PHYSICS 115/242

Homework 1

Due in class, Monday, April 8.

Note:

- **Please read these notes carefully.**
- The answers to the questions must include the computer code and output, in addition to any writing that might be needed. If you are not sure what is required please ask me, preferably *before* the last evening before the homework is due.
- Please assemble all parts of a question *together*. I don't want to have the code for all the questions together, followed by all the output, followed by all the explanations. Rather, I require all parts of question 1, followed by all parts of question 2, etc.
- To print the output of your code, please save the output to a file and print the file (rather than dump the screen). Using the editor you can then combine the code and the output into one file, and possibly also add any text needed to explain your answer.

  In all computer systems that I'm aware of you can redirect the output of a command to a file using the ">" symbol. For example, if "a.out" if the name of the executable, to put the output from it in a file called "out" give the command

  ```
  a.out > out
  ```

- You need control over the number of digits to be presented in your output. I recommend you use the function `printf` which works with C++ as well as C compilers. (You need to include the statement `#include <stdio.h>`.) The `cout << ...` command in C++ doesn't seem to give enough control. The function `scanf` for reading in parameters from the command line is also useful. I suggest you familiarize yourself with `printf` and `scanf`.
- In some problems there will be many lines of output only a few of which are relevant. For example, in Qu. 2(b) there will be about 309 lines of output for the double precision part. To save paper just print the first 2 or 3 lines and the last 2 or 3 lines. You could indicate the missing lines by a line with something like .....
- In problems like Qu. 5 where you have to show that the error goes as a particular of a small parameter h, choose values for h which decrease in a *geometric* (rather than arithmetic) sequence (i.e. divide by 2 or 10, whichever is most appropriate, each time).

1. (a) Kernighan and Ritchie, who created the C language, advise that "The first program to write in any language is the same for all languages: Print the words '`Hello world.`'. This is the basic hurdle; to leap over it you have to be able to create the program text somewhere, compile it successfully, load it, run it, and find out where your output went. With these mechanical details mastered, everything else is comparatively easy." Write a `hello world` program.

   (b) i. Print the number 19 billion in scientific notation, i.e. the result should look something like `1.9e+10` (e.g. use the `%e` format in `printf`.

   ii. Print

   `The value of the golden mean is ...`

   where you replace "..." by the numerical value of $(\sqrt{5}-1)/2$, which you should give to 8 decimal places.

2. (a) Write a program which shows how many bits are used to represent integers with your compiler, and hence state what is the largest positive integer that can be represented. *Hint:* As illustrated in class, you could start with 1 and keep doubling. Just show the few relevant lines of output.

   (b) Write a program which shows (at least roughly) what is the largest floating point number that can be represented using your compiler. Most compilers have two levels of precision, e.g. "float" and "double" in C. Give results for both precisions.

   (c) Write a program which shows (at least roughly) what is the smallest (in magnitude) floating point number, different from zero, that can be represented using your compiler. Again give results for both single and double precision.

   (d) Write a program which shows (at least roughly) how many digits of precision there are for floating point numbers using your compiler. Another way of expressing this is: what is the closest number to 1 that can be distinguished from 1. As in the previous parts give results for two levels of precision.
   Note: It is because there are only a finite number of digits of precision in floating point numbers that *roundoff errors* arise in numerical computation.

3. Even calculating the sum of a simple series can require some care. Consider the series:

$$S^{(up)} = \sum_{n=1}^{N} \frac{1}{n},$$

which is finite as long as N is finite. When summed analytically, it does not matter if you sum the series upwards from $n = 1$ or downwards from $n = N$,

$$S^{(down)} = \sum_{n=N}^{1} \frac{1}{n}.$$

However, because of roundoff error, when summed numerically, $S^{(up)} \neq S^{(down)}$.

(a) Write a program to calculate $S^{(up)}$ and $S^{(down)}$ in single precision for $N = 10^p$, with $p = 2, 3, 4, 5, 6$ and 7.

(b) Also calculate $S^{(up)}$ and $S^{(down)}$ in double precision.

(c) Show that in double precision $S^{(up)}$ and $S^{(down)}$ agree to high accuracy.

(d) Taking the double precision result to be correct within the desired accuracy, show that, with single precision, $S^{(up)}$ is less accurate than $S^{(down)}$ and that the error increases with increasing N.

(e) Explain in words why $S^{(up)}$ is less accurate than $S^{(down)}$.

4. This problem shows that in certain algorithms roundoff errors increase exponentially. We say that such algorithms are *unstable*. The first rule of numerical analysis is to avoid unstable algorithms.
The "Golden Mean", ϕ, is given by $\phi = (\sqrt{5} - 1)/2 \simeq 0.61803399$.

(a) Determine the n-th power of ϕ, using successive multiplications,

$$\phi^0 = 1, \quad \phi^n = \phi \cdot \phi^{n-1},$$

for $n = 1, 2, 3, \cdots$.

