Programmiermethoden des Wissenschaftlichen Rechnens

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

Exercise sheet 1. Submission no later than the start of lecture on 2018-10-31

- The website of this lecture can be found at `https://ins.uni-bonn.de/teachings/ws-2018-228-p2e1-p4e1-praktikum-progr/`
- Please submit your homework assignments in groups of up to 3 students.

Exercise 1. (A polynomial class)

Write a class `Polynomial`, where you assume that a polynomial p is given in a monomial basis. It should include methods to

a) call the polynomial p at a point x (i.e. compute $p(x)$)

b) add two polynomials

c) multiply two polynomials

d) test if two polynomials are equal

e) compute the derivative and antiderivative of polynomials, and illustrate them for $p(x) = 3x^2 + 2x + 1$

Test all your methods extensively.

Exercise 2. (A rudimentary matrix class)

Define a rudimentary class for the representation of matrix computations. It should include

a) A matrix multiplication of the form $A * B$,

b) A method to print a matrix to the screen, in the usual form as a vertical sequence of rows, such as e.g.

```
1. 2.
3. 4.
```

c) A test for equality using the infix-operator '`==`'.

d) Give an example to illustrate that multiplication is associative.

For the solution of the following excercises, include these methods in suitable subclasses of your rudimentary matrix class.

e) Write a program for the LU decomposition of suitable regular matrices. Test the program for the inputs

$$A = \begin{pmatrix} 3 & 2 & 1 \\ 6 & 6 & 3 \\ 9 & 10 & 6 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

f) Compute the inverse of the matrix

$$A = \begin{pmatrix} 3 & -1 & 2 \\ -3 & 4 & -1 \\ -6 & 5 & -2 \end{pmatrix}.$$

Exercise 3. (Numerical integration)

A common approach to the approximate computation of the integral of real-valued functions f over an interval $[a, b]$ is based on the following (simple) *trapezoidal rule*:

$$\int_a^b f(x)\,dx \approx (b-a)\frac{f(a)+f(b)}{2}.$$

If f is twice continuously differentiable in $[a, b]$, then the following estimate for the deviation $E(f, a, b)$ from the exact value holds:

$$(*) \qquad |E(f,a,b)| \le \frac{(b-a)^3}{12} \max_{a \le x \le b} |f''(x)|.$$

To better approximate the integral, we subdivide the interval $[a, b]$ into n adjacent equal-size subintervals of length $h = \frac{b-a}{n}$. In each subinterval we apply the simple trapezoidal rule, and then add the resulting approximations. This gives the *composite* trapezoidal rule.

a) Define a Python function

```
trapeze(f,a,b,n)
```

to compute the integral of a function f over an interval $[a, b]$ according to the composite trapezoidal rule for n subintervals. Avoid repeated computations of the individual function values $f(x)$.

b) Use the `trapeze` function to approximate the integral $I := \int_0^\pi \sin(x)\,dx$, such that the result becomes correct to 6 decimal places. To do this, use the exact value $I = 2$, which results from the evaluation of the antiderivative $-\cos$ of sin.

In the refinement of the approximation it is a good idea to consider n-values of the form 2^m, i.e. powers of 2, since you can then reuse the already calculated values from the previous iteration round.

c) Calculate the approximation for $n = 2^m$, $m = 1, \dots, 10$. Store the results in a list. Print the result of the individual calculations as well as the deviations from the exact value.

d) What assumption about convergence speed can be deducted?

e) Check your assumption for the integral

$$\int_0^2 f(x)\,dx \quad \text{for} \quad f(x) := 3^{3x-1}.$$

f)* For example, consider the function $f(x) = e^{x^2}$. Compute the integral

$$\int_0^1 f(x)\,dx$$

to 6 decimal places. For this purpose, determine the number n of the necessary equidistant support points by applying the above error estimate $(*)$ to each individual trapeze over the subintervals.

Exercise 4. (Taylor series)

The Taylor series plays an important role in error estimation of approximative methods. For instance, the estimate $(*)$ above is based on the evaluation of a suitable Taylor series. In general, the Taylor series allows the approximation

$$f(x+h) \approx \sum_{i=0}^{n} \frac{h^i}{i!} f^{(i)}(x)$$

of a sufficiently often differentiable function f in a vicinity of x.
Here, we consider the exponential function. For this function we even have

$$(**) \qquad e^x = \lim_{n\to\infty} \sum_{i=0}^{n} \frac{1}{i!} x^i, \quad x \in \mathbb{R}.$$

Write a Python function
`exp_approx(x)`
for the approximative computation of e^x to 6 decimal places. Formulate the loop condition without reference to the true value. Try to use as few elementary arithmetic operations $+, *, -, /$ as possible.

