Benedict Bleidt
Deborah Kornwolf
Marc Messmer
Sarah Naja
Benedict Sondershaus

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Programming in C++

Sheet 1

Submission date: 23.09.2025 12:00

In this first exercise we will take a look at function calling and parameter passing (including functions as parameters), default values for function arguments, conditional statements and basic loops.

Your task is to estimate the root of strictly monotonic increasing or decreasing, linear functions. The root is defined as the x-value at which the function is zero ($f(x) = 0$). Such functions only have one root point at the x-axis. To find the x-value, we will use interval bisection:
We start with an interval defined by `xLower` and `xUpper` (with `xLower` $<$ `xUpper`). In each step, the interval is halved at its midpoint. The interval half which has the positive-negative transition of $f(x)$ must contain the root and is therefore taken as the new interval for the next loop iteration. The bisection process stops if the interval is sufficiently small (which means, the root has been determined with the required precision).

Add your code to the file `student_template_1.1/submission/estimate_function_root.cpp`. No further includes are required/allowed. Add a comment line with your first and last names at the top of each submitted source file (.cpp). To assist in coding, we added several smaller helpers to test the functionality. Particularly, the `testEstimateFunctionRoot()`-function will help you to test your implementation.

Before submitting your solution make sure your program compiles, can be executed and does not produce a huge amount of output. Then you can zip the `submission` folder and submit it on InfoMark.

1.1 Estimate the root of a given linear function (100 Points)

C++

a) Complete the **roundValToNDecimals**-function in the **estimate_function_root.cpp** file. The function takes a float value and an unsigned integer specifying the number of decimal places to round the value to. Hint: you can make use of the `std::round` function which computes the nearest integer value to a given float and the power function `std::pow`.

b) Complete the function **isAlmostEqual** which takes 3 float values as arguments: `x1`, `x2` and `epsilon`. If the absolute difference between x1 and x2 is smaller or equal than the specified epsilon-value, the function should return true, otherwise false. Note that the default `epsilon`-value is set to 10^{-5}. Hint: To calculate the absolute value you can use the function `std::abs`.

c) Implement the interval estimate of the root using the interval bisection algorithm as described above in **estimateFunctionRoot**. The arguments are the linear monotonically **increasing** or **decreasing** function (`*linearFunc`), the initial search interval bounds [`xLower`, `xUpper`] and an unsigned integer `nDecimals` which specifies the precision in number of decimal digits.
To check whether we are done use the `isAlmostEqual`-function implemented in b) to check if `xLower` and `xUpper` are almost equal (using the default `epsilon`). In case we are done, return the result rounded to `nDecimals` number of decimal digits using the `roundValToNDecimals` function implemented in a). To find the next interval bisect the current interval in the middle. Calculate the $f(x)$-values of the three interval bounds lower, middle and upper with the provided `linearFunc(x)`. If there's a sign change between $f($`lower`$)$ and $f($`middle`$)$, lower and middle are taken as interval bounds for the next iteration. Otherwise, if there's a sign change between the $f($`middle`$)$ and the $f($`upper`$)$, this interval is chosen for the next iteration.

d) Consider the special case if one of the starting interval x-values is equal to the root of the function. We provide two test functions to debug your code.

e) Another special case occurs if the starting interval does not contain the root of the linear function. Make sure to handle and test this case as well in **estimateFunctionRoot**. Return **NaN** (Not a Number) if the root is not in the given interval.

f) Make sure to test your implementation for **decreasing** linear functions as well! You can use the provided `testAndPrintDecreasing`-function.