Computational Statistics
M.Sc. Mathematical Engineering
Prof. A. Manzoni

A.A. 2021 – 2022
Politecnico di Milano

Hands-On 3
Gaussian Processes & Karhunen-Loève Expansion

In this session we learn how to generate realizations of a Gaussian process, and how to perform dimensionality reduction through the Karhunen-Loève Expansion.

Recall that a stochastic process $X_t, t \in I \subset \mathbb{R}^d$ is Gaussian if and only if, for any finite subset $\{t_1, \dots, t_n\} \subset I$ – being $\mathbb{R}^d$ the input space – the random vector $\mathbf{X} = (X_{t_1}, \dots, X_{t_n})$ has a multivariate Gaussian distribution. Here we will focus on the cases $d = 1$ and $d = 2$, and scalar quantities to model.

A Gaussian process is entirely specified by its mean $\mu_X : I \subset \mathbb{R}$ and its covariance function $C_X : I \times I \to \mathbb{R}$. The mean is the trend around which the realizations vary, while C_X describes the regularity and characteristic length scale of the model. We will consider the case of weakly stationary GPs, for which the positive definite covariance function $C_X(s,t)$ only depends on $s - t$. The covariance function is the most important term of a GP, since it controls the smoothness and the scale of the approximation.

Two popular choices for the covariance function are:

- the stationary Gaussian (or squared exponential) function, respectively isotropic or anisotropic:

$$C_X(s,t) = \sigma^2 exp\left(-\frac{\|t-s\|^2}{2l_c^2}\right), \qquad C_X(s,t) = \sigma^2 exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(t_i - s_i)^2}{l_{c,i}^2}\right),$$

  where σ^2 is the variance of the field, and l_c its correlation length;

- the exponential function, respectively isotropic or anisotropic:

$$C_X(s,t) = \sigma^2 exp\left(-\frac{\|t-s\|}{l_c}\right), \qquad C_X(s,t) = \sigma^2 exp\left(-\sum_{i=1}^{d}\frac{|t_i - s_i|}{l_{c,i}}\right).$$

The provided `Matlab` function `randomfield` generates *approximate* realizations of a Gaussian random field on a user-defined computational mesh. In its basic format,

```
[F, KL] = randomfield (corr, mesh)
```

the function returns:

- `F`: a matrix of random field realizations; each column is a realization of the random field with each element corresponding to a point supplied in the required input `mesh`;
- `KL`: a struct containing the components of a Karhunen-Loève (KL) expansion of the random field. This struct includes:
  - `KL.mean`: the mean of the random field. If mean was supplied by the user, then this is the same vector. If the field was conditioned on data, then this is equivalent to Kriging interpolant;
  - `KL.bases`: the eigenvectors covariance matrix;
  - `KL.sv`: the square root of the eigenvalues of the covariance matrix.

Required inputs are:

- `mesh`: a matrix of size $n_x \times d$, where n_x is the number of points in the mesh and d is the dimension.
- `corr`: a struct containing the correlation information;

in one dimension is straightforward to generate a good mesh. In higher dimension we need a way to feed the function with a good mesh. That's why we'll rely on an other function that focuses on good mesh construction

this function will evaluate the covariance function on the mesh (either one-dim / two-dim). This function discretize the covariance object.

so that, by using a sum of N terms weighted $1/\sqrt{\lambda_i}$, λ_i eigenvalues, times the eigenvectors times some random input we can get the reconstruction in finite dimension of the random field (KL-expansion)

bases → eigenvectors
SV → eigenvalues (singular values)

Among others, the input struct `corr` may contain the following fields:

- `corr.name`: taking two points `x1` and `x2` from the mesh, specifies the correlation type from `'gauss'` (Gaussian, also referred to as squared exponential) or `'exp'` (exponential), given respectively by

$$\texttt{sigma}\ exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{(\texttt{x1(i)}-\texttt{x2(i)})^2}{\texttt{c0(i)}}\right) \qquad \texttt{'gauss'}$$

$$\texttt{sigma}\ exp\left(-\frac{1}{2}\sum_{i=1}^{d}\frac{\texttt{x1(i)}-\texttt{x2(i)}}{\texttt{c0(i)}}\right) \qquad \texttt{'exp'}$$

- `corr.c0`: the scaling parameters for the correlation function. `c0` may be a scalar for isotropic correlation or a vector for anisotropic correlation. In the anisotropic case, the vector must have d elements, where d is the dimension of a mesh point.
- `corr.sigma`: the variance scaling parameter; it can be either a scalar or a vector with the size of the mesh.

