CS 514/ Math 514 Numerical Analysis

Spring 2018

Homework 1

Given: Jan. 18, 2018; Due: Feb. 1, 2018 11:45 A.M. (15 minutes before class)

You are permitted to discuss these problems with **one** other student, and you should indicate the name of your collaborator in your answer to the first problem in this set. (Many of you expressed a preference to not collaborate with anyone, and that is fine; please indicate this on your answer to the first question.) *You should write up your own solution, and you are not permitted to either share a written copy of your solution or copy some one else's written solution.* Similarly with programs, you are not permitted to share your code with another student. If you need help with debugging your code, you should describe what the problem is without sharing or showing your code to another student. If you are stuck on a problem or on a code, you could ask the TA for help on Piazza, or in a meeting. Please do not discuss solutions to the HW problems on Piazza before they are graded.

If you copy someone else's work, or let your work be copied, you will get zero points for the entire HW, a loss of one letter grade in the course, and you will be reported to the Dean of Students. I follow the course policies described by Professor Gene Spafford (of Purdue Computer Science department) at the URL `http://spaf.cerias.purdue.edu/~spaf/cpolicy.html` If you are not familiar with this policy, please read it! Please submit your assignments using Blackboard, preferably using either a Tex-generated or Word document.

1. If you collaborated with a student in answering these questions, please write down your collaborator's name. If you did not collaborate with anyone, please indicate that too.

2. **15 points** In this problem, we explore how integers are stored in Matlab, a system that conforms to the IEEE standard. Recall from lecture that you can see the hexadecimal (base 16) representation of any number by changing to `format hex` before you display the number. You can then convert the hexadecimal number to binary to see the binary representation.

   There are four ways to represent an integer in Matlab, using 8, 16, 32, or 64 bits; there data types are called `int8`, `int16`, and so on. You can see this from the Matlab documentation, going to 'Language Fundamentals', then 'data types', and then to 'numeric types'.

   (a) Let's work with `int8`, this data type uses `8` bits to store an integer. You can see how the integer zero is stored in this data type by displaying `int8(0)`. Explain how the numbers `0`, `1`, and `int8(intmax)` are stored by considering their binary representations. What is the decimal value of `int8(intmax)`? (You might be surprised by this answer, since we could store larger positive integers using 8 bits. But all positive integers have their most significant bit equal to zero, and all negative integers have their most significant bit equal to one, in this representation.)

   (b) We consider negative integers in the `int8` data type. Negative integers are represented using "two's complement arithmetic", which makes it possible to add and subtract using the addition operation. Here to find the negative of an integer, write the integer in binary, flip every bit (change zero to one and vice-versa), and then add one in binary arithmetic. (You can get the same result by using the identity $i + (-i) = 0$, with the addition performed in binary arithmetic, and discarding any carry beyond the 8 binary digits. So write down the binary representation of i, and then find the binary number whose addition would result in zeros in the last 8 bits, ignoring the carry.)

How is -1 represented as a binary number in this system? What is the decimal value of `int8(intmin)`? Show that the binary representations satisfy the equation $\texttt{int8}(\texttt{intmax}) + \texttt{int8}(\texttt{intmin}) = -1$, where again the computation is performed in binary arithmetic.

(c) Type $\texttt{int8}(256) - \texttt{int8}(128) - \texttt{int8}(128)$ in Matlab, and report your answer. Explain your result.

(d) Now look at the binary representation of 1 without explicitly stating a data type (you should type 1 and NOT `int8(1)`). Also look at the binary representation of 1.0, `int64(1)`; and the binary representations of -1, -1.0, and `int64(-1)`. Explain what you observe. What do you conclude about how Matlab stores an integer for which an explicit type is not declared?

3. **20 points**
The IEEE floating point standard defines a single precision word to have 32 bits. Of these, one bit is for the sign of the fraction, 8 bits are used for the exponent, and the remaining 23 bits are for the fraction. A normalized representation is used for all the numbers, except zero and the subnormal numbers. Assume that the extreme values at either end of the exponent range are reserved for representing zero, Inf, NaN, subnormal numbers, etc. Answer the following questions by comparing with the way double precision floating point numbers are stored in the IEEE standard.

(a) What are the largest and smallest values that are stored in the exponent field? Assume a biased exponent value is computed from the stored values, and that the value of the bias is chosen to be the largest integer that is smaller than half the maximum value of the unbiased exponent. (You should see that this choice is similar to the 64 bit double precision numbers.) What is the range of biased exponent values in single precision?

(b) Roughly how many decimal digits are stored in the fraction in this system?

(c) What is the largest real binary number (not `Inf`) that can be represented on this system? What is its decimal value?

(d) What is the decimal value of the smallest positive normalized real number that can be represented on this system? (Ignore subnormal numbers for this question.)

(e) What is the decimal value of the largest subnormal number represented in this system?

(f) What is the value of the unit roundoff if rounding is used?

