R. Hiptmair
S. Hosseini
C. Pagliantini

Spring Term 2014

Numerical Analysis I

ETH Zürich
D-MATH

Homework Problem Sheet 0

Introduction. Introduction to MATLAB and floating point representation of numbers.

Problem 0.1 Introduction to MATLAB

On the course website, you find an introduction "Learning MATLAB by doing MATLAB". Work through this manual step by step (no submission).

Problem 0.2 Approximation of Euler's Number e

We will discuss different methods to approximate Euler's number e based on the following two properties of e:

(i) Finite approximation $\mathrm{e}_L(n)$ of the limit of a sequence:

$$\mathrm{e} = \lim_{n\to\infty} \left(1+\frac{1}{n}\right)^n \quad \Longrightarrow \quad \mathrm{e}_L(n) = \left(1+\frac{1}{n}\right)^n. \tag{0.2.1}$$

(ii) Finite approximation $\mathrm{e}_S(n)$ of an infinite series:

$$\mathrm{e} = \sum_{k=0}^{\infty} \frac{1}{k!} \quad \Longrightarrow \quad \mathrm{e}_S(n) = \sum_{k=0}^{n-1} \frac{1}{k!}. \tag{0.2.2}$$

(0.2a) Write a MATLAB-function...

... `calcEulerLimit(n)` that calculates $\mathrm{e}_L(n)$,

... `calcEulerSum(n)` that calculates $\mathrm{e}_S(n)$ using the MATLAB-function `factorial`,

... `calcEulerSum2(n)` that calculates $\mathrm{e}_S(n)$ without using `factorial` but instead by saving the denominator of the preceding summand.

(0.2b) Use the MATLAB-functions `tic` and `toc` to measure the execution times of your three functions from subproblem (0.2a) for $n = 1, \ldots, 1000$ and visualize the measured times in a plot with logarithmic y-axis. Repeat the measurement a few times until you get a characteristic plot without too many outliers. Which of your functions is the fastest? Is there a function for which the execution time does not depend on the input number n? Why?

(0.2c) Plot the relative errors

$$\mathrm{err}_{\mathrm{rel},X}(n) := \frac{|\mathrm{e}_X(n) - \exp(1)|}{\exp(1)} \qquad \text{for } X \in \{L, S\}$$

of the two approximations for $n = 1, \ldots, 20$ in one logarithmic plot. Which approximation converges faster?

(0.2d) For which n is the absolute error $\mathrm{err}_{\mathrm{abs},S}(n) := |\mathrm{e}_S(n) - \exp(1)|$ of the approximation via the infinite series smaller than 10^{-7}? What is the smallest possible absolute error of the approximation via the limit? To answer this question, calculate $\mathrm{err}_{\mathrm{abs},L}(n)$ for $n = 10^0, 10^1, \ldots, 10^{16}$ and state the smallest of these errors. Can you explain what happens for large n?

Listing 0.1: Testcalls for Problem 0.2

```
errors_abs
```

Listing 0.2: Output for Testcalls for Problem 0.2

```
>> test_call

n =

    12

E =

                   0   0.718281828459046
   1.000000000000000   0.124539368359043
   2.000000000000000   0.013467999037517
   3.000000000000000   0.001357896223452
   4.000000000000000   0.000135901634120
   5.000000000000000   0.000013591266748
   6.000000000000000   0.000001359363292
   7.000000000000000   0.000000134326964
   8.000000000000000   0.000000030111688
   9.000000000000000   0.000000223552515
  10.000000000000000   0.000000224775742
  11.000000000000000   0.000000224898065
  12.000000000000000   0.000241667578192
  13.000000000000000   0.002171794372145
  14.000000000000000   0.002171794372023
  15.000000000000000   0.316753378090216
  16.000000000000000   1.718281828459046

min =

   8.000000000000000   0.000000030111688
```

Problem 0.3 Approximation of π

We discuss different methods to approximate the number π.

(0.3a) The *Monte-Carlo*-method is based on the fact that the area of the unit circle is π.

