Dr. Ole Klein
M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 23, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 1

Exercise 1.1 [Working with the C++ compiler]

Take some time to get familiar with the compiler output.
The output for errors and warnings is structured as follows:

[file]:[line]:[position]: [type]: [description of the error or warning]

The output of all warnings can be activated bei adding -Wall (Warnings all) to the compiler call:

```
>> g++ -Wall wrong.cc -o wrong
```

Fix all errors and warnings in the file "wrong.cc". The file should compile without errors or warnings

```
#include <iostream>

namespace local
{
  int answer() { int answer = 42; return; }
}

main()
{
  int dummy = 0;

  double a; a = 1.0;
  double const b; b = 2.0;
  double c = a * b;

  std::cout << "The answer is: " << answer() << "\n";
  std::cout << "a = " << a << \n;
  std::cout << "b = " < b << "\n"
  std::cout << "a * b =  << c << "\n";

} // main
```

Exercise 1.2 [Factorial]

Write a program `factorial.cc`, that computes computes for an input `int n` the factorial

$$n! = n \cdot (n-1) \cdot \ldots \cdot 1$$

The following commands will compile and run the file.

```
>> g++ factorial.cc -o factorial
>> ./factorial
```

Test your program for different values of `int n`. What do you notice? change your program for, such that it uses only `unsigned int` and test your program again.

Exercise 1.3 [Euler's Constant]

Euler's constant is defined by the infinite sum

$$e = \sum_{k=0}^{\infty} \frac{1}{k!} = 2.718281828459045\ldots.$$

This formula allows us to compute the value of e to arbitrary precision. Using $e_n := \sum_{k=0}^{n} \frac{1}{k!}$ we can compute the precision with $e_n - e_{n-1}$.

Write a program/function, that computes the approximation of e for a given integer n and print that number on the screen. Make sure to efficiently compute the values of $k!$ by using the previous value.

(a) Compute e_n for $n = 5, 10, 20$, by summing upwards ($k = 0, 1, 2, \ldots$) resp. downwards ($k = n, n-1, n-2, \ldots$). Compare the results and precision. Make sure to use the right data types

(b) Change your program/function, such that the computations can also be done with `float` and compare the results. Furthermore, compute the error to `std::exp(1.0)` from the standard library `<cmath>`.

(c) Optional: Write a program/function, that approximates e to a given precision `tol`.

Bonus Problem 1.4 [Binomial coefficient]

Write a program `binomial.cc`, that computes for specific inputs $k, n \in \mathbb{N}$ the binomial coefficient

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Use the data types `unsigned int` and treat possible exceptional cases ($k > n$?) with `if` conditions. Test your program intensively.

The allowed range for $n, k \in \mathbb{N}$ can be extended by **not** using all factorials:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

Furthermore, the efficiency can be enhanced by exchanging k with $min(k, n-k)$, due to the symmetry of the binomial coefficient.
Use the above aspects to improve your program and compare your results with the first version.

Optional: Compute the binomial coefficient for $n, k \in \mathbb{N}$ using the product

$$\binom{n}{k} = \prod_{j=1}^{k} \frac{n+1-j}{j}.$$

This product can be implemented using a loop that switches between multiplication and division, **without** a division remainder. Why?

Bonus Problem 1.5 [Greatest common divisor]

To compute the greatest common divisor (gcd) of two numbers, the euclidian algorithm can be used:

> Start by dividing the bigger by the smaller number. After that divide the smaller number by the remainder until the remainder is zero. The last divisor is the gcd. If the remainder is 1 both numbers don't have a common divisor.
> The following example illustrates this algorithm with the numbers 13575 and 345:

- $13575 = 39 \cdot 345$ remainder 120
- $345 = 2 \cdot 120$ remainder 105
- $120 = 1 \cdot 105$ remainder 15
- $105 = 7 \cdot 15$ remainder 0

The greatest common divisor is 15.

Implement this algorithm for two positive integers in C++ and test it on some examples.

