General Information: This assignment contains written and/or programming tasks. Combine all the answers to the written tasks in a single PDF document, named `{lastname}-written.pdf`. You can also scan or take pictures of (readable) handwritten papers. JPEG/PNG image files are accepted in this case and they should be named `{exercisenumber}-{lastname}-written.{jpeg/png}`. Make sure that we can follow the manual calculations. Do not combine too many small steps into one. The programming tasks have to be solved in *Julia* and the source code files have to be submitted using the following naming scheme: `{exercisenumber}-{lastname}.jl`.

(1) (2.5 points) Use the Julia template file `f.jl` for all implementation parts of this exercise. Given

$$f : \mathbb{R} \to \mathbb{R} \tag{1}$$
$$f(x) = \tanh(x) \tag{2}$$
$$= \frac{\sinh(x)}{\cosh(x)} \tag{3}$$
$$= \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \tag{4}$$
$$= \frac{\exp(2x) - 1}{\exp(2x) + 1} \tag{5}$$

a) (0.5 points) Find $\frac{\mathrm{d}f}{\mathrm{d}x}$ analytically.

b) (0.5 points) Finite difference schemes can be used to approximate the derivatives of a function numerically. There are different schemes available and three basic schemes are the *forward*, *backward* and *central difference scheme*:

$$\text{forward: } D_+ f(x) = \frac{f(x+h) - f(x)}{h}, \tag{6}$$
$$\text{backward: } D_- f(x) = \frac{f(x) - f(x-h)}{h}, \tag{7}$$
$$\text{central: } D_c f(x) = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h}. \tag{8}$$

Calculate the analytical solution for the first derivative $f'(x)$ at $x = 3$. Then use the three finite difference schemes with $h = 0.1$ to approximate the derivative numerically. For the numerical approximation evaluate $f(x)$ at $2.9, 2.95, 3, 3.05, 3.1$ and write down the resulting values. Compare the results by computing the error between the approximations and the analytical solution using the absolute difference: e.g.for the forward difference scheme:

$$\varepsilon^+ = \mid D_+ f(3) - f'(3) \mid \tag{9}$$

c) (0.5 points) Implement the analytical solution of the derivative $f'(x)$ in the `f1(x::Float64)::Float64` method.

d) (0.5 points) Implement the finite difference schemes `forward()`, `backward()` and `central()` with the signature `func(x::Float64, h::Float64, f::Function)::Float64` in the module `FiniteDiff`. The resulting plots should look like Figure 1.

e) (0.5 points) Look at the result and describe what you can observe. What is the reason for some schemes being more accurate (smaller errors) compared to others? Provide a detailed description.

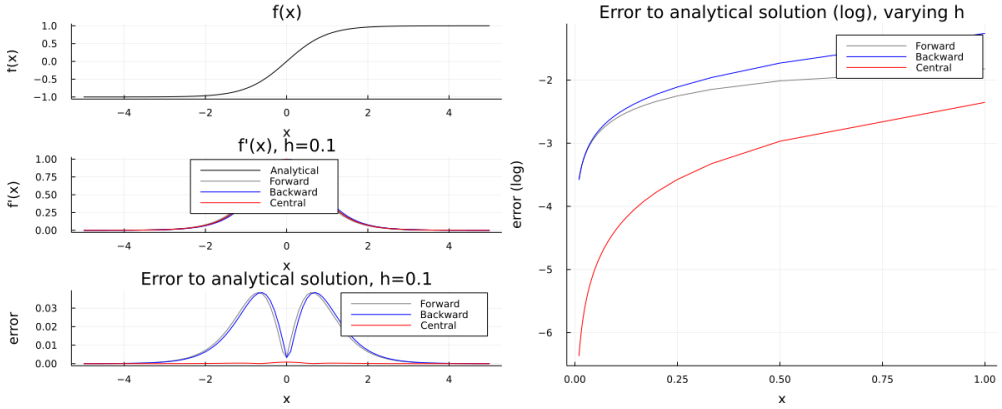

Figure 1: $f(x)$

(2) (2.5 points) Use the Julia template file `quadrature.jl` for all implementation parts of this exercise. Given integral of Equation 1

$$F(x) = \int f(x)\mathrm{d}x \tag{10}$$

$$= \log(\cosh(x)) + C \tag{11}$$

a) (0.1 points) Calculate $\int_{-1}^{5} f(x)\mathrm{d}x$ analytically

b) (0.3 points) Calculate $\int_{-1}^{5} f(x)\mathrm{d}x$ numerically with 3 equidistant intervals. Use *right sums*, *trapezoid*, and *Simpson's rule* (see Equation 12) for numerical quadrature to evaluate the integral.

