# Graded Homework 1 Foundations of Computational Math 1 Fall 2020

**The solutions are due by 11:59PM on September 28, 2020**

## Written Exercises

## Problem 1.1

Let $f(\xi_1,\ \xi_2,\ldots,\xi_k)$ be a function of $k$ real parameters $\xi_i,\ 1 \le i \le k$. Recall, the relative condition number of $f$ with respect to $\xi_1$ can be expressed

$$\kappa_{rel} = \max(1, c(\xi_1, \xi_2, \ldots, \xi_k))$$

where $0 \le c(\xi_1, \xi_2, \ldots, \xi_k)$ is a value that indicates the sensitivity of $f$ to small relative perturbations to $\xi_1$ as a function of the parameters $\xi_i,\ 1 \le i \le k$. If $c(\xi_1, \xi_2, \ldots, \xi_k) \le 1$ then $f$ is considered well-conditioned. Additionally, however, when $c < 1$ its value gives important information. The smaller $c$ is the less sensitive $f$ is to a relative perturbations in $\xi_1$.

Let $n \ge 2$ be an integer and $\beta > 0$. Consider the polynomial equation

$$p(x) = x^n + x^{n-1} - \beta = 0.$$

**1.1.a**. Show that the equation has exactly one positive root $\rho(\beta)$.

**1.1.b**. Derive a formula for $c(\beta, n)$ that indicates the sensitivity of $\rho(\beta)$ to small relative perurturbations to $\beta$.

**1.1.c**. Derive añ upper bound on $c(\beta, n)$.

**1.1.d**. Comment on the conditioning of $\rho(\beta)$ with respect to $\beta$.

## Problem 1.2

Each question below has a brief answer and justification.

**1.2.a** Explain the idea of a "hidden bit" in a floating point system with base $\beta = 2$ and the benefit achieved by using it.

**1.2.b** Consider a floating point representation system with base $\beta = 8$, precision $t = 12$ and exponent range determined by $L = -15$ and $U = 16$. How many bits are required to represent a floating point number in this system?

**1.2.c** A communications laboratory has a signal processor that computes with precision high enough to satisfy easily all of the requirements by applications of interest to the laboratory. Suppose for one of the applications the data must be collected using sensors that measure the appropriate signals in the environment and the resulting problem to be solved using the data has a condition number of $\kappa \approx 10^5$.

If at least 2 decimal digits of accuracy in the solution are required, how many decimal digits must the sensors accurately measure in the input signals assuming the problem is solved with an unconditionally very stable algorithm?

**1.2.d** Let $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^n$ be two vectors with

$$x = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_n \end{pmatrix}, \quad y = \begin{pmatrix} \eta_1 \\ \eta_2 \\ \vdots \\ \eta_n \end{pmatrix}.$$

$$|\xi_i| \geq 1 \quad |\eta_i| \geq 1$$

Consider the evaluation of the two inner products

$$\mu = x^T x$$

$$\gamma = x^T y$$

Suppose you have a robust backward stable routine to compute the sum of $n$ numbers and it is used to compute the two inner products. Which of the two inner products would you expect to be less sensitive to the perturbations caused by the finite precision of IEEE floating point arithmetic? Justify your answer.

## Problem 1.3

Consider the summation $\sigma = \sum_{i=1}^{n} \xi_i$ using the following "binary fan-in tree" algorithm described below for $n = 8$ but which clearly generalizes easily to $n = 2^k$:

$$\sigma = \{[(\xi_0 + \xi_1) + (\xi_2 + \xi_3)] + [(\xi_4 + \xi_5) + (\xi_6 + \xi_7)]\}$$

or equivalently

$$\sigma_j^{(0)} = \xi_j, \ 0 \leq j \leq 3$$

$$\sigma_0^{(1)} = \xi_0 + \xi_1, \quad \sigma_1^{(1)} = \xi_2 + \xi_3, \quad \sigma_2^{(1)} = \xi_4 + \xi_5, \quad \sigma_3^{(1)} = \xi_6 + \xi_7$$

