HOMEWORK
David F. Gleich
October 31, 2018

Homework 8

Please answer the following questions in complete sentences in a clearly prepared manuscript and submit the solution by the due date on Blackboard (around Sunday, November 4th, 2018.)

Remember that this is a graduate class. There may be elements of the problem statements that require you to fill in appropriate assumptions. You are also responsible for determining what evidence to include. An answer alone is rarely sufficient, but neither is an overly verbose description required. Use your judgement to focus your discussion on the most interesting pieces. The answer to "should I include 'something' in my solution?" will almost always be: Yes, if you think it helps support your answer.

Problem 0: Homework checklist

- Please identify anyone, whether or not they are in the class, with whom you discussed your homework. This problem is worth 1 point, but on a multiplicative scale.
- Make sure you have included your source-code and prepared your solution according to the most recent Piazza note on homework submissions.

Problem 1: Accurate summation

Consider a list of n numbers. For simplicity, assume that all numbers are positive so you don't have to write a lot of absolute values.

1. Show that the following algorithm is backwards stable.

   ```
   function mysum(x::Vector{Float64})
     s = zero(Float64)
     for i=1:length(x)
       s += x[i]
     end
     return s
   end
   ```

   Which requires showing that mysum($\mathbf{x}$) $= \sum \hat{x}_i$ where $\|\hat{\mathbf{x}} - \mathbf{x}\|/\|\mathbf{x}\| \leq C_n \varepsilon$ where ε is the unit-roundoff for Float64.

2. Consider adding three positive numbers together a, b, c. Describe how to compute $s = a + b + c$ with the greatest accuracy.

3. Use the results of part 2 to describe a way to permute the input $\mathbf{x}$ to `mysum` to attain the greatest accuracy. Find an input vector $\mathbf{x}$ where this new ordering gives a measurable change in the floating point accuracy as determined by the number of correct digits in the mantissa. (Hint, this means you should know the true sum of your vector so that you can identify it's best floating point representation.)

4. Lookup the Kahan summation algorithm and implement it to sum a vector. Compare the accuracy with what you found in part 3.

Problem 2: Quadratic equations

Read through the stack exchange post on solving quadratic equations. https://math.stackexchange.com/questions/311382/solving-a-quadratic-equation-with-precision-when-using-floating-point-variables

This suggests a number of approaches to compute the roots of a quadratic equation through closed form solutions.

An alternative approach is to use an iterative algorithm to estimate that root of an equation. In this case, we can use a simple bisection approach, which works quite nicely for finding the root. Of course, there are floating point issues here too! Read about how to do bisection in floating point https://www.shapeoperator.com/2014/02/22/bisecting-floats/

Your task for this problem is to implement a bisection algorithm to return all the solutions of $ax^2 + bx + c = 0$ when $c \neq 0$.

```
""" Return all the solutions to ax^2 + bx + c. It is acceptable to return
NaN instead of a root as well. """
function roots(a::Float64,b::Float64,c::Float64)
end
```

Compare the accuracy of this procedure to the methods suggested on the stack exchange page and explain your results. Note that you may need to look for extremal inputs.

Problem 3: Inner-products are backwards stable.

1. Show that computing an inner-product $\mathbf{x}^T\mathbf{y}$ is backwards stable.
2. Show that computing a matrix-vector product $\mathbf{y} = \boldsymbol{A}\mathbf{x}$ is backwards stable.

Problem 4: The advantages of Float64

Consider the Candyland problem from HW2 where we worked out the expected length of a game based on an infinite summation. Repeat this analysis with Float16 arithmetic and also with BigFloat analysis. (This problem may require julia to use both Float16 and BigFloat.) Make sure that all intermediate computations use these types. To declare a vector of Float16 or BigFloats, use `Vector{Float16}` or `Matrix{BigFloat}` also helpful are `zero(Float16), one(Float16)`. If there are questions about using these types, please post to Piazza. Which answer is more accurate?