Problem 3.1 (Gaussian Processes, case $d = 1$)

1. Using the Matlab function `randomfield`, generate 10 samples from a Gaussian process defined over the set $D = (-1, 1)$, discretized introducing a uniform partition D_h with mesh size $h = 0.05$, and resulting number of points N_h. Take as mean $\mu_X = 0$ and (isotropic) exponential covariance function with $\sigma = 1$ and correlation length $l_c = 2$, so that `c0`$= l_c/2 = 1$.
2. Generate 10 more samples exploiting the computed KL expansion, check that the KL modes are orthonormal, visualize the first 10 (spatial) modes and the decay of the singular values.
3. Check now that the truncated KL expansion

$$X_N(t,\omega) = \mu_X(t) + \sum_{i=1}^{N}\sqrt{\lambda_i}b_i(t)Y_i(\omega)$$

provides a good approximation to the GP X_t, and compare the error decay on a sample of 250 generated instances. Consider the field approximation obtained by taking all the N_h modes as *truth* solution.

4. Repeat point 1 by adding a mean function $\mu_X = t$ by evaluating this function over the grid defined in `mesh`, and check the obtained samples.

1. Sampling from the Gaussian process can be done using the following commands:

```
corr.name = 'exp';
corr.c0 = 1;
corr.sigma = 1;

mesh = linspace(-1,1,401)';                % generate a mesh
[F,KL] = randomfield(corr, mesh, 'nsamples', 10);

figure(1)
plot(mesh,F)
```

2. Once the domain D is discretized, all the operations related to the KL expansion become finite-dimensional. Indeed, denoting by D_h the discretization introduced over D, the covariance is a two-point statistics that models the spatial correlation of the stochastic field between two points $t_i, t_j \in D_h$ and results in a matrix $\mathbf{K} \in \mathbb{R}^{N_h \times N_h}$. This matrix, being symmetric and positive definite, has N_h non-negative real eigenvalues $\lambda_1 \geq \ldots \geq \lambda_{N_h} \geq 0$ and corresponding eigenvectors $\boldsymbol{\zeta}_i \in \mathbb{R}^{N_h}$ such that

$$\mathbf{K}\boldsymbol{\zeta}_i = \lambda_i \boldsymbol{\zeta}_i, \qquad i = 1, \ldots, N_h$$

that form an orthonormal basis for $\mathbb{R}^{N_h}$. We can use the following commands (see Figure 1, right and Figure 2, left):

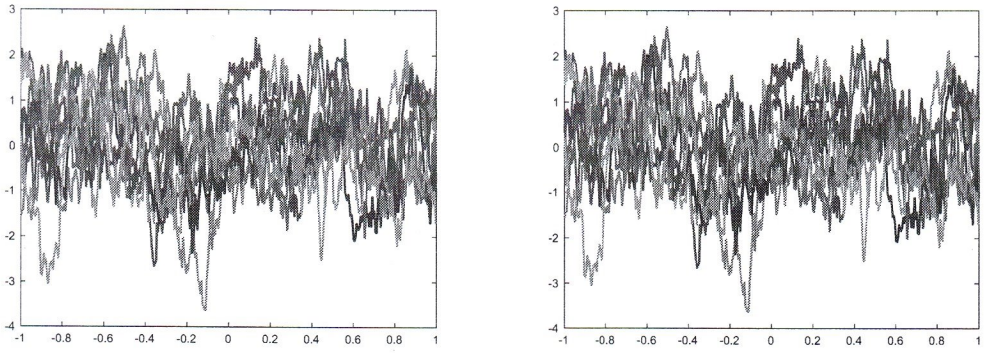


Figure 1: Left: 10 realization of the GP; right: 10 realizations of the GP obtained through the (full) KL expansion.

```
% to generate Nreal more samples using the KL
trunc = length(KL.sv);                    % get the truncation level
Nreal = 10;
W = randn(trunc,Nreal);
F1 = repmat(KL.mean,1,Nreal) + KL.bases*diag(KL.sv)*W;

% plot the realization
figure(2)
plot(mesh,F1,'lineWidth',2)

% plot the eigenvectors
figure(3)
plot(mesh,KL.bases(:,1:10),'lineWidth',2)

%check that KL modes are orthonormal
KL.bases(:,1:10)'*KL.bases(:,1:10)

% plot the eigenvalues
figure(4)
semilogy(KL.sv.^2)
```

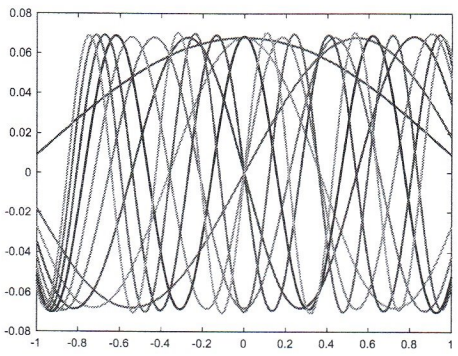
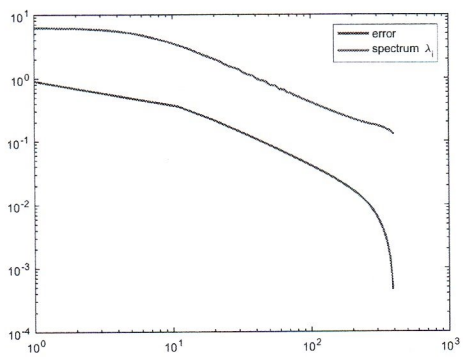


Figure 2: Left: first 10 modes of the KL expansion.

