

Scientific programming for interdisciplinary mathematics - Worksheet 4

Topics: Eigenvalue iterations, 2D quadrature, data structure for 2D FEM

Approximation of eigenvalues

Many applications (e.g. in structural engineering, physics) require the calculation of eigenvalues of operators or matrices. Here, we consider symmetric, positive definite matrices $A \in \mathbb{R}^{n \times n}$ and consider an eigenvalue problem of finding eigenpairs $(x, \lambda) \in \mathbb{R}^n \times \mathbb{R}$ such that

$$Ax = \lambda x.$$

Here, we consider an approach based on iterative procedures, so called **power iteration**, **inverse iteration** and **Rayleigh quotient iteration**.

idea of power iteration: Let $x_0 \in \mathbb{R}^n$ and sort the eigenvalues of A by $|\lambda_1| \geq |\lambda_2| \geq \dots$. As we have a basis of eigenvectors, we can write

$$x_0 = \sum_{i=1}^n \alpha_i v_i \quad \implies \quad Ax_0 = \sum_{i=1}^n \alpha_i Av_i = \sum_{i=1}^n \alpha_i \lambda_i v_i$$

and thus inductively for $\ell \geq 1$

$$A^\ell x_0 = \sum_{i=1}^n \alpha_i \lambda_i^\ell v_i = \lambda_1^\ell \sum_{i=1}^n \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^\ell v_i.$$

If the largest two eigenvalues are separated, there holds $\left| \frac{\lambda_i}{\lambda_1} \right|^\ell \rightarrow 0$ for $\ell \rightarrow \infty$ and thus the vector $A^\ell x_0 / \|A^\ell x_0\|$ converges to the first eigenvector. Once an (approximative) eigenvector is known, the corresponding (approximative) eigenvalue can be calculated with the Rayleigh quotient

$$x^T Ax = \lambda x^T x \quad \implies \quad \lambda = \frac{x^T Ax}{\|x\|_2^2} =: R(x)$$

Power iteration now is the iterative vector iteration $x_{\ell+1} = \frac{Ax_\ell}{\|Ax_\ell\|_2}$ with some starting vector $x_0 \in \mathbb{R}^n$ with $\|x_0\|_2 = 1$. In each step, this produces an approximation $\lambda_{\ell+1} = x_{\ell+1}^T Ax_{\ell+1}$ to the largest eigenvalue of A . *2 lines of codes*

Inverse iteration does power iteration for the matrix A^{-1} and thus approximates the smallest eigenvalue of A (why is that the case?). Thus, the iteration reads as $x_{\ell+1} = \frac{\hat{x}_{\ell+1}}{\|\hat{x}_{\ell+1}\|_2}$, where $\hat{x}_{\ell+1}$ is the solution of $A\hat{x}_{\ell+1} = x_\ell$. *method for smallest eval. we do the same procedure for A^{-1} we also normalize*

Rayleigh quotient iteration works similarly to inverse iteration, but solves the system $(A - R(x_\ell))\hat{x}_{\ell+1} = x_\ell$. *we shift A by approx eval. then we are closer to smith that converges fast*

Problem 1:

Write MATLAB functions that compute an approximation to an eigenpair (x, λ) with $Ax = \lambda x$ with power iteration, inverse iteration and Rayleigh quotient iteration:

- (a) `[x,lam] = powerIteration(A,x0,maxit)`
- (b) `[x,lam] = inverseIteration(A,x0,maxit)`
- (c) `[x,lam] = RayleighIteration(A,x0,maxit)`

Here, the input arguments are the matrix A , a starting vector x_0 and the number `maxit` $\in \mathbb{N}$ of iterations made.

Condition number

An important quantity for a matrix is its so called condition number. Let $A \in \mathbb{R}^{n \times n}$ and $\|\cdot\|$ be a matrix norm. Then, the condition number of A with respect to the norm $\|\cdot\|$ is given as

$$\kappa(A) := \|A\| \|A^{-1}\| \quad \rightarrow \text{like stability}$$

which measures the impact perturbations have on a linear system. For a symmetric, positive definite matrix, the spectral condition number (where the matrix norm is given as $\|A\|_2 := \max\{\lambda : \lambda \text{ is eigenvalue of } A\}$) can be calculated as

$$\kappa_2(A) := \frac{\lambda_{\max}}{\lambda_{\min}}, \quad \begin{array}{l} \leftarrow \text{Borne iterateur} \\ \leftarrow \text{Inverse} \end{array}$$

where λ_{\max} and λ_{\min} denote the maximal and minimal eigenvalue of A .

