our computer allows us to do. In figure 2 we see a plot of a realisation of this random field along with its eigenvalue decay and its first 6 eigenfunctions (corresponding to the six largest magnitude eigenvalues).

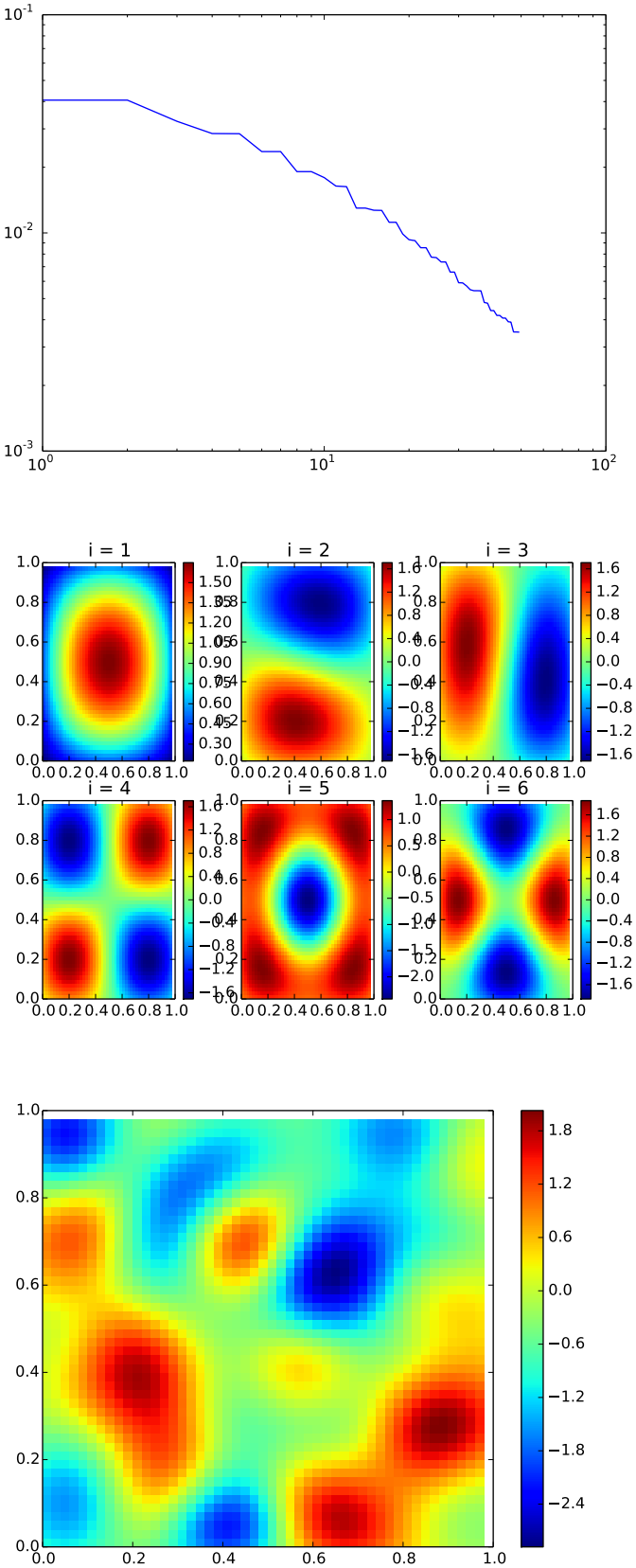

Figure 2: (Top) The first 100 eigenfunctions of the exponential covariance of section 2.2. (Middle) First six eigenfunctions. Note that some eigenfunctions are degenerate. (Bottom) A sample realisation of this exponential Gaussian field.

