1 Introduction

In this document, we will recapitulate our code and provide further explanations on selected excerpts of our code

2 Mesh

We start of with our mesh generator 'mesh_generate'. Our domain $\Omega = [0, 1]^2$ will not be subject to change for this task, therefore our only input will be the mesh size h. The result will be two matrices, namely *vertex_matrix* and *cell_matrix*. *vertex_matrix* will store the coordinates of each vertex, while *cell_matrix* only stores the numbers of the four vertices of each cell.
Example: $h = 0.5$

$$vertex\_matrix = \begin{pmatrix} 0 & 0 \\ 0.5 & 0 \\ 1 & 0 \\ 0 & 0.5 \\ 0.5 & 0.5 \\ 1 & 0.5 \\ 0 & 1 \\ 0.5 & 1 \\ 1 & 1 \end{pmatrix}, \ cell\_matrix = \begin{pmatrix} 1 & 2 & 4 & 5 \\ 2 & 3 & 5 & 6 \\ 4 & 5 & 7 & 8 \\ 5 & 6 & 8 & 9 \end{pmatrix}$$

3 Shape Functions

An essential concept for finite element methods would be shape functions. We have to set up certain points, so called *nodes* on a reference cell $\hat{T} = [0, 1]^2$ such that our shape functions p_i are uniquely defined by two properties:

1. $p_i(x_j) = \delta_{ij}, \ x_j$node

2. p_i is a polynomial on each edge.

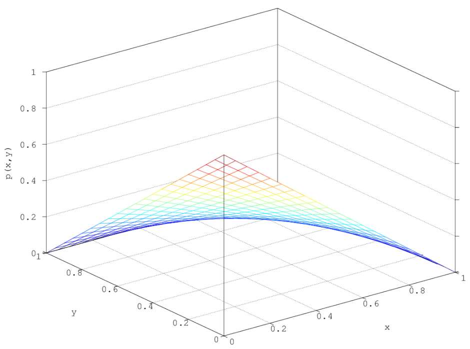
A way to achieve this, is to choose the location of the nodes in the same way as our mesh. Afterwards, we choose appropriate base-polynomials. If we want polynomials of degree k on the edges, the appropriate basis would be $\{x^i y^j, \ i, j = 0...k\}$. Although the resulting polynomials are of order $2k$, they are only of order k in each variable and therefore only of order k on each edge. A mesh generated with mesh-size $\frac{1}{k}$ will yield $k+1$ nodes on each edge; the interpolation is well-posed.

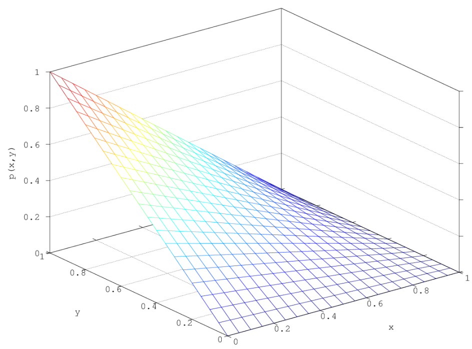
Remark: The way we order our basis has to be consistent for the entire code, because we only store the coefficients when we want to store a polynomial. In particular, this means that one entry of the coefficient vector must refer to the same polynomial basis element, even for different orders of polynomials. Effectively, this means that the basis for a higher order polynomial can only add basis elements at the end of any lower order polynomial basis.

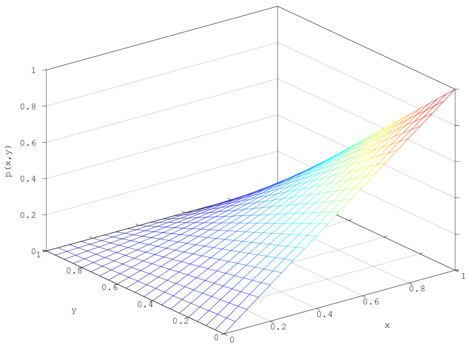
sf_generate follows this scheme closely. Since the shape functions only depend on the polynomial degree k we want to have on the edges, our only input will be the integer k. The function will create the nodes using *mesh_generate* with mesh-size $\frac{1}{k}$. Next, it will create a data-set, containing the coordinates and the value at these coordinates for each node. In total, we get $(k+1)^2 := n$ nodes, their values are computed using the Kronecker Delta. This will ensure, that property 1) holds true when interpolating over all these points. The interpolation is done by solving