Write a MATLAB-function `calcPiMC(n)` in which you generate n pairs (x_i, y_i) of uniformly distributed pseudorandom numbers in $[0,1]^2$ using `rand` in a `for` loop. Determine how many pairs (x_i, y_i) lie inside the first quadrant of the unit circle: The ratio of that number by n is an approximation of $\frac{1}{4}\pi$. Let then the resulting approximation of π be the function's return value.

Test your function for $n = 10^i$, $i = 1, \ldots, 7$ and list the results.

Now write a vectorized version `calcPiMCVec(n)` of the above function that does not use any explicit loops as e.g. `for`. In addition, write a script `execution_times.m` to measure the time for $n = 10^i$, $i = 1, \ldots, 7$ it takes to calculate the approximation of π via `calcPiMC(n)` and `calcPiMCVec(n)`, respectively. Comment on your results.

(0.3b) Partial sums of known infinite sums for π generate approximations of π. For each of the following two sums, write a function `calcPiSumA(n)` and `calcPiSumB(n)`, that calculates an approximation of π on the first n terms of the sums.

$$\text{(A)} \qquad \frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots. \tag{0.3.1}$$

$$\text{(B)} \qquad \frac{\pi^2 - 8}{16} = \sum_{k=1}^{\infty} \frac{1}{(4k^2-1)^2} = \frac{1}{3} + \frac{1}{15} + \frac{1}{35} + \ldots. \tag{0.3.2}$$

(0.3c) For $n = 10^i$, $i = 1, \ldots, 4$, calculate the errors of the approximations comparing your results to the MATLAB-constant `pi` for all three methods. Plot the errors using `loglog` in one log-log diagram.

Which of the discussed methods converges the fastest? Can you analytically explain why?

Listing 0.3: Testcalls for Problem 0.3

```
execution_times
```

Listing 0.4: Output for Testcalls for Problem 0.3

```
>> test_call
i = 1: with for-loop:    Elapsed time is 0.207747 seconds.
i = 1: without for-loop: Elapsed time is 0.019630 seconds.
i = 2: with for-loop:    Elapsed time is 0.000708 seconds.
i = 2: without for-loop: Elapsed time is 0.000122 seconds.
i = 3: with for-loop:    Elapsed time is 0.008695 seconds.
i = 3: without for-loop: Elapsed time is 0.000291 seconds.
i = 4: with for-loop:    Elapsed time is 0.065324 seconds.
i = 4: without for-loop: Elapsed time is 0.001642 seconds.
i = 5: with for-loop:    Elapsed time is 0.661119 seconds.
i = 5: without for-loop: Elapsed time is 0.041397 seconds.
i = 6: with for-loop:    Elapsed time is 5.444969 seconds.
i = 6: without for-loop: Elapsed time is 0.106453 seconds.
i = 7: with for-loop:    Elapsed time is 50.537209 seconds.
i = 7: without for-loop: Elapsed time is 1.565055 seconds.

```

```
pi =

   2.000000000000000   3.200000000000000
   3.280000000000000   3.120000000000000
   3.148000000000000   3.080000000000000
   3.137200000000000   3.146800000000000
   3.138680000000000   3.145000000000000
   3.140108000000000   3.141520000000000
   3.141388400000000   3.141259600000000
```

Problem 0.4 Lucas Numbers

The sequence L_n ($n = 0, 1, \ldots$) of Lucas numbers is defined as follows:

$$F_n = \begin{cases} 1 & \text{if } n = 1, \\ 3 & \text{if } n = 2, \\ F_{n-1} + F_{n-2} & \text{if } n > 2. \end{cases} \tag{0.4.1}$$

(0.4a) Write a MATLAB-function `lucas(n)` that takes a positive integer n as an input and returns the Lucas numbers $L_1, \ldots, L_n$ in a vector. Print the result for $n = 10$.

(0.4b) Write a MATLAB-function `lucasQuot(n)` that takes a positive integer n as an input and returns the quotients $q_k = \frac{L_{k+1}}{L_k}$ for $k = 1, \ldots, n$ in a vector. Print the result for $n = 10$.

