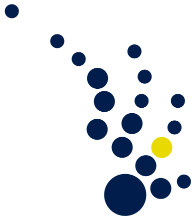

Scientific Computing I

Winter Semester 2013 / 2014
Prof. Dr. Beuchler
Bastian Bohn and Alexander Hullmann

Excercise sheet 1.

Closing date **22.10.2013**.

Theoretical exercise 1. (Polynomials on the reference triangle [5 points])

Consider the reference triangle given by

$$\hat{T} = \{(\xi_1, \xi_2) : 0 \leq \xi_1, \xi_2 \leq 1, \xi_1 + \xi_2 \leq 1\} .$$

We want to find polynomials ϕ_α on $\hat{T}$ that satisfy the following condition on the nodes $\hat{P}_\beta = \hat{P}(\xi^{(\beta)})$:

$$\phi_\alpha(\xi^{(\beta)}) = \delta_{\alpha\beta} = \begin{cases} 1 & \text{for} \quad \alpha = \beta \\ 0 & \text{for} \quad \alpha \neq \beta \end{cases} \tag{1}$$

for $\alpha, \beta \in \{1, 2, \dots, \hat{N}\}$.

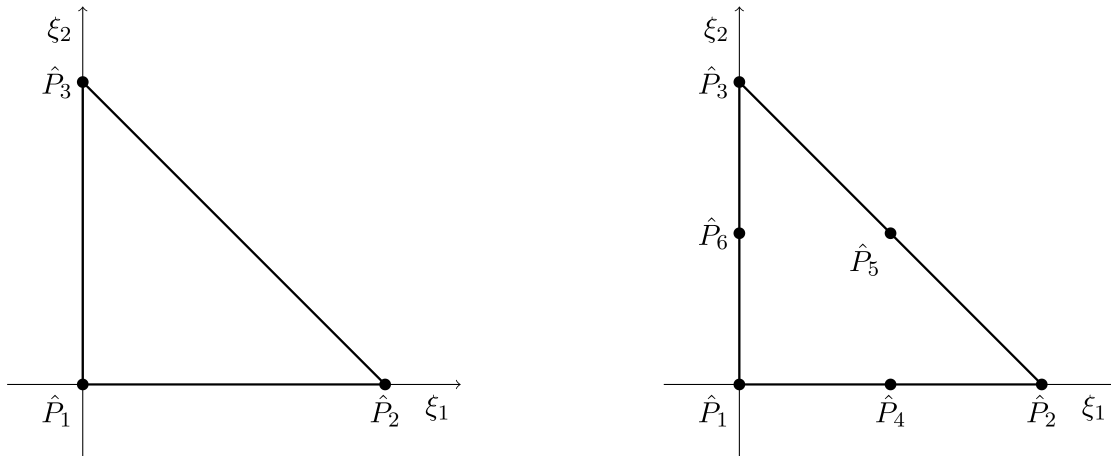

Figure 1: Reference triangles for linear (left) and quadratic (right) polynomials

a) Find linear polynomials that satisfy (1) with $\hat{N} = 3$ and

$$\xi^{(1)} = (0,0), \xi^{(2)} = (1,0), \xi^{(3)} = (0,1) ,$$

see the left triangle in Fig. 1.

b) Find quadratic polynomials that satisfy (1) with $\hat{N} = 6$ and

$$\xi^{(1)} = (0,0), \xi^{(2)} = (1,0), \xi^{(3)} = (0,1), \xi^{(4)} = (\tfrac{1}{2}, 0), \xi^{(5)} = (\tfrac{1}{2}, \tfrac{1}{2}), \xi^{(6)} = (0, \tfrac{1}{2}) ,$$

see the right triangle in Fig. 1.

c) Compute the partial derivatives of the polynomials with respect to ξ_1 and ξ_2.

General information on the programming exercises:

- The main tasks in these exercises will consider the construction of a two-dimensional Finite Element code during the semester. The programming exercises will guide you through the construction bit by bit.
- All exercises have to be programmed in C or C++. References and tutorials to these programming languages can be found on the homepage of this lecture.
- It is necessary that the prescribed function interfaces in the programming exercises are adhered to. The interfaces will be built such that they support C and C++. However, internally you should feel free to use e.g. the `vector` class from the STL (`http://www.cplusplus.com/reference/stl/`) as long as the function interfaces remain the predefined ones.
- Wasteful use of runtime and memory leaks will lead to fewer points awarded.
- Mail completed exercises (compilable source code) to your tutor before the closing date.

