Project.

Finish until: **31. July 2025**

Problem formulation

In this project, we consider the Poisson problem in the random domain. To this end, let $D_r \subset \mathbb{R}^2$ be a bounded and simply connected domain with boundary Lipschitz boundary ∂D_r, which we call the *reference domain*. We consider a complete probability space $(\Omega, \mathcal{F}, \mathbb{P})$ with separable set Ω, σ-field $\mathcal{F} \subset 2^\Omega$, and probability measure $\mathbb{P}$. We then define $\mathbf{V} : \overline{D_r} \times \Omega \to \mathbb{R}^2$ as a random vector field such that $\mathbf{V}(\cdot, \omega) \in C^1(D_r; \mathbb{R}^2)$. The *random domain* is the image of the reference domain under the random domain mapping $\mathbf{V}$, i.e.,

$$D(\omega) = \mathbf{V}(D_r, \omega) \quad \text{and} \quad \partial D(\omega) = \mathbf{V}(\partial D_r, \omega).$$

We also define the respective hold-all as

$$\mathcal{D} = \bigcup_{\omega \in \Omega} D(\omega).$$

Thus, the Poisson problem in the random domain $D(\omega)$ reads as: for given $\mathbf{V} \in C^1(D_r; \mathbb{R}^2)^2$ and $f \in L^2(\mathcal{D})$, find $u(\omega) \in H^1_0\big(D(\omega)\big)$ satisfying

$$\begin{cases} -\Delta u(\mathbf{x}, \omega) = f(\mathbf{x}) & \text{for } \mathbf{x} \in D(\omega), \\ \quad u(\mathbf{x}, \omega) = 0 & \text{for } \mathbf{x} \in \partial D(\omega). \end{cases} \tag{1}$$

Modelling

We model the random domain mapping by means of the *Karhunen-Loève expansion*, i.e.,

$$\mathbf{V}(\mathbf{x}, \omega) = \mathbb{E}[\mathbf{V}](\mathbf{x}) + \sum_{k=1}^{\infty} \sqrt{\lambda_k} \boldsymbol{\phi}_k(\mathbf{x}) Y_k(\omega), \tag{2}$$

where $\{(\lambda_k, \boldsymbol{\phi}_k)\}_{k \in \mathbb{N}}$ are eigenpairs of the *covariance operator*

$$\mathcal{C}_{\mathbf{V}}\langle \mathbf{v} \rangle(\mathbf{x}) := \int_{D_r} \mathrm{Cov}[\mathbf{V}](\mathbf{x}, \mathbf{x}') \mathbf{v}(\mathbf{x}') \, \mathrm{d}\mathbf{x}.$$

Here, the *two-point covariance* defined as

$$\mathrm{Cov}[\mathbf{V}](\mathbf{x}, \mathbf{x}') := \mathbb{E}\Big[\big(\mathbf{V}(\mathbf{x}, \cdot) - \mathbb{E}[\mathbf{V}](\mathbf{x})\big) \otimes \big(\mathbf{V}(\mathbf{x}', \cdot) - \mathbb{E}[\mathbf{V}](\mathbf{x}')\big)\Big].$$

For the discretization we assume that the random variables $\{Y_k\}_{k \in \mathbb{N}}$ are independent and uniformly distributed in $[-\sqrt{3}, \sqrt{3}]$, the sequence $\{\|\sqrt{3\lambda_k}\boldsymbol{\phi}\|_{W^{1,\infty}(D_r;\mathbb{R}^2)}\}_{k \in \mathbb{N}}$ is at least in $\ell^1(\mathbb{N})$, and that the mean satisfies $\mathbb{E}[\mathbf{V}] = \mathbf{x}$. These model assumptions allow for truncation of Karhunen-Loève expansion for some $0 < M < \infty$ and after parametrization we obtain

$$\mathbf{V}(\mathbf{x}, \omega) \longrightarrow \mathbf{V}(\mathbf{x}, \mathbf{y}) = \mathbf{x} + \sum_{k=1}^{M} \sqrt{3\lambda_k} \boldsymbol{\phi}_k(\mathbf{x}) y_k, \quad \mathbf{y} = (y_1, \dots, y_M) \in \square := [-1, 1]^M.$$