Problem 2:

Test your implementation from problem 1 on the matrix $A = \begin{pmatrix} 10 & 1 & 0 \\ 1 & 9 & 0 \\ 0 & 0 & 12 \end{pmatrix}$, the starting vector

$x_0 = (1, 1, 1)^T$. Calculate the eigenvalues of A using the MATLAB function `eigs`. To which eigenvalue does each iteration converge?

Plot the corresponding errors $|\lambda_{\text{maxit}} - \lambda|$ over the number of iterations made. Redo your calculation for the Rayleigh quotient iteration with $x_0 = (-1, 1, 1)^T$.

we don't know which eigenvalue we converge, so we do it for several vectors

Additionally, write a MATLAB function

```
coNum = conditionNumber(A,maxit)
```

cond₂(A) = $\frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$

↳ depends on the norm

that computes an approximation to the spectral condition number $\kappa_2(A)$ for a SPD matrix A .

Test your function by calculating an approximation to the condition number of the

```
A = gallery('poisson',k)
```

for different values $k \in \{10, 20, \dots, 100\}$ using power iteration and inverse iteration and plot the calculated condition numbers over k^2 .

2D quadrature formulas for rectangles

By exploiting the Fubini theorem, 1D quadrature formulas can be extended naturally to quadrature formulas for rectangles in 2D. Let

$$Q_n^{1D} g := \sum_{i=1}^n w_i^{1D} g(x_i^{1D}) \approx \int_0^1 g \, dx$$

be a 1D quadrature formula on $[0, 1]$. From Fubini's theorem, we obtain

$$\begin{aligned} \int_{[0,1]^2} f(x, y) \, d(x, y) &= \int_0^1 \int_0^1 f(x, y) \, dx \, dy \approx \sum_{j=1}^n w_j^{1D} \int_0^1 f(x, x_j^{1D}) \, dx \\ &\approx \sum_{i=1}^n \sum_{j=1}^n w_i^{1D} w_j^{1D} f(x_i^{1D}, x_j^{1D}). \end{aligned}$$

Hence,

$$Q_{n,n}^{R_{\text{ref}}} f := \sum_{\ell=1}^{n^2} w_{\ell} f(x_{\ell}) \stackrel{!}{=} \sum_{i=1}^n \sum_{j=1}^n w_i^{1D} w_j^{1D} f(x_i^{1D}, x_j^{1D}) \approx \int_{R_{\text{ref}}} f(x, y) \, d(x, y)$$

defines a 2D quadrature formula on the *unit square* $R_{\text{ref}} := [0, 1]^2$, where the quadrature node $x_{\ell} = (x_i^{1D}, x_j^{1D}) \in [0, 1]^2$ corresponds to the quadrature weight $w_{\ell} = w_i^{1D} w_j^{1D} \in \mathbb{R}$.

Problem 3 (Implement 2D quadrature on rectangles):

Use the Fubini theorem to derive a quadrature formula

$$Q_{n,n}^R f := \sum_{\ell=1}^{n^2} w_{\ell} f(x_{\ell}) \approx \int_R f \, d(x, y)$$

on a general rectangle $R := [a, b] \times [c, d] \subset \mathbb{R}^2$. Write a MATLAB function


`[x, w] = tensorQuadrature(x1D, w1D, a, b, c, d)`

that takes a 1D quadrature on $[0, 1]$ in terms of its nodes $\mathbf{x1D} \in [0, 1]^n$ with $x_j^{1D} = \mathbf{x1D}(j)$ and its weights $\mathbf{w1D} \in \mathbb{R}^n$ with $w_j^{1D} = \mathbf{w1D}(j)$. With $N := n^2$, the function should return the quadrature nodes $\mathbf{x} \in \mathbb{R}^{2 \times N}$ and weights $\mathbf{w} = (w_1, \dots, w_N) \in \mathbb{R}^{1 \times N}$ on $[a, b] \times [c, d]$, where $x_j = \mathbf{x}(:, j)$ is the node corresponding to the weight w_j .