3. We can check the error decay using the following commands (see Figure 3, left):

```
Ntest_sample = 250;
```

```
Nmodes = 1:10:size(mesh,1);
for j=1:Ntest_sample

    W1=randn(size(mesh,1),1);
    F_real=(KL.bases * diag(KL.sv)) * W1 + KL.mean;
    for n = 1:length(Nmodes)

      weights = diag(KL.sv(1:Nmodes(n),:));
      F_red=(KL.bases(:,1:Nmodes(n)) *  weights) * W1(1:Nmodes(n)) + KL.mean;

      field_error(n,j) =  ((F_real-F_red)'*((F_real-F_red)));
      norm_F                =   ((F_red)'* ((F_red)));
    end
    field_error_rel(:,j)       = field_error(:,j)./norm_F; % relative error
    fprintf('k_sample = %d \n',j);
end

field_error_av        =  mean(field_error,2);
field_error_rel_av    =  mean(field_error_rel,2);
field_error_max       =  max(field_error,[],2);
field_error_rel_max   =  max(field_error_rel,[],2);

figure(5)
loglog(Nmodes,field_error_rel_av,'lineWidth',2)
hold on
loglog(KL.sv,'lineWidth',2)
```



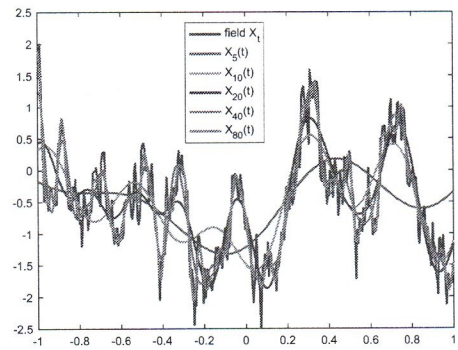


Figure 3: Left: spectrum and error decay; right: KL expansions for different numbers N of modes.

4. Adding the mean function is rather simple, using the following commands (see Figure 4):

```
mean_func = @(t) t;
mean_mesh=mean_func(mesh);
[F,KL] = randomfield(corr, mesh, 'nsamples', 10, 'mean', mean_mesh);
```

Problem 3.2 (Gaussian Processes, case $d = 2$)

1. Generate a Gaussian random field $Y(x, \omega)$ in dimension $d = 2$ on $D = (0, 1) \times (0, 5)$, discretized introducing a partition made by triangular elements:

```
[vertices, boundaries, elements] = initmesh('mesh_square','Jiggle','minimum', ...
    'Hgrad',1.01,'Hmax', 0.1);
for i = 1 : 3
```

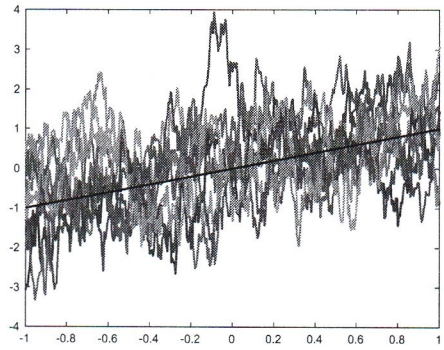


Figure 4: 10 samples from the GP with mean function $\mu_X(t) = t$.

```
    [vertices, boundaries, elements] = refinemesh('mesh_square', ...
       vertices, boundaries, elements);
end
```

Take as mean $\mu_Y = 4$ and an anisotropic exponential covariance function defined by

```
corr.name = 'gauss';
corr.c0 = 0.05;    corr.sigma = 1;
```

2. Generate 4 samples exploiting the computed KL expansion, with different dimension $N = 3, 6, 9, 150$, evaluating the error between the Gaussian random field and the truncated KL expansion

$$Y_N(t, \omega) = \mu_X(t) + \sum_{i=1}^{N} \sqrt{\lambda_i} b_i(t) Y_i(\omega).$$

Consider the field approximation obtained by taking all the N_h modes as *truth* solution.