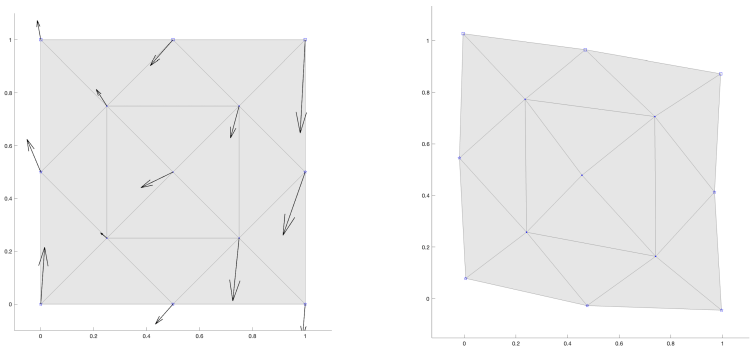

Figure 1: The square as reference domain with one sample of the shifted random map $\mathbf{V}(\mathbf{x}, \mathbf{y}^*) - \mathbf{x}$ and the asscoiated random domain.

Using this parametrization, we can rewrite the problem (1) in the random domain by

$$\begin{cases} -\Delta u(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}, \mathbf{y}) & \text{for } \mathbf{x} \in D(\mathbf{y}), \\ u(\mathbf{x}, \mathbf{y}) = 0 & \text{for } \mathbf{x} \in \partial D(\mathbf{y}), \end{cases} \tag{3}$$

where $D(\mathbf{y}) = \mathbf{V}(D_r, \mathbf{y})$ and $\partial D(\mathbf{y}) = \mathbf{V}(\partial D_r, \mathbf{y})$, compare Figure 1 for an illustration.

Exercise 1. Compute the random domain-mapping (2) by using the pivoted Cholesky decomposition. This needs an update of the pivoted Cholesky decomposition for the vector case. Implement the function

```
[Phi1, Phi2, D] = random_dom_map(kf, mesh_ref, tol),
```

where $\texttt{Phi}_i = [\phi_{i,k}]_{k=1,\dots,m}$, $i = 1, 2$, are matrices with the eigenfunctions and `D` is a matrix with the eigenvalues. The inputs are `kf`, which is a discretization of the matrix covariance function

$$\mathrm{Cov}[\mathbf{V}](\mathbf{x}, \mathbf{x}') = \frac{1}{100} \begin{bmatrix} 5\exp(-4\|\mathbf{x} - \mathbf{x}'\|\_2^2) & \exp(-0.1\|2\mathbf{x} - \mathbf{x}'\|_2^2) \\ \exp(-0.1\|\mathbf{x} - 2\mathbf{x}'\|_2^2) & 5\exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2) \end{bmatrix}, \tag{4}$$

`mesh_ref` is the mesh of the reference domain generated by `mesh_ref=mesh_generation()`, and `tol` is the desired tolerance for the pivoted Cholesky decomposition.

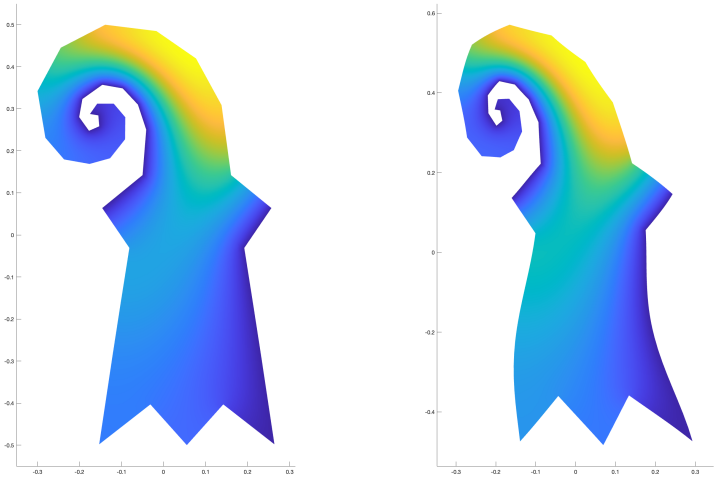

Figure 2: Solution $u_r(\mathbf{x})$ on the reference domain and solution $u(\mathbf{x}, \mathbf{y}^*)$ on the random domain.