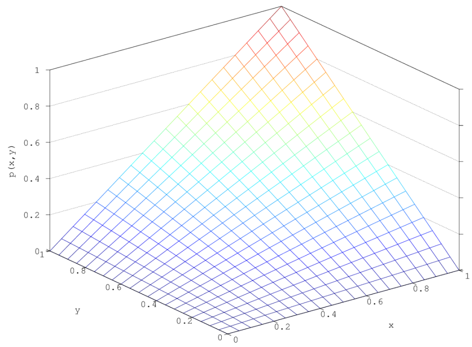
$$\begin{pmatrix} 1 & x_0 & \cdots & x_0^k y_0^k \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \cdots & x_n^k y_n^k \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} b_0 \\ \vdots \\ b_n \end{pmatrix}$$

The output of our function is the matrix SF, which stores the coefficient vectors of each shape function in its rows. Again, we see that a consistent numbering is needed when we want to evaluate a shape function. Let's look at some shape functions:
Example: Let the polynomial degree on the edges be 1. The four resulting shape functions:









You can clearly see, that each shape function has the value 1 at one vertex, and 0 at all the other vertices

4 Stiffness Matrix

Next up is the assembly of the stiffness matrix, this will be the left hand side of our linear system. The original idea behind the stiffness matrix was to build every combination of our basis functions in our bilinear form, namely $A_{ij} = a(\phi_i, \phi_j)$. As seen in the lecture, it is more efficient to compute a *local* stiffness matrix for each cell, and then insert these values in the right way in a global stiffness matrix. This approach proves to be even more efficient, considering that in our case, the local stiffness matrices are identical for each cell (*see also*: Theoretical Construct, Chapter 'Integrals and Transformations').

The computation of the local matrix is done by *sm_assemble_local*. As already shown, the computation will not depend on a specific cell, therefore we only need the mesh-size h and the shape function coefficients SF. Furthermore, we need several help functions which will compute the required integrals

- *hf_eval_poly* will evaluate a given polynomal at a given point, by using our specific polynomial basis and a coefficient vector
- *sf_derivate* will return the coefficient vectors for the derivative in x and y direction for a given polynomial.
- *int_gauss_weights* will return quadrature points and weigths for a given order and a given quadratic domain.
- *int_gauss* will evaluate a given function at the quadrature points and sum over the product of the quadrature weigths.

Note that the computation of the quadrature points and the actual quadrature is split up into two function. The transformations we use enable us to only integrate over the unit square. Therefore, we only have to compute the quadrature points and weights once. Since this operation is rather expensive, this will save alot of recources.
Having these functions at hand, the local matrix can be easily computed

In order to apply some of the results from the lecture, we need to derive the weak formulation of the given problem

$$\text{Find } u \in C^2(\Omega) : -\Delta u + u = \cos(\pi x)\cos(\pi y) \qquad \text{in } \Omega \tag{1}$$

$$\partial_n u = 0 \qquad \text{on } \partial\Omega. \tag{2}$$

Multiplying with an arbitary $v \in C^2(\Omega)$ and integrating over Ω gives us

$$-\int_\Omega \Delta u v \, \mathrm{d}\boldsymbol{x} + \int_\Omega uv \, \mathrm{d}\boldsymbol{x} = \int_\Omega fv \, \mathrm{d}\boldsymbol{x}$$

where $f = \cos(\pi x)\cos(\pi y)$ and $\boldsymbol{x} = (x, y)$. Using Green's first formula and (2) we can obtain the weak formulation

$$\int_\Omega \nabla u \nabla v \, \mathrm{d}\boldsymbol{x} + \int_\Omega uv \, \mathrm{d}\boldsymbol{x} = \int_\Omega fv \, \mathrm{d}\boldsymbol{x}$$

From now on, we will denote the left hand side of the equation by $a(u, v)$ and the right hand side by $F(v)$. Thus, we obtain the weak formulation

$$\text{Find } u \in H^1(\Omega) : \quad a(u, v) = F(v) \quad \forall v \in H^1(\Omega) \tag{3}$$

5 Existence and Uniqueness of a Solution

We can already see, that our bilinear form $a(\cdot\,,\cdot)$ is the inner product associated with the norm on our function space $H^1(\Omega)$. We want to use the Riesz representation theorem to prove existence and uniqueness of a solution. In order to do so, it remains to show that our functional $F(\cdot)$ is linear and bounded. Linearity follows from the properties of integration. Using Hoelder's inequality, we show that

$$\begin{aligned} F(u) &= \|fu\|_{L^1(\Omega)} \\ &\overset{\text{Hld.}}{\leq} \|f\|_{L^2(\Omega)} \|u\|_{L^2(\Omega)} \\ &\leq \|1\|_{L^2(\Omega)} \left(\|u\|_{L^2(\Omega)} + \|u\|_{L^2(\Omega)} \right) \\ &\leq c\, \|u\|_{H^1(\Omega)}, \end{aligned}$$

where c depends on our domain Ω. For our case, we have $\Omega = [0,1]^2$, in particular this means that Ω is bounded and our constant c is finite. Therefore, $F(\cdot)$ is a bounded, linear functional and we can apply the Riesz representation theorem.

