Programmiermethoden des Wissenschaftlichen Rechnens

Winter semester 2018/2019
Prof. Dr. Marc Alexander Schweitzer
Clelia Albrecht and Albert Ziegenhagel

UNIVERSITÄT BONN

Exercise sheet 3.

Exercise 24. (Linear algebra in C++)

a) Consider the linear algebra types (`CsrMatrix`, `Vector`) and operations from the last exercise. Transfer all the functionality into modern `C++` code.

b) Explain the differences between your `C` and `C++` implementation. In particular elaborate on:

- Your method of construction, use of members, types, operators and memory management.
- Do you need an explicit destructor? Why?
- How do you access data members of your classes? What are the benefits and potential drawbacks of your implementation?
- How robust is your implementation against potential future changes (e.g. a different storage format)?

c) Write a CMake for your project that includes:

- Compiling the linear algebra types and operations into a library.
- Build a number of executables where each represents a reasonable test for the functionality provided in the library.

Exercise 25. (Iterative solvers for sparse matrices)

We want to write a small solver library that uses the linear algebra library from exercise 24. To this end:

a) Implement the Richardson and Conjugate-Gradient (CG) method, both utilizing either a Gauss-Seidel or Jacobi preconditioner (`https://web.stanford.edu/class/cme324/saad.pdf`).

b) Think of an interface to solve an equation system with arbitrary combination of solver and preconditioner from above. Can you implement this without any code duplication? How many changes would be necessary if your functions need to be extended to support another type of solver and/or preconditioner?

c) Extend your CMake file to generate another library for the solvers that uses the linear algebra library.

d) Write additional tests for the solvers.

Exercise 26. (B-Spline curves)

Read the introduction to B-Splines at e.g. `http://ftp.cs.wisc.edu/Approx/bsplbasic.pdf`. The notation used in the following is that a B-Spline curve $C(u)$ of degree p is represented by a node vector $U = \{u_0, \dots, u_m\}$ and two dimensional control points $\{P_i\}$, $i = 0, \dots, n$.

a) Write a 2-dimensional B-Spline class that needs to support:

- Construction of a B-Spline curve of degree p from a given knot vector U and control points P_i.
- Evaluation of the curve at a parameter u.
- Evaluation of the curve at n uniformly distributed points and write the results into a simple text file (one pair of x, y values per line).

b) Explain the design of your class with respect to at least the following points:

- How do you manage memory within your spline class?
- What operators may be overloaded for the class? How does that influence the interface of the class? What implications does that have on future changes of the interface, the implementation or the addition of additional features to your class?
- How strong does your implementation depend on the restriction to two space dimensions?
- How do you handle the 2-dimensional control points? What are the benefits of your representation? What other ways would be possible to represent those points? What are the benefits/drawbacks compared to your choice?

c) Write a small Python script that reads the x, y data from the file and plots the linear approximation to the curve via *matplotlib*.

d) Test your B-Spline class with at least the following data: $p = 2$, $U = \{0.0, 0.0, 0.0, 0.3, 0.5, 1.0, 1.0, 1.0\}$, $\{P_i\} = \{(0, 1), (1, -0.2), (3, 2), (6, 1.3), (7, 1.5)\}$.

Exercise 27. (B-Spline interpolation)

Given a number of points $\{Q_k\}$, $k = 0, \ldots, n$. We now want to write an algorithm to create a B-Spline curve $C(u)$ that interpolates those points at specific locations $C(\bar{u}_k) = Q_k$. To this end:

a) Create your interpolation parameters as $\bar{u}_0 = 0$, $\bar{u}_n = 1$, $\bar{u}_k = \bar{u}_{k-1} + \frac{|Q_k - Q_{k-1}|}{d}$ with $d = \sum_{k=1}^{n} |Q_k - Q_{k-1}|$.

b) Choose your knot vector $U = \{u_0, \ldots, u_m\}$ as $u_0 = \cdots = u_p = 0$, $u_{m-p} = \cdots = u_m = 1$, $u_{j+p} = \frac{1}{p} \sum_{i=j}^{j+p-1} \bar{u}_i$, $j = 1, \ldots, n - p$.

c) Set-up a system of linear equations to solve $C(\bar{u}_k) = \sum_{i=0}^{n} N_{i,p}(\bar{u}_k) P_{i,r} = Q_{k,r}$ where $N_{i,p}$ are the B-Spline basis functions, P_i are the unknown control points and r is the index of the space component. Please note that the matrix representing the equation system has at most $p + 1$ non-zero entries per row.

d) Solve the systems of linear equations with one of the solvers from the previous exercise and create a B-Spline curve from the resulting knot vector and control points.

e) Use the interpolation points from the file `b_spline_interpolation.txt` (to be downloaded from the course website) to test your implementation. Use $p = 3$.

f) Plot the curve using *matplotlib* via the approach from the previous exercise.