Hands-On 3

Gaussian Process & Karhunen-Loève Expansion

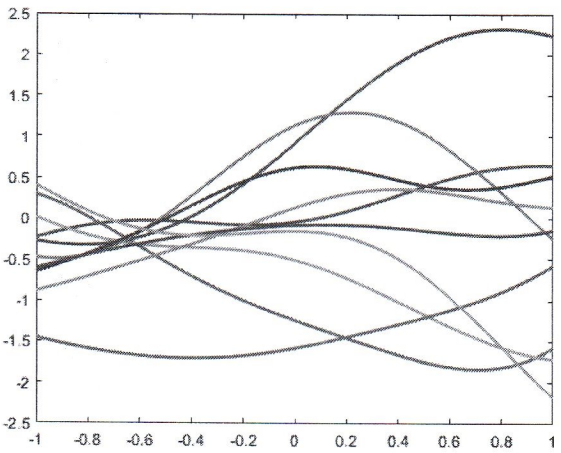
Problem 3.1

1.

```
corr.name  = 'gauss';
corr.c0    = 0.5;
corr.sigma = 1;

mesh   = linspace(-1, 1, 401)'; % generate a mesh
[F,KL] = randomfield(corr, mesh, 'nsamples', 10);
```

```
Computing the full spectral decomposition of the 401-by-401 covariance matrix.
eigs converged!
```

```
figure(1)
plot(mesh, F, 'lineWidth', 2)
```



These are 10 discrete realizations of the random field, that is a gaussian random process in one dimension, with given mean (zero) and covariance structure.

To do so, the function `randomfield.m` has computed some eigen-decomposition since we see "`eigs converged!`".

Because of the definition of gaussian random process, if, for every realization, we take a subset of points and we take the joint distribution of these points we get a gaussian vector.

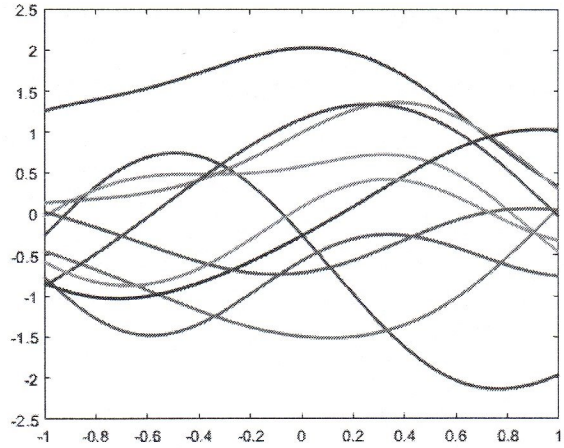
```
KL
```

```
KL = struct with fields:
     mean: [401×1 double]
    bases: [401×401 double]
       sv: [401×1 double]
```

In KL there are 401 singular values, 401 bases (columns) and the mean of the gaussian process.

2.

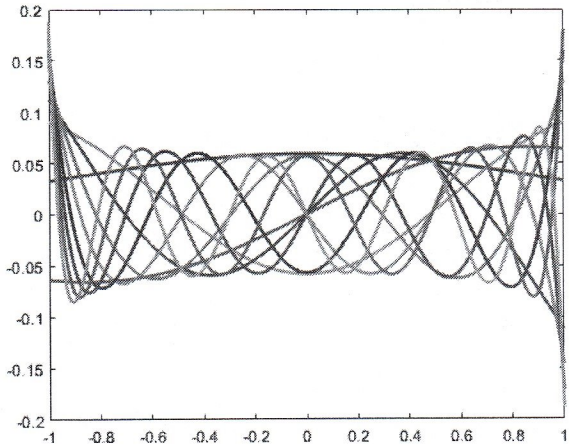
```
% to generate Nreal more samples using the KL
trunc = length(KL.sv);    % get the truncation level (now we're considering all the 401 singular values,
                          % later on we'll se what happens if we consider only part of them)
Nreal = 10;
W     = randn(trunc, Nreal);
F1    = repmat(KL.mean, 1, Nreal) + KL.bases * diag(KL.sv) * W;

% plot the realization
figure(2)
plot(mesh, F1, 'lineWidth', 2)
```



These are 10 new realization of the gaussian process.

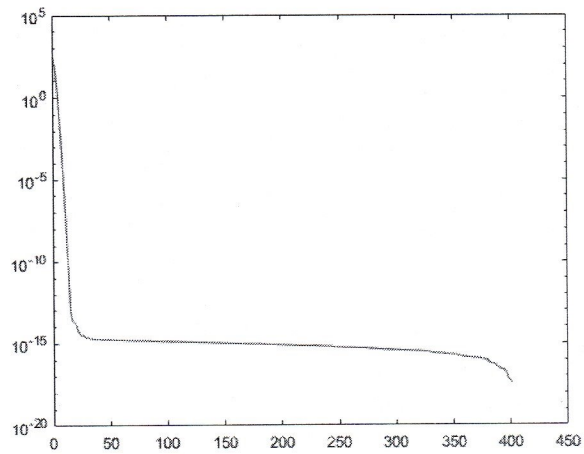
3.

```
% plot the eigenvectors
figure(3)
plot(mesh, KL.bases(:, 1:10), 'lineWidth', 2)
```