6 Finding the Analytical Solution

Now that we know that a unique solution exists, we want to actually compute it. We will use the ansatz $u = C\cos(\pi x)\cos(\pi y)$, with its gradient $\Delta u = 2\pi^2 u$. Inserting in (1) gives us

$$-2C\pi^2\cos(\pi x)\cos(\pi y) + C\cos(\pi x)\cos(\pi y) = \cos(\pi x)\cos(\pi y) \tag{4}$$

$$\Rightarrow -2C\pi^2 + C = 1 \tag{5}$$

$$\Leftrightarrow C = \frac{1}{1-2\pi^2} \tag{6}$$

This leaves us with the solution $u = \frac{1}{1-2\pi^2}\cos(\pi x)\cos(\pi y)$.

7 Integrals and Transformations

The transformations we have to use for our FEM-code are very simple, since our mesh will be generated uniformly. Ultimately, we will only have to scale and displace our reference cell. We decided, that we will hard-code this property, since it will heavily simplifiy the computation of the local stiffness-matrices and the right-hand-side of the linear system, as we will see in brief.

Let T be a cell of the mesh with vertices v_1, v_2, v_3, v_4, where v_1 is the vertex on the bottom left and h is the length of every edge. $\hat{T} = [0,1]^2$ shall be our reference cell. Then, our transformation F will look like

$$F\colon \hat{T} \to T, \quad \hat{\boldsymbol{x}} = \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} \mapsto \begin{pmatrix} h & 0 \\ 0 & h \end{pmatrix} \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix} + v_1 := \begin{pmatrix} x \\ y \end{pmatrix} := \boldsymbol{x}$$

We immediately can see, that the jacobian J is given by

$$J(\hat{\boldsymbol{x}}) = J = \begin{pmatrix} h & 0 \\ 0 & h \end{pmatrix}, \tag{7}$$

in particular, it is independet of $\hat{\boldsymbol{x}}$. Consequently, we also get

$$|\det(J(\hat{\boldsymbol{x}}))| = |\det(J)| = h^2 \text{and } J^{-1} = h^{-1}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{8}$$

Using the results from the lecture on the transformation of our bilinear form $a(\cdot\,,\cdot)$ and using (7) and (8), we can compute the stiffness matrix in the following way

$$a_{ij}^{(T)} = \int_{\hat{T}} \nabla p_i \cdot \nabla p_j \,\mathrm{d}\hat{\boldsymbol{x}} + h^2 \int_{\hat{T}} p_i p_j \,\mathrm{d}\hat{\boldsymbol{x}},$$

where p_i and p_j are our shape functions.

Remark: Note that the ∇ in our formula refers to the gradient with respect to $\hat{\boldsymbol{x}} = \begin{pmatrix} \hat{x} \\ \hat{y} \end{pmatrix}$. *Remark:* Note that for affine transformations, the local stiffness matrices do not depend on the cell T, for which we want to compute it. Therefore, every local matrix looks the same and we only have to compute it once.

For the right-hand side, we have to look at the substitution formula for higher dimension integrals. But first of all, we need do split the intagral up. Let ϕ_i be a basis function of our domain Ω, $\mathfrak{T}_h$ our meshcells and $\mathfrak{T}_i$ the mesh cells, on which $\phi_i \neq 0$. Then we get

$$\begin{aligned}
\int_\Omega f(\boldsymbol{x})\phi_i(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} &= \sum_{T\in\mathfrak{T}_h} \int_T f(\boldsymbol{x})\phi_i(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \\
&= \sum_{T\in\mathfrak{T}_i} \int_T f(\boldsymbol{x})\phi_i(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \\
&= \sum_{T\in\mathfrak{T}_i} \int_T f(\boldsymbol{x}) p_{i^{(T)}}^{(T)}(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x} \\
&= \sum_{T\in\mathfrak{T}_i} \int_{\hat{T}} f(F^{(T)}(\hat{\boldsymbol{x}})\hat{p}_{i^{(T)}}(\hat{\boldsymbol{x}})|\mathrm{det}(J)|\ \mathrm{d}\hat{\boldsymbol{x}} \\
&= h^2 \sum_{T\in\mathfrak{T}_i} \int_{\hat{T}} f(h\hat{\boldsymbol{x}} + v_1^{(T)})\hat{p}_{i^{(T)}}(\hat{\boldsymbol{x}})\ \mathrm{d}\hat{\boldsymbol{x}}
\end{aligned}$$