The script to generate the plots in figure 2 are is as follows:

```
# Test the 2D Karhunen-Loeve simulator: exponential covariance realisation
from GaussRF.GaussRF import GaussF_KL2D, grid_2D
import numpy as np
import matplotlib.pyplot as plt
#Parameters of the 2D exponential covariance
method = "KL_EOLE"
N = 50
lims = [0., 1., 0., 1.]
n, m = [50, 50] # number of grid points along each direction
def K2D(x, y):
    return np.exp(-np.linalg.norm( x - y, 2, axis = 1)/ 0.1)
X_sim = GaussF_KL2D(N, n, m, lims, K2D)
phi, L = X_sim.eigens() #Get the eigenfunctions and values
X = X_sim.Gfield()
xx, yy = grid_2D(n, m, lims, method)
# plot the eigenfunctions
for i in range(6):
    plt.subplot(2,3,i+1).set_title('i = {}'.format(i+1)) # add subplot
    e_func = np.array(phi[:,i]).reshape(n,m) # extract and reshape EFs
    plt.pcolor(xx,yy,e_func)
    plt.colorbar()
plt.savefig("2D_exp_cov_EFs.pdf")
plt.show()
#plot the eigenvalues
plt.loglog(range(N), L[:N])
plt.savefig("2D_exp_cov_EVs.pdf")
plt.show()
#plot the field
plt.pcolor(xx,yy, X) # note we have to extract the first
plt.colorbar()
plt.savefig("2D_exp_cov_eg.pdf")
plt.show()
```

2.3 2D inhomogeneous Poisson Process

We have recently set out to add point process functionality to `GaussRF`, creating the module `point_process`. There is currently support for the simulation of inhomogeneous Poisson processes in two-dimensions. We use our simulation procedure to simulate the inhomogeneous Poisson process with the intensity function

$$\lambda(x, y) = 300(x^2 + y^2),$$

on the unit square $[0, 1] \times [0, 1]$, an example given in [7]. The class `PoissF_2D` can be used to generate a realisation of such a Poisson process:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from GaussRF.point_process import PoissF_2D
#Define the inhomogeneous intensity function
def lamb(x,y):
    return 300 * ( x**2 + y**2 )
lims = [0., 1., 0., 1.]
points = PoissF_2D(lims, lamb).Poi_field() # generate points
# plot the points on top of the intensity function.
plt.plot(points[:,0], points[:,1], 'o')
plt.xlabel('x')
plt.ylabel('y')
plt.title(r'Poisson process with intensity $\lambda(x,y)$')
xx, yy = np.meshgrid(np.linspace(0., 1., 100), np.linspace(0., 1., 100))
L = lamb(xx,yy) # get intensity field
plt.pcolor(xx, yy, L, cmap = cm.gray)
plt.colorbar()
plt.show()
```

In figure 2 we see a sample of this Poisson process overlaid on its intensity function.We can see few points in the lower left corner of the domain, where $\lambda(x, y)$ decreases its quickest.

2.4 2D homogeneous Poisson process

The previous example was an example of an inhomogeneous Poisson process on a 2D rectangular domain. Homogeneous Poisson point processes are also supported. The class `PoissHF_2D` simulates homogeneous Poisson processes on a rectangle, `PoissHF_disk` simulates an homogeneous process on a circular disk and `PoissHF_sphere` samples the process on a sphere. In figure 4 we see simulations of a process with on the unit square, the unit circle and the unit sphere. The code to produce the plots of figure 4 is presented here:

```
import numpy as np
import matplotlib.pyplot as plt
from GaussRF.point_process import PoissHF_2D, PoissHF_disk
#Parameters
lims = [0., 1., 0., 1.] # limits of rectangular geometry
lamb = 100 # process intensity
R = 1. # radius of disk domain

#Rectangular geometry
X = PoissHF_2D(lims, lamb)
X_points = X.realisation()
#plot
plt.scatter(X_points[:, 0], X_points[:, 1])
plt.axis([0,1,0,1])
plt.title(r'Poisson process with $\lambda = 100$ ')
```

```
plt.savefig('homog_poisson.pdf')
plt.show()

#Next, test the circulation geometry
X = PoissHF_disk(R, lamb)
X_points = X.realisation()
#plot
plt.scatter(X_points[:, 0], X_points[:, 1])
plt.axis([-1.5, 1.5, -1.5, 1.5])
plt.title(r'Poisson process with $\lambda = 100$ on a disk')
plt.savefig('homog_poisson_disk.pdf')
plt.show()