(a) Implement the function
- `unsigned int gcD(unsigned int, unsigned int)`. Use the `%`-Operator for divisions with remainder.

(b) Write a `main`-method, that tests your function on the given example. Output the solution to the terminal.

Bonus Problem 1.6 [Bisection algorithm]

The bisection algorithm is used to find the root of a continuous function $f\colon \mathbb{R} \to \mathbb{R}$ in the interval $[a, b] \subset \mathbb{R}$. It works as follows:

If $f(a)$ and $f(b)$ have different signs there exists a root in $[a, b]$. If you seperate the interval into two equal parts $[a, m]$ and $[m, b]$ by using $m = \frac{a+b}{2}$, the root has to be in one of the two intervals. If $f(m) = 0$ we have found the root. Otherwise we have to check for the signs again. If $f(a) \cdot f(m) < 0$ we set $b = m$ and start from the beginning. Otherwise if $f(b) \cdot f(m) < 0$ we set $a = m$ and start from the beginning.

Write a program that consist of the following parts
- a function `f`, for computing the value $f(x) = e^{\sin(x)} - 0.5$
- a function `bisec`, for Finding the root of $f(x)$ on an interval given by a and b using the described algorithm
- a `main`-method.

The (iterative) implementation should terminate if
- a maximal number of iterations `max_iter` has been reached
- for the given tolerance `tol`, $f(m) <$`tol` holds

Print the values for a and b in every iteration to the terminal. Test your program with the accuracy `tol`$= 10^{-6}$ for the interval $I := [-1, 1]$ and verify your results.

Afterwards change your functon `f` and determine the roots of the following functions on the same interval
- $f(x) = \frac{1}{\sqrt{x^2+1}} - \cos(x + 1)$,
- $f(x) = \sin(\cos(x + 1))$,
- $f(x) = \arctan(x^2 + 2x) - 0.2$,
- $f(x) = x^2 + 1$.

Dr. Ole Klein
M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 24, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 2

Exercise 2.1

The exponential function $\exp(x)$ can be seen as a power series for $x \in \mathbb{R}$. The recursive formula for the computation reads

$$\begin{aligned} y_1 &:= x, & f_1 &:= 1 + y_1, \\ y_n &:= \frac{x}{n} y_{n-1}, & f_n &:= f_{n-1} + y_n. \end{aligned}$$

Write a program `powerseries.cc` which computes the approximation of $\exp(x)$ for given point x and iteration count n. The datatype for x should be variable (i.e. `double` or `float`). User input shall be over the command line. Please ask for x, n and the datatype.

(a) Test your program with $x = 5$ and $x = -10$ for 100 iterations and different datatypes. Print the difference between the exact value of `std::exp` and your approximation f_n

$$e_n := |\exp(x) - f_n|$$

(b) especially for values $x \ll 0$ the result is off by many orders of magnitude. Use the properties of the exponential function and change the algorithm to get a smaller error. Test this with $x = -20$ and `float`.

Hints:

- The exact value of $\exp(x)$ can be approximated by `long double`:
  `long double exact = std::exp(x);`
- $\exp(-x) = \exp(x)^{-1}$ might be helpful

Exercise 2.2

Consider the tent map

$$f\colon [0,1] \to [0,1], \quad x \mapsto \begin{cases} 2x, & \text{for } x \in [0, 0.5) \\ 2 - 2x, & \text{for } x \in [0.5, 1] \end{cases}.$$

This is a simple example for a non-linear mapping for a dynamical system. For $x_0 \in [0,1]$ define $(x_i)_i$ by

$$x_i := f(x_{i-1}), \quad i \in \mathbb{N}. \tag{1}$$

The tent map is a chaotic system, i.e. little changes in the initial value lead to large effects to later values and the values reached seem unpredictable. However, this is invalid if x_0 can be represented as finite binary number. Thus, we can not reproduce this behavior on the computer.
During this exercise we examine the tent map for finite binary numbers and how to implement the desired chaotic behavior.