Hint: Use the transformation theorem to reduce $\int_{[a,b] \times [c,d]} f(x, y) \, d(x, y)$ to $\int_{[0,1]^2} \tilde{f}(\tilde{x}, \tilde{y}) \, d(\tilde{x}, \tilde{y})$ and then proceed as outlined before. Equivalently, you can first derive 1D quadrature formulae on $[a, b]$ and $[c, d]$ and then proceed as outlined before.

Problem 4 (Validate 2D quadrature on rectangles):

Use the Gauss quadrature on $[0, 1]$ from last week to check your implementation from Exercise 3. Write a MATLAB script that performs the following tests:

Ex. $\int_{[0,1]^2} f \, d(x, y) = \int_0^1 \int_0^1 f \, dx \, dy$

 Apply our quadrature in 2D on $[0, 1]^2$
 $\int_0^1 \left(\int_0^1 f(x, y) \, dx \right) dy = \sum_{j=1}^n w_j^{1D} \int_0^1 f(x, y_j^{1D}) \, dx$
 iteratively apply 1D quadrature
 $\approx \sum_{i=1}^n w_i^{1D} \left(\sum_{j=1}^n w_j^{1D} f(x_i^{1D}, y_j^{1D}) \right) = \sum_{\ell=1}^{n^2} w_{\ell} f(x_{\ell})$

10 mins

- (a) Consider $f(x, y) = x^j y^k$ for $j, k \in \mathbb{N}_0$ and choose $n \in \mathbb{N}$ with $2n - 1 \geq \max\{j, k\}$. If your implementation is correct, on $R_{\text{ref}} = [0, 1]^2$ you should get the value

$$\int_{R_{\text{ref}}} x^j y^k \, d(x, y) = \int_0^1 x^j \, dx \int_0^1 y^k \, dy = \frac{1}{j+1} \frac{1}{k+1} = Q_{n,n}^{R_{\text{ref}}}(x^j y^k).$$

In addition, for any rectangle $R := [a, b] \times [c, d]$, your quadrature should return

$$Q_{n,n}^R 1 = (b-a)(d-c) = \int_R 1 \, d(x, y).$$

- (b) If you have validated your code, consider $f(x, y) = \exp(-(x^2 + y^2))$ on $R := [0, 1] \times [0, 2]$. With

$$Qf := \int_R f(x, y) \, d(x, y) \approx 0.658759669726125149 =: Q_{\text{ref}},$$

consider the error $e_n := |Q_{\text{ref}} - Q_{n,n}^R f|$. Plot e_n on the x-axis over n on the y-axis in a semilog-plot. What convergence rate $e_n = \mathcal{O}(e^{-\alpha n})$ do you observe? What happens for large n ?

going from an unit square \rightarrow unit triangle.

2D quadrature formulas on reference triangle

Suppose that we are given a quadrature formula on the unit square $R_{\text{ref}} = [0, 1]^2$, i.e.,

$$Q_{n,n}^{R_{\text{ref}}} g := \sum_{i=1}^n \sum_{j=1}^n w_i^{1D} w_j^{1D} g(x_i^{1D}, x_j^{1D}) \approx \int_{R_{\text{ref}}} g(x, y) \, d(x, y).$$

Let $T_{\text{ref}} := \{(x, y) \in \mathbb{R}_{\geq 0}^2 : x + y \leq 1\} = \text{conv}\{(0, 0), (1, 0), (0, 1)\}$ be the reference triangle. We consider the so-called *Duffy transformation* $\Phi(x, y) := (x, (1-x)y)$. One can show that $\Phi: R_{\text{ref}} \rightarrow T_{\text{ref}}$ is a diffeomorphism. Hence, the transformation theorem proves that

$$\begin{aligned} \int_{T_{\text{ref}}} f(x, y) \, d(x, y) &= \int_{R_{\text{ref}}} f(\Phi(x, y)) |\det D\Phi(x, y)| \, d(x, y) \\ &= \int_0^1 \int_0^1 f(x, (1-x)y) (1-x) \, dx \, dy \\ &\approx \sum_{i=1}^n \sum_{j=1}^n w_i^{1D} w_j^{1D} (1-x_i^{1D}) f(x_i^{1D}, (1-x_i^{1D})x_j^{1D}) =: Q_{n^2}^{T_{\text{ref}}} f, \end{aligned}$$

and we derive a quadrature formula $Q_{n^2}^{T_{\text{ref}}} f$ on the reference triangle T_{ref} . \rightarrow

