Programming sheet 1.

Meeting week: **3.–7. March 2025**

Information on submission

There will be a meeting with the assistant to present the programming exercises. In this meeting, you must be able to explain your code and answer questions about it. Your code must be uploaded to ADAM by the **last Sunday before the meeting week**. Start working on the programming assignments early and do not hesitate to ask questions: **viacheslav.karnaev@unibas.ch**.

Poisson problem with a random right-hand side

Let $D \subset \mathbb{R}^2$ be a bounded and connected domain with smooth boundary ∂D and $(\Omega, \Sigma, \mathbb{P})$ be a complete probability space. For $u \in L^2_{\mathbb{P}}(\Omega) \otimes H^1_0(D)$, we consider the following boundary value problem

$$\begin{cases} -\Delta u(\mathbf{x}, \omega) = f(\mathbf{x}, \omega) \ \text{ in } \ D, \\ \quad u(\mathbf{x}, \omega) = 0 \ \text{ on } \ \partial D, \end{cases}$$

with random right-hand side $f \in L^2_{\mathbb{P}}(\Omega) \otimes L^2(D)$, where $\omega \in \Omega$ indicates the random event. In case of u being a Gaussian random field, we can represent it by its *Karhunen-Loève expansion*

$$u(\mathbf{x}, \omega) = \mathbb{E}[u](\mathbf{x}) + \sum_{i=1}^{\infty} \sqrt{\lambda_i} \phi_i(\mathbf{x}) \psi_j(\omega), \tag{1}$$

where $\mathbb{E}[u](\mathbf{x})$ is the *expectation*, i.e.,

$$\mathbb{E}[u](\mathbf{x}) := \int_{\Omega} u(\mathbf{x}, \omega) \, \mathrm{d}\mathbb{P}(\omega), \quad \mathbf{x} \in D,$$

$\{\psi_i\}_{i=1}^{\infty}$ is a series of independent, standard normally distributed random variables, and $\{(\lambda_i, \phi_i(\mathbf{x}))\}_{i=1}^{\infty}$ are the normalized eigenpairs of the *covariance operator*:

$$\mathcal{C}_u\langle\phi\rangle(\mathbf{x}) := \int_D \mathrm{Cov}[u](\mathbf{x}, \mathbf{y}) \phi(\mathbf{y}) \, \mathrm{d}\mathbf{y} = \lambda \phi(\mathbf{x}), \quad \mathbf{x} \in D. \tag{2}$$

Here, the function $\mathrm{Cov}[u] : D \times D \to \mathbb{R}$ is called the *covariance function* and defined as

$$\mathrm{Cov}[u](\mathbf{x}, \mathbf{y}) := \int_{\Omega} u(\mathbf{x}, \omega) u(\mathbf{y}, \omega) \, \mathrm{d}\mathbb{P}(\Omega) - (\mathbb{E}[u](\mathbf{x}))^2, \quad \mathbf{x}, \mathbf{y} \in D.$$

If we know the expectation $\mathbb{E}[f] \in L^2(D)$, it is easy to get $\mathbb{E}[u] \in H^1_0(D)$ as the solution of the boundary value problem

$$\begin{cases} -\Delta \mathbb{E}[u](\mathbf{x}) = \mathbb{E}[f](\mathbf{x}) \ \text{ in } \ D, \\ \quad \mathbb{E}[u](\mathbf{x}) = 0 \ \text{ on } \ \partial D. \end{cases} \tag{3}$$

In the case of covariance, things are more complicated. We can obtain the covariance $\mathrm{Cov}[u] \in H^1_0(D) \otimes H^1_0(D)$ from the covariance of the right-hand side $\mathrm{Cov}[f] \in L^2(D) \otimes L^2(D)$ by solving the tensor-product boundary value problem

$$\begin{cases} (\Delta \otimes \Delta) \, \mathrm{Cov}[u](\mathbf{x}, \mathbf{y}) = \mathrm{Cov}[f](\mathbf{x}, \mathbf{y}) \ \text{ in } \ D \times D, \\ \qquad\qquad \mathrm{Cov}[u](\mathbf{x}, \mathbf{y}) = 0 \ \text{ on } \ \partial(D \times D). \end{cases} \tag{4}$$

This is a boundary value problem in $\mathbb{R}^4$ and computationally expensive. However, we can efficiently avoid the high dimensionality by employing a low rank-approximation.