Programming exercise 1. (preconditioned CG with CSR matrices [10 points])

CSR matrix

An efficient format to store sparse (i.e. many zero entries) matrices is the CSR (**c**ompressed **s**parse **r**ow) format, see e.g. `http://netlib.org/utk/papers/templates/node91.html`. Here, a matrix is defined by the three arrays

- `Values`,
- `ColInd`,
- `RowPtr`.

The array `Values` has size `#non-zero values in the matrix`. It contains the non-zero values in the matrix. The ordering is given by a row-wise traversal of the matrix.

The array `ColInd` has size `#non-zero values in the matrix`. It contains the column indices of the entries in `values`.

The array `RowPtr` has size `#rows of the matrix`. The i-th entry contains the position in the `Values` array (beginning by index 0) where the first entry of the i-th row is found. If there only exist zero values in the i-th row, it contains the position in the `Values` array where the first entry of the $i + j$-th row is found, where $j \in \mathbb{N}$ is the smallest number such that the $i + j$-th row has non-zero entries. If there exists no such j then `#non-zero values in the matrix` is stored there.

Note that in the link given above the indexing of the row pointer goes from 1 to `#non-zero values in the matrix`+1 instead of 0 to `#non-zero values in the matrix`. For your implementation you can use either indexing.

Conjugate gradient method

The conjugate gradient method is a very common and successfully used iterative solver for systems of linear equations (see e.g. `http://en.wikipedia.org/wiki/Conjugate_gradient_method`). To achieve better convergence results for ill-conditioned systems $Ax = b$ a preconditioning can be done. In our case we will use the inverse of the diagonal of A as preconditioning matrix. This has been introduced in Algorithmische Mathematik I/II for example. A code for a preconditioned CG-algorithm on

CSR matrices was implemented during the exercises there. It can be found here: `http://www.ins.uni-bonn.de/teaching/vorlesungen/WissRechWs13/` (vector.h and LGSiterativ.h) Feel free to re-use this (or your own) code to complete the following exercises:

Tasks:

a) [3 points] Implement a class/struct `CSRMatrix`. It consists of the following information

- number of rows
- number of columns
- number of non-zero entries
- array/vector of non-zero entries (see `Values`)
- array/vector of column indices (see `ColInd`)
- array/vector of row pointer (see `RowPtr`).

Furthermore the following (member) functions have to be implemented:

- `int getNumRows()` – returns the number of rows
- `int getNumCols()` – returns the number of columns
- `void setEntries(const int* is, const int* js, const double* entries, int len)` – sets the values, column indices and row pointer of the CSRMatrix according to the row index in `is`, the column index in `js`, the values in `entries`. The entries are assumed to be already sorted in a row-wise fashion (lexicographically by index). `len` gives the length of the three parameter arrays.
- `void printMatrix()` – prints out the non-zero entries (e.g. 1 1), the entries of the row pointer (e.g. 0 1 2) and the number of non-zero entries (e.g. 2). Then the method prints out the matrix as a full matrix, i.e. with zero entries in a standard format (e.g. $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$).

b) [7 points] Implement the following member functions:

- `void multiplyInto(const double* x, int lengthx, double* y, int lengthy)` – calculates $Ax = y$ for the CSRMatrix A and a given vector `x`. The vector `y` contains the result.
- `int pcCG(double* x const double* b, int maxIt, double eps)` – calculates the solution of $Ax = b$ for the CSRMatrix A and the right hand side vector `b` by a diagonally preconditioned conjugate gradient algorithm. `maxIt` contains the maximum number of iterations and `eps` contains the relative accuracy that is demanded. The vector `x` contains the starting vector and should contain the solution at the end.

Test your CSRMatrix class:

- Write a main function which creates a CSRMatrix to represent the 4×3 matrix

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

  and call your `printMatrix()` routine. Check if everything is working correctly.

- Now create a CSRMatrix to represent the 10×10 matrix

$$A := \begin{pmatrix} 4 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 16 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 64 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 256 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1024 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4096 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 16384 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 65536 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 262144 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1048576 \end{pmatrix}$$

  and solve the system $Ax = b$ for $b = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1)^T$ with pcCG for starting vector $x = \vec{0}$ with `maxIt`=100 and `eps`=10^{-13}. Print out the solution and the number of iterations.

You can find an incomplete source code on the website of this lecture. You can implement the functions there or write your own C/C++-code.