MATH 315, Fall 2020
Homework 2

James Nagy

Homework Team 3: Judy Hao, Sirui Hu, Laura Neff, Diego Gonzalez

---

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?

Answer: By factorizing the first expression, we can easily find that its two roots are $r_1 = 1$ and $r_2 = 2$, which are the solutions that the script correctly provides. The roots for the second expression are the same, as it is equal to the first expression multiplied by 10^{160}. However, the code returns "NaN" as the solutions for the second equation, which is incorrect.

(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.

Answer: The reason why `QuadFormula1.m` is not behaving as expected is because of an overflow, which means that we have a number that is too large that cannot be represented because of the limited number of bits used to represent numbers in MATLAB. MATLAB determines that big numbers that don't have enough bits to represent them must be infinity, or `Inf`, which greatly reduces the amount of values we can calculate on the practical scale. In our problem, we have b^2 inside of the discriminant of the Quadratic Formula, which overflows with $b = 3 * 10^{160}$. This causes the discriminant to be equal to `Inf`, which consequently leads to incorrect solutions. One way to solve this problem is to use a scaling factor s. We can use an expression such as:

$$s\sqrt{\left(\frac{b}{s}\right)^2 - 4\left(\frac{a}{s}\right)\left(\frac{c}{s}\right)}$$

Which leaves the result of the discriminant unchanged. Lastly, we choose s to be equal to the largest number of the absolute values of the coefficients. By choosing that as our scaling factor s, we risk having underflows. That is, we may have an operation with a result much smaller than can be represented in the computer (again, because of the limited number of bits used to store each number), making the result of such calculation equal to zero. However, this is a more desirable calculation error to have inside of the discriminant and is much safer than overflow.

(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:

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

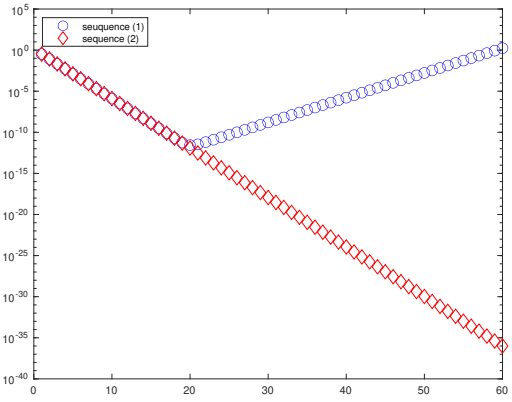
Looking at (2), what can you say about the terms x_k as k increases?

Answer: Looking at the exponint in (2), we can tell that $x_{k+1} < x_k$ for all k. In other words, the values that are part of this sequence are always decreasing, and they approach 0 as k approaches infinity.

(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)?

Answer: No. From part (a), we expect the terms in both sequences to always decrease, and to approach 0 as k increases. However, while the points for sequence (2) behave as expected, the terms for sequence (1) begin to increase when x is greater than 20.

(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 cancellation". 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?

Answer: Using $k = 19$ for sequence (2), we can obtain

$$2.25x_{k+1} = 2.25x_{20} = 2.728484105318785 * 10^{-12}$$

$$0.5x_k = 0.5x_{19} = 2.425319204727808 * 10^{-12}$$

Therefore,

$$2.25x_{k+1} - 0.5x_k = 3.031649005909770 * 10^{-13}$$

We can calculate the relative difference with the formula

$$\text{relative difference} = \left| \frac{2.25x_{k+1} - 0.5x_k}{2.25x_{k+1}} \right| \approx 0.1111 = 11.11\%$$

Because the relative difference is not a small number, subtracting $0.5x_k$ from $2.25x_{k+1}$ will not result in a number that is small enough to be represented as 0 in MATLAB. You can tell by comparing our relative difference to

$$\frac{1}{2}\epsilon = 2.220446000000000 * 10^{-16}$$

Since we are using double precision in our script, $\epsilon = 2.220446010^{16}$. Comparing this number with the relative difference that we got, we can conclude that catastrophic cancellation is not the issue here, given that the difference between the two numbers is large enough to avoid such problem.

(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.

Answer: The sequence given by (1) is of the form:

$$x_{k+2} = ax_{k+1} + bx_k$$

Where $a = 2.25$ and $b = -0.5$. From the assumptions that we are making, we have that:

$$\hat{x}_1 = x_1$$

$$\hat{x}_2 = x_2$$

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

If we keep computing a few more elements in the sequence, we get:

$$\hat{x}_4 = a(x_3 + \delta\,) + bx_2 = x_4 + a\delta$$

$$\hat{x}_5 = a(x_4 + a\delta\,) + b(x_3 + \delta\,) = x_5 + (a^2 + b)\delta$$

$$\hat{x}_6 = a(x_5 + (a^2 + b)\delta\,) + b(x_4 + a\delta\,) = x_6 + (a^3 + 2ab)\delta$$

$$\hat{x}_7 = a(x_6 + (a^3 + 2ab)\delta\,) + b(x_5 + (a^2 + b)\delta\,) = x_7 + (a^4 + 3a^2b + b^2)\delta$$

As we can appreciate, the expression multiplying δ doesn't show a pattern that makes it easy to represent using a polynomial. However, we can use a recurrence relation. By closely looking at the expressions multiplying δ, we find (perhaps unsurprisingly) that they follow the same recurrence relation that is used to define

(1). Thus, if we use the correspondent values for a and b, then we are left with the expressions:

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

Where

$$c_{k+2} = 2.25c_{k+1} - 0.5c_k$$

With $c_1 = 0, c_2 = 0$ and $c_3 = 1$, as our original assumption stated.

(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.

Answer: The creation of this plot can be seen in the script `TestingGenSequence.m`. To be able to calculate actual numeric values, we had to choose a value for δ. Since the assumption was that δ was the result of a round-off error, we decided that a good guess would be the machine precision $\epsilon = 2.2204 * 10^{-16}$ obtained from MATLAB.