Programming Assignment 01

The Effects of Accumulated Roundoff in Creating a Sequence of Numbers

Nick Handelman
1/24/2018

Project Description

This report discusses the effects of accumulated roundoff in creating a sequence of numbers. The terms of array x are calculated without relying on previous calculations. The sequences: p, q, r and s are calculated sequentially, using previously calculated terms of the sequence. The provided Octave code was run to create the array and sequences and plot the results. The first 30 terms of x and each sequence were calculated. The term by term error relative to x is calculated for p, q, r and s.

$$x(k) = \left(\frac{2}{3}\right)^{(k-1)} \tag{array x}$$

$$p(k+1) = \frac{2*p(k)}{3}, p(1) = 1 \tag{sequence p}$$

$$q(k+2) = \frac{7*q(k+1)}{6} - \frac{q(k)}{3}, q(1) = 1, q(2) = \frac{2}{3} \tag{sequence q}$$

$$r(k+2) = 2*r(k+1) - \frac{8*r(k)}{9}, r(1) = 1, r(2) = \frac{2}{3} \tag{sequence r}$$

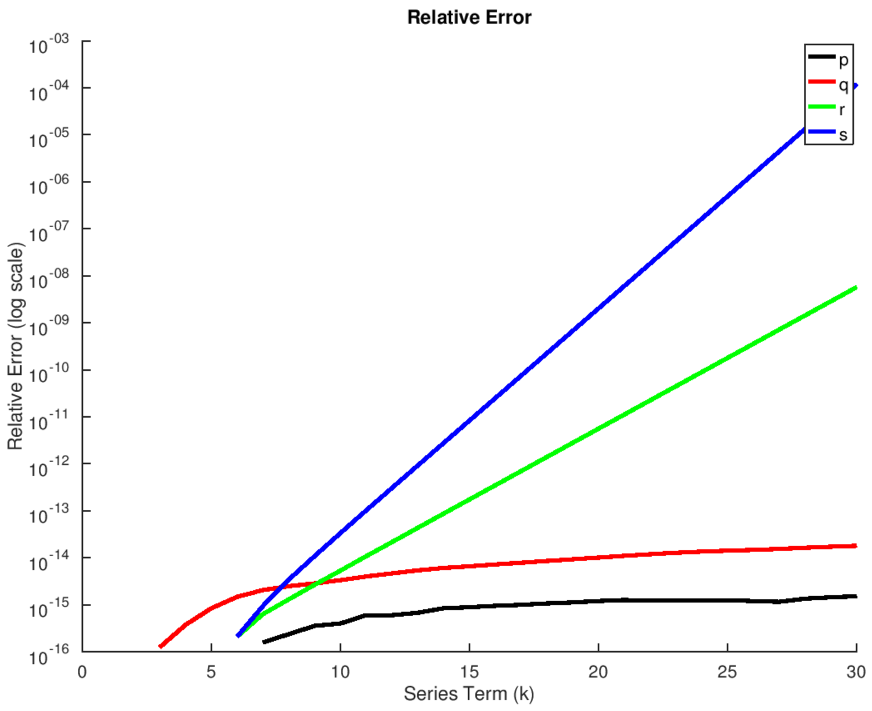
$$s(k+2) = \frac{8*s(k+1)}{3} - \frac{4*s(k)}{3}, s(1) = 1, s(2) = \frac{2}{3} \tag{sequence s}$$

Analysis

Appendix A has Sage code that, if run, confirms that the four sequences output the same numbers in exact arithmetic. I was not sure how, or if, this could be done in Octave.
In Octave, each term in each sequence is calculated. The error of the i$^{\text{th}}$ term in each of the sequences is compared relatively to the corresponding i$^{\text{th}}$ term in x as shown in equation 1.

$$relative\_error = \frac{|x_i - z_i|}{x_i} \tag{1}$$

The following graph displays the results of the accumulated roundoff that occurs due to the sequential calculations in each of the sequences.



Graph 1 - Relative Error for Each Sequence

For sequences q, r and s, once the relative error becomes nonzero, the graph clearly indicates that the error continues to grow but at various rates. The errors for r and s increase linearly with time, but s increases at a greater rate. q grows very slowly, possibly logarithmically, and nearly appears to be leveling off. Graphically, p appears to have leveled off, though the numbers indicate the error is continuing to grow. Interestingly, the relative error in p decreases for the 26th and 27th terms.

Conclusions

Sequence s has the greatest accumulation of roundoff error, probably due to repeated multiplications of constants $\frac{8}{3}$ and $\frac{1}{3}$. r has the second greatest accumulation of roundoff error. I am surprised at this since it only performs repeated fractional multiplication for one constant term: $\frac{8}{9}$. I am also surprised that q manages to maintain a really low error since it contains two constant fractional terms: $\frac{7}{6}$ and $\frac{1}{3}$. q is the first sequence to reach a nonzero error though.

Sequence p has the lowest accumulated round off error, for two reasons that I can see. First, it doesn't perform any subtraction. Second, it doesn't perform any repeated multiplication of a constant fraction. Instead, it performs integer multiplication then division. I would not have assumed that subtraction would have a major effect on round off error. In fact, I'm still not convinced since sequence q performs subtraction but maintains a low error. I think the integer multiplication has the greatest effect on keeping the error low, particularly since the integer is 2. Multiplication by 2 has the third lowest chance (behind 1 and 0) of increasing the number of digits that need to be represented. For example, $2 * 456,789.456 = 913,578.912$ but $3 * 456,789.456 = 1,370,368.368$. The number of digits needed to represent the new number has increased by 1.

Appendix A - Sage Code

```
plist=[]
qlist=[]
rlist=[]
slist=[]

plist.extend([1,2/3])
qlist.extend([1,2/3])
rlist.extend([1,2/3])
slist.extend([1,2/3])

for x in range(0,2):
        print("p" + str(x+1) + " : " + str(plist[x]))
        print("q" + str(x+1) + " : " + str(qlist[x])
        print("r" + str(x+1) + " : " + str(rlist[x]))
        print("s" + str(x+1) + " : " + str(slist[x]))
        print

for x in range(2,30):
        plist.append(plist[x-1]*(2/3))
        qlist.append((7/6)*qlist[x-1]-(1/3)*qlist[x-2])
        rlist.append((2)*rlist[x-1]-(8/9)*rlist[x-2])
        slist.append((8/3)*slist[x-1]-(4/3)*slist[x-2])
        print("p" + str(x+1) + " : " + str(plist[x]))
        print("q" + str(x+1) + " : " + str(qlist[x]))
        print("r" + str(x+1) + " : " + str(rlist[x]))
        print("s" + str(x+1) + " : " + str(slist[x]))
        print
```