Write a program without any safeguards against overflow or zero divide (*this time only!*)[1]. 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 (4) and $M_n = \max_k a_{n,k}$. For large n, one should be `inf` and the other `NaN`. Why?

(c) To go beyond $n \approx 1000$, we need a new method. We must avoid computing $a_{n,k}$ and 2^n separately. One way to do this is to use the recurrence relation for the largest terms, $a_{n,n/2}$. This is easiest when $n = 2m$ is even. If we define

$$c_m = 2^{-2m} a_{2m,m}$$

then

$$c_{m+1} = \frac{2n+1}{2n+2} c_m \quad . \tag{5}$$

From this, and the known value of c_1, we can compute all the c_m. For even n, this gives us $2^{-n} a_{n,n/2}$. We can get the rest of the values of $2^{-n} a_{n,k}$ needed for (4) by using either $a_{n,k+1} = \frac{n-k}{k+1} a_{n,k}$ (if $k > n/2$) or $a_{n,k} = \frac{k+1}{n-k} a_{n,k+1}$ (if $k < n/2$). Show that this allows us to compute $E(k)$ for n=10^4, which would not be possible directly.

(d) (*Extra credit, and lots of it!*) Find a way to compute

$$S(k) = \sum_{k=0}^{n} (-1)^k \sin(k/n) a_{n,k}$$

with good relative accuracy for large n. This is very hard, so don't spend too much time on it.

[1] One of the purposes of the IEEE floating point standard was to *allow* a program with overflow or zero divide to run and print results

Assignment 3.

Given February 12, due February 26.

Objective: To explore computer arithmetic.

(1) The fibonacci numbers, f_k, are defined by $f_0 = 1$, $f_1 = 1$, and

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

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

$$f_{k+1} = c \cdot f_k + f_{k-1} \tag{2}$$

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

(a) For various n values, compute the f_k for $k = 2, 3, \ldots, n$ using (1). Then use the computed f_n and f_{n-1} to recompute f_k for $k = n-2, n-3, \ldots, 0$. Compare the recomputed f_0 with its true value, 1, and see how the error depends on n. It will help if you notice the size of the largest f_k. What n values result in no accuracy for the recomputed f_0? How do the results in single and double precision differ?

(b) Repeat (a) for the pib numbers. Comment on the striking difference in the way precision is lost in these two cases. Which is more typical? *Extra credit*: predict the order of magnitude of the error in recomputing p_0 using what you may know about recurrence relations and what you should know about computer arithmetic.

(2) The binomial coefficients, $a_{n,k}$, are defined by

$$a_{n,k} = \left(\begin{array}{c} n \\ k \end{array} \right) = \frac{n!}{k!(n-k)!} \tag{3}$$

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 fixed of n, 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 it that roundoff is not a problem here as it was in problem (1)?

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

$$E(k) = \frac{1}{2^n} \sum_{k=0}^{n} k a_{n,k} = \frac{n}{2} \quad . \tag{4}$$