Homework #1, due September 13

Objectives: (1) To warm up to programming with good habits, (2) To illustrate basic numerical ideas.

(1) Explore truncation error and rounding error, and resolution and accuracy checking using the following integral equation:

$$f(x) = \int_0^1 \cos\left(xt^2\right)dt \quad . \tag{1}$$

This equation is related to the Fresnel integral (Abramowitz and Stegun, Section 7.3):

$$C(z) = \int_0^z \cos\left(\frac{\pi}{2}t^2\right)dt \quad . \tag{1}$$

The equation $f(x)$ can be expressed in terms of the Fresnel integral:

$$f(x) = \sqrt{\frac{\pi}{2x}}\,C\left(\sqrt{\frac{2x}{\pi}}\right).$$

- **(a)** Write a program to estimate the integral using a panel integration method (such as trapezoid rule and Simpson's rule), with the number of panels $n = 1/\Delta t$ as input. The program should be modular, well documented, This routine should be written so that another person could easily substitute a different quadrature by changing a few lines of code. (You can use standard subroutines such as these in NAG, Numerical Recipes, Netlib: http://www.netlib.org, MATLAB: type help funfun.) Turn in a listing of this code.
- **(b)** Verify the correctness of the program by checking that it gives the right answer for small x. We can estimate the integral for small x using a few terms of its Taylor series. This series can be computed by integrating the Taylor series for $\cos(xt^2)$ term by term. Turn in a table showing your Taylor series approximation and the value returned by your code for a few small values of x, say, $x = 0.1, 0.2, 0.3$.
- **(c)** With $x = 1$, do a convergence study to verify the second order accuracy of the trapezoid rule (trapzd.f routine of Numerical Recipes, Section 4.2) and the fourth order accuracy of Simpson's rule (qsimp.f routine of Numerical Recipes, Section 4.2). Turn in a table showing the results for different step sizes Δt, or a log-log plot of the error, and explain how your results demonstrate the correct order of accuracy. Try different ways if checking accuracy such as comparing with "correct" answer computed numerically by a very fine grid; using relation of $(A(4h) - A(2h))/(A(2h) - A(h))$.
- **(d)** As in **(c)**, what happens when n is very large for a given precision (single or double)? Find the value n_0 such that the accuracy check in **(c)** falls when $n \geq n_0$. What is the connection with rounding error? You will also find that accuracy check in **(c)** falls when n is very small. This is related to the resolution and will be explained in **(e)**.
- **(e)** For large values of x, the integrand will undergo many oscillations within the limits of the integration. Therefore, in order to achieve accuracy, the value of h must decrease in order to resolve the features of the integrand. Perform a convergence study for large x, say $x = 100, 1000$, and find out how many grid points per-wave structure in the integrand is needed in order to obtain a clean accuracy check. You shall use relative error in the convergence study.

(f) Write a routine that uses the the trapezoidal rule and Richardson extrapolation to compute the integral to within a specified (absolute) error tolerance. The desired error bound should be input. The output should be the estimated value of the integral and the number of function evaluations used. You can use or modify the routine qtrap.f from Numerical Recipes, Section 4.2. This routine should be robust enough to quit and report failure if it is unable to achieve the requested accuracy. Turn in a few examples with your input and output from the routine. Try to find an example where your code reports failure.

(g) For large x, we have the following asymptotic approximation for the integration (See Abramowitz and Stegun, Eqn 7.3.9)

$$f(x) \sim \sqrt{\frac{\pi}{8x}} + \frac{1}{2x}\sin(x) - \frac{1}{4x^2}\cos(x) + \cdots \ . \tag{2}$$

Make a few plots showing f and its approximations using one, two and all three terms on the right side of (2) for x in the range $1 \le x \le 1000$. In all cases we want to evaluate f so accurately that the error in our f value is much less than the error of the approximation (2). Note that even for a fixed level of accuracy, more points are needed for large x as explained in **(e)**.

(2) Explore an ill conditioned matrix.

For this assignment you will need software that estimates the condition number of a matrix. This can be MATLAB, or a standard scientific subroutine package (e.g. NAG) or software from netlib. If you don't have access to such software, you will have to skip part **(a)**. If you don't find a routine that estimates condition number but you can find one that computes the singular values or the singular value decomposition, the condition number is $\sigma_{\max}/\sigma_{\max}$. You can use svdcmp.f in Numerical Recipes, Section 2.9 or svd in MATLAB.

For a fixed n, define A to be the $n \times n$ matrix with all entries zero except the $a_{ii} = 1$ and $a_{i,i+1} = 2$ for all i. That is, A has 1 on the diagonal and 2 immediately above the diagonal and all the rest zero. The determinant of this matrix is 1 and its norm is about 3 (depending on exactly which norm is used).

(a) Find the (estimated) condition number of A as a function of n for $n = 2, 3, \ldots$ until the condition number becomes (in the computer) infinite.

(b) Let $x \in R^n$ be the vector with n components whose components are given by $x_j = 1$ if j is even and $x_j = 1/3$ if j is odd. The value 1/3 is not stored exactly in the computer in floating point. Compute $b = Ax$ by simple matrix multiplication. Now compute y by solving the equations $Ay = b$ where b has just been computed. Write a program to do this in MATLAB, FORTRAN, C, or C++ using back substitution on an upper triangular matrix that is bidiagonal (see Kahaner, Moler, and Nash). Compute $\|x - y\|$ for some norm and plot this as a function of n so long as possible. Compute $|Ay - b|$ for the y you have and check its dependence on n. Comment on the contrast between this and the dependence of $\|x - y\|$ on n.

(c) Is the error $\|x - y\|$ of the order predicted by the size of roundoff error (in single or double precision) and the condition number of A? Turn in a table or semi-log plot showing the error $y - x$, the residue $Ay - b$, and $|y - x|/\text{cond}$.

(3) Explore computer arithmetic and the problem of $0 \cdot \infty$.

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.

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

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

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? Turn in a table showing $E(n)$, the relative error $(E(n) - n/2)/(n/2)$, and $a_{n,n/2}$. 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

$$c_m = 2^{-2m} a_{2m,m} \tag{5}$$

then

$$c_{m+1} = \frac{2m+1}{2m+2} c_m \quad . \tag{6}$$

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 $E(n)$ for n=10^4, which would not be possible directly.

[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