(0.4c) Use the MATLAB-function `semilogy` and the function `lucasQuot` you came up with in subproblem (0.4b) to plot $|q_k - \phi|$ with $\phi = \frac{1+\sqrt{5}}{2}$ for $k = 1, \ldots, N$ as a function of k in a graph with logarithmic y-axis. For this, set $N = 30$. Explain the behaviour of the graph and try to explain what happens for $N = 60$.

HINT: You can find the detailed documentation of `semilogy` in the MATLAB-Documentation or by typing `help semilogy` into the MATLAB-console.

Listing 0.5: Testcalls for Problem 0.4

```
L = lucas(10)

Q = lucasQuot(10)
```

Listing 0.6: Output for Testcalls for Problem 0.4

```
>> test_call

L =

     1
     3
     4
     7
    11
    18
```

```
    29
    47
    76
   123

Q =

    3.0000
    1.3333
    1.7500
    1.5714
    1.6364
    1.6111
    1.6207
    1.6170
    1.6184
    1.6179
```

Problem 0.5 Floating-Point Arithmetic

Every element $x \in \mathbb{F}$, where $\mathbb{F}$ denotes the set $\mathbb{F} = \mathbb{F}(\beta, t, e_{\min}, e_{\max})$ of floating point numbers with $\beta \in \mathbb{N}$, $\beta \geqslant 2$, $t \in \mathbb{N}$ and $e_{\min} \leqslant e_{\max} \in \mathbb{Z}$, is of the form

$$x = \pm\beta^e \sum_{i=1}^{t} \frac{d_i}{\beta^i}, \quad \text{where} \quad \begin{cases} \{d_i\}_{i=1}^t \subset \{0, 1, \ldots, \beta - 1\}, \\ x = 0 \iff d_1 = 0, \\ e \in \mathbb{Z} \cap [e_{\min}, e_{\max}]. \end{cases}$$

To approximate a real number $x \in \mathbb{R}$ by a number $\mathrm{rd}(x) \in \mathbb{F}$, it is reasonable to take the "nearest" floating point number, i.e. a number $\mathrm{rd}(x) \in \mathbb{F}$ such that $|\mathrm{rd}(x) - x| = \min_{y \in \mathbb{F}} |y - x|$. If the latter minimum is not unique, i.e. if there are two $y \in \mathbb{F}$ minimizing $|y - x|$, $\mathrm{rd}(x)$ is defined as the one with d_t *even*.

Now, let $\mathbb{F} = \mathbb{F}(2, 3, -1, 1)$.

(0.5a) Determine all numbers in $\mathbb{F}$. Do that first by hand, and then write a MATLAB function that handles it.

(0.5b) Mark $\mathbb{F}$ on the real number line. You may want to do this in MATLAB.

(0.5c) Sketch the graphs of the functions

$$\mathrm{rd} : [x_{\min}; x_{\max}] \to \mathbb{F},\ x \mapsto \mathrm{rd}(x) \quad \text{and} \quad \mathrm{err} : [x_{\min}; x_{\max}] \to \mathbb{F},\ x \mapsto |\mathrm{rd}(x) - x|,$$

where $x_{\min} := \min\{x \in \mathbb{F} \mid x > 0\}$ and $x_{\max} := \max \mathbb{F}$.

Listing 0.7: Testcalls for Problem 0.5

```
F = ComputeF(2,3,-1,1)
```

Listing 0.8: Output for Testcalls for Problem 0.5

```
>> test_call

F =

    0.2500
    0.3750
    0.3125
    0.4375
    0.5000
    0.7500
    0.6250
    0.8750
    1.0000
    1.5000
    1.2500
    1.7500
```

Published on February 17, 2014.

To be submitted on February 25/26, 2014.

MATLAB: Submit all files in the online system. Include the files that generate the plots. Label all your plots. Include commands to run your functions. Comment on your results.

Written exercises: Hand-in the solutions during the exercise class or in the labeled boxes in HG G 53.x.

References

[NMI] Lecture Slides for the course "Numerical Analysis I".

Last modified on February 17, 2014