Problem 5 (Implement 2D quadrature on reference triangle):

Write a MATLAB function

`[x, w] = DuffyQuadratureTref(x1D, w1D)`

\downarrow
we call what
we've done prev.
on a different (transformed)
P.

that takes a 1D quadrature on $[0, 1]$ in terms of its nodes $\mathbf{x}^{1D} \in [0, 1]^n$ with $x_j^{1D} = \mathbf{x}^{1D}(j)$ and its weights $\mathbf{w}^{1D} \in \mathbb{R}^n$ with $w_j^{1D} = \mathbf{w}^{1D}(j)$. With $N := n^2$, the function should provide the quadrature nodes $\mathbf{x} \in \mathbb{R}^{2 \times N}$ and weights $\mathbf{w} = (w_1, \dots, w_N) \in \mathbb{R}^{1 \times N}$ of $Q_{n^2}^{T_{\text{ref}}} f$, where $x_\ell = \mathbf{x}(:, \ell)$ is the node corresponding to the weight w_ℓ .

Problem 6 (Validate 2D quadrature on reference triangle):

Use the Gauss quadrature on $[0, 1]$ from last week to check your implementation from Exercise 5. Write a MATLAB script that performs the following tests:

- (a) Consider $f(x, y) = x^k y^j$ for $j, k \in \mathbb{N}_0$ on T_{ref} and note that

$$\begin{aligned} \int_{T_{\text{ref}}} x^j y^k \, d(x, y) &= \int_0^1 x^j \int_0^{1-x} y^k \, dy \, dx = \frac{1}{k+1} \int_0^1 x^j (1-x)^{k+1} \, dx \\ &= \frac{1}{k+1} Q_n^{1D}(x^j (1-x)^{k+1}), \end{aligned}$$

where the 1D integral on the right-hand side can be computed exactly with a 1D Gauss quadrature Q_n^{1D} with n nodes and $2n - 1 \geq j + k + 1$, i.e., $n \geq (j + k + 2)/2$.

- (b) If you have validated your code, consider $f(x, y) = \sin(2\pi(x + y))$. With

$$Qf := \int_{T_{\text{ref}}} f(x, y) \, d(x, y) \approx -0.15915494309189533577 =: Q_{\text{ref}},$$

consider the error $e_n := |Q_{\text{ref}} - Q_{n^2}^{T_{\text{ref}}} f|$. Plot e_n on the y-axis over n on the x-axis in a semilog-plot. What convergence rate $e_n = \mathcal{O}(e^{-\alpha n})$ do you observe?

→ now from reference triangle → general triangle

2D quadrature formulas on arbitrary triangles

Suppose that we are given a quadrature formula on the reference triangle T_{ref} .

$$Q_{n^2}^{T_{\text{ref}}} g := \sum_{\ell=1}^{n^2} w_\ell g(x_\ell) \approx \int_{T_{\text{ref}}} g(x, y) \, d(x, y).$$

Let $T := \text{conv}\{z_1, z_2, z_3\}$ be a triangle. Provided that T has positive area $|T| > 0$, the mapping $\Psi_T(x, y) := z_1 + x(z_2 - z_1) + y(z_3 - z_1)$ is an affine diffeomorphism $\Psi_T: T_{\text{ref}} \rightarrow T$. Then, the transformation theorem proves that

$$\int_T f(x, y) \, d(x, y) = \int_{T_{\text{ref}}} f(\Psi_T(x, y)) |\det D\Psi_T(x, y)| \, d(x, y).$$

With $z_j = (z_{j1}, z_{j2})$, we note that

$$\det D\Psi_T(x, y) = \det \begin{pmatrix} z_{21} - z_{11} & z_{31} - z_{11} \\ z_{22} - z_{12} & z_{32} - z_{12} \end{pmatrix} = \det \begin{pmatrix} 1 & 1 & 1 \\ z_{11} & z_{21} & z_{31} \\ z_{12} & z_{22} & z_{32} \end{pmatrix}.$$