(g) How many distinct normalized numbers are stored in this system?

(h) How many distinct subnormal numbers are represented on this system?

4. **10 points** In studying computer arithmetic, we need to convert from one number system to another. Here is an example of how to convert a decimal integer (base 10) to binary (base 2). We divide the integer by 2 (the base of the binary system) and write down the quotient and the remainder. Repeat the process with the quotient.

   Consider the decimal integer 43. Then

$$\begin{aligned} 43/2 &= 21 + 1/2; \cdots 1 & 21/2 &= 10 + 1/2; \cdots 1 & 10/2 &= 5 + 0/2; \cdots 0 \\ 5/2 &= 2 + 1/2; \cdots 1 & 2/2 &= 1 + 0/2; \cdots 0 & 1/2 &= 0 + 1/2. \cdots 1 \end{aligned}$$

   Now write down the remainder from the last to the first, to get 101011, which is the binary representation of 43.

   To convert the fraction in decimal to binary, we multiply the number by 2 and write down the integer part. Repeat with the fractional part of the answer. Consider the binary representation of 0.4 in decimal arithmetic. Then

$$\begin{aligned} 0.4 \times 2 &= 0.8; \cdots 0 & .8 \times 2 &= 1.6; \cdots 1 & .6 \times 2 &= 1.2; \cdots 1 \\ .2 \times 2 &= 0.4; \cdots 0 & .4 \times 2 &= 0.8; \cdots 0 & .8 \times 2 &= 1.6; \cdots 1 \\ .6 \times 2 &= 1.2; \cdots 1 \end{aligned}$$

   Write down the integer part (0 or 1) from beginning to end after the binary point. Hence 0.4 is $0.011\ 0011\ 0011 \cdots$ in binary. Notice that we have observed a recurring pattern of four binary digits. We find that the decimal 0.4 cannot be represented exactly in a binary floating point number system since an infinite number of binary digits are required. So in IEEE double precision arithmetic, we will have to round the number to 52 binary digits.

   (a) Convert the following decimal number to a correctly rounded binary number with 8 digits after the binary point: 22/7

   (b) Convert the following binary number to a decimal number: 0.10101010

5. **10 points** A method to compute twice the unit roundoff of a binary floating point arithmetic system was devised by William Kahan, who was the prime leader in creating the IEEE floating point standard. Here it is:

```
a = 4/3;
b = a - 1;
threeb = b + b + b;
epsilon = abs(threeb - 1);
```

   (a) Run this program in Matlab (or Python on a system that implements the IEEE double precision standard for floating point arithmetic). Report the computed value of `epsilon`.

(b) Write down the binary representation of $4/3$ in the binary number system, and round it to 52 binary digits. Now write down what happens as a result of each floating point arithmetic operation when you compute b, `threeb`, and `epsilon`. Thus explain why this program computes twice the unit roundoff. Matlab also calls this `eps(1)`.

6. **10 points** Consider a decimal floating point arithmetic system. One difference between it and a binary arithmetic system is that a normalized representation of the fraction is $fl(x) = d_0 \cdot d_1 \cdots d_{t-1}$, and now since d_0 can take any value from 1 to 9, we need to store it. Show that if we use truncation, then the unit roundoff is $10^{-(t-1)}$, and if we use rounding, then the unit roundoff is 5×10^{-t}, You should show this result by an analysis similar to the one we did in class for a binary arithmetic system.

7. **10 points** Read Example 2.2 in the text book, which estimates the rounding error in single precision arithmetic by computing a function $g(t)$ in double precision and then rounding the result to single precision. We will examine if this is a reasonable approach by first computing the function $g(t)$ using single precision arithmetic, and secondly by computing it again in double precision arithmetic. The latter can be taken to be correct value of the function, and the difference in the two values (computed in double precision) estimates the rounding error in single precision computation of the function.

   Look at the code in the example, and compute the function $g(t)$ in single precision arithmetic by declaring t to be a single precision variable. Type `help single` in Matlab to see how to do this. Functions computed with a single precision variable will be evaluated in single precision.

   Plot the round-off error as done in Figure 2.2 by this approach. Visually compare your results with Figure 2.2 and comment on similarities and differences. Submit your Matlab code and the plot of the round-off error, and your comments.

8. **10 points** In this problem, we write programs to compute the sum of the reciprocals of the numbers $1/i$, for $i = 1, \ldots, n = 100,000$.

   (a) Compute the sum in single precision arithmetic, in two different orderings: First compute the sum in increasing order of i with i equal to 1 to n, and then in the reverse order, from i equal to n to 1. How do these two results compare? (Examine 7 digits after the decimal point.)

   (b) Next repeat the computations with the two different orderings in double precision arithmetic. Compare the two values obtained with each other (now you have 15 digits after the decimal point in the result), and with the values you obtained in single precision arithmetic. Which ordering to sum these numbers gives more accurate results in single precision? Give a reason for why this might be so.