Lab Manual

Calculus in Professional Practice

Table of Contents

[Introduction 1](#_Toc530076481)

[Grading 2](#_Toc530076482)

[Additional learning goals 2](#_Toc530076483)

[Input format for functions 2](#_Toc530076484)

[Assignments 3](#_Toc530076485)

[Assignment 1: functions 3](#_Toc530076486)

[Assignment 2: derivatives 4](#_Toc530076487)

[Assignment 3: integrals 4](#_Toc530076488)

[Assignment 4: McLaurin series 4](#_Toc530076489)

[Assignment 5: n-polynomial 4](#_Toc530076490)

[Assignment 6: Poisson distribution 4](#_Toc530076491)

[Assignment 7: network connectivity 5](#_Toc530076492)

[Extra assignments 5](#_Toc530076493)

[Additional assignment: validation 5](#_Toc530076494)

[Additional assignment: enhancements 5](#_Toc530076495)

[Tree-like structural representation of a function 5](#_Toc530076496)

# Introduction

This document describes the assignments for the course CPP for the specialization route Academic Preparations as part of the pre-master program for the Tue (6 ECTS).

Make sure that your lecturer is properly informed about all smart activities that you have done during this course. This could be done for example by writing an accompanying document. Please note: when your lecturer is only looking at your code, he/she might quite easily overlook your intelligent solutions, which would be a pity.

The assignments can be done in any modern object oriented language (we advise C# and Java). The assignments differ in difficulty. The next table gives a global indication (1 = relatively easy; 4 = relatively difficult), together with an advised week planning. (YMMV)

|  |  |  |
| --- | --- | --- |
| **assignment** | **difficulty** | **week (for 6 ECTS)** |
| 1 | 3 | 1 |
| 2 | 2 | 2 |
| 3 | 1 | 3 |
| 4 | 2 | 4 |
| 5 | 3 | 5 |
| 6 | 2 | 6 |
| 7 | 1 | 7 |

# Grading

To pass this course (grade 6), all assignments are implemented and work properly with an easy-to-use GUI (even on another machine (in particular: the lecturer's)).

For higher grades: incorporate the following aspects:

* good software design (classes, interfaces, SOLID principles, Design Patterns, …)
* clear documentation of your actual design and your design decisions
* proof of the robustness of your code (thorough test cases, code analysis, code coverage)
* robust recovery for incorrect user input
* other smart inventions and spectacular new features

All assignments have to be submitted in Canvas before their deadlines. Except the last assignment, a snapshot of your software project is sufficient. After the last deadline, there will be an individual meeting with the lecturer with a discussion and explanation of your work.

# Additional learning goals

Besides the specific contents of the assignments, the following general aspects of software engineering play a role:

* algorithms
* UML modelling
* refactoring (in particular when your initial UML modelling was not that optimal)
* testing (module tests and system tests)
* code analysis (coverage, complexity)
* user interface design

# Input format for functions

Normally, we write functions like:

f(x) = 7 + x

f(x) = sin(πx)

But, for easy parsing in your code, in this course we write them in prefix notation and limit all words to single characters. Above functions become:

+(7,x)

s(\*(p,x))

The building blocks of our functions are:

|  |  |  |
| --- | --- | --- |
| normal notation | prefix notation | remark |
| x | x | we will only use functions with one parameter, being x |
| 0..9 | 0 1 2 3 4 5 6 7 8 9 | natural numbers 0 till 9 (syntactic sugar, because we can also use n(A)) |
| other natural numbers | n(A) | where A is a natural number (positive, negative or zero) |
| real numbers like -4.2 and 987.654 | r(A) | where A is a real number written in decimal notation |
| A + B | +(A,B) | where A and B are functions as well |
| A \* B | \*(A,B) | " |
| A - B | -(A,B) | " |
| A / B | /(A,B) | " |
| AB | ^(A,B) | where A is a function and B is an natural number |
| sin(A) | s(A) | where A is a function |
| cos(A) | c(A) | " |
| exp(A) aka eA | e(A) | " |
| ln(A) | l(A) | letter el |
| A! | !(A) | where A is a positive natural number |
| π | p | 3.14… |

So:

is written as:

/(\*(-(x,3),+(c(n(-73)),r(1.6))), e(!(5)))

# Assignments

## Assignment 1: functions

Write a program that reads a function from a text box and plot it on a canvas.

Show the tree-structure of the function (see Appendix Tree-like structural representation of a function) and print the function in a more human-readable ASCII format.

Tips:

* use *recursion* to read the formula
* use one base class (or interface) for each node and let method ToString() return the formula in human-readable notation
* as the next assignments build on this first assignment: spend a serious amount of time to make a clean design; you will benefit from it

## Assignment 2: derivatives

Determine the derivative of the function in two ways:

* analytically
* with Newton's difference quotient

For both solutions, plot the resulting function. For the analytical result: show the function as a tree and in text.

The analytical derivative function of the previous week will contain many useless branches, like 1 \* x, x + 0, x1, etc. Simplify them (note: simplify (x + 3) + 5 into (x + 8) might be too complex, so you might skip them).

## Assignment 3: integrals

Calculate the Riemann integral of the function (where the user can indicate the lower and upper boundaries), and show it in the plot.

## Assignment 4: McLaurin series

Calculate the McLaurin series of the function. Show the versions with increasing number of terms (1 .. 8) in the plot.

## Assignment 5: n-polynomial

When a user has clicked on n+1 spots on an empty canvas, calculate an nth-order polynomial function that covers those points. Show a plot and the ASCII text of that function.