Exercise 2. Given the reference domain D_r as `mesh_ref`, a sample of the random domain $D(\mathbf{y}^*)$ corresponds to a mesh `mesh_rand` by just mapping the vertices using the approximate Karhunen-Loève expansion. Solve the Poisson problem by

```
uh_rand = solve_poisson_problem(ff, mesh_rand)
```

with

```
ff = @(x) x(1) + 6*(x(2)+.5).^2.
```

Compare the solution $u(\mathbf{x}, \mathbf{y}^*)$ with the solution on the reference domain $u_r(\mathbf{x})$.

Quantity of interest

In the case of random boundary value problems, we are interested in *Quantities of Interest (QoI)* such as the expectation and the variance of the solution with respect to the reference domain. If the parameter domain $\square$ is equipped with the M-dimensional (normalized) Lebesgue measure μ^M, i.e., $2^{-M}\mu^M(\square) = 1$, these QoIs can be expressed as high-dimensional integrals

$$\mathbb{E}[u](\mathbf{x}) = 2^{-M} \int_{\square} u\big(\mathbf{V}(\mathbf{x}, \mathbf{y}), \mathbf{y}\big)\, \mathrm{d}\mathbf{y},$$
$$\mathbb{V}[u](\mathbf{x}) = 2^{-M} \int_{\square} u^2\big(\mathbf{V}(\mathbf{x}, \mathbf{y}), \mathbf{y}\big)\, \mathrm{d}\mathbf{y} - \big(\mathbb{E}[u](\mathbf{x})\big)^2.$$

We aim in this project on the approximation of these QoIs with the help of a *sparse grid quadrature*. We will base it on either the *composite trapezoidal rule*, the *Gauss-Legendre quadrature*, or the *Clenshaw-Curtis quadrature*. The composite trapezoid rule differs from the other two quadrature formulas in that it achieves higher accuracy by refinement (*h-convergence*), while the other two formulas achieve higher accuracy by increasing the degree of the polynomial exactness (*p-convergence*).

One-dimensional quadrature formulas

In all formulas below, we set $n = 2^{J-1} + 1$ for $J \in \mathbb{N}_{>1}$. For $J = 1$, we use the midpoint rule, i.e., $\xi_1 = 0$ and $w_1 = 2$.

Composite trapezoidal rule. We set quadrature points $\xi_k = (k-1)h - 1$ for $k = 1, \dots, n$ with $h = 2^{-J+2}$ and define the associated weights

$$w_k = \begin{cases} h, & \text{if } 1 < k < n, \\ h/2, & \text{otherwise.} \end{cases}$$

Gauss-Legendre quadrature. The computation of the points and weights of the one-dimensional Gauß-Legendre quadrature formula is performed using the *three-way recursion* of the *orthonormalized* Legendre polynomials, i.e,

$$u_{-1}(x) = 0, \quad u_0(x) = \frac{1}{\sqrt{2}}, \quad \frac{n+1}{\sqrt{4(n+1)^2 - 1}} u_{n+1}(x) = x u_n(x) - \frac{n}{\sqrt{4n^2 - 1}} u_{n-1}(x)$$

for $n = 0, 1, \dots$. The points and weights for the quadrature formula in zeros of orthonormalized polynomials with respect to the density $\rho(x)$

$$\beta_{k+1} u_{k+1}(x) = (x - \alpha_k) u_k(x) - \beta_k u_{k-1}(x)$$

are obtained from the eigenvalues or first entries of the corresponding eigenvectors of the Jacobi matrix

$$\mathbf{J}_n = \begin{bmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{n-1} \\ & & \beta_{n-1} & \alpha_n \end{bmatrix}.$$

If $\{(\lambda_k, \mathbf{v}_k)\}_{k=1}^n$ are the eigenpairs of $\mathbf{J}_n$, then the positions and weights are given by

$$\xi_k = \lambda_k \quad \text{and} \quad w_k = v_{k,1} \int_{-1}^{1} \rho(x)\,\mathrm{d}x.$$

In particular, in our case the weights are given by $w_k = 2v_{k,1}$. It is well known that the exactness of the Gauss-Legendre quadrature with n points is $2n-1$.