In particular, we see that $|\det D\Psi_T(x, y)| = 2|T|$ is constant. Overall, we thus obtain

$$\int_T f(x, y) d(x, y) = 2|T| \int_{T_{\text{ref}}} f(\Psi_T(x, y)) d(x, y) \approx 2|T| \sum_{\ell=1}^{n^2} w_\ell f(\Psi_T(x_\ell)) =: Q_{n^2}^T f,$$

and this formula holds also if T degenerates, i.e., $|T| = 0$. In particular, we note that the area $|T|$ of the triangle can be computed by

$$|T| = \int_T 1 d(x, y) = \int_{T_{\text{ref}}} |\det D\Psi_T(x, y)| d(x, y) = \frac{1}{2} \left| \det \begin{pmatrix} 1 & 1 & 1 \\ z_{11} & z_{21} & z_{31} \\ z_{12} & z_{22} & z_{32} \end{pmatrix} \right|.$$

Problem 7 (Implement 2D quadrature on arbitrary triangles):

Write a MATLAB function

$$[\mathbf{x}, \mathbf{w}] = \text{DuffyQuadratureTriangle}(\mathbf{x1D}, \mathbf{w1D}, \mathbf{z})$$

that takes a 1D quadrature on $[0, 1]$ in terms of its nodes $\mathbf{x1D} \in [0, 1]^n$ with $x_j^{1D} = \mathbf{x1D}(j)$ and its weights $\mathbf{w1D} \in \mathbb{R}^n$ with $w_j^{1D} = \mathbf{w1D}(j)$ as well as $\mathbf{z} \in \mathbb{R}^{2 \times 3}$ specifying the vertices $z_j = \mathbf{z}(:, j) \in \mathbb{R}^2$ of a triangle $T = \text{conv}\{z_1, z_2, z_3\}$. With $N := n^2$, the function should provide the quadrature nodes $\mathbf{x} \in \mathbb{R}^{2 \times N}$ and weights $\mathbf{w} = (w_1, \dots, w_N) \in \mathbb{R}^{1 \times N}$ of $Q_{n^2}^T f$, where $x_\ell = \mathbf{x}(:, \ell)$ is the node corresponding to the weight w_ℓ .

understand how we transform ref triangle to a general Δ .

Problem 8 (Validate 2D quadrature on arbitrary triangles):

TESTING

Use the Gauss quadrature on $[0, 1]$ to check your implementation from Exercise 7. Write a MATLAB script to perform the following tests:

- (a) Consider test functions $f_1(x, y) = x^k$ and $f_2(x, y) = y^k$ to validate your implementation with

$$\mathbf{z} = \begin{pmatrix} 0.5 & 4.2 & 0.7 \\ 0.5 & 0.7 & 4.2 \end{pmatrix}.$$

Do you observe the same values in x and y ? The reference value for $k = 2$ is $Q_{\text{ref}} = 27.03837499999$. Permute the vertices and check whether $Q_{n^2}^T 1$ is independent of the order of the vertices!

- (b) Let $f(x, y) = \sin(2\pi(x + y))$ for $x, y \in T = \text{conv}\{(0.5, 0.5), (4.2, 0.7), (0.7, 4.2)\}$ with $Q_{\text{ref}} = -0.464018408541078$. Implement a MATLAB script that computes a vector \mathbf{Qf} of approximate values $\mathbf{Qf}(n) = Q_{n^2}^T(f)$. Compute the error $e_n = |Q_{n^2}^T(f) - Q_{\text{ref}}|$ and plot e_n on the y-axis over n on the x-axis in a semilog-plot. What rate $\mathcal{O}(e^{-an})$ do you observe?

usually data structure of 2D FEM is given by positions of vertices in 2D.



• elements $\{\{1,2,6\}, \{2,3,6\} \dots\}$
 set of combinations of vertices for each element
 • boundary edges like $(1,2)$
 (exterior) $(2,3)$ and etc.
 each of the integers are coordinates. coord $(\bar{x}_1, \bar{x}_2, \bar{x}_3)$

Data structure for 2D FEM

We consider conforming triangulations $\mathcal{T} = \{T_1, \dots, T_N\}$ of $\Omega \subset \mathbb{R}^2$, i.e., each $T_j \in \mathcal{T}$ is a compact triangle and, for $j \neq k$, the intersection $T_j \cap T_k$ is either empty, or a joint node, or a joint edge of both triangles T_j and T_k . The classical data structure to store such a triangulation internally reads as follows:

- The (double) matrix **coordinates** $\in \mathbb{R}^{M \times 2}$ stores all vertices of \mathcal{T} , i.e., $z_j = \text{coordinates}(j, :) \in \mathbb{R}^2$. By convention, all these vertices are pairwise disjoint (i.e., no vertex is stored multiple times)
- The (integer) matrix **elements** $\in \mathbb{N}^{N \times 3}$ stores the indices of the vertices of all elements, i.e., $T_\ell = \text{conv}\{z_i, z_j, z_k\}$ is stored as **elements** $(\ell, :) = (i, j, k) \in \mathbb{N}^3$. Usually, one employs the convention that z_i, z_j, z_k are stored in counterclockwise order (i.e., mathematically positive orientation). Clearly, each T_ℓ is stored only once in **elements**.
- Finally, the edges on the (Dirichlet) boundary $E_\ell \in \mathcal{E}^{\partial\Omega}$ are stored in an (integer) matrix **dirichlet** $\in \mathbb{N}^{D \times 2}$, where $E_\ell = \text{conv}\{z_i, z_j\}$ is stored as **dirichlet** $(\ell, :) = (i, j) \in \mathbb{N}^2$. Then, it holds $\partial\Omega = \bigcup_{\ell=1}^D E_\ell$, and each edge E_ℓ is stored only once in **dirichlet**. By convention, the order of z_i and z_j is chosen such that it is counterclockwise with respect to $\partial\Omega \supset E_\ell$.

Problem 9 (Getting along with 2D FEM data structure): \rightarrow playing around w data structures of this.
 Download from TUWEL the data files **Lshape-elements.dat**, **Lshape-coordinates.dat**, and **Lshape-dirichlet.dat**, which provide the conforming triangulation \mathcal{T} of some $\Omega \subset \mathbb{R}^2$. Visualize the domain (and the triangulation), where the edges of triangles are drawn in black. The edges on the boundary should be highlighted in red. To this end, you can use **plot** (or high-level functions like **trisurf**). Compute the area of Ω and the length of $\partial\Omega$ by means of

$$|\Omega| = \sum_{T \in \mathcal{T}} |T| \quad \text{and} \quad \text{length}(\partial\Omega) = \sum_{E \in \mathcal{E}^{\partial\Omega}} \text{length}(E).$$

\rightarrow summing up all the areas

Problem 10 (Quadrature for 2D FEM):

Use the data from Exercise **9** for the L-shape domain and your quadrature rule from Exercise **7** (based on Gauss quadrature on $[0, 1]$) to approximate the integral

$$\int_{\Omega} f(x, y) \, d(x, y) = \sum_{T \in \mathcal{T}} \int_T f(x, y) \, d(x, y) \approx \sum_{T \in \mathcal{T}} Q_{n^2}^T f =: Q_{N,n}^{\Omega} f$$

for $f(x, y) := \exp(-(x^2 + y^2))$ with $\int_{\Omega} f(x, y) \, d(x, y) \approx 1.673238856053101 =: Q_{\text{ref}}$. Download the MATLAB function **refine.m** from TUWEL. By calling

[coordinates, elements, dirichlet] = refine(coordinates, elements, dirichlet, 1:size(elements,1));
 \rightarrow no need to understand, it's tough.
 & we refine our structure so that as an output we get a triangulation that gives better approx.

you can refine your triangulation (i.e., each element $T \in \mathcal{T}$ is refined into four elements T' with area $|T'| = |T|/4$). For various $n \in \{1, 2, 3, 4, 5\}$, plot the quadrature error $e_N := |Q_{N,n}^{\Omega} f - Q_{\text{ref}}|$ on the y-axis over the number $N := \#\mathcal{T}$ of triangles on the x-axis. What rate $e_N = \mathcal{O}(N^{-\alpha})$ do you observe?

check the efficiency of
refine function.

Problem 11 (Getting along with MATLAB's sparse):

Use the triangulation $\mathcal{T}_1 := \mathcal{T}$ from TUWEL. Employ `refine` from TUWEL to generate a sequence of triangulations \mathcal{T}_j for $j = 1, \dots, 7$, where \mathcal{T}_j is the refinement of \mathcal{T}_{j-1} .