(b) Now lets try to be "clever". Show that the following recursion relation holds:

$$\phi^{n+1} = \phi^{n-1} - \phi^n. \tag{1}$$

Hint: reduce this to a quadratic equation.
Hence we can calculate ϕ^n using only subtraction rather than multiplication by constructing an array of numbers $\Phi[0], \Phi[1], \cdots$ from the following relation

$$\Phi[n+1] = \Phi[n-1] - \Phi[n], \tag{2}$$

with the "boundary conditions" $\Phi[0] = 1, \Phi[1] = \phi$. The value of $\Phi[n]$ will then be ϕ^n.

(c) Use Eq. (2), in both single and double precision to compute ϕ^n, for $n = 1, 2, \cdots, 50$. Compare the results with those in part (a). Is the recursion relation unstable?

(d) Show that there is another number, $\tilde{\phi}$, which satisfies the recursion relation in Eq. (2). What is the *general* solution of Eq. (2)? Using the form of the general solution, explain the instability.

5. (a) Show analytically that the following "midpoint" expression for the second derivative

$$f''_{mid}(x) = \frac{f(x+h) + f(x-h) - 2f(x)}{h^2}$$

has an error of order h^2.

(b) Confirm this numerically for a function and x value of your choice.
Note: Do not let h become too small or roundoff errors will spoil your analysis.

6. *For Physics 242 students only.* This material of this question is discussed in Sec. 3.7 of the recommended text, Landau and Páez, which is is on reserve on the library. (I also have a copy from which you may photocopy that section.) See also Numerical Recipes, Sec. 6.7 (and Sec. 6.5) for further details.

   Write a program to compute the spherical Bessel functions $j_l(x)$ for $x = 0.1, 1$ and 10 and $l = 3, 5, 8$ and 12. Choose a value for x and use the recurrence relation

$$j_{l-1}(x) = \frac{2l+1}{x} j_l(x) - j_{l+1}(x), \tag{3}$$

   starting from a *large* value l (e.g. $l = 50$) with some assumed values (say 1) for $j_l(x)$ and $j_{l+1}(x)$ (the answer will not depend on these values) and then determine the ratio of the resulting value of $j_0(x)$ with that obtained from the correct function

$$j_0(x) = \frac{\sin(x)}{x}.$$

   Multiply your values for the higher l values by this factor. This method works for x smaller than the largest value of l used in the downward iteration.
   Note:

   - Results for the first three values of n are given in the book but it is not enough to quote the result; you must show the computer output and the code that produced it.
   - One can rewrite Eq. (3) so it is an *upward* recursion

$$j_{l+1}(x) = \frac{2l+1}{x} j_l(x) - j_{l-1}(x). \tag{4}$$

     It looks tempting to use this to compute the value of $j_l(x)$ for any l since we know the first two functions

$$j_0(x) = \frac{\sin x}{x}, \qquad j_1(x) = \frac{\sin x - x\cos x}{x^2}\,.$$

     Unfortunately, Eq. (4) is *unstable* at least for small values of x, and so is not useful in practice. You may want to check this. The reason is that there is a *second* function which satisfies the recurrence relations, the spherical *Neumann* function, $n_l(x)$. Using the upward recursion, the amount of the unwanted Neumann function function grows so the algorithm is unstable. This is similar to the problem with calculating the golden mean from the recurrence relation Eq. (2) in Qu. (4). However, with the downward recursion relation, the amount of the Neumann function diminishes (at least if x is smaller than the order, l, at which the iterations start) and eventually becomes negligible even if is significant to start with.

     The stability, or otherwise, of these recursion relations depends on the fact that $|n_l(x)| \gg |j_l(x)|$ for l significantly greater than x. We now explain why the downward recursion relation is stable. Writing the estimated value of $j_l(x)$ as $j_l^{\rm est}(x)$, then the initial choices $j_l^{\rm est}(x)$ and $j_{l+1}^{\rm est}(x)$ can be written as

$$\begin{aligned} j_{l+1}^{\rm est}(x) &= A j_{l+1}(x) + B n_{l+1}(x), \\ j_l^{\rm est}(x) &= A j_l(x) + B n_l(x), \end{aligned}$$

where A and B are constants. Since the initial values are just guesses, the two terms in this equation, $Aj_l(x)$ and $Bn_l(x)$, will, in general, be of comparable magnitude. However, for $l \gg x$, $|n_l(x)| \gg |j_l(x)|$ and so $B \ll A$. Hence, iterating down to smaller values of l where $|n_l(x)/j_l(x)|$ is not so large, the $Aj_l(x)$ term dominates. We then just need to determine the coefficient A which is done by matching to the known function $j_0(x)$.

Essentially the same argument shows that the upward recursion relation is unstable.