Exercise 5. (The Newton method)

The Newton method is a standard method for the numerical determination of zeros of nonlinear functions. Let $f\colon \mathbb{R} \to \mathbb{R}$ be a continuously differentiable function. Consider the recursively defined sequence

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad x_0 \text{ given.}$$

If this sequence converges, then to a root of f. This can again be shown using the Taylor series expansion mentioned above.

a) Write a Python function

   `newton(f, f_prime, x)`

   which performs the above Newton method for the initial value x_0, until $|x_{n+1} - x_n| < 10^{-7}$. In order to intercept the case that the method does not converge, it should be interrupted with an error message after 100 unsuccessful steps.

   Note that in addition to the function f itself, the derivative f' must also be supplied explicitly.

b) Test the procedure for $f(x) := x^2 - 2$ and the initial value $x = 1$. Compare the result with the exact solution $\sqrt{2}$.

For the following exercises, you are allowed to use the modules `scipy`, `numpy` and `matplotlib`.

Exercise 6. (The Newton method II)

We continue with the Newton method in Exercise 5. Extend the function ‘`newton` to a Python function
`newton_ext(f,x)`
that now also computes the necessary values of the derivative.

a) Test both functions `newton` and `newton_ext` for

$$f(x) := e^x + 2x,$$

   and compare the results. Choose a suitable initial value x_0 by first plotting the graph of f.

b) The equation $\cosh(x) = 2x$ has two solutions.

   1) Use the Newton method to determine both.
   2) Find an initial value x_0, for which the computation diverges.

   Again it might be useful to inspect the graph.

Exercise 7. (Multidimensional Newton method)

For the computation of the zeros of a function $f\colon \mathbb{R}^n \to \mathbb{R}^n$, $n > 1$, with Newton's method, one has to 'left-multiply' with the inverse of a Jacobian matrix instead of dividing by a derivative. Consider $f = (f_1, f_2, f_3)$ with

$$\begin{aligned} f_1(x,y,z) &:= 9x^2 + 36y^2 + 4z^2 - 36, \\ f_2(x,y,z) &:= x^2 - 2y^2 - 20z, \\ f_3(x,y,z) &:= x^2 - y^2 + z^2. \end{aligned}$$

a) Compute the Jacobian

$$J(v) := \left(\frac{\partial f_i(v)}{\partial v_j}\right)_{1\le i,j\le 3}, \qquad v = (x,y,z).$$

b) Solve the equation system $f(v) = 0$ approximatively by

$$u_{k+1} := u_k - J(u_k)^{-1} \cdot f(u_k)^T$$

   with initial values $u_0 = (\pm 1.0,\ \pm 1.0,\ 0)$.

Exercise 8. (Numerical integraton II)

Compute the integrals

$$\int_0^\pi \sin(x)\,dx, \qquad \int_0^2 3^{3x-1}\,dx, \qquad \int_0^1 e^{x^2}\,dx,$$

using different (at least 3) methods in the package `scipy.integrate`.

Exercise 9. (Gauss-Seidel method)

The Gauss-Seidel method is an iterative method to solve a linear system of equations $Ax = b$. The matrix A is decomposed into a diagonal matrix D, a strictly lower triangular component L, and a a strictly upper triangular component U, such that $A = L + D + U$. Starting with an initial value $x^{(0)}$, the following sequence is then computed iteratively:

$$x^{(k+1)} = (D+L)^{-1}(b - Ux^{(k)}).$$

Consider the linear equation system:

$$\begin{pmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 24 \\ 30 \\ -24 \end{pmatrix}.$$

Perform three iteration steps using the Gauss-Seidel method, starting from the initial approximation $x^{(0)} = (3,3,3)^T$, and compare the results with the exact solution.

Exercise 10. (Solving ODEs)

a) Solve the IVP $u(x)$, specified by

$$u'(x) + u(x) = x, \quad u(0) = 1,$$

with the method '`odeint`' in the package `scipy.intergrate`, and plot the solution together with the exact solution

$$u(x) = x - 1 + 2e^{-x}.$$

b) * Use `odeint` to solve

$$(3x-1)u'' - (3x+2)u' - (6x-8)u = 0, \quad u(0) = 2,\ u'(0) = 3.$$

Note that in the present case an initial value is specified also for the *derivative* u'.

Exercise 11. (Boundary value problem)

Consider the boundary value problem

$$\begin{aligned} u''(x) &= -|u(x)|, \quad x \in [0,4], \\ u(0) &= 0,\ u(4) = -2. \end{aligned}$$

The equation has *two* solutions u_1, u_2. Use `solve_bvp` in the package `scipy.integrate` to determine both, and plot them. As a hint to the solver, which one to compute, you can use $u_1'(0) > 0$, $u_2'(0) < 0$.

Exercise 12. Consider the following program:

```
from numpy import *
n = 51
x = linspace(0, 1, n)
y = linspace(0, 1, n)
L = zeros((n,n))
L[0,:] = y * (y-1)

iter = 70
for _ in range(iter):
    for i in range(1, n-1):
        for j in range(1, n-1):
            Lt = L
            L[i,j] = (Lt[i+1,j] + Lt[i-1,j] + Lt[i,j+1] + Lt[i,j-1]) / 4
```

a) Plot `L` as a set of function values over the square determined by `x` and `y`.

b) What is computed by the program? How and why does it work?