#Next, test a sphere
fig = plt.figure()
#ax = Axes3D(fig)
ax = fig.add_subplot(111, projection='3d')
ax.set_aspect('equal')
lamb = 70 # reset the intensity
X = PoissHF_sphere(R, lamb)
X_points = X.realisation()
#plot
u, v = np.mgrid[0: 2 * np.pi :20j, 0: np.pi : 10j]
x = np.cos(u) * np.sin(v)
y = np.sin(u) * np.sin(v)
z = np.cos(v)
ax.plot_surface(x, y, z, rstride = 1, cstride = 1, color = 'w', edgecolor = 'r', alpha = 0.3) #
ax.scatter(X_points[:, 0], X_points[:, 1], X_points[:, 2], color = 'k')
ax.set_title(r' Poisson process with $\lambda = 70$ on a sphere')
plt.savefig('homog_poisson_sphere.pdf')
plt.show()
```

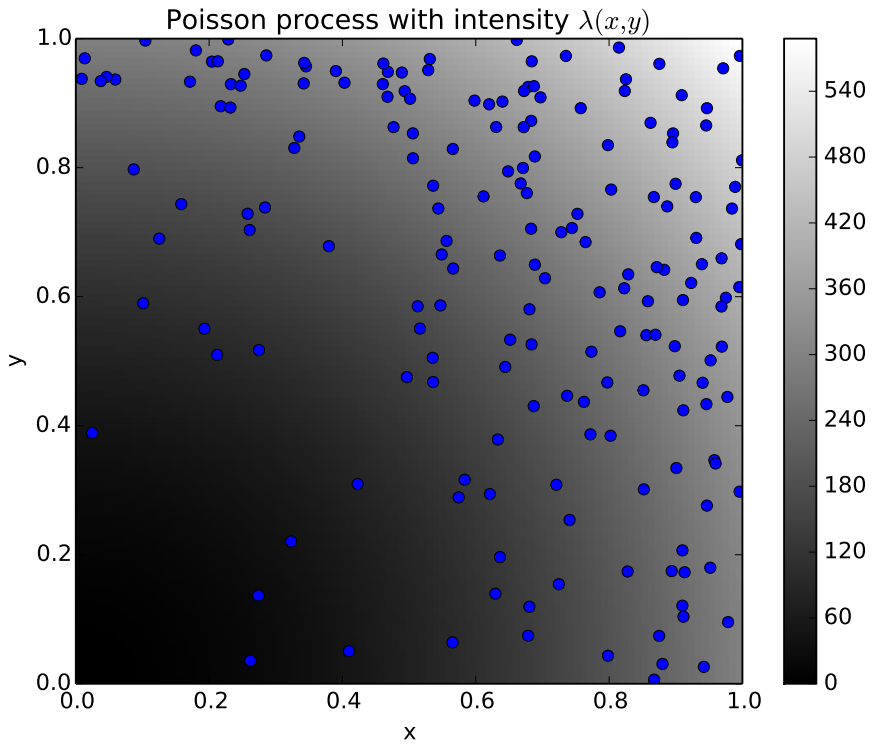

Figure 3: Inhomogeneous Poisson process with intensity $\lambda(x, y)$ on $[0, 1] \times [0, 1]$.

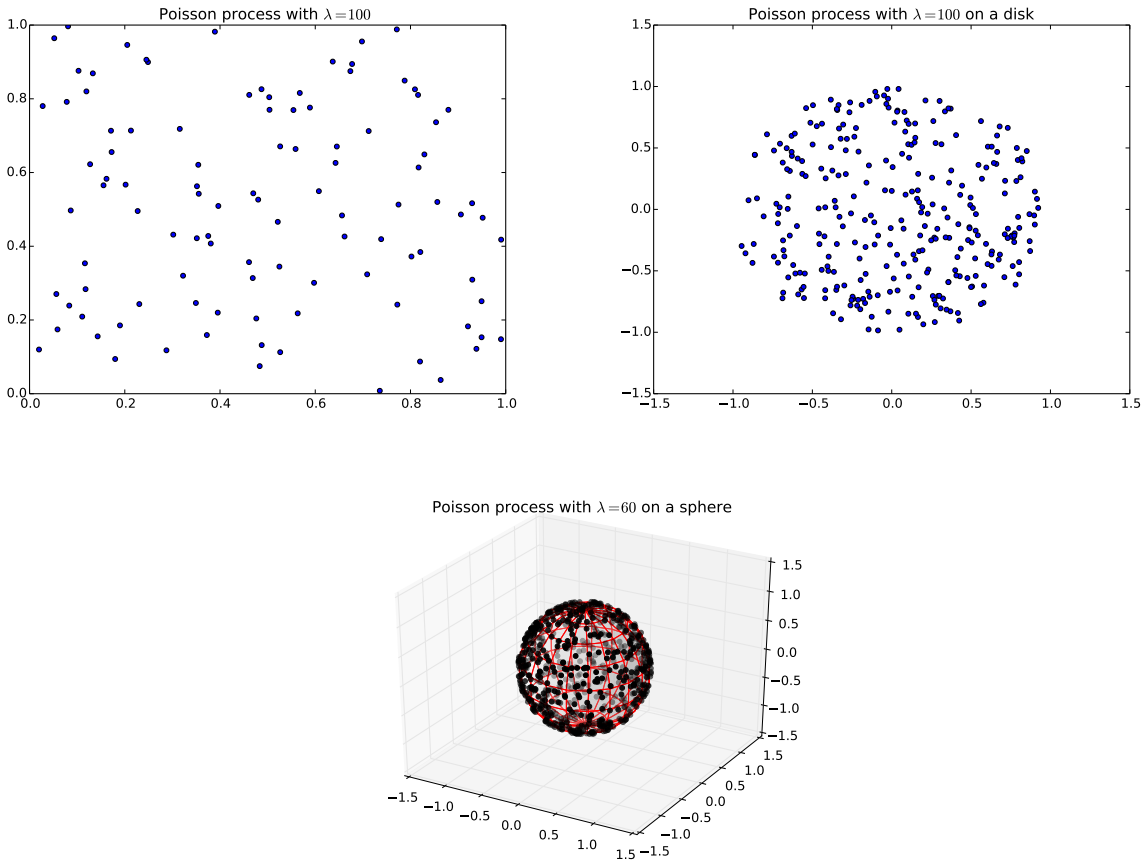

Figure 4: (Left) an homogeneous Poisson process on the square $[0, 1] \times [0, 1]$. (Right) an homogeneous Poisson process on the disk $x^2 + y^2 < 1$. In both cases the rate is $\lambda = 100$.(Bottom center) an homogeneous Poisson process on the unit sphere with rate $\lambda = 60$.

References

