MTH 308: Numerical Analysis & Scientific Computing I
Department of Mathematics and Statistics
Indian Institute of Technology - Kanpur January 11, 2017

Problem Set 1

The following problem is due **Monday, January 30** at **5:00 PM**.
Steps you should follow :

1. While submitting, all of your programs should be arrange as follows: main directory should be called HW1-ID where ID stands for your roll number. For example, if your roll number is 12345, then the directory name will be HW1-12345; if your roll number is 123456, then the directory name will be HW1-123456.

2. The directory should contain five subdirectories named Problem1, Problem2, Problem3, Problem4 and Problem5. The codes corresponding to Problem J should go in the subdirectory ProblemJ.

3. Upon completion of the work, once you are ready for submission, go to the directory that contains the directory HW1-ID and compress it using following command at the xterm/terminal command prompt

   $ tar cvfz HW1-ID.tgz HW1-ID

   which will create a file called HW1-ID.tgz .

4. Attach this file in an email with subject HW1-ID and send it to akasha@iitk.ac.in by 5.00 pm on January 30. The supporting writeup should also be submitted by 5:00 pm in hard-copy in Akash Anand's mail-box in the Department of Mathematics and Statistics. Late submission will not get any credit.

Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Aim for concise solution and focus on the key idea of the problem.

I. We have seen that using computer arithmetic, we can evaluate polynomials exactly (up to the rounding error). Let the polynomial $p(x)$ of degree n be given by

$$p(x) = \sum_{k=0}^{n} a_k x^k. \tag{1}$$

Note that $p(x)$ can be rewritten as

$$p(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + xa_n)\cdots))). \tag{2}$$

Explain why using the expression in (2) is more efficient over the expression in (1)for evaluating the polynomial. Write a Matlab program for implementing the polynomial evaluation using (2) and use it to compute π with the help of

$$p_n(x) = \sum_{k=0}^{n} (-1)^k \frac{x^{2k+1}}{2k+1}, \qquad |x| \leq 1$$

the nth order Taylor series approximation to $\tan^{-1} x$. Complete the following table

n	$\lvert\pi - 6p_n(1/\sqrt{3})\rvert/\pi$	$\lvert\pi - 4p_n(1)\rvert/\pi$
8	–	–
16	–	–
32	–	–
64	–	–
128	–	–
256	–	–
512	–	–
1024	–	–
2048	–	–
4096	–	–

Comment on the result.

II. Consider the function

$$f(x) = (e^x - 1)/x.$$

Use L'Hospital's rule to show that

$$\lim_{x \to 0} f(x) = 1.$$

(a) Check this result empirically by writing a program to compute $f(x)$ for $x = 10^{-k}, k = 1, 2, \ldots, 15$. Do your results agree with theoretical expectations? Explain why.

(b) Perform the experiment in part (a) again, this time using the mathematically equivalent formulation

$$f(x) = (e^x - 1)/\log(e^x),$$

evaluated as indicated, with no simplification. Does this work better? If yes, explain why?

III. Show that $f_{j,k} = \sin(x_0 + (j-k)\pi/3)$ satisfies the recurrence relation

$$f_{j,k+1} = f_{j,k} - f_{j+1,k}. \tag{3}$$

We view this as a formula that computes the f values on level $k+1$ from the f values on level k. Let $f^a_{j,k}$ for $k \geq 0$ be the floating point numbers that come from implementing $f_{j,0} = \sin(x_0 + j\pi/3)$ and (3) (for $k > 0$) in double precision floating point. If $\left|f^a_{j,k} - f_{j,k}\right| \leq \epsilon$ for all j, show that $\left|f^a_{j,k+1} - f_{j,k+1}\right| \leq 2\epsilon$ for all j. Thus, if the level k values are very accurate, then the level $k+1$ values still are pretty good.

Write a program that computes $e_k = \left|f^a_{1,k} - f_{1,k}\right|$ for $1 \leq k \leq 60$ and $x_0 = 1$. Print the e_k and see whether they grow monotonically. Plot the e_k with respect to k and see that the numbers seem to go bad suddenly at around $k = 50$.

Repeat this computation in single precision and comment on the result. Note that, if you are using MATLAB, by default, it does computations in *double precision*.

IV. The Fibonacci numbers, f_k, are defined by $f_0 = 1, f_1 = 1$, and

$$f_{k+1} = f_k + f_{k-1} \tag{4}$$

for any integer $k > 1$. A small perturbation of them, the *pib numbers* ("p" instead of "f" to indicate a perturbation), p_k, are defined by $p_0 = 1, p_1 = 1$, and

$$p_{k+1} = c\, p_k + p_{k-1}$$

for any integer $k > 1$, where $c = 1 + \sqrt{3}/100$.

(a) Plot the f_n and p_n together on a log scale plot. On the plot, mark $1/\epsilon_{mach}$ for single and double precision arithmetic. This can be useful in answering the questions below.

(b) Rewrite (4) to express f_{k-1} in terms of f_k and f_{k+1}. Use the computed f_n and f_{n-1} to recompute f_k for $k = n-2, n-3, \ldots, 0$. Make a plot of the difference between the original $f_0 = 1$ and the recomputed $\hat{f}_0$ as a function of n. What n values result in no accuracy for the recomputed f_0? How do the results in single and double precision differ?

(c) Repeat (b) for the *pib numbers*. Comment on the striking difference in the way precision is lost in these two cases. Which is more typical?

V. The binomial coefficients, $a_{n,k}$, are defined by

$$a_{n,k} = \binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

To compute the $a_{n,k}$, for a given n, start with $a_{n,0} = 1$ and then use the recurrence relation

$$a_{n,k+1} = \frac{n-k}{k+1} a_{n,k}.$$

(a) For a range of n values, compute the $a_{n,k}$ this way, noting the largest $a_{n,k}$ and the accuracy with which $a_{n,n} = 1$ is computed. Do this in single and double precision. Why is roundoff not a problem here as it was in problem IV? Find n values for which $\hat{a}_{n,n} \approx 1$ in double precision but not in single precision. How is this possible, given that roundoff is not a problem?

(b) Use the algorithm of part (a) to compute

$$E(k) = \frac{1}{2^n} \sum_{k=0}^{n} k a_{n,k} = \frac{n}{2}. \tag{5}$$

Write a program without any safeguards against overflow or zero divide. Show (both in single and double precision) that the computed answer has high accuracy as long as the intermediate results are within the range of floating point numbers. As with (a), explain how the computer gets an accurate, small, answer when the intermediate numbers have such a wide range of values. Why is cancellation not a problem? Note the advantage of a wider range of values: we can compute E(k) for much larger n in double precision. Print $E(k)$ as computed by (5) and $M_n = \max_k a_{n,k}$. For large n, one should be `inf` and the other `NaN`. Why?

(c) For fairly large n, plot $a_{n,k}/M_n$ as a function of k for a range of k chosen to illuminate the interesting "bell shaped" behavior of the $a_{n,k}$ near $k = n/2$. Combine the curves for $n = 10, n = 20$, and $n = 50$ in a single plot. Choose the three k ranges so that the curves are close to each other. Choose different line styles for the three curves.