553.481/681 Numerical Analysis – Homework 1 Solutions

Matthew Hudes

Problem 1. (a) Any binary number $(\cdots b_2 b_1 b_0 . b_{-1} b_{-2} \cdots)_2$ can be written as the decimal $\sum_i b_i 2^i = \sum_i (b_{4i+3} \cdot 2^3 + b_{4i+2} \cdot 2^2 + b_{4i+1} \cdot 2 + b_{4i}) 2^{4i}$. Since $\forall i : b_i \in \{0, 1\}$, we must have that $b_{4i+3} \cdot 2^3 + b_{4i+2} \cdot 2^2 + b_{4i+1} \cdot 2 + b_{4i} \in \{0, \ldots, 15\}$, i.e. is represented by one hexadecimal symbol, and noting that $2^{4i} = 16^i$, we're done.

(b) Coded in `MATLAB`.

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

h=[1 f b c 4 7 a 3];
n=length(h);

bb=[];
for ii=1:n
        z=h(ii);
        z0=mod(z,2);
        z=(z-z0)/2;
        z1=mod(z,2);
        z=(z-z1)/2;
        z2=mod(z,2);
        z3=(z-z2)/2;
        bb=[bb, [z3 z2 z1 z0]];
end

bb=num2str(bb);
bb= bb(~isspace(bb))
```

(c) With `h='f593c48b'`, we obtain the binary representation `11110101100100111100010010001011`.

Problem 2. (a) We can convert $(bff)_{16} = 11 * 16^2 + 15 * 16^1 + 15 * 16^0 = 3071$. Since $3071 > 2^{11} = 2048$, the sign is $\sigma = -1$ and we subtract the bias plus $2^{11} = 2048$ from this characteristic to find the exponent

$$E = 3071 - (2^{r-1} - 1 + 2^1 1) = 3071 - (2^1 0 - 1 + 2048) = 3071 - 3071 = 0$$

(b) We can convert

$$\begin{aligned} F = (0.75d9cb07e7400)_{16} &= (7, 5, d, \ldots, 4, 0, 0) \cdot (16^{-1}, 16^{-2}, 16^{-3}, \ldots, 16^{-11}, 16^{-12}, 16^{-13}) \\ &= 0.4603545088 \end{aligned}$$

(c) Per IEEE format we know the number is

$$\begin{aligned} \sigma[1 + F] \cdot 2^E &= -1[1 + 0.4603545088] \cdot 2^{3071} \\ &= -1.4603545088 \end{aligned}$$

.
This is the value of the Riemann zeta function $\zeta(s)$ at $s = 1/2$

Problem 3. (a) Note $(1.0\ldots0)_\beta \cdot \beta^L = \beta^L$.

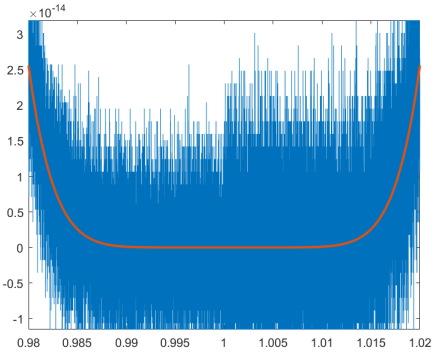
(b) Let $\xi := \beta - 1$ and write

$$\begin{aligned}(\xi.\xi\cdots\xi)_\beta \cdot \beta^U &= \xi \sum_{i=0}^{p} \beta^{-k} \cdot \beta^U \\ &= (\beta-1)\sum_{i=0}^{p} \beta^{-k} \cdot \beta^U \\ &= (\beta-1)\frac{1/\beta^p - 1}{1/\beta - 1} \cdot \beta^U \\ &= (\beta - \beta^p)\cdot\beta^U.\end{aligned}$$

(c) Write $(0.0\cdots01)_\beta \cdot \beta^L = \beta^{-p}\cdot\beta^L = \beta^{L-p}$.

Problem 4. Code in `MATLAB`. Graph below.