[1] Betz, W., Papaioannou, I., & Straub, D.(2014). Numerical methods for the discretization of random fields by means of the Karhunen-Loève expansion. *Computer Methods in Applied Mechanics and Engineering, 271*, 109-129.

[2] Dietrich, C.R., & Newsam, G. N. (1993). A fast and exact method for multidimensional Gaussian stochastic simulations. *Water Resources Research, 29(8)*, 2861-2869.

[3] Chan, G., & Wood, A.T. (1999). Simulation of stationary Gaussian vector fields. *Statistics and computing, 9(4)*, 265-268.

[4] Atkinson, K.E. (1967). The numerical solution of Fredholm integral equations of the second kind. *Siam Journal on Numerical Analysis, 4(3)*, 337-348.

[5] Phoon, K. K., Huang, S. P., & Quek, S. T. (2002). Implementation of Karhunen-Loeve expansion for simulation using a wavelet-Galerkin scheme. *Probabilistic Engineering Mechanics, 17(3)*, 293-303.

[6] Cai, D., Vassilevski, P. S. (2019). Eigenvalue Problems for Exponential-Type Kernels. *Computational Methods in Applied Mathematics, 20(1)*, 61-78.

[7] Kroese, D. P., Botev, Z. I. (2015) Spatial Process Simulation. *Stochastic geometry, spatial statistics and random fields*, Springer, 369–404.