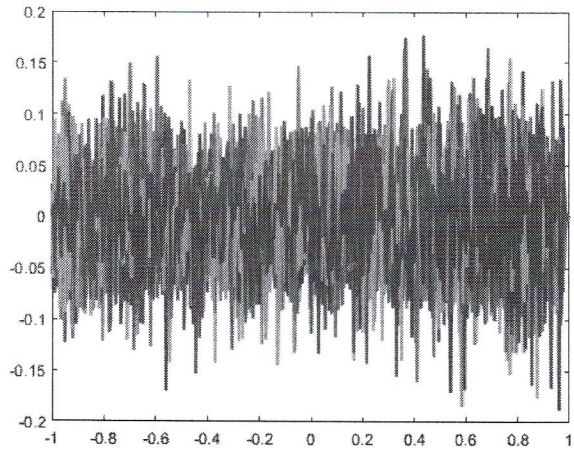
These are the (first 10) basic bricks that KL uses to get the realizations.

```
semilogy(KL.sv.^2)
```



This is how the eigenvalues decay. This is a good news because, from the plot, we see that if we chop the expansion at about N ~ 20/30 modes we already reach a good approximation (we discard very limited information)

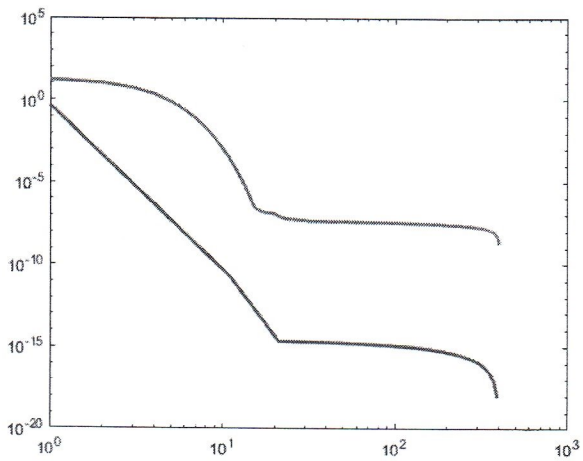
```
% how are the last eigenvectors?
figure(4)
plot(mesh, KL.bases(:, end-10:end), 'lineWidth', 2)
```



```
% consider now an increasing number of nodes
Ntest_sample = 250;
Nmodes       = 1:10:size(mesh,1);

for j = 1:Ntest_sample
    W1     = randn(size(mesh,1),1);
    F_real = (KL.bases * diag(KL.sv)) * W1 + KL.mean;
    for n = 1:length(Nmodes)
        weights          = diag(KL.sv(1:Nmodes(n),:));
        F_red            = (KL.bases(:,1:Nmodes(n)) * weights) * W1(1:Nmodes(n)) + KL.mean;
        field_error(n,j) = ((F_real-F_red)' * ((F_real-F_red)));
        norm_F           = ((F_red)' * ((F_red)));
    end
    field_error_rel(:,j) = field_error(:,j)./norm_F; % relative error
    fprintf('k_sample = %d \n',j);
end
```

```
k_sample = 1
k_sample = 2
..
k_sample = 249
k_sample = 250
```

```
field_error_av      = mean(field_error,2);
field_error_rel_av  = mean(field_error_rel,2);
field_error_max     = max(field_error,[],2);
field_error_rel_max = max(field_error_rel,[],2);

figure(5)
loglog(Nmodes, field_error_rel_av, 'lineWidth', 2)
hold on
loglog(KL.sv, 'lineWidth', 2)
hold off
```



The decay of the error looks like the decay of the eignevalues.

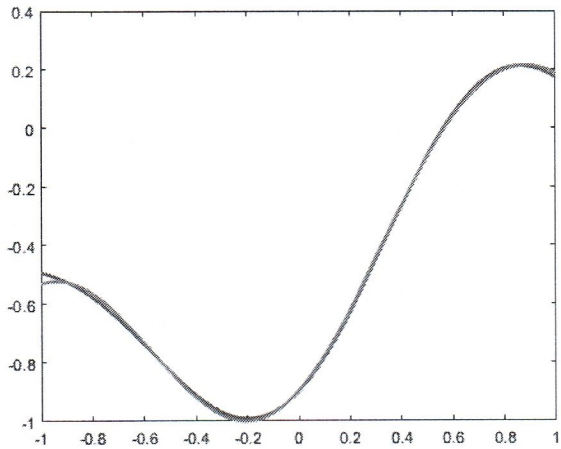
```
% plot KL expansions for different numbers of modes
corr.name  = 'gauss';
corr.c0    = 0.5;
corr.sigma = 1;
mesh       = linspace(-1, 1, 401)';
[F,KL]     = randomfield(corr, mesh, 'nsamples', 10);
```

```
Computing the full spectral decomposition of the 401-by-401 covariance matrix.
eigs converged!
```

```
figure(6)
Ntest_sample = 1;
Nmodes       = [5 10 20 40 80];

for j = 1:Ntest_sample
        W1     = randn(size(mesh,1),1);
        F_real = (KL.bases * diag(KL.sv)) * W1 + KL.mean;
        plot(mesh, F_real, 'lineWidth', 2)
        hold on
        for n = 1:length(Nmodes)
                weights = diag(KL.sv(1:Nmodes(n),:));
                F_red   = (KL.bases(:,1:Nmodes(n)) * weights) * W1(1:Nmodes(n)) + KL.mean;
        plot(mesh, F_red, 'lineWidth', 2)
        end
        fprintf('k_sample = %d \n', j);
end
```

```
k_sample = 1
```

```
hold off
```