(a) Write a program `tent_map.cc`, that computes the sequence $(x_i)_i$ for $x_0 = 0.01401$ and $i = 1, \ldots, 100$ and print the results in the terminal. It should look like the following:

```
$ g++ -o tent_map tent_map.cc
$ ./tent_map
0.01401
0.02802
...
```

Write the values in a file `data.dat` by passing the terminal output to this file:

```
$ ./tent_map > data.dat
```

Visualize the data using `gnuplot`. Therefore, start `gnuplot` in the same directory of the data `data.dat` in the terminal with the command `gnuplot`. Afterwards visualize the data using the command

```
plot 'data.dat'
```

(b) The results are chaotic in the beginning, however, we can see that the values form a pattern in the end. If $x_0 = (0.m_1 \dots m_r)_2 \in [0,1]$ is a fixed-point number in binary representation with at most r non-zero decimals and (x_i) defined by (1) then it is possible to show that $x_{r+1} = 0$.

The proof illustrates that for irrational initial values $x_0 \in [0,1]$ the sequence $(x_i)_i$ will be non-periodic. However, this will not help us with our programming task. To achieve non-periodically sequences with finite binary representations, we change the tent map by

$$\widetilde{f}\colon [0,1] \to [0,1], \quad x \mapsto \begin{cases} 1.999999x, & \text{for } x \in [0, 0.5) \\ 1.999999 \cdot (1-x), & \text{for } x \in [0.5, 1] \end{cases}.$$

Use the new function $\widetilde{f}$ in your code from (a) and visualize your results using `gnuplot`.

Dr. Ole Klein
M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 25, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 3

Exercise 3.1 [Gaussian Elimination with hdnum]

Write a new headerfile `gauss.hh`, that implements the template function

```
template<typename NUMBER>
void gauss( hdnum::DenseMatrix<NUMBER>& A, // Input A
       hdnum::Vector<NUMBER>& x        // Output x
       hdnum::Vector<NUMBER>& b        // Input b
           )
{
  ...
}
```

for solving the linear equation system $Ax = b$ using gaussian elimination. This headerfile will be needed to compile and execute the program `gaussmain.cc`. Compile the program for both data types `double` and `float` and check the maximal dimension n for which the equation system gives a correct solution.
Hint: To get access to the `hdnum` library type in the following command

```
git clone https://parcomp-git.iwr.uni-heidelberg.de/Teaching/hdnum
```

Exercise 3.2 [Polynomial Interpolation]

All programmed functions in this exercise should accept a *template* parameter to allow different representations of the real numbers (`float`, `double`, etc.).

(a) Write a function that evaluates the interpolating polynomial for a given function $f : \mathbb{R} \to \mathbb{R}$ at a given point x. The function should be given as points $(x_i)_{i=1}^n \in \mathbb{R}$ and matching values $(y_i)_{i=1}^n$, i.e.

```
template<typename T>
T interpolation(T x, std::vector<T> x_i, std::vector<T> y_i){...}
```

(b) Write a program that interpolates the following functions on the interval $I = [-1, 1]$ with equidistant points $x_i = -1 + ih, \quad i = 0, \ldots, n$ with $h = 2/n$. The degree of the interpolating polynomial n should be chosen as $n = 5, 10, 20$.

$$f_1(x) = \frac{1}{1 + x^2}$$
$$f_2(x) = \sqrt{|x|}$$

Evaluate the polynomials on a fine grid (1000 grid points) and plot the results using gnuplot. Compare the plots to the actual functions. What do you see?

Exercise 3.3 [Numerical Differentiation]

(a) Write a program `numdiff.cc`, that computes the second derivative of $\sinh(x)$ at $x = 0.6$ with the second differential quotient for a given h

$$a(h) := \frac{\sinh(x + h) - 2\sinh(x) + \sinh(x - h)}{h^2} \approx \frac{d^2}{dx^2} \sinh(x).$$

Calculate the values $a(h_i)$ for $h_i = 2^{-i}$, $i = 1, \ldots, 20$ and compare with the exact value of the second derivative.
Hint: $\frac{d^2}{dx^2}\sinh(x) = \sinh(x)$. The function $f(x) = \sinh(x)$ is available in C++: Using the library `<cmath>` you can call the function `std::sinh(double x)`.

