Numerical Methods I, Fall 2010
Assignment I: Numerical Computing

Aleksandar Donev
Courant Institute, NYU, *donev@courant.nyu.edu*

September 9th, 2010
Due: September 23rd, 2010

Choose the problems that interest you, including any of the extra credit ones. Anything above 70 points is excellent. "Extra credit" simply marks problems that are more free-style and do not come with very specific directions.

1 [Up to 20 points] Floating-Point Operations

Choose a "processor" to work with, meaning some programming tool and some machine to run on. For example, you may choose MATLAB on your laptop, or you may choose the GNU C compiler on a Courant server (report what you chose). If you know how to use a compiled language, it is recommended (extra credit!) that you try both MATLAB and the compiled language.

1.1 [10 pts] Non-normalized numbers

Using single and/or double precision, do some simple calculations that lead to exceptions and report what your processor gives you. Where you can, verify that the bit string (use hex format in MATLAB) for the numbers you get corresponds to what the IEEE standard specifies. Try the following:

- Overflow other than division by zero (e.g., two to a large power). Verify that the result would have been larger than the largest IEEE number if it had not overflowed (for example, by pen-and-paper, repeating the same calculation with higher precision or in Maple/Mathematica).
- Divide a normal number by zero and infinity.
- Generate some NaNs and then perform some arithmetic operations between a normal number and a NaN.
- Perform a few comparisons (e.g., $x < y$, $x > y$, $x == y$) between infinities, $NaNs$, and normal numbers to determine if and how these are ordered.
- Generate some denormalized numbers if your processor supports that. Generate a signed $+0$ and -0 and take their square roots.

1.2 [10pts extra credit] IEEE exceptions

Can you control the behavior that occurs? For example:

- Can you trap/detect exceptions when they occur or at the end of a computation?
- Can you enable/disable denormals and gradual underflow? Does that affect the speed of calculations involving denormals?
- Can you go beyond double precision, e.g., quad precision?

2 [20 points] Stability and Error Propagation

[From Dahlquist & Bjorck, also Exercise 9 (second ed) / 10 (first ed) in second chapter of textbook]
Review from the lecture slides:
Consider error propagation in evaluating

$$y_n = \int_0^1 \frac{x^n}{x+5} dx$$

based on the identity (you may want to derive this yourself)

$$y_n + 5y_{n-1} = n^{-1}.$$

Since $y_n < y_{n-1}$, we have that

$$6y_n < y_n + 5y_{n-1} = n^{-1} < 6y_{n-1},$$

$$0 < y_n < \frac{1}{6n} < y_{n-1},$$

so for large n we have tight bounds

$$\frac{1}{6(n+1)} < y_n < \frac{1}{6n} \tag{1}$$

2.1 [10pts] Forward iteration

Calculate y_n for $n = 1, 2, \cdots$ using the forward iteration $y_n = n^{-1} - 5y_{n-1}$, starting from $y_0 = \ln(1.2)$, using both single and double precision. Plot the results together with the bounds (1) [choose your axes wisely, e.g., plot ny_n instead of y_n]. Report the result for the largest $n = n_{max}$ where you actually trust the answer to 4 significant digits, and explain your choices and observations the best you can.

Hints: Maple tells us that $y_{14} = 0.01122918662647553$, $y_{15} = 0.01052073353428904$ and $y_{16} = 0.00989633232855484$.

2.2 [10pts] Backward iteration

Now start with $n = 2n_{max}$ and repeat the calculation going backward, $y_{n-1} = (5n)^{-1} - y_n/5$, starting with both the lower and upper bound for y_n in (1), at least for double precision. Plot the results on top of the plot from the forward iteration. How many digits do you trust in the answer for $y_{n_{max}}$ now?

3 [Up to 85 points] Numerical Cancellation

Choose some of these problems, trying to earn 40 points.

3.1 [20 points] Numerical Differentiation

The derivative of a function $f(x)$ at a point x_0 can be calculated using finite differences, for example the first-order *one-sided difference*

$$f'(x = x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

or the second-order *centered difference*

$$f'(x = x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h},$$

where h is sufficiently small so that the approximation is good. Consider a simple function such as $f(x) = \sin(x)$ and $x_0 = \pi/4$ and calculate the above finite differences for several h on a logarithmic scale (say $h = 2^{-m}$ for $m = 1, 2, \cdots$) and compare to the known derivative. For what h can you get the most accurate answer? Obtain an estimate of the *truncation error* in the one-sided and the centered difference formulas by performing a Taylor series expansion of $f(x_0 + h)$ around x_0. Also estimate what the *roundoff error* is due to cancellation of digits in the differencing. At some h, the combined error should be smallest (optimal, which usually happens when the errors are approximately equal in magnitude). Estimate this h and compare to the numerical observations.