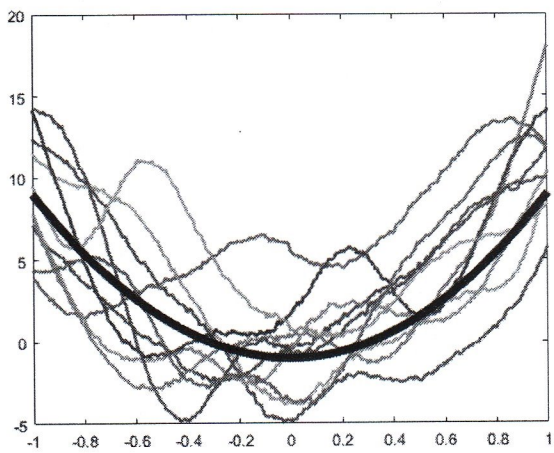
Different realization of the random field for different number of nodes.

4.

```
corr.name  = 'gauss';
corr.c0    = 0.05;
corr.sigma = 10;
mean_func  = @(x) 10*x.^2-1;
mean_mesh  = mean_func(mesh);
[F,KL]     = randomfield(corr, mesh, 'nsamples', 10, 'mean', mean_mesh);
```

```
Computing the full spectral decomposition of the 401-by-401 covariance matrix.
eigs converged!
```

```
figure(7)
plot(mesh, F, 'lineWidth', 2)
hold on
plot(mesh, mean_mesh, 'k-', 'lineWidth', 5)
hold off
```



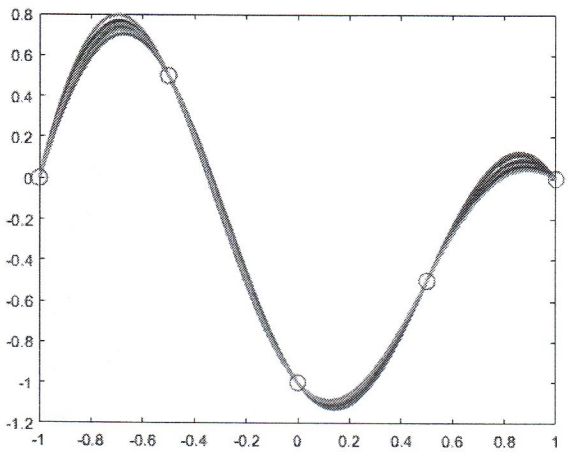
[Extra] KRIGGING - Gaussian Process that interpolates some points

It is possible to obtain a behavior of a gaussian random field, but constrained to the fact that some points must be interpolated.

```
% Version 1
corr.name  = 'gauss';
corr.c0    = 0.5;
corr.sigma = 1;
mesh       = linspace(-1,1,401)';

data.x  = [-1; -0.5; 0; 0.5; 1];
data.fx = [0; 0.5; -1; -0.5; 0];  % specify data

[F2,KL] = randomfield(corr, mesh, 'nsamples', 10, 'data', data);
```

```
Computing the full spectral decomposition of the 401-by-401 covariance matrix.
eigs converged!
```

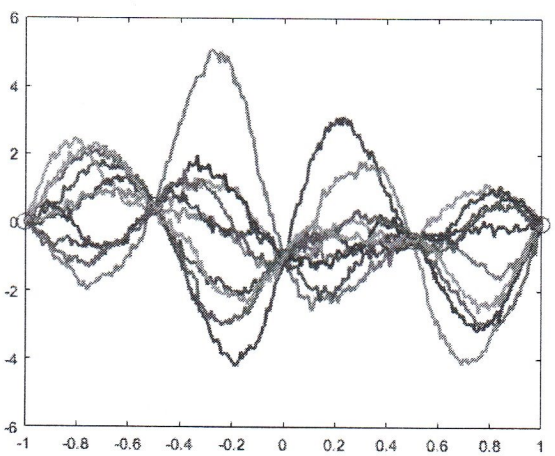
```
plot(mesh, F2, 'lineWidth', 2)
hold on
plot(data.x, data.fx, 'o', 'MarkerSize', 10)
hold off
```




```
% Version 2
corr.name  = 'gauss';
corr.c0    = 0.05;
corr.sigma = 10;
mean_func  = @(x) 10*x.^2-1;
mean_mesh  = mean_func(mesh);

data.x  = [-1; -0.5; 0; 0.5; 1];
data.fx = [0; 0.5; -1; -0.5; 0];  % specify data

[F2,KL] = randomfield(corr, mesh, 'nsamples', 10, 'data', data);
```