- (a) Use the MATLAB commands `tic` and `toc` to measure the runtime of the function `assemblyLaplace` found on TUWEL. Plot the runtime on the y-axis over the number of elements $N := \#\mathcal{T}_j$ on the x-axis in a loglog-plot. What growth of the computational time $= \mathcal{O}(N^\alpha)$ do you expect and what growth do you observe for this “slow” implementation.
- (b) Use the advanced properties of the MATLAB command `sparse` to modify the template `assemblyLaplaceFast` found on TUWEL. Use `tic` and `toc` to measure also the runtime of `assemblyLaplaceFast`. Add the plot of this runtime to the plot of Exercise 11(a). What growth of the computational time $= \mathcal{O}(N^\alpha)$ do you observe for each of the implementations?

Take-home messages

- (i) Quadrature rules on rectangles of the form $[a, b] \times [c, d]$ in 2D can be derived by exploiting the Fubini theorem. The quadrature nodes and weights can be computed by tensor products of 1D quadrature nodes and weights.
- (ii) Quadrature rules on the unit triangle in 2D can be derived from quadrature rules on the unit square $(0, 1)^2$ by exploiting the Duffy transformation. Furthermore, the quadrature rules on arbitrary triangles can be derived from the quadrature rules on the unit triangle by an affine transformation.
- (iii) An important starting point for the implementation of the \mathbb{P}_1 -FEM in 2D is the data structure for the triangulation of the domain. The data structure consists of the vertices, the elements, and the boundary. This provides all the things needed for the implementation of 2D FEM in the NumPDE course.

let's
start with C.

Outline

Finish the MATLAB exercises. Start to read the C-slides (found on TUWEL). In the coming four weeks, we are going to extend your knowledge on C programming to prepare you for parallel computing on the VSC:

- April 11: C slides 1–50 (variables, if-else, functions, recursion)
- May 2: C slides 51–89 (static arrays, loops)
- May 9: C slides 90–136 (pointers, dynamic arrays)
- May 16: C slides 137–159 (strings, structures)

Please read the C slides ahead of the respective lessons.

Problem 11 (Getting along with MATLAB's sparse):

Use the triangulation $\mathcal{T}_1 := \mathcal{T}$ from TUWEL. Employ `refine` from TUWEL to generate a sequence of triangulations \mathcal{T}_j for $j = 1, \dots, 7$, where \mathcal{T}_j is the refinement of \mathcal{T}_{j-1} .

- (a) Use the MATLAB commands `tic` and `toc` to measure the runtime of the function `assemblyLaplace` found on TUWEL. Plot the runtime on the y-axis over the number of elements $N := \#\mathcal{T}_j$ on the x-axis in a loglog-plot. What growth of the computational time $= \mathcal{O}(N^\alpha)$ do you expect and what growth do you observe for this “slow” implementation.
- (b) Use the advanced properties of the MATLAB command `sparse` to modify the template `assemblyLaplaceFast` found on TUWEL. Use `tic` and `toc` to measure also the runtime of `assemblyLaplaceFast`. Add the plot of this runtime to the plot of Exercise 11(a). What growth of the computational time $= \mathcal{O}(N^\alpha)$ do you observe for each of the implementations?

Take-home messages

- (i) Quadrature rules on rectangles of the form $[a, b] \times [c, d]$ in 2D can be derived by exploiting the Fubini theorem. The quadrature nodes and weights can be computed by tensor products of 1D quadrature nodes and weights.
- (ii) Quadrature rules on the unit triangle in 2D can be derived from quadrature rules on the unit square $(0, 1)^2$ by exploiting the Duffy transformation. Furthermore, the quadrature rules on arbitrary triangles can be derived from the quadrature rules on the unit triangle by an affine transformation.
- (iii) An important starting point for the implementation of the \mathbb{P}_1 -FEM in 2D is the data structure for the triangulation of the domain. The data structure consists of the vertices, the elements, and the boundary. This provides all the things needed for the implementation of 2D FEM in the NumPDE course.

Outline

Finish the MATLAB exercises. Start to read the C-slides (found on TUWEL). In the coming four weeks, we are going to extend your knowledge on C programming to prepare you for parallel computing on the VSC:

- April 11: C slides 1–50 (variables, if-else, functions, recursion)
- May 2: C slides 51–89 (static arrays, loops)
- May 9: C slides 90–136 (pointers, dynamic arrays)
- May 16: C slides 137–159 (strings, structures)

Please read the C slides ahead of the respective lessons.