(b) The error decreases until $i = 12$. Calculate with your output the number j such that the error is of order $\mathcal{O}(h^j)$. Explain why the approximations for smaller h become worse.

(c) Examine how the extrapolation to the limit can be used to improve the numerical values. Write a function, that has the input arguments $\widetilde{h}_j$, $j \in \{0, \ldots, k\}$ and computes $a(0)$ using the extrapolation of the limit. Calculate for $h_i = 2^{-i}$, $i = 1, \ldots, 10$ the approximation of $a(0)$ with two values $(h_i, h_i/2)$, resp. three values $(h_i, h_i/2, h_i/4)$. Examine the error.

(d) **Bonus:** Use the fact that $a(h)$ can be represented as series in h^2 for the extrapolation and use the pairs $(h_i^2, a(h_i))$ instead of $(h_i, a(h_i))$. Explain how this change affects the rate of convergence of the error.
N.B.: This modification is also called Richardson extrapolation.

Prof. Dr. Peter Bastian, Prof. Dr. Thomas Wick
Dr. Ole Klein, M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 28, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 4

Exercise 4.1 [The pendulum model – Theory]

Recapitulate the model for the pendulum

$$\frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\sin(\phi(t)) \qquad \forall t > t_0.$$

with the two initial conditions

$$\phi(0) = \phi_0, \qquad \frac{d\phi}{dt}(0) = \phi_0'.$$

For *small* deflection angle ϕ derive the *approximation*

$$\frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\phi(t)$$

Show that it has the general solution $\phi(t) = A\cos(\omega t)$ and determine the constants A, ω from the initial conditions

Exercise 4.2 [The pendulum model – Solver]

(a) **Recap, Method 1**

In the first method, begin by rewriting the second order ODE as a first order system

$$\frac{d\phi(t)}{dt} = u(t), \qquad \frac{d^2\phi(t)}{dt^2} = \frac{du(t)}{dt} = -\frac{g}{\ell}\sin(\phi(t)).$$

Replacing the derivatives by difference quotients

$$\begin{aligned}\frac{\phi(t+\Delta t) - \phi(t)}{\Delta t} &\approx \frac{d\phi(t)}{dt} = u(t),\\ \frac{u(t+\Delta t) - u(t)}{\Delta t} &\approx \frac{du(t)}{dt} = -\frac{g}{\ell}\sin(\phi(t)),\end{aligned}$$

yields the *one step* scheme

$$\begin{aligned}\phi^{n+1} &= \phi^n + \Delta t\, u^n & \phi^0 &= \phi_0\\ u^{n+1} &= u^n - \Delta t\,(g/\ell)\,\sin(\phi^n) & u^0 &= u_0\end{aligned}$$

Where ϕ^n approximates $\phi(n\Delta t)$ for a chosen Δt using recursion (*Euler*).

(b) **Recap, Method 2**

Now, we derive a method that directly approximates the second-order ODE. It uses a *central difference quotient* for the second derivative

$$\frac{\phi(t+\Delta t) - 2\phi(t) + \phi(t-\Delta t)}{\Delta t^2} \approx \frac{d^2\phi(t)}{dt^2} = -\frac{g}{\ell}\sin(\phi(t)).$$

Solving for $\phi(t+\Delta t)$ yields the *two step* scheme ($n \geq 2$):

$$\phi^{n+1} = 2\phi^n - \phi^{n-1} - \Delta t^2\,(g/\ell)\,\sin(\phi^n), \tag{1}$$

with the initial condition

$$\phi^0 = \phi_0, \qquad \phi^1 = \phi_0 + \Delta t\, u_0. \tag{2}$$

The starting value ϕ^1 is derived with one step of method 1.

(c) **Actual Task**

(i) Write a C++ program implementing methods 1 and 2 using a time step Δt that can be entered by the user. For the constants choose $\ell = 9.81$ and $g = 9.81$.