c) (0.1 points) Calculate the errors to the analytical solution using absolute differences.

d) (0.25 points) Implement code of `f(x::Float64)` in `F(x::Float64)`

e) (1.5 points) Implement code of `right(), trapez(), simpson()`. Signature of all of these functions is `func(s::Float64, e::Float64, N::Int, f::Function)::Float64`, with `s` the lower and `e` the upper border of integration, `N` the number of intervals, and `f(x::Float64)::Float64` the function to be integrated. See Figure 2.

f) (0.25 points) Look at the resulting plots and describe what you can observe. Which method performs best? Why do you think this is the case?

$$\mathcal{I}_R = \sum_{i=1}^{N} f(x_i) \cdot h \tag{12}$$

$$\mathcal{I}_T = \sum_{i=1}^{N} \frac{f(x_{i-1}) + f(x_i)}{2} \cdot h \tag{13}$$

$$\mathcal{I}_S = \sum_{i=1}^{N} \frac{f(x_{i-1}) + 4f((x_i + x_{i-1})/2) + f(x_i)}{6} \cdot h \tag{14}$$

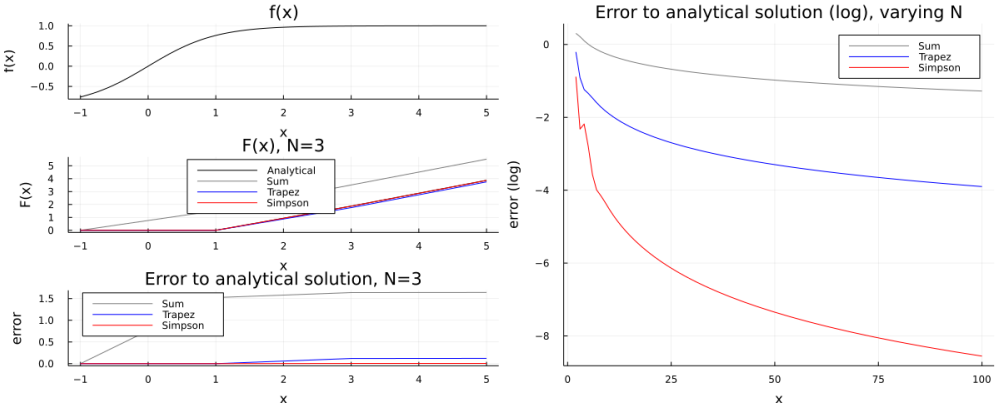

Figure 2: $f(x), F(x)$ and error

(3) (2 points) The "diffusion" equation is a model of ... "diffusion" processes like the propagation of heat over time in a given region. It is a parabolic partial differential equation and its normalized form in one dimension is given by Equation 15. State $u(0,t)$ at the boundary (i.e. boundary condition) and the initial state $u(x,0)$ (i.e. initial condition) have to be known to solve the problem. Assume (see Equation 16) a homogeneous *Dirichlet* boundary condition (i.e. "specifying the values that a solution needs to take along the boundary of the domain"), initial heat distribution for $-1 < x < 1$ and $t > 0$.