Low-rank approximation

To derive a low-rank approximation, we use the pivoted Cholesky decomposition. We discretize the covariance $\mathrm{Cov}[f]$ by the matrix $\mathbf{C}_f$ such that $[\mathbf{C}_f]_{i,j} = \mathrm{Cov}[f](\mathbf{x}_i, \mathbf{x}_j)$, where $\{\mathbf{x}_i\}_{i=1}^n$ are the vertices of the underlying finite element mesh. Since the correlation is a symmetric and positive semidefinite function, the matrix $\mathrm{Cov}[f]$ is also symmetric and positive semidefinite. We can thus apply the following algorithm:

Algorithm Pivoted Cholesky decomposition

Input: covariance function $\mathrm{Cov}[f] : D \times D \to \mathbb{R}$, points $\mathbf{x}_1, \dots, \mathbf{x}_n$, accuracy $\texttt{tol} \geq 0$
Output: $\mathbf{L} \in \mathbb{R}^{n \times m}$ with $\mathbf{L}\mathbf{L}^\intercal \approx \mathbf{C}_f$, permutation vector $\mathbf{p}$

setze $\mathbf{p} := [1, 2, \dots, n]$, $\mathbf{d} := \left[\mathrm{Cov}[f](\mathbf{x}_j, \mathbf{x}_j)\right]_{j=1}^n$, $\epsilon := \|\mathbf{d}\|_1$, $m := 1$
while $m \leq n$ and $\epsilon \geq \texttt{tol}$ **do**
  set pivot $:= \mathrm{argmax}_{\ell \in \{m, m+1, \dots, n\}} \mathbf{d}_{p_\ell}$
  swap $p_{\text{pivot}} \leftrightarrow p_m$
  set $\mathbf{L}_{p_m, m} := \sqrt{\mathbf{d}_{p_m}}$
  set $\mathbf{L}_{p_{m+1:n}, m} := \left[\mathrm{Cov}[f](\mathbf{x}_{p_{m+1}}, \mathbf{x}_{p_m}), \dots, \mathrm{Cov}[f](\mathbf{x}_{p_n}, \mathbf{x}_{p_m})\right]^\intercal$
  update $\mathbf{L}_{p_{m+1:n}, m} := \left(\mathbf{L}_{p_{m+1:n}, m} - \mathbf{L}_{p_{m+1:n}, 1:m-1} \cdot \mathbf{L}_{p_m, 1:m-1}^\intercal\right) / \mathbf{L}_{p_m, m}$
  update $\mathbf{d}_{p_{m:n}} := \mathbf{d}_{p_{m:n}} - \left[\mathbf{L}_{p_m, m}^2, \mathbf{L}_{p_{m+1}, m}^2, \dots, \mathbf{L}_{p_n, m}^2\right]^\intercal$
  update $\epsilon := \|\mathbf{d}_{p_{m:n}}\|_1$
  set $m := m + 1$
end while

Note that the matrix $\mathbf{C}_f$ does not need not be completely known for the algorithm to work. It is sufficient to just provide the elements that enter the computation of the Cholesky factors. Especially for large matrices, this greatly reduces the memory and computation requirements.

Exercise 1. Implement a Matlab function

```
function [L, p] = pivoted_cholesky(cov, xs, tol),
```

which calculates the pivoted Cholesky decomposition. The argument `cov` is a `function handle`, which evaluates a vectorized function $\mathrm{Cov}[f](\mathbf{x}, \mathbf{y})$. Note that the matrix $\mathbf{C}_f$ is never set up explicitly in the algorithm. The $(2 \times n)$-vector `xs` specifies the vertices of the mesh and `tol` is the termination accuracy.

Test your implementation with the following code, the result of which should be of the order of 10^{-10}:

```
cov = @(x , y) exp(-(x(1,:)-y(1,:)').^2-(x(2,:)-y(2,:)').^2);
[x1, x2] = meshgrid(linspace(0, 1, 101), linspace(0, 1, 101));
xs = [x1(:), x2(:)]';
C = cov(xs , xs);
[L , p] = pivChol(cov , xs , 1e-9);
disp(norm(C - L*L.', "fro"));
```

Once we have the low-rank approximation $\mathbf{C}_f \approx \mathbf{L}\mathbf{L}^\intercal$ at hand, we are going to solve $\mathbf{A}\mathbf{U}_{:,i} = \mathbf{M}\mathbf{L}_{:,i}$ for each $i = 1, 2, \dots, m$. Here, $\mathbf{A} \in \mathbb{R}^{n \times n}$ and $\mathbf{M} \in \mathbb{R}^{n \times n}$ denote the finite element stiffness and mass matrix, respectively. This leads us directly to the approximation

$$\mathbf{C}_u = [\mathrm{Cov}[u](\mathbf{x}_i, \mathbf{y}_j)]_{i,j} \approx \mathbf{U}\mathbf{U}^\intercal.$$

We hence avoided the solution of the computationally expensive problem (4) in the product domain $D \times D$.