```
f=@(x) x.^8-8*x.^7+28*x.^6-56*x.^5+70*x.^4-56*x.^3+28*x.^2-8*x+1;
fplot(f,[0.98,1.02])
rr=roots([1 -8 28 -56 70 -56 28 -8 1])
g=@(x) (x-1).^8;
hold on
fplot(g,[0.98,1.02],'LineWidth',2)
ss=roots([1 0 0 0 0 0 0 0]);
rr=ss+1
```



(a) No, the result does not look like the plot of a smooth polynomial.

(b) No, there is not agreement to double precision accuracy. Checking that $f(1) = 0$ is straightforward, but all of the computed roots are different from $x = 1$ by a good amount. For example, one computed root is $1.0152 + 0.0154i$. The roots are close to the region where floating point addition errors are substantial.

(c) The plot of $(x-1)^2$ using the code at the start of this problem is the red curve in the previous figure. This plot is much more accurate. The only floating point arithmetic errors in this formulation are from multiplication, which is very well-conditioned. By contrast, the formula in (a) sums up 9 terms with alternating signs that almost completely cancel each other since $1 - 8 + 28 - 56 + 70 - 56 + 28 - 8 + 1 = 0$. Thus, there are big loss of significance errors in the expression in part (a).

(d) Yes. Using the code at the beginning of the problem, the computed roots are uniformly 1.

Problem 5. (a) By substitution it's easy to see that $x(\alpha) = 1/(1-\alpha^2)$ and $y(\alpha) = -\alpha/(1-\alpha^2)$. We have

$$K_x(\alpha) := \frac{\alpha x'(\alpha)}{x(\alpha)} = \frac{2\alpha^2/(1-\alpha^2)^2}{1/(1-\alpha^2)} = \frac{2\alpha^2}{1-\alpha^2},$$

and

$$K_y(\alpha) := \frac{\alpha y'(\alpha)}{y(\alpha)} = \frac{2\alpha(\alpha^2+1)/(1-\alpha^2)^2}{2\alpha/(1-\alpha^2)} = \frac{\alpha^2+1}{1-\alpha^2}.$$

By inspection it's evident that as $\alpha \to 1$, $K_x(\alpha), K_y(\alpha) \to \pm\infty$.

(b) Note $z(\alpha) = x(\alpha) + y(\alpha) = (1-\alpha)/(1-\alpha^2) = 1/(1+\alpha)$. Then

$$K_z(\alpha) = \frac{\alpha z'(\alpha)}{z(\alpha)} = \frac{-\alpha/(1+\alpha)^2}{1/(1+\alpha)} = -\frac{\alpha}{1+\alpha},$$

and as $\alpha \to 1$, $K(z) \to -1/2$.

(c) In `MATLAB`:

```
al=1-1e-8;

A=[1 al; al 1];
b=[1; 0];
x=A\b;
z1=sum(x)

z2=1/(1+al)

Kx=2*al^2/(1-al^2)
```

The method by addition estimates $z \approx 0.5$ and the method by direct computation estimates $z \approx 0.5000000025$. The relative difference is $4.999999975000001 \cdot 10^{-9}$, much bigger than double-precision error. The direct computation should be more accurate because $\alpha = 0.99999999$ is very close to 1, where $K_x(\alpha), K_y(\alpha)$ become big. Thus, inaccuracy in approximations to x and y carry over to inaccuracy in the approximate value of z.

Problem 6. (a) Note that

$$\frac{1}{n(n+1)} = \frac{1}{n} - \frac{1}{n+1}$$

Therefore we can get the formula for S_k:

$$S_k = \sum_{n=1}^{k} \frac{1}{n(n+1)} = \sum_{n=1}^{k} \left[\frac{1}{n} - \frac{1}{n+1}\right] = 1 - \frac{1}{k+1}$$

Note then that

$$S = \lim_{k\to\infty} S_k = \lim_{k\to\infty} (1 - \frac{1}{k+1}) = 1$$

We also can find a formula for the remainder:

$$R_k = 1 - S_k = 1 - (1 - \frac{1}{k+1}) = \frac{1}{k+1}$$

(b) We find that the algorithm terminates when n reaches 5793, with $S_{5793} = 0.9999$ and relative error $1.4728 \cdot 10^{-04}$. The exact remainder R_{5793} is $1.7259 \cdot 10^{-4}$, which is the same order of magnitude as the relative error.

(c) Single precision was not obtained in (b). The last term computed for $n = 5793$ was $1/n(n+1) \doteq 2.9793 \cdot 10^{-8}$ which is smaller than the unit round $2^{-24} \doteq 5.9605 \times 10^{-8}$ in single precision arithmetic. Since the sum of all earlier terms was close to 1, adding this small value did not change the result to single precision and the series appeared to "converge". However, the remainder was $1.4728 \cdot 10^{-4}$, which is much larger than single precision round-off error! This is an example where the last term in the sum is much smaller than the remainder and does not estimate the remainder well.

To avoid this problem, one should always sum numbers in floating-point arithmetic from small terms to large ones, not from large to small. To decide how large an n must be considered, we use the remainder formula to find an N so that $R_N = 1/(N+1) \doteq eps$ and thus find that $N = 1/eps = 2^{23} = 8,388,608$. Below is a `MATLAB` script that implements an improved algorithm:

```
NN=round(1/eps('single'))
nn=NN;
RR=1/(nn+1)
dS=single(1/nn/(nn+1));
SS=dS;
for nn=NN-1:-1:1
    dS=single(1/nn/(nn+1));
    SS=SS+dS;
end
SS=SS
relerr=abs(SS-1)
```

Running this script yields $S_{8388608} = 1.0000$ with a relative error $1.1921 \cdot 10^{-07}$, which is exactly the machine epsilon in single precision arithmetic or 2^{-23}. Thus, the correct answer is obtained to single precision.