$$\sigma_0^{(2)} = \sigma_0^{(1)} + \sigma_1^{(1)}, \qquad \sigma_1^{(2)} = \sigma_2^{(1)} + \sigma_3^{(1)}$$

$$\sigma = \sigma_0^{(3)} = \sigma_0^{(2)} + \sigma_1^{(2)}$$

In general, there will be $k = \log n$ levels and $\sigma = \sigma_0^{(k)}$. Level $i$ has $2^{k-i}$ values of $\sigma_j^{(i)}$, each of which corresponds to a sum

$$\sigma_j^{(i)} = \xi_{2^i j} + \ldots + \xi_{2^i (j+1)-1}.$$

The algorithm is easily adaptable to $n$ that are not powers of 2.

**1.3.a**. Derive an expression for the absolute forward error of the method for a fixed $n = 8$ or $n = 16$ and then generalize to $n = 2^k$.

**1.3.b**. Derive an expression for the absolute backward error of the method for $n = 8$ or $n = 16$ then generalize to $n = 2^k$.

**1.3.c**. Bound the errors and discuss stability relative to the simple sequential summation algorithm given by, for $n = 8$ but easily generalizable to any $n$,

$$\sigma = (((((((\xi_1 + \xi_2) + \xi_3) + \xi_4) + \xi_5) + \xi_6) + \xi_7) + \xi_8)$$

or equivalently

$$\sigma = \xi_1$$

$$\sigma \leftarrow \sigma + \xi_i, \quad i = 2, \ldots, 8$$

## Programming Exercise

## Problem 1.4

### 1.4.a Overview

The purpose of this exercise is the become familiar with defining and organizing a set of experiments that include specific individual tests and large groups of related tests in order to demonstrate the correctness of your codes and to assess empirically theoretical properties such as conditioning and numerical stability. This exercise considers the two most basic approaches to summing $n$ numbers and an important modification of the most basic one:

1. the recursive algorithm of the notes (simple accumulation)
2. the binary fan-in tree algorithm of the notes
3. double precision accumulation

You must design experiments to demonstrate the correctness of your code and check conditioning and stability bounds.

### 1.4.b Implementation Requirements

#### Data for the codes to be tested

The accumulation sum algorithm and the fan-in algorithm should be implemented in a compiled and typed language. Since these are the codes to be evaluated all operations, input data and output data are to be single precision unless otherwise indicated.

Specifically, your input data $\xi_1, \ldots, \xi_n$ will come from one of three sources:

1. file input
2. manual input in response to a request by the code
3. closed form expressions

For the first two the data should be read into a double precision array, e.g., XI_DP(1:n). These values will serve as the "exact" real numbers to be summed. The closed form expressions, e.g., $1/i^2$, $a^{-k}$, should be computed in double precision and stored in XI_DP(1:n). The input to the single precision codes to be tested will be generated by copying the data in XI_DP(1:n) to a single precision input data array XI_SP(1:n). This will coerce the type and create a single precision of the data for consumption by the algorithms.

The three algorithms will therefore compute SUM(XI_SP(1:n)) to be compared against the "exact" sum $\sigma_{exact} = \sum_{i=1}^{n} \xi_i$.

### Specifications for the codes to be tested

All three algorithms should be implemented to be as efficient as possible in space and operations. Your solutions must be clear and concise about justifying your design.

Using the single precision input data XI_SP(1:n) all operations for the accumulation algorithm and the fan-in algorithm should be single precision. The double precision accumulation algorithm is the same as the accumulation algorithm in that it operates on the same single precision input data and produces a single precision output. However, it differs in that the variable S_DP in which the sum is accumulated is double precision and the addition that updates it is double precision. Specifically, the two algorithms have the fundamental steps:

- basic accumulation algorithm:
  - S_SP single precision, XI_SP(1:n) single precision
  - accumulation statement S_SP = S_SP + XI_SP(I)
- double precision accumulation algorithm:
  - S_SP single precision, S_DP double precision, XI_SP(1:n) single precision
  - accumulation statement S_DP = S_DP + double(XI_SP(I))
  - double(XI_SP(I)) converts the single precision value into a double precision representation. It is not XI_DP(I). This conversion