[10pts extra credit] Higher order derivatives can also be calculated in this way, for example, the second-order derivative centered approximation

$$f''(x = x_0) \approx \frac{f(x_0 + h) - 2f(x_0) + f(x_0 - h)}{h^2}$$

is commonly used. Explore the best possible accuracy for this formula with double precision.

3.2 [20 points] Computing π

There are many methods to compute many digits of π, and lots of them suffer from numerical accuracy problems. Here is one of them due to Archimedes: Start with $t_0 = 1/\sqrt{3}$ and then iterate

$$t_{i+1} = \frac{\sqrt{1 + t_i^2} - 1}{t_i} \tag{2}$$

and for large i you can get a good approximation $\pi \approx 6 \cdot 2^i \cdot t_i$.

Do this calculation with both single and double precision, and report how many digits of accuracy you get and after how many iterations (Note: $\pi = 3.14159265358979323846264338 3\cdots$ and MATLAB has a built-in constant *pi*), accompanied with some plots of the convergence. Are there numerical problems and can you explain why they occur when they occur? Find a way to rewrite the iteration (2) so that you avoid cancellation errors in the numerator and get much better accuracy and repeat the calculation.

3.3 [20 points] Beneficial Cancellation: Computing $\ln(1 + x)$ for small x

[Due to William Kahan / David Goldberg]

Introduction: When calculating $\ln(1 + x)$ for $0 < x \ll 1$ that is close to machine precision, the argument $1 + x$ has a large roundoff error and the relative error in the result is large. The MATLAB function *log*1*p* is designed so as to avoid this problem and give an accurate result.

You may choose to work with $\frac{\ln(1+x)}{x}$ instead to make the behavior near zero easier to study. Also, Exercise 13 (second ed) / 14 (first ed) in the second chapter of textbook contains a similar example but for $\exp(x) - 1$. If you wish you can do that instead of $\ln(1+x)$. The corresponding MATLAB function is *expm*1.

Explore what *log*1*p* does by doing these steps:

1. [5pts] Calculate $log(1 + x)$ for logarithmically-spaced (small) values of x using MATLAB, and comment the relative error compared to the built-in function *log*1*p* (a picture is worth a thousand words!).
2. [10pts] Do you trust the built-in function? Give some justification why, for example, by comparing to the Taylor series of $\ln(1 + x)$ for small x. If you did not have the built-in function, an alternative would have been to use the naive direct calculation of $log(1 + x)$ for $x > x_0$ and the Taylor series for $x \le x_0$. Try to find an x_0 that is optimal, that is, one for which the worst relative accuracy over the interval $0 < x < 1$ is smallest.
3. [5pts] A wise person that knows a lot about IEEE arithmetic has shown that using the following alternative calculation

$$\ln(1 + x) = \begin{cases} x & \text{if } x < \epsilon \\ \frac{x \ln(1+x)}{(1+x)-1} & \text{otherwise} \end{cases}$$

   gives a relative error of several ulps (i.e., within machine precision) for all $0 \le x < 3/4$ if the log is computed to within half an ulp [see paper by David Goldberg on IEEE].
   Try this and see how it compares to the built-in *log*1*p*.
   [Extra credit:] Can you explain why this magic works (no proof necessary, just look at an example)?

3.4 [Up to 25 points extra credit] The Fibonacci sequence

[Due to Jonathan Goodman]

The Fibonacci numbers are defined by the recurrence relation

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

with $f_0 = 1$ and $f_1 = 1$. Consider also the apparently-related sequence

$$p_{k+1} = \left(1 + \frac{\sqrt{3}}{100}\right) p_k + p_{k-1}, \tag{4}$$

with $p_0 = 1$ and $p_1 = 1$.

1. [5 pts] For both single and double precision arithmetic, plot f_k and p_k on a log-scale plot and comment on how large k can be before there are numerical problems in computing f_k.
2. [10 pts] For a set of logarithmically-spaced k's, run the calculation backward for the Fibonacci sequence, $f_{k-1} = f_{k+1} - f_k$, starting from the computed f_k and f_{k-1} and going backward to f_0. Compare the result to the correct value $f_0 = 1$ and comment on why there is an error for larger k, for both single and double precision.
3. [10 pts] Now also run the modified sequence (4) backward and compare to the correct $p_0 = 1$ and comment on the difference with (3) [Note: Problem 2 above is related]. Can you provide an estimate for the order of magnitude of the error in p_0 based on how the error propagates?