MA 3257 – SPRING 2023 C-TERM
HOMEWORK I (DUE JAN 19TH, 2023)

Problem 1. (Computing in finite precision – MATLAB; 5 points)

(1) (1 point) As we have spoken in class, there are gaps between consecutive numbers in floating point number representation. The size of these gaps depends on the size of the number and on the precision. MATLAB provides the function `eps()`, which returns, for a number, the distance to the next floating point number in the same precision (typing `eps` itself yields the **machine epsilon**). Using the form of double and single floating point number representation, explain the values you find for

```
eps(1)
eps(single(1))
eps(2^40)
eps(single(2^40))
```

(2) (1 point) Try the following experiment and explain the behaviour[1]:

```
a=0.5;
b=0.7-0.2;
a==b;
sprintf('%20.18f',a)
sprintf('%20.18f',b)
```

(Hint: you are welcome to try decimal-floating-point-conversion, and use it to explain the above. An extra 3 points will be given to those who write a correct algorithm that converts any decimal input to its 64-bit floating point representation, following what the webpage says.)

(3) (3 points) We can compare the speed of mathematical operations when using single vs double precision. Consider the following program that performs 10^9 additions of (random) floating point numbers, and measures the time it takes for doing that.

```
N=1e5;
M=1e4;
a=rand(N,1);
b=rand(N,1);
tic
for␣i=1:M
a.*b;
end
toc
```

Report the run time of this program. Now modify N and M such that their product remains the same, and again report the run times (do for several pairs of N and M, enough to make a point). If you wish to loop over N and M over an index j for these new experiments, you may consider using

```
tmul(j)=toc;
```

to record the run time of each program. Be careful that each `toc` measures the run time to the previous `tic`.

Lastly, repeat the exercise but use single precision random numbers (the default is double precision) by replacing the definitions of the vectors a and b as follows

```
a=single(rand(N,1));
b=single(rand(N,1));
```

The differences you will observe are due to the need to read numbers from memory, which takes time, before they can be added by the processing unit (the CPU).

[1] a==b is checking whether the number a has exactly the same floating-point representation as the number b.

Problem 2. (5 points) Use 4-digit rounding with the quadratic formula to find the roots of the following equation

$$x^2 + 62.1x + 1 = 0.$$

The exact solutions are $x_1 = -0.01610723$ and $x_2 = -62.083892$.

(1) (2 points) Find the relative error of the approximations $\widehat{x_1}$ and $\widehat{x_2}$.
(2) (3 points) How do we alleviate the large rounding error of one of the roots? Hint: manipulate the quadratic formula or use the relationship between the two roots of a quadratic equation.

Problem 3. (Reversing the order is better?!; 5 points)

(1) (2 points) Use three-digit chopping arithmetic to compute the sum $\sum_{i=1}^{10} \left(1/i^2\right)$ first by $1/1 + 1/4 + \cdots + 1/100$ and then by $1/100 + 1/81 + \cdots + 1/1$. Which method is more accurate, and why?
(2) (3 points) Write a program (pseudocode is fine) that compute the sum $\sum_{i=1}^{N} x_i$ in the reverse order.

Problem 4. (Approximating e; 5 points + Extra credit 5 points)

In class, we spoke very briefly about several ways to approximate e. One such way is using the Maclaurin series

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}, \quad n! = n\,(n-1)\,(n-2)\cdots(2)\,(1)\,, \quad n \neq 0,$$

and $0! = 1$. Use four-digit chopping arithmetic to compute the following approximations to e, and determine the absolute and relative errors. (Presentation 1 point)

(1) (1 point) $\sum_{n=0}^{5} \frac{1}{n!}$;
(2) (1 point) $\sum_{n=0}^{10} \frac{1}{n!}$;
(3) (1 point) $\sum_{n=0}^{5} \frac{1}{(5-n)!}$;
(4) (1 point) $\sum_{n=0}^{10} \frac{1}{(10-n)!}$.
(5) (Extra credit 5 points) Consider the sums

$$\sum_{n=0}^{N} \frac{1}{n!}, \quad \sum_{n=0}^{N} \frac{1}{(N-n)!}.$$

Plot their absolute errors to approximate e as a function of N in the same figure. Do the same for relative error.

Problem 5. (Relative error for rounding arithmetic; 5 points) In lecture, we found that k-digit chopping for a machine representation of a real number

$$y = 0.d_1 d_2 \ldots d_k d_{k+1} d_{k+2} \ldots \times 10^n$$

is given by

$$error_{relative} = \left| \frac{y - fl\,(y)}{y} \right| \leq \frac{1}{0.1} \times 10^{-k} = 10^{-k+1}.$$

Recall that rounding involves first adding $5 \times 10^{n-(k+1)}$ and then perform k-digit chopping. Determine the relative error for rounding $error_{round}$ and provide a suitable upper bound.