Clenshaw-Curtis quadrature. The points of the Clenshaw-Curtis quadrature formula are given by extremes of the Chebyshev polynomials

$$T_{n-1}(x) = \cos\big((n-1)\arccos(x)\big)$$

with respect to the boundary points, i.e.

$$\xi_k = \cos\left(\frac{(k-1)\pi}{n-1}\right) \quad \text{for} \quad k = 1, \dots, n.$$

The determination of the associated weights is a more complex than in the case of the Gauß-Legendre quadrature. There holds (see [3] for example)

$$w_k = \frac{1}{1-\xi_k^2}\frac{2\sin(\theta_{k-1})}{n-1}\sum_{s=0}^{n-3}\sin\left(\frac{(s+1)(k-1)\pi}{n-1}\right)\lambda_s \quad \text{for} \quad k = 2, 3, \dots, n-1,$$

where $\theta_k = k\pi/(n-1)$ and

$$w_1 = \frac{1}{2(n-1)}\left(\gamma_0 + 2\sum_{s=1}^{n-2}\gamma_s + \gamma_{n-1}\right), \quad w_n = \frac{1}{2(n-1)}\left(2\sum_{s=0}^{n-1}(-1)^j\gamma_s - \gamma_0 + (-1)^n\gamma_{n-1}\right).$$

Here

$$\gamma_s = \frac{1+\cos(\pi s)}{1-s^2} \quad \text{and} \quad \lambda_s = \frac{2\cos(\pi s)+2}{-s^3-3s^2+s+3}.$$

The implementation of these formulas in Matlab can be realized by the *discrete sine transformation* `dst`.

A notable advantage of the Clenshaw-Curtis quadrature is its efficiency when dealing with nested points, which leads to a significant reduction in the number of quadrature points required in high dimensions. However, this is at the expense of the polynomial exactness. For example, the Clenshaw-Curtis quadrature with n points has only the exactness $n-1$.

Exercise 3. Implement the one-dimensional quadrature formulas in the Matlab functions

```
function Q = init_tz(J),
function Q = init_gl(J),
function Q = init_cc(J),
```

which set up all points in the array `xi` and all weights in the cell-array `w` for the associated quadrature formulas. The points and weights should be stored in `Q`. Test your quadrature formulas by calculating integrals over polynomials.

Sparse grid quadrature

Let Q_J denote the square operator of any quadrature formula with n_J points, i.e.,

$$Q_J : C([-1,1]) \to \mathbb{R}, \quad Q_J f = \sum_{k=1}^{n_J} w_{J,k} f(\xi_{J,k}). \tag{5}$$

It applies for the composite trapezoidal rule and the Clenshaw-Curtis quadrature with $n_J = 1$ for $J = 1$ and $n_J = 2^{J-1} + 1$ for $J \geq 2$ as well as $n_J = J$ for the Gauß-Legendre quadrature.

A quadrature formula for functions in $C(\square)$ is obtained by tensorization

$$Q_J : C(\square) \to \mathbb{R}, \quad \mathbf{Q}_J f = (Q_J \otimes \cdots \otimes Q_J) f = \sum_{k_1=1}^{n_J} \cdots \sum_{k_M=1}^{n_J} w_{J,k_1} \cdots w_{J,k_M} f(\xi_{J,k_1}, \ldots, \xi_{J,k_M}).$$

It can be rewritten by means of the *difference quadrature formulas*

$$\Delta_J f := (Q_J - Q_{J-1}) f \quad \text{with} \quad Q_0 = 0.$$

in accordance with

$$\mathbf{Q}_J f = \sum_{\|\mathbf{j}\|_\infty \leq J} (\Delta_{j_1} \otimes \cdots \otimes \Delta_{j_M}) f.$$

The associated *sparse grid quadrature formula* is then obtained by restricting the index set $\{\mathbf{k} \in \mathbb{N}_{\geq 1}^M : \|\mathbf{j}\|_\infty \leq J\}$ of the tensor product formula to $\{\mathbf{j} \in \mathbb{N}_{\geq 1}^M : \|\mathbf{j}\|_1 \leq J\}$, i.e.,

$$\mathbf{Q}_J^{\mathrm{SG}} f = \sum_{\|\mathbf{j}\|_1 \leq J+M-1} (\Delta_{j_1} \otimes \cdots \otimes \Delta_{j_M}) f. \tag{6}$$