$$\frac{\partial}{\partial t}u(x,t) = \frac{\partial^2}{\partial x^2}u(x,t) \tag{15}$$

$$u(-1,t) = u(1,t) = \pi \tag{16}$$

$$u(x,0) = e^{-x} \tag{17}$$

Such a problem can be approximated numerically using finite differences: the space domain is discretized into $x_0, \ldots, x_I$ with a step size h and the time domain is discretized into $t_0, \ldots, t_N$ with a step size τ. On such a grid, $u(x,t)$ can be approximated at each point $u(x_i, t_n)$. To do that an explicit method can be derived by using a forward difference scheme (see $D_+f(x)$ in Equation 6) to calculate the time derivative $\frac{\partial}{\partial t}u(x_i,t_n)$ at position x_i.

$$\frac{\partial}{\partial t}u(x_i,t_n) = \frac{u(x_i,t_{n+1}) - u(x_i,t_n)}{\tau} \tag{18}$$

A second-order central difference scheme can be used to estimate the space derivative $\frac{\partial^2}{\partial x^2}u(x_i,t_n)$. The formula for a second-order central difference scheme is obtained by applying the first-order central difference twice, resulting in Equation 19 so $\frac{\partial^2}{\partial x^2}u(x_i,t_n)$ can be succesfully estimated.

$$D_c^{(2)}f(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{19}$$

$$\Rightarrow \frac{\partial^2}{\partial x^2}u(x_i,t_n) = \frac{u(x_{i+1},t_n) - 2u(x_i,t_n) + u(x_{i-1},t_n)}{h^2} \tag{20}$$

a) (0.5 points) Derive the formula for $u(x_i, t_{n+1})$: substitute $\frac{\partial}{\partial t}u(x,t)$ and $\frac{\partial^2}{\partial x^2}u(x,t)$ in the diffusion equation with the forward difference scheme over the time and the second-order central difference scheme over space and solve for $u(x_i, t_{n+1})$.

b) (1.5 points) Implement the update step of $u(x,t)$ for one time step in the template file `diffusion.jl` (i.e. `explicitStep(us::Vector, h::Float64, tau::Float64)::Vector`). Take Figure 3 as reference.