(ii) Write the results to a file, where every line contains

$$t_i \qquad \phi^i \qquad u^i.$$

(iii) you can visualize the results using `gnuplot` as follows
`plot "filename"  u 1:2`
where the x-axis uses the first column and the y-axis uses the second column.

Exercise 4.3 [The pendulum model – implementations]

(a) For method 1: choose an initial deflection angle $\phi_0 = 0.1$ and a time step $\Delta t = 0.1$ and compute the solution up to time 4.0. What do you observe?

(b) Repeat the experiment with successively smaller time steps, say 0.01, 0.001, 0.0001. What do you observe?

(c) Try to compute the solution for longer times with the small timesteps. What happens?

(d) Repeat the same experiments with method 2. Is there a difference?

(e) Compare the solution of the full model and the reduced model for different initial angles $\phi_0 = 0.1, 0.5, 3.0$. Use your favourite method and a timestep Δt that is small enough to avoid any visibly numerical error.

(f) Recapitulate the concepts stability, discretization error and modeling error in the light of the results of exercise 1.

Prof. Dr. Peter Bastian, Prof. Dr. Thomas Wick
Dr. Ole Klein, M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 29, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 5

Exercise 5.1 [Condition for explicit Euler]

Consider the linear, scalar model problem

$$u'(t) = \lambda u(t), \quad u(0) = 1, \quad \mathbb{R} \ni \lambda < 0.$$

Derive the explicit Euler scheme. What is the condition on Δt such that the explicit Euler scheme produces bounded approximations for all $t > 0$? Confirm your result with the implementation in file `eemodelproblem.cc` provided in the exercise yesterday.

Exercise 5.2 [Object oriented ODE solver]

Download the file `linearoscillator.cc` available on the cloud. It solves the problem

$$u'(t) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} u(t) \text{ in } (0, 20\pi], \qquad u(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

using the methods

#	Scheme	#	Scheme
0	Explicit Euler	4	Implicit Euler
1	Heun 2nd order	5	Implicit midpoint
2	Heun 3rd order	6	Alexander
3	Runge-Kutta 4th order	7	Crouzieux
		8	Gauß 6th order

and provides errors $e(T)$ and convergence rates for all schemes. What conclusions can you draw from the tables?

Exercise 5.3 [Van der Pol oscillator]

In this exercise we explore the nonlinear Van der Pol oscillator

$$\begin{aligned} u_0'(t) &= -u_1(t) & u_0(0) &= 1 \\ u_1'(t) &= 1000 \cdot (u_0(t) - u_1^3(t)) & u_1(0) &= 2 \end{aligned}$$

which is an example for a stiff ODE system.
Download an updated version of the file `vanderpol.cc` from the cloud. Compile and run the following four combinations of methods and timesteps:

- `RKF45` method is an adaptive embedded Runge Kutta method of 5th order. Run it with tolerances $TOL_1 = 0.2$ and $TOL_2 = 0.001$ using an initial time step $\Delta t = 1/16$.
- The implicit Euler method. Run it with $\Delta t_1 = 1/16$ and $\Delta t_2 = 1/512$.

The output file contains in each line

$$t_i \quad u_0(t_i) \quad u_1(t_i) \quad \Delta t_i$$

Compare the solutions, especially $u_1(t)$ as well as the time step sizes Δt_i for all four runs. What do you observe?

Prof. Dr. Peter Bastian, Prof. Dr. Thomas Wick
Dr. Ole Klein, M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 30, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 6

Exercise 6.1 [Poisson's equation]

Let $\alpha \in \mathbb{R}$. We are given the Poisson problem in 1D on the interval $\Omega = (0, 1)$:

$$\begin{aligned} -\alpha u''(x) &= f \quad \text{in } \Omega \\ u(0) = u(1) &= 0 \end{aligned}$$

with $\alpha = 1$ and the right hand side $f = -a$ with $a > 0$. The code of this example can be found on the cloud in `fem1d_linear.cc`.