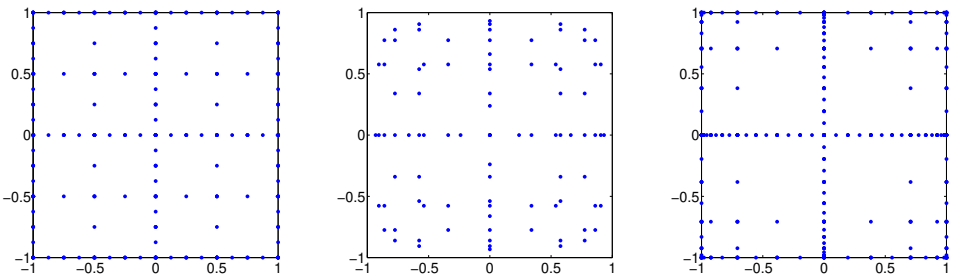

Figure 3: Set of quadrature points in 2D: trapezoidal rule (left), Gauß-Legendre quadrature (center), and Clenshaw-Curtis quadrature (right).

Implementation

A meaningful numbering of the multi-indices is required for the implementation of the sparse grid quadrature (6). A specific numbering is provided by the *droplet algorithm* listed below, compare [2]. However, the efficient implementation depends significantly on the number of required function evaluations. For the non-nested Gauß-Legendre quadrature, the *combination technique* can be used to evaluate the quadrature formula (6). Thus, only the one-dimensional quadrature formulas (5) are required, which are combined according to the formula

$$\mathbf{Q}_J^{\mathrm{SG}} f = \sum_{J \leq \|\mathbf{j}\|_1 \leq J+M-1} (-1)^{J+M-\|\mathbf{j}\|_1 - 1} \binom{M-1}{\|\mathbf{j}\|_1 - J} (Q_{j_1} \otimes \cdots \otimes Q_{j_M}) f. \tag{7}$$

With this formula, only the function value in zero is evaluated several times.

Algorithm. Droplet algorithm

Input: dimension M and level J
Output: matrix $\mathbf{K}$ with all permissible indices $\mathbf{j} = [j_1, \dots, j_M]$

set $\mathbf{j} = [1, \dots, 1]$
set $\ell = 1$
while $j_M \leq J$ **do**
  if $\|\mathbf{j}\|_1 > J + M - 1$ **then**
    set $j_\ell = 1$
    set $\ell = \ell + 1$
  else
    save $\mathbf{j}$ in $\mathbf{K}$
    set $\ell = 1$
  end
  set $j_\ell = j_\ell + 1$
end while

For the nested quadrature formulas, the evaluation is more complicated. In this case, we have

$$Q_J^{\mathrm{SG}} f = \sum_{\|\mathbf{j}\|_1 \leq J+M-1} \sum_{k_1 = n_{j_1} - m_{j_1} + 1}^{n_{j_1}} \cdots \sum_{k_M = n_{j_M} - m_{j_M} + 1}^{n_{j_M}} w_{\mathbf{j},\mathbf{k}} f(\boldsymbol{\xi}_{\mathbf{j},\mathbf{k}}), \tag{8}$$

see [1]. The numbers m_j correspond to the number of new quadrature points at level j, i.e. $m_j = n_j - n_{j-1}$, where we assume a hierarchical numbering of the quadrature points. Furthermore, the quadrature points are $\boldsymbol{\xi}_{\mathbf{j},\mathbf{k}} := (\xi_{j_1,k_1}, \dots, \xi_{j_M,k_M})$ and the quadrature weights are

$$w_{\mathbf{j},\mathbf{k}} = \sum_{\|\mathbf{j}+\mathbf{q}\|_1 \leq J+2M-1} v_{j_1+q_1,k_1} \cdots v_{j_M+q_M,k_M} \tag{9}$$

with $\mathbf{q} \in \mathbb{N}_{\geq 1}^M$ and

$$v_{(j+q),k} := \begin{cases} w_{j,k}, & \text{if } q = 1, \\ w_{j+q-1,k} - w_{j+q-2,k}, & \text{otherwise.} \end{cases}$$

Exercise 4. Write a Matlab function

```
function K = droplet(dim,lvl)
```

which realizes the droplet algorithm. Implement the functions

```
function SQ = sparseQuadrature(dim, lvl, Q),
function SQ = sparseQuadratureNested(dim, lvl, Q),
```

which evaluate the integral by the sparse grid quadrature formulas (7) and (8). The functions receive a level `lvl`, a dimension `dim` and an one-dimensional quadrature rule stored in `Q` as arguments.