## Assignment 6: Poisson distribution

Part A:

Write methods to generate random variates for the following distributions:

* Poisson distribution with parameter λ  
  Implement it as a sequence of Bernouilli experiments (eg. number of sequences = 20 λ)
* Exponential distribution with parameter λ

Both methods may only be implemented with the help of the (uniformly distributed) random generator as offered by your programming language.

"Prove" both implementations by collecting a sample, calculate its mean, variance and standard deviation. Show the resulting histogram in a chart, together with the probability mass (resp. density) function.

Part B:

"Prove" by means of a simulation that a series of m events (whose inter-arrival times are exponential distributed with parameter λ) approximates a Poisson distribution λ. Let the user choose λ, the number of events m and the Poisson interval.

Further information can be found on:

* <https://en.wikipedia.org/wiki/Poisson_distribution>
* [https://en.wikipedia.org/wiki/Binomial\_distribution#Poisson\_approximation](https://en.wikipedia.org/wiki/Binomial_distribution" \l "Poisson_approximation)
* <https://en.wikipedia.org/wiki/Probability_mass_function>
* <https://en.wikipedia.org/wiki/Exponential_distribution>
* <https://en.wikipedia.org/wiki/Probability_density_function>
* <https://en.wikipedia.org/wiki/Cumulative_distribution_function>
* <https://stackoverflow.com/questions/2106503/pseudorandom-number-generator-exponential-distribution>

## Assignment 7: network connectivity

"Prove" (by means of a simulation) the threadhold formula of the [Erdős–Rényi model](https://en.wikipedia.org/wiki/Erdős–Rényi_model).

Part A: The user can select the n (number of vertices) and the p (probability that an edge is present). Then, your program generates a graph and checks if it is connected (and show the graph via graphviz (for reasonable n and p)).

Part B: for one n and all probabilities (0.00 ≤ p ≤ 1.00) perform m experiments and calculate the averages of connected graphs (for each probability). Show those averages in a chart, together with the n∙ln(n) value.

# Extra assignments

## Additional assignment: validation

For a cross-validation of your results, show the outcomes of (reliable) websites regarding plots, derivatives, definite integrals, Taylor series, etc. Your program connects to such a website (with GET/POST parameters) and shows the results (of course in a way that a comparison is easy).

Examples of those websites are:

* <https://www.emathhelp.net/calculators/calculus-1/online-graphing-calculator/?f=sin(x)>
* <https://www.intmath.com/functions-and-graphs/graphs-using-svg.php?function1=x^3-2x^2%2B5&function2=1.5+cos(2x)&xMin=-5&xMax=5&yMin=-2&yMax=6>
* <https://www.graphsketch.com/?eqn1_eqn=sin(x)%2Fe^x&eqn2_eqn=e^x>
* <https://www.wolframalpha.com/input/?i=sin(x)>

(and for sure there are many more possibilities)

## Additional assignment: enhancements

Simplify a function when the following parts appear inside that function:

* e0 , eln a , ea ∙ eb, ea / eb, (ea)b  
  where a and b can be any function
* ln 1, ln e, ln a + ln b, ln a - ln b, ln ab  
  where a and b can be any function
* a + b, a - b, a ∙ b, a / b, ab  
  where a and b are constants

# Tree-like structural representation of a function

To add a structural picture of a function (for example: f(x) = sin(πx)/(x+5)), please follow the following steps:

1. install GraphViz
2. adapt your $PATH (Linux) or %Path% (Windows) environment variable
3. in your application:
   1. generate a text file (e.g. abc.dot), similar to this:

graph calculus {

node [ fontname = "Arial" ]

node1 [ label = "/" ]

node1 -- node2

node2 [ label = "sin" ]

node2 -- node4

node4 [ label = "\*" ]

node4 -- node8

node8 [ label = "pi" ]

node4 -- node9

node9 [ label = "x" ]

node1 -- node3

node3 [ label = "+" ]

node3 -- node6

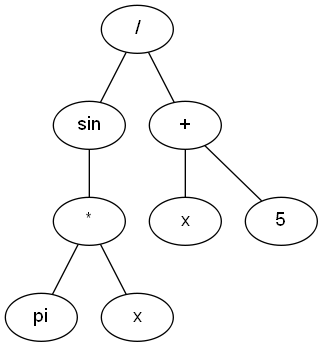
node6 [ label = "x" ]

node3 -- node7

node7 [ label = "5" ]

}

* 1. start the GraphViz executable  
     dot -Tpng -oabc.png abc.dot
  2. (wait until this executable is finished)
  3. show a picture (e.g. abc.png), for example in a PictureBox



In C#, the steps 3.2 - 3.4 *could* look like:

WriteDot("abc.dot"); // your method to write a function

// into a dot-format file

Process dot = new Process();

dot.StartInfo.FileName = "dot.exe";

dot.StartInfo.Arguments = "-Tpng -oabc.png abc.dot";

dot.Start();

dot.WaitForExit();

myPictureBox.ImageLocation = "abc.png";

In Java, the steps *could* look like:

String[] cmd = { "dot.exe", "-Tpng", "-oabc.png", "abc.dot" };

Process p = Runtime.getRuntime().exec(cmd);

p.waitFor();

File file = new File("abc.png");

Image image = new Image(file.toURL().toString());

myWidget.setImage(image);

(you can do some experiments for steps 3.1 - 3.4 in a text editor and on the command line, or you can check it on <http://www.webgraphviz.com/>)

Ideas for other nice features of GraphViz are welcome.