Fundementals of Simulation Methods: Assignement 1

By: Max Argus, Constantin Pape

Excercise 1:

The machine epsilon (eps) is determined with a loop that runs as long as 1 + eps evaluates to something unequal to 1.
During every cycle of this loop eps is divided by a divisor for which a value >~ 1 is chosen to approximate the actual value for epsilon by this method as accurately as possible.
For the source code see a1.cpp.

Results: a) eps_float = 5.96047E-08
b) eps_double = 1.11022E-16
c) eps_long = 5.42101E-20

Excercise 2:

For this calculations we get the following results (source code see a2.cpp):

x = 1
y = 0

x is the correct result. In this case the law of associativity is broken, because when calculating y b + c = -1.0E17 + 1 evaluates to -1.0E17. Therefore y yields the wrong result.

Excercise 3:

a):

We get the following results: (see a3.cpp for the source code)

i = 100
j = 99
k = 100

As we can see i and k are correct whereas j is off by one.
Apparently this error results from casting float to double with the line

double y = x;

We do not know what exactly causes this error.

b):

The rational number that is presented by x is determined by calculating the first 24 significant bits of the binary representation of 0.01 and then converting this binary number again to a decimal number.
This yields (see a3b.py for details):

x = 0.00999999977648

Converted to x = n/m this yields:

n = 62499998603
m = 6250000000000

Excercise 4:

In the interval between 1.0 and 2.0 there are approximately as many numbers as different permutation of the mantissa. Thus:

$N\_1 = 2^p$ where p is the length of the mantissa.

For float (p=23) and double (p = 52) this yields

N_1_float = 2^23 = 8388608

N_1_double = 2^52 ≃ 4.5E15

In the Interval between 256.0 and 512.0 there are as many numbers as in the interval 1.0 and 2.0. Therefore an estimation of the amount of numbers in the interval between 511.0 and 512.0 is given by:

```
N_511-512    =    1 / (512 – 256) * N_1
             =    2^-8 * N_1
             =    2^(p-8)
```

This yields:

N_511-512_float = 2^15 = 32768

N_511-512_double = 2^44 ≃ 1.8E13

Excercise 5:

Reading in the numbers and summing them up is done with C++ (see a5.cpp for source code)

a): Summing up the values in the order they were read in yields:

sum_a = -6.51624E+15

b): Summing up the values in reverse order yields:

sum_b = 9.69531E+25

c): Sorting the values by magnitude and then summing them up yields:

sum_c = 0

d): Following the same procedure as in c) but using a long double as summation variable yields (We did not enable „true“ quadruple precision):

sum_d = 0

At this point we believed that this result is correct, because it agrees with c.

e): Summing up the sorted value using GMP-Library functions for calculation yields:

sum_e = 42

This surely is the correct result :).