Numerical experiments

Exercise 5. Consider the integral

$$\int_{[0,1]^M} (1+1/M)^M \prod_{k=1}^{M} x_k^{1/M} \, \mathrm{d}\mathbf{x} = 1.$$

Transform the integral to the domain $\square$ and approximate its value by using the sparse grid combination technique for $M = 5$ and $J = 1, \ldots, 6$. The numerical results from [1] should be used as a reference.

Exercise 6. Consider the problem (3) for $M = 5$ and with the reference domain, covariance and right-hand side from Exercise 1. Take `EU` from `data.mat` as the reference solution obtained by quasi-Monte Carlo quadrature with 10^5 samples. Determine the expected solution of the problem using the different quadrature methods introduced above for $J = 1, \ldots, 6$. Visualize the L^2-error of the respective expectations with respect to the reference solution against the number of solved differential equations in a `loglog` plot. The outcome should look like seen in Figure 4.

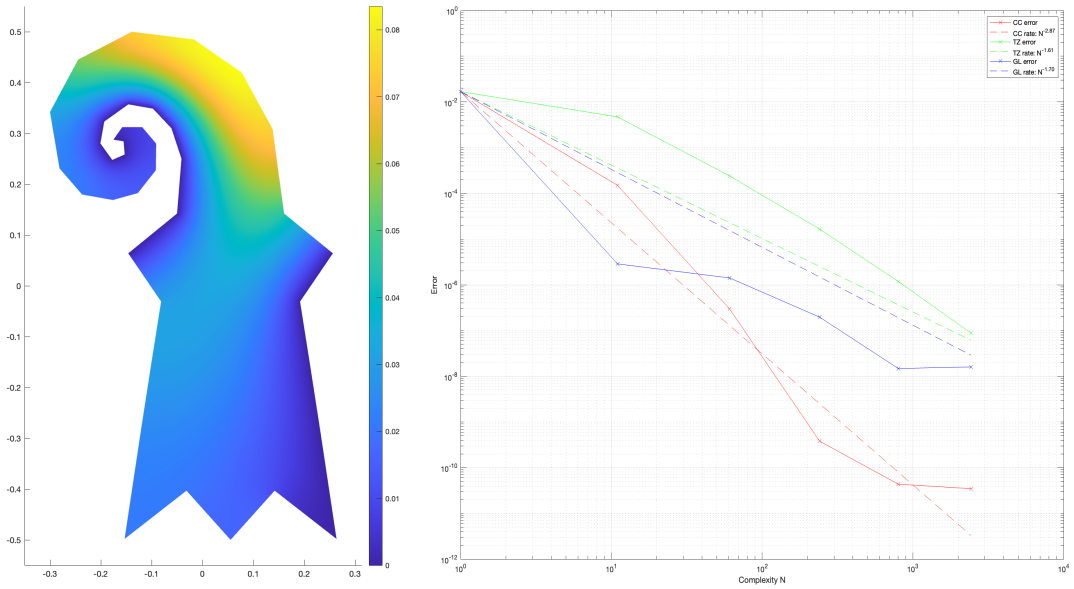

Figure 4: Reference solution (left) and convergence histories of the different quadrature methods (right).

References

[1] Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numer. Algorithms*, 18(3-4):209–232, 1998.

[2] Astrid Meyer (geb. Fischer). *Interpolation von vektorwertigen Funktionen mit adaptiven dünnen Gittern*. Diplomarbeit, Institut für Numerische Simulation, Universität Bonn, 2007.

[3] Alvise Sommariva. Fast Construction of Fejér and Clenshaw-Curtis Rules for General Weight Functions. *Comput. Math. Appl.*, 65(4):682–693, 2013.