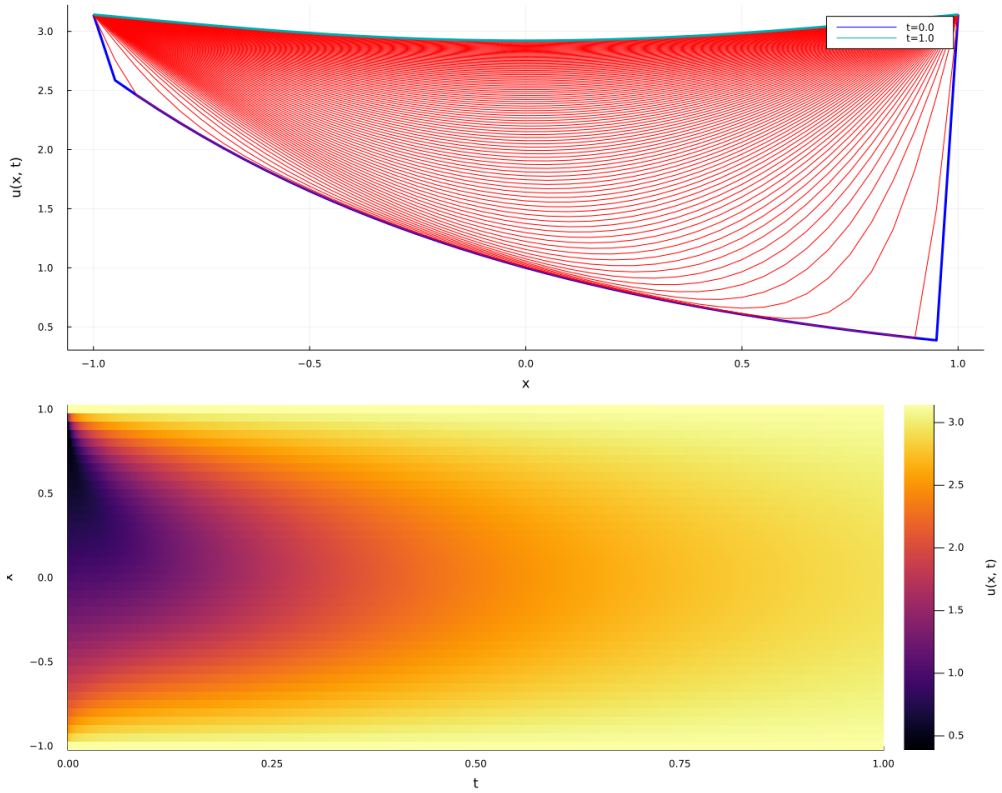

Figure 3: Solution of Equation 16

(4) **Bonus**: (1 points) Transform the following polynomial using Horner's scheme. Compute the number of arithmetic operations before and after transformation. Perform the calculations for (a) and (b) explicitly by hand.

(a) (0.25 points) $2x^6 - 5x^5 + x^2 - 6x + 1$

(b) (0.5 points) $12x^7 + 2x^4y^6 + x^3y^4 + x^2(-3 - 7y^4) - 2 + 9y + 2y^5$.

Hint: sort the terms by x-, y- and xy-terms and transform each of the three groups separately.

(c) (0.25 points) Implement the transformed function from (b) via the prepared Julia function `h(x::Float64, y::Float64)::Float64` in `bonus.jl`. The script measures the runtime for evaluating the polynomial for each cell on a grid of size $N \times N, N = 1024$ and then reports the average runtime for both functions over three program executions. In addition, it plots the results and the difference between them as heat maps (see Figure 4). Use the average runtimes to compute the speed-up compared to the original function.

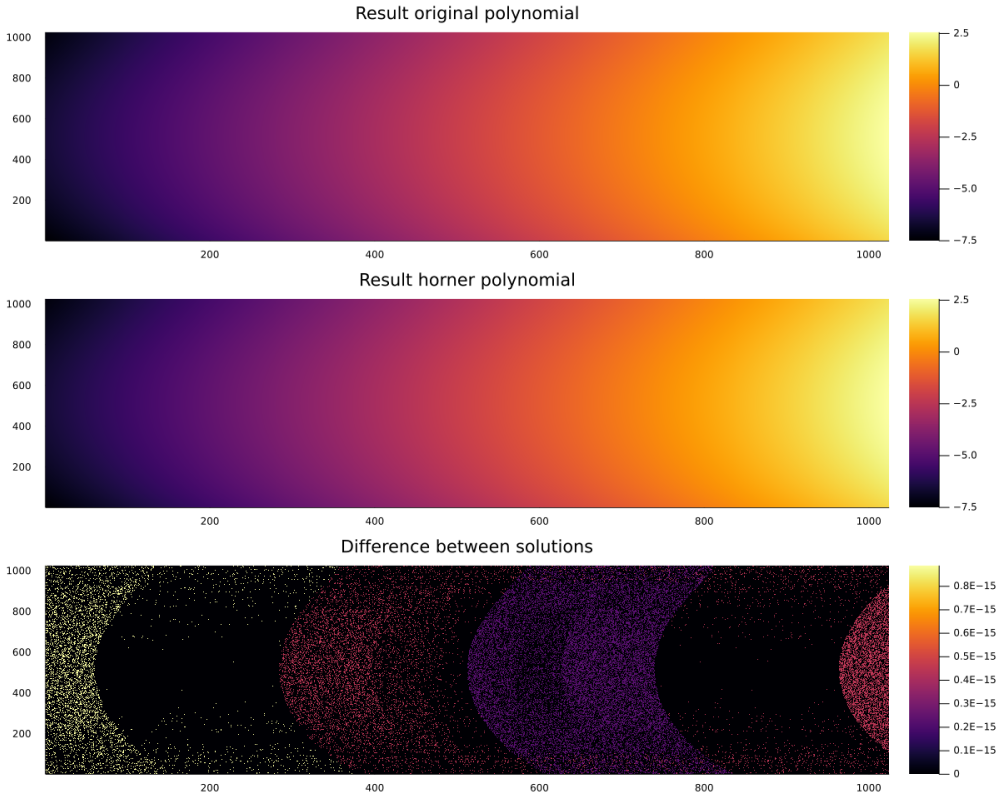

Figure 4: Solution of 4c