Exercise 2. Consider the Gaussian covariance

$$\mathrm{Cov}[f](\mathbf{x},\mathbf{y}) = \frac{1}{\sqrt{2\pi\sigma^2}} \mathrm{e}^{\frac{-\|\mathbf{x}-\mathbf{y}\|_2^2}{2\sigma^2}} \tag{5}$$

with correlation length $\sigma = 0.4$. Generate a mesh by

```
mesh = mesh_generation().
```

The vertices of the mesh are then located in `mesh.P`. Compute the pivoted Cholesky decomposition with `tol=1e-9` and solve for each i the finite element problem $\mathbf{A}\mathbf{U}_{:,i} = \mathbf{M}\mathbf{L}_{:,i}$ by using

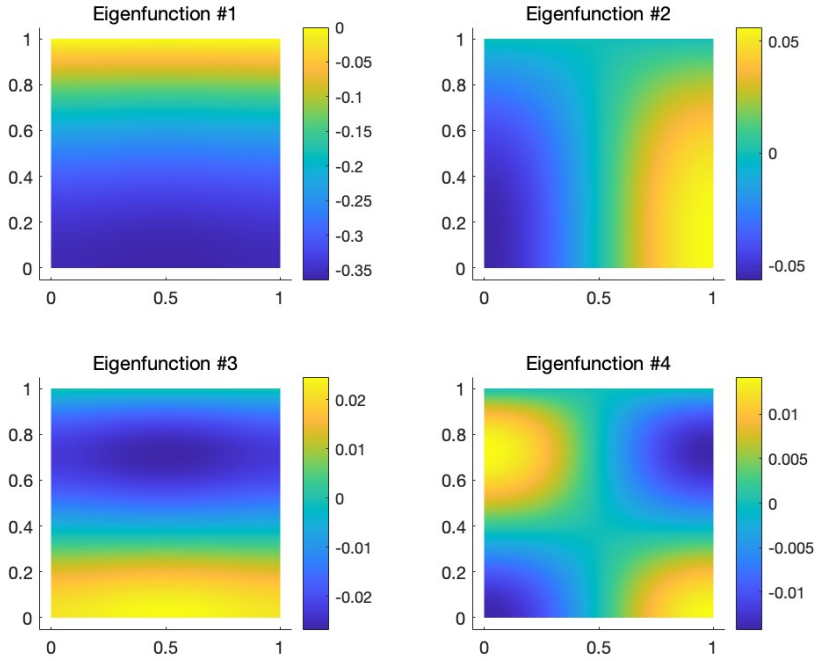
```
[uh,~] = solve_poisson_problem(lh, mesh, 1e-9).
```

Note that the multiplication of $\mathbf{L}_{:,i}$ with the mass matrix $\mathbf{M}\mathbf{f}_i$ is already included in the code. Plot the mesh function $\mathrm{diag}(\mathbf{U}\mathbf{U}^\intercal)$ by using

```
mesh_function_plot(diag(U*U'), mesh, 'none').
```

It corresponds to the variance of the Gaussian random field $u(\mathbf{x},\omega)$.

Simulation of random fields



In order to obtain the expansion (1), we need to find eigenpairs of the operator $\mathcal{C}_u$. The Galerkin discretization of the eigenvalue problem (2) yields a generalized eigenvalue problem

$$\mathbf{M}\mathbf{C}_u\mathbf{M}\mathbf{v} = \lambda\mathbf{M}\mathbf{v}. \tag{6}$$

We replace $\mathbf{C}_u$ by its low-rank approximation to arrive at

$$\mathbf{M}\mathbf{U}\mathbf{U}^\intercal\mathbf{M}\mathbf{v} = \mathbf{M}^{1/2}(\mathbf{M}^{1/2}\mathbf{U}\mathbf{U}^\intercal\mathbf{M}^{1/2})\mathbf{M}^{1/2}\mathbf{v} = \lambda\mathbf{M}\mathbf{v} \iff \mathbf{M}^{1/2}\mathbf{U}\mathbf{U}^\intercal\mathbf{M}^{1/2}\mathbf{v} = \lambda\mathbf{v}.$$

