MATH 315, Fall 2020
Homework 2

James Nagy

Homework Team: Z

This assignment is associated with Chapter 2, computing with floating point numbers. It illustrates that while mathematical formulas might be equivalent, how they behave when implemented on a computer might be quite different.

1. This first problem considers two different approaches to compute the roots of a quadratic polynomial, $p(x) = ax^2 + bx + c$. We know that the values r such that $p(r) = 0$, are given by the quadratic formula:

$$r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

   (a) A naive MATLAB implementation to compute the roots might look like:

```
function r = QuadFormula1(coeffs)
%
%  Given coefficients of a quadratic polynomial, p(x), this function
%  computes the roots of p(x) = 0 using the quadratic formula.
%
%  Input: coeffs - vector containing the coefficents of p(x),
%                  [a, b, c], where p(x) = a*x^2 + b*x + c.
%
%  Output:     r - vector containing the two roots of p(x) = 0.
%
%
r = zeros(2,1);
a = coeffs(1);
b = coeffs(2);
c = coeffs(3);
d = sqrt(b^2 - 4*a*c);
a2 = 2*a;
r(1) = (-b + d)/a2;
r(2) = (-b - d)/a2;
```

On your computer, create the MATLAB function m-file `QuadFormula1.m`, using the above code. You should type in the comment lines as well.

(b) Test your code using two simple problems:

$$\begin{aligned} p(x) &= x^2 - 3x + 2 \\ p(x) &= 10^{160}x^2 - 3 * 10^{160}x + 2 * 10^{160} \end{aligned}$$

In each case, you should be able to compute the roots by hand, and compare with what is computed by your MATLAB code. Does the code compute good approximations of the true roots for these two polynomials?

(c) Briefly explain why you did not get good approximations for the roots in one of the polynomials, and explain how you can fix it.

(d) Copy the code in `QuadFormula1.m` into a new function called `QuadFormula2.m`, and modify the code so that it implements your fix. Use `QuadFormula2.m` to compute the roots from the above examples and show that your new code obtains accurate approximations for both of these polynomials.

2. (a) Consider the sequence of numbers:

$$x_{k+2} = 2.25x_{k+1} - 0.5x_k\,, \quad k = 1, 2, \ldots \tag{1}$$

with starting values of $x_1 = 1/3$ and $x_2 = 1/12$.

It can be shown that this sequence of numbers can also be written as:

$$x_k = \frac{4^{1-k}}{3}\,, \quad k = 1, 2, \ldots \tag{2}$$

Looking at (2), what can you say about the terms x_k as k increases?

(b) Write a MATLAB code that will generate the two sequences of numbers given in (1) and (2). Your code should be written as a function m-file, with input n (number of points to generate) and output two vectors containing values $x_1, x_2, \ldots, x_n$ for each of (1) and (2).

More specifically, the beginning of your code should contain the following:

```
function [x1, x2] = GenSequence(n)
%
%  This function computes the entries of two sequences:
%     x(k+2) = 2.25*x(k+1) - 0.5*x(k), x(1) = 1/3, x(2) = 1/12
%  and
%     x(k) = (4^(1-k))/3
%
%  Mathematically, these sequences should produce the exact same values.
%
%  Input:  n  - integer, number of values of the sequence
%
%  Output: x1 - computed values of the first sequence of values
%          x2 - computed values of the second sequence of values
%
```

(c) Run the code using $n = 60$, and plot the resulting x_k values using MATLAB's `semilogy` function. Plot both sets of points on the same axes, using different symbols and colors (e.g., blue circles for (1) and red diamonds for (2)).

(d) Do the points generated from your code behave as you expect, considering what you found in part (a)?

(e) We should try to understand what why some of the computed values from these sequences are so different. First you should explain why it is not "catastrophic cancelation". Hint: You could look at some of the values:

$$2.25x_{k+1} \quad \text{and} \quad 0.5x_k$$

and see if they are nearly equal. For example, look at these values when $k = 19$. Are these numbers nearly equal?

(f) Now consider an analysis similar to problems 2.3.3 and 2.3.4 in Chapter 2 of the book. That is, assume that x_1 and x_2 are computed exactly, but we get roundoff error when we compute x_3. That is, suppose the computed x_3 is denoted by $\hat{x}_3$, and

$$\hat{x}_3 = x_3 + \delta\,,$$

where δ is small. Then, notice how this small initial error accumulates when computing the recursion.

Your goal here should be to get a formula of the form:

$$\hat{x}_{k+2} = x_{k+2} + \_\_\_\_\_\delta$$

where you have to find an expression that fills in the above blank.

(g) You can visualize this as follows: Assume that the computed values of sequence (2) are the "true" x_{k+2}. Add to these values the estimated error, $\_\_\_\_\_\delta$.

Now plot something similar to part (c), but instead of plotting sequences (1) and (2), plot sequence (2) and sequence (2) + error.

This plot should look very similar to the one you obtained in part (c), but you are using a very rough approximation of the error, so you should not expect them to be precisely the same.