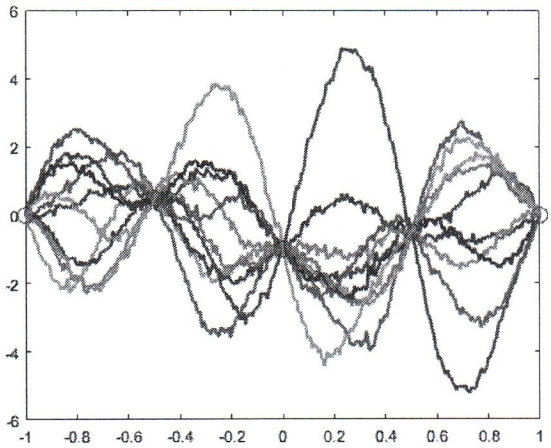
```
Computing the full spectral decomposition of the 401-by-401 covariance matrix.
eigs converged!
```

```
plot(mesh, F2, 'lineWidth', 2)
hold on
plot(data.x, data.fx, 'o', 'MarkerSize', 10)
hold off
```




```
% to generate 100 more samples using KL
trunc = length(KL.sv);
```

```
Nreal = 10;
W     = randn(trunc, Nreal);
F2    = repmat(KL.mean, 1, Nreal) + KL.bases * diag(KL.sv) * W;

% plot the realization
plot(mesh, F2, 'lineWidth', 2)
hold on
plot(data.x, data.fx, 'o', 'MarkerSize', 10)
hold off
```



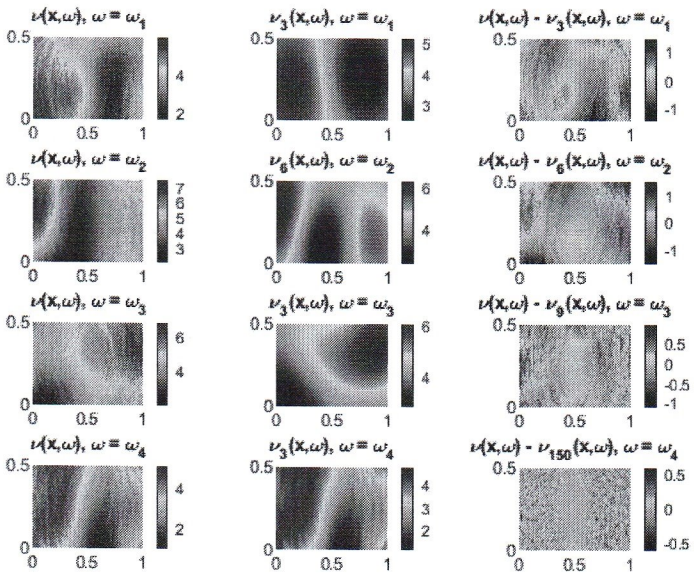
Problem 3.2

```
[vertices, boundaries, elements] = initmesh('mesh_square', 'Jiggle', 'minimum', 'Hgrad', 1.01, 'Hmax', 0.1);
for i = 1:3
        [vertices, boundaries, elements] = refinemesh('mesh_square', vertices, boundaries, elements);
end

corr.name  = 'gauss';
corr.c0    = 0.05;
corr.sigma = 1;

[F,KL] = randomfield(corr, vertices', 'mean', 4*ones(size(vertices,2),1));
```

```
Computing the full spectral decomposition of the 5305-by-5305 covariance matrix.
eigs converged!
```

```
% plot some realizations

figure(1)

W1          = randn(size(vertices,2),1);
F_real1     = (KL.bases * diag(KL.sv)) * W1 + KL.mean;
Nmodes      = 3;
weights     = diag(KL.sv(1:Nmodes,:));
F_red_real1 = (KL.bases(:,1:Nmodes) * weights) * W1(1:Nmodes) + KL.mean;

subplot(4,3,1)
pdesurf(vertices, elements, F_real1); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega), \omega = \omega_1'])

subplot(4,3,2)
pdesurf(vertices, elements, F_red_real1); view(2); colormap(jet); colorbar;
title(['\nu_3(x,\omega), \omega = \omega_1'])

subplot(4,3,3)
pdesurf(vertices, elements, F_real1 - F_red_real1); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega) - \nu_3(x,\omega), \omega = \omega_1'])

%

W2          = randn(size(vertices,2),1);
F_real2     = (KL.bases * diag(KL.sv)) * W2 + KL.mean;
Nmodes      = 6;
weights     = diag(KL.sv(1:Nmodes,:));
F_red_real2 = (KL.bases(:,1:Nmodes) * weights) * W2(1:Nmodes) + KL.mean;

subplot(4,3,4)
pdesurf(vertices, elements, F_real2); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega), \omega = \omega_2'])

subplot(4,3,5)
pdesurf(vertices, elements, F_red_real2); view(2); colormap(jet); colorbar;
title(['\nu_6(x,\omega), \omega = \omega_2'])

subplot(4,3,6)
pdesurf(vertices, elements, F_real2 - F_red_real2); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega) - \nu_6(x,\omega), \omega = \omega_2'])
```

```
%

W3          = randn(size(vertices,2),1);
F_real3     = (KL.bases * diag(KL.sv)) * W3 + KL.mean;
Nmodes      = 9;
weights     = diag(KL.sv(1:Nmodes,:));
F_red_real3 = (KL.bases(:,1:Nmodes) * weights) * W3(1:Nmodes) + KL.mean;

subplot(4,3,7)
pdesurf(vertices, elements, F_real3); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega), \omega = \omega_3'])

subplot(4,3,8)
pdesurf(vertices, elements, F_red_real3); view(2); colormap(jet); colorbar;
title(['\nu_3(x,\omega), \omega = \omega_3'])

subplot(4,3,9)
pdesurf(vertices, elements, F_real3 - F_red_real3); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega) - \nu_9(x,\omega), \omega = \omega_3'])

%

W4          = randn(size(vertices,2),1);
F_real4     = (KL.bases * diag(KL.sv)) * W4 + KL.mean;
Nmodes      = 150;
weights     = diag(KL.sv(1:Nmodes,:));
F_red_real4 = (KL.bases(:,1:Nmodes) * weights) * W4(1:Nmodes) + KL.mean;

subplot(4,3,10)
pdesurf(vertices, elements, F_real4); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega), \omega = \omega_4'])

subplot(4,3,11)
pdesurf(vertices, elements, F_red_real4); view(2); colormap(jet); colorbar;
title(['\nu_3(x,\omega), \omega = \omega_4'])

subplot(4,3,12)
pdesurf(vertices, elements, F_real4 - F_red_real4); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega) - \nu_{150}(x,\omega), \omega = \omega_4'])
```