Since the (nonzero) eigenvalues of $\mathbf{M}^{1/2}\mathbf{U}\mathbf{U}^\intercal\mathbf{M}^{1/2}$ are the same as for $\mathbf{U}^\intercal\mathbf{M}\mathbf{U} \in \mathbb{R}^{m\times m}$, it remains to compute the eigenvalues of the small matrix $\mathbf{U}^\intercal\mathbf{M}\mathbf{U}$. As one readily verifies, the associated eigenvectors $\hat{\mathbf{v}}_1,\dots,\hat{\mathbf{v}}_m$ are related to the original ones by $\mathbf{v}_i = \mathbf{U}\hat{\mathbf{v}}_i$, $i = 1,\dots,m$. Finally, we normalize the eigenvectors: $\overline{\mathbf{v}}_i := \mathbf{v}_i/\sqrt{\mathbf{v}_i^\intercal\mathbf{M}\mathbf{v}_i}$.

Exercise 3. Compute the eigenvalues of the matrix $\mathbf{U}^\intercal\mathbf{M}\mathbf{U}$ by using the Matlab function `eig` in order to approximately solve the generalized eigenvalue problem (6). Obtain the approximate eigenpairs $\{\lambda_i, \mathbf{v}_i\}_{i=1}^m$. To get the matrix $\mathbf{M}$, use

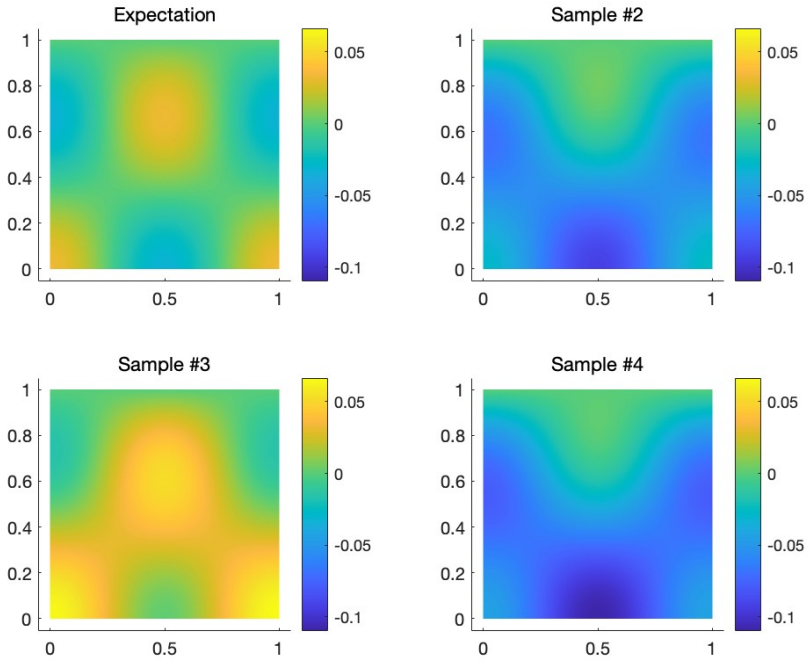
```
[M,~] = assemble_mass_and_stiffness(mesh).
```

Plot the first four eigenfunctions $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4$ by using the function `mesh_function_plot`.

We are now in the position to simulate the discretized random field by evaluating

$$\mathbf{u}_{\text{rand}} = \mathbf{u}_0 + \sum_{i=1}^{m} \sqrt{\lambda_i}\mathbf{v}_i r_i, \tag{7}$$

where $r_i \sim \mathcal{N}(0,1)$ are independent, standard normally distributed random numbers and $\mathbf{u}_0$ is the finite element solution of (3).



Exercise 4. Consider the right-hand side f with the Gaussian covariance (5) and the expectation

```
ff0 = @(x) 2*cos(2*pi*x(1))*cos(1.5*pi*x(2)).
```

We discretize this function by means of

```
fh0 = compute_nodal_interpolation(ff0, mesh).
```

and compute the expectation $\mathbf{u}_0$ by

```
[uh,~] = solve_poisson_problem(lh, mesh, 1e-9).
```

Finally, we determine the low-rank approximation of $\mathbf{C}_f$ by the pivoted Cholesky decomposition with `tol=1e-8` and compute the expression (7) by solving the generalized eigenvalue problem (6) as described before. Visualize $\mathbf{u}_0$ and three random realizations of $\mathbf{u}_{\text{rand}}$.