Computational Finance
Computer Lab 7

The aim of the Lab is to learn to use finite difference approximations of derivatives of a function and to derive finite difference methods for boundary value problems of ordinary differential equations.
Let us start from the numerical differentiation by finite difference approximations. Well-known finite difference approximations are as follows:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \qquad (\text{error} \le ch^2),$$
$$f''(x) \approx \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \qquad (\text{error} \le ch^2).$$

Exercise 1. Write a function `myDerivative` that takes four arguments - a name of a function, the value of x, the value of h and the order of the derivative (1 or 2) - and computes the value of the specified derivative at x using the finite difference approximation given above. Using this function, verify the accuracy of the error estimates in the case of $f(x) = e^x$, $f(x) = \sqrt{x}$ at $a = 1$: compute the derivatives for $h = 1, \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{2^{10}}$ and find the quotient of the error to h^2 for both the first and second order derivatives. The quotients should approach a constant value.

Exercise 2. Compute the errors of the finite difference approximation of $f'(1)$ in the case $f(x) = e^x$ and $h = \frac{1}{2^i},\ i = 1, 2, \ldots, 50$. Can you see the effect of round-off errors for small value of h?

Often we can not compute the value of a function for all x values, but have the values of the function available only at $n+1$ equally spaced points in an interval $[a, b]$. This means that we have a vector of values $(f_0, f_1, \ldots, f_n)$, where $f_i = f(x_i)$ for $x_i = a + i \cdot \frac{b-a}{n}$. In such case we still can use finite difference formulas for computing derivatives at the points $x_i,\ i = 1, \ldots, n-1$, but we can not use arbitrary h values. Instead, it makes sense to use the formulas with $h = \frac{b-a}{n}$. So, if we want to compute approximately $f'(c)$, where c is a grid point inside $[a, b]$, we first have to find i such that $c = x_i$ and then compute the approximate value of the derivative by $\frac{f_{i+1} - f_{i-1}}{2h}$.

Exercise 3. Define a function `myDer2` which for given vector `f` and for given numbers `a,b,x` computes approximately $f'(x)$. Assume that `f` corresponds to the uniform grid in the interval $[a, b]$ and that `x` corresponds to some interior grid point x_i. Check the correctness of your function in the case `f=[1,0,1,4,9]`, `a=−1`, `b=3` and x=1 (the answer should be 2).

Finite difference approximations of derivatives can be used for deriving numerical methods for solving differential equations.
Let us consider the following problem: find y such that

$$y''(x) + x\, y'(x) = f(x),\ x \in [a, b] \tag{1}$$
$$y(a) = 0,\ y(b) = 1, \tag{2}$$

where f is a given functions. The procedure for deriving a finite difference approximation for the problem above consists of the following steps.

1. Choose a set of points at which we want to find approximate values of the unknown function. Usually this set of points is chosen by dividing the interval $[a, b]$ into n equal subintervals: we get points $x_i = a + ih,\ i = 0, \ldots, n$ where $h = \frac{b-a}{n}$.

2. In order to determine the approximate values of the solution y at the $n+1$ points, we need $n+1$ equations. We get those equations by using the boundary conditions (two equations) and by writing down the differential equation at a set of $n-1$ points $\bar{x}_i,\ i = 1, \ldots, n-1$ and replacing the derivatives by approximations that use only the function values at points $x_i,\ i = 0, \ldots, n$. The points $\bar{x}_i,\ i = 1, \ldots, n-1$ do not have to be the same as the points $x_i, i = 1, \ldots, n-1$, but in the case of the current problem let us use $\bar{x}_i = x_i,\ i = 1, \ldots, n-1$.

If $x_i,\ i = 0\ldots,n$ are equally spaced (with step size $h = \frac{b-a}{n}$), then we can use the finite difference approximation discussed above:

$$y'(x_i) \approx \frac{y(x_{i+1}) - y(x_{i-1})}{2h} \qquad (\text{error} \leq ch^2)$$
$$y''(x_i) \approx \frac{y(x_{i-1}) - 2y(x_i) + y(x_{i+1})}{h^2} \qquad (\text{error} \leq ch^2).$$

Hence, writing out the differential equation at points $x_i,\ i = 1,\ldots,n-1$ and replacing derivatives with finite difference approximations we get a system of equations

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + x_i \frac{y_{i+1} - y_{i-1}}{2h} = f(x_i),\ i = 1,\ldots,n-1,$$

where y_i denotes the approximate values of $y(x_i)$. We can view the boundary conditions as two additional equations

$$y_0 = 1,\ y_n = -1.$$

Since i-th equation contains only 3 unknowns y_{i-1}, y_i, y_{i+1} the resulting matrix of the system of equations has non-zero entries only on three diagonals, so we obtain a three diagonal system of equations.

Exercise 4. Consider the following problem: find y such that

$$y''(x) + xy'(x) = f(x),\ x \in [a,b]$$

$$y(a) = 0,\ y(b) = 1$$

One possibility to solve a linear system of equations is to use the Python command `linalg.solve(M,z)` from the subpackage `linalg` of SciPy (that has to be imported first with `from scipy import linalg`), which returns the solution y of the system of equations $My = z$. Use this command to find the values of the approximate solution of the problem with the finite difference method in the case $n = 10$, $a = 0$, $b = 1$, $f(x) = 3x^3 + 6x$. Find the errors between the approximate solution and the exact solution $y(x) = x^3$.

Exercise 5(*). If the system matrix has only some nonzero diagonals then it is actually a waste of computer memory to store the full matrix. For solving such system it is actually possible to use the command `linalg.solve_banded((l,u),Dgs,z)`, where the rows of the matrix Dgs contain the diagonals of M starting from the highest one, l is the number of diagonals below the main diagonal and u is the number of diagonals above the main diagonal (in our case $l = u = 1$).NB! In the matrix Dgs for the diagonals that are above the main diagonal the first elements are actually not used (for the diagonal directly above the main diagonal the first element is not used, for the next diagonal two steps above the main diagonal the first two elements are not used etc); for diagonals below the main diagonal last elements are not used. Write a function that for a given n solves the problem of the previous exercise with the command `linalg.solve_banded((l,u),Dgs,z)` and returns the maximal error at the points $x_i,\ i = 1,\ldots,n-1$. Determine how many times the error is reduced if we increase the number of points two times.