Note: Please note that the above form is only correct when α is constant. The general formulation is

$$-\frac{d}{dx}(\alpha u')$$

which reduces to the above one, when α is constant.

(a) Run the code and observe the results using gnuplot, with $a = 1$ and $h = 0.1$.
Hint: Please work in the optimized compiling mode

(b) We play now with two parameters:

(i) Vary the *discretization parameter* h and use other parameters. What do you observe?

(ii) Vary now the *model parameter* α. What do you observe?

(iii) Choose now a different *right hand side* f. What do you observe?

(c) Check your solution by observing whether the maximum principle is fulfilled or not.

(d) We study in this final task the structure of the code. Go into the code and try to understand the different functions and methods that are implemented therein. Please have a specific look into the `assemble_system` method.

Prof. Dr. Peter Bastian, Prof. Dr. Thomas Wick
Dr. Ole Klein, M.Sc. Robin Görmer, M.Sc. Philipp Thiele

Lima, Oct. 31, 2019

Spring School on the Introduction on Numerical Modelling of Differential Equations – Programming Exercise 7

Exercise 7.1 [Poisson's equation – P2 elements]

(a) Implement P_2 finite elements to solve the problem from Exercise 6. Recapitulate quadratic shape functions for yourself by hand first.

(b) Go into the code `fem1d_quadratic.cc` and implement the necessary modifications.

(c) Implement a numerical quadrature rule in order to evaluate the integrals locally.

(d) Check your code using your 'physical intuition'. This means, does the code deliver results that are 'similar' to those from yesterday?
Hint: On purpose we do not perform a rigorous computational convergence analysis in this exercise because in 1D the finite element method is actually 'too simple' and would yield for point-wise errors exactly zero.

Remarks on quadratic elements:
First we define the discrete space

$$V_h = \{v \in C[0,1] |\ v|_{K_j} \in P_2\}.$$

The space V_h is composed by the basis functions:

$$V_h = \{\phi_0, \ldots, \phi_{n+1}, \phi_{\frac{1}{2}}, \ldots, \phi_{n+\frac{1}{2}}\}.$$

The dimension of this space is $\dim(V_h) = 2n+1$. The mid-points represent degrees of freedom as the two edge points. For instance on each $K_j = [x_j, x_{j+1}]$ we have as well $x_{j+\frac{1}{2}} = x_j + \frac{h}{2}$, where $h = x_{j+1} - x_j$.
On the element $K^{(1)}$ (unit element), we have

$$\begin{aligned}\phi_0(\xi) &= 1 - 3\xi + 2\xi^2,\\ \phi_{\frac{1}{2}}(\xi) &= 4\xi - 4\xi^2,\\ \phi_1(\xi) &= -\xi + 2\xi^2.\end{aligned}$$

These basis functions fulfill the property $\phi_i(\xi_j) = \delta_{ij}$, for $i,j = 0, \frac{1}{2}, 1$. On the master element, a function has therefore the representation

$$u(\xi) = \sum_{j=0}^{1} u_j \phi_j(\xi) + u_{\frac{1}{2}} \phi_{\frac{1}{2}}(\xi).$$

Using these three shape functions we can now evaluate

$$A_{i,j} = \int_0^1 \phi_i' \phi_j' \, dx$$

and

$$b_j = \int_0^1 (-a)\phi_j \, dx$$

with the Simpson rule to obtain the *local* stiffness matrix

$$A = \frac{1}{h}\begin{pmatrix} 7 & -8 & 1 \\ -8 & 16 & -8 \\ 1 & -8 & 7 \end{pmatrix}$$

and the *local* right hand side

$$b = \frac{h}{6}(-a, -4a - a)^T$$

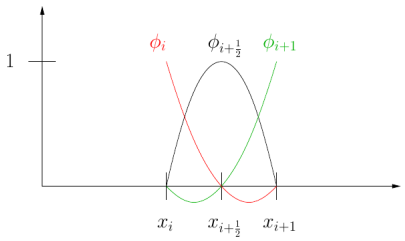

Figure 1: Example for quadratic elements.