Homework No.1, 553.481/681, Due February 9, 2024.

Problem 1. (a) Show that to convert between binary and hexadecimal representations, each hecadecimal digit b is replaced, in order, by the four binary digits a, a', a'', a''' that satisfy

$$b = 2^3 a + 2^2 a' + 2a'' + a'''.$$

(b) Use part (a) to write a Matlab script `hextobin.m` which converts any n-dimensional vector $[h_1, ..., h_n]$ of the values $h_i \in \{0, 1, 2, ..., 9, a, b, ...f\}$, with the definitions

```
a=10;b=11;c=12;d=13;e=14;f=15;
```

into a vector $[b_1, ..., b_{4n}]$ of the values $b_i \in \{0, 1\}$. *Hint:* Use Matlab's `mod` function.

(c) Apply your script from part (b) to find the binary equivalent of the hexadecimal string `f593c48b`.

Problem 2. In IEEE Standard Floating-Point representation for numbers, a 12-bit binary number is used first to encode the sign σ and exponent E, followed by a 52-bit binary to encode the fraction F. This can also be represented by a 3-digit hexadecimal number, followed by a 13-digit hexadecimal number, as in `MATLAB`. Consider the following IEEE Standard Floating-Point number in hexadecimal form:

```
x=bff75d9cb07e7400
```

(a) Use the first 3 hexadecimal digits to determine the sign σ and exponent E of `x` in decimal format.

(b) Convert the remaining 13 hexadecimal digits to a decimal representation of the fraction F for `x`.

(c) Calculate the decimal representation of the double-precision number `x`. You can use the Matlab intrinsic function `hex2num` to check your answer in (c), but you must explain how the result follows from parts (a) and (b).

Problem 3. This problem concerns a floating-point arithmetic with base β, precision p, and exponents E in the range $L \leq E \leq U$.

(a) Show that the underflow level for normalized numbers is $UFL = \beta^L$.

(b) Show that overflow level is $OFL = (\beta - \beta^{-p})\beta^U$.

(c) Show that the underflow level for denormalized numbers is $UFL_* = \beta^{L-p}$.

Problem 4. (a) Use Matlab's function `fplot` to plot the function

$$f(x) = x^8 - 8x^7 + 28x^6 - 56x^5 + 70x^4 - 56x^3 + 28x^2 - 8x + 1$$

over the x-interval $[0.98, 1.02]$. Does the result look like the plot of a smooth polynomial?

(b) Matlab's function `roots` approximates the n roots of a general nth-degree polynomial $p(x) = c_n x^c + c_{n-1} x^{n-1} + \cdots c_1 x + c_0$ by numerically computing the eigenvalues of a so-called "companion matrix". Calculate the 8 roots of the polynomial in part (a) using the function `roots` and compare them with the exact roots = 1 obtained by factorizing the polynomial. Do the approximate roots agree with the exact roots to double precision accuracy? Explain why this should have been expected based upon the results in part (a).

(c) Write the function in part (a) in factorized form

$$f(x) = (x-1)^8$$

and then use `fplot` to plot the function in this form over the same interval $[0.98, 1.02]$. How does the plot compare with that in part (a)? Explain why the plots differ so drastically.

(d) Use the Matlab function `roots` to find the zeros of the polynomial $g(y) = f(1+y) = y^8$ and those of $f(x)$ by setting $x = y + 1$. Are the roots now correct to double precision?

Problem 5. Consider the following linear system

$$\begin{aligned} x + \alpha y &= 1 \\ \alpha x + y &= 0 \end{aligned}$$

for the real parameter α.

(a) Find the exact solutions $x(\alpha)$, $y(\alpha)$ as functions of α. Calculate the infinitesimal condition numbers $K_x(\alpha)$, $K_y(\alpha)$ of these functions. How do these behave for $\alpha \to 1$?

(b) Find the function $z(\alpha) = x(\alpha) + y(\alpha)$ and simplify its form as much as possible. Calculate the infinitesimal condition number $K_z(\alpha)$ and determine its behavior for $\alpha \to 1$.

(c) Evaluate the function $z(\alpha)$ for $\alpha = 1 - 10^{-8}$ by solving the given linear system using the Matlab function `mldivide(A,b)` or, equivalently, `A\b` and then evaluate the function $z(\alpha)$ by summing the two components of the solution vector. Find a second approximation to $z(\alpha)$ for the same value of α by directly evaluating the expression in (b). What is the relative difference between the two approximations? Which value do believe is more accurate and why?

Problem 6*. (a) Show that the following infinite series can be exactly evaluated as

$$S := \sum_{n=1}^{\infty} \frac{1}{n(n+1)} = 1.$$

Show furthermore that the kth partial sum S_k and its remainder $R_k = 1 - S_k$ are given by

$$S_k = \sum_{n=1}^{k} \frac{1}{n(n+1)} = 1 - \frac{1}{k+1}, \quad R_k = \frac{1}{k+1}.$$

(b) Consider this Matlab script to evaluate the infinite series in single-precision arithmetic:

```
nn=1;
dS=single(1/nn/(nn+1));
Sold=0;
SS=dS;
while (abs(SS-SSold)>0)&(nn<1e16)
    nn=nn+1;
    dS=single(1/nn/(nn+1));
    Sold=SS;
    SS=SS+dS;
end
```

Run the code and state the approximate value of S obtained, the error, and the largest value of n employed. Is the error consistent with the exact result for the remainder?

(c) Was single precision accuracy obtained in part (b)? If so, explain why. Otherwise, explain why the algorithm failed to achieve a single precision result and, if you can, devise a more accurate numerical approach using only single-precision arithmetic.