CS 514/ Math 514 Numerical Analysis
Spring 2018
Homework 2
Given: Feb. 1, 2018; Due: Feb. 15, 2017 11:45 A.M. (15 minutes before class)

You are permitted to discuss these problems with **one** other student, and you should indicate the name of your collaborator in your answer to the first problem in this set. If you worked on your own, please indicate this on your answer. You should write up your own solution, and you are not permitted to either share a written copy of your solution or copy some one else's written solution. Similarly with programs, you are not permitted to share your code with another student. If you need help with debugging your code, you should describe what the problem is without sharing or showing your code to another student. If you are stuck on a problem or on a code, you could ask the TAs for help on Piazza. Please do not discuss solutions to the HW problems on Piazza before they are graded.

If you copy someone else's work, or let your work be copied, you will get zero points for the entire HW, a loss of one letter grade in the course, and you will be reported to the Dean of Students. I follow the course policies described by Professor Gene Spafford (of Purdue Computer Science department) at the URL `http://spaf.cerias.purdue.edu/~spaf/cpolicy.html` If you are not familiar with this policy, please read it! Please submit your assignments using Blackboard, preferably using either a Tex-generated or Word document.

0. If you collaborated with a student in answering these questions, please write down your collaborator's name. If you did not collaborate with anyone, please indicate that too. This answer is worth one point.

1. **10 points**
Consider the problem of evaluating the integrals in Example 1.6 of the textbook. There the recurrence $y_n = 1/n - 10y_{n-1}$, for $n = 1, 2, \ldots$, is used to compute the integrals. By substituting $y_{n-1} = 1/(n-1) - 10y_{n-2}$, we can write y_n in terms of y_{n-2}. By continuing this process, we can write y_n in terms of y_0, and other terms that do not involve any of the y_i. Obtain this expression, and use it to explain why computing the recurrence in the forward direction (i.e., computing y_n from y_{n-1}) is numerically unstable.

2. **15 points**
(See Problem 2 in Chapter 1, and Problem 14 in Chapter 3.)
We compute a finite difference approximation to the derivative of $\sin(1.2)$, using the centered difference $(\sin(1.2+h) - \sin(1.2-h))/(2h)$. This is similar to the approach shown in Example 1.3, except that the centered difference is used here.

   (a) Show that the truncation error in the centered difference approximation to the derivative of a function $f(x)$ is given in absolute value by $h^2|f'''(\xi)|/6$, where ξ is some point in the interval $[x-h, x+h]$. (Expand $f(x+h)$ and $f(x-h)$ about the point x in Taylor series.)

(b) Assume that the rounding error in computing the centered difference is dominated by the rounding error in computing the function $f(h)$, and that this error is at most ϵ/h, a constant value. (This is a simplifying assumption, we will check how valid it is in the rest of the problem.) Thus the sum of the truncation error and rounding error is at most $h^2|f'''(\xi)|/6+\epsilon/h$. Show that this error is at most $h^2/6+\epsilon/h$ for $f = \cos(x)$. Minimize this error with respect to h, and show that the minimizer is $h = (3\epsilon)^{1/3}$. Choose values of $\epsilon = 10^{-16}$, 10^{-15} and 10^{-14}, and compute the corresponding values of h from this equation. (We assume that the computations are being done in double precision, so we choose small multiples of the rounding error for ϵ.)

(c) Now compute the absolute value of the difference between $\cos(1.2)$ and its approximate value obtained using the centered difference formula, using values of h from 10^{-1}, 10^{-2}, $\cdots$, 10^{-16}. Plot the error in a loglog plot as done in Figure 1.3. Is the minimizer of the error in the plot approximately equal to any of the values of h that you computed in the previous part?

3. **15 points**
In this problem we investigate how to rewrite expressions involving subtraction of nearly equal quantities to avoid cancellation error as far as possible. In each case, compute the values obtained by the original and rewritten expressions using double precision arithmetic.

(a) Rewrite $1 - \cos x$, near $x = 0$.

(b) Rewrite $\cos x - \sin x$, near $x = \pi/4$.

(c) Explain why $a^2 - b^2$ can be more accurately computed from the expression $(a + b) * (a - b)$.

4. **20 points**
Compute the absolute and relative condition numbers of the following functions $f(x)$, and indicate for what values of x the function becomes ill-conditioned.

(a) x^n

(b) a^x, here a is a constant.

(c) $\cot x$

(d) $\arcsin x$

5. **15 points**
We develop a fixed point iteration to compute the value of the square root of a number y. Consider finding a root of the function $f(x) = x^2 - y$, and the corresponding fixed point methods, (1) $g_1(x) = y - x^2 + x$; (2) $g_2(x) = 1 + x - x^2/y$; and (3) the function $g(x)$ that corresponds to Newton's method. For each method, find the largest interval $[-a, a]$ for which the fixed point iteration converges.

Use the code for Newton's method to compute the square root of 3, using 1.0 as the initial guess.

6. **20 points**
Implement the Secant method to compute a root of a nonlinear function $f(x)$ as a Matlab or Python function (similar to the function for Newton's method discussed in the lecture whose Matlab code has been provided for you in `newton2.m` on Blackboard). Note that you will need to provide two initial guesses for the Secant method to work. Use the termination criterion as in Newton's method by requiring the absolute value of the function to be reduced by a factor of 10^{-12} relative to the function value at the initial guess x_0, and at most 20 iterations are performed.

   (a) Submit your code; it will be evaluated for correctness and documentation.

   (b) Run your code for the Secant method on the function $f(x) = \sin(10x) - x$, using the initial guesses $x = 1$ and 0.9. Report the root that you find, and the number of iterations taken by your code.

   (c) How many roots does this function have in the interval $[0.1\ 1]$? Find them all using your program with different initial guesses. Report the roots found and the number of iterations taken.