These are 4 different realizations of the gaussian random field in 2 dimensions:

- on the left the true realization is obtained by considering the discretization over the whole mesh
- in the middle there is the discretization with `Nmodes = [3, 6, 9, 150]`
- on the right there is the error (evaluate as: true random field - discretized random field)

```
% Otherwise, let's focus on the difference of the approximation based on the number of modes

[vertices, boundaries, elements] = initmesh('mesh_square', 'Jiggle', 'minimum', 'Hgrad', 1.01, 'Hmax', 0.1);
for i = 1:3
        [vertices, boundaries, elements] = refinemesh('mesh_square', vertices, boundaries, elements);
end

corr.name  = 'gauss';
corr.c0    = 0.05;
corr.sigma = 1;

[F,KL] = randomfield(corr, vertices', 'mean', 4*ones(size(vertices,2),1));
```

```
Computing the full spectral decomposition of the 5305-by-5305 covariance matrix.
eigs converged!
```

```
figure(2)

W1          = randn(size(vertices,2),1);
F_real1     = (KL.bases * diag(KL.sv)) * W1 + KL.mean;
Nmodes      = 3;
weights     = diag(KL.sv(1:Nmodes,:));
F_red_real1 = (KL.bases(:,1:Nmodes) * weights) * W1(1:Nmodes) + KL.mean;

subplot(2,3,1)
pdesurf(vertices, elements, F_real1); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega), \omega = \omega_1'])

subplot(2,3,2)
pdesurf(vertices, elements, F_red_real1); view(2); colormap(jet); colorbar;
title(['\nu_3(x,\omega), \omega = \omega_1'])

subplot(2,3,3)
pdesurf(vertices, elements, F_real1 - F_red_real1); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega) - \nu_3(x,\omega), \omega = \omega_1'])


Nmodes      = 150;
weights     = diag(KL.sv(1:Nmodes,:));
F_red_real1 = (KL.bases(:,1:Nmodes) * weights) * W1(1:Nmodes) + KL.mean;

subplot(2,3,4)
pdesurf(vertices, elements, F_real1); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega), \omega = \omega_1'])

subplot(2,3,5)
pdesurf(vertices, elements, F_red_real1); view(2); colormap(jet); colorbar;
title(['\nu_3(x,\omega), \omega = \omega_1'])

subplot(2,3,6)
pdesurf(vertices, elements, F_real1 - F_red_real1); view(2); colormap(jet); colorbar;
title(['\nu(x,\omega) - \nu_{150}(x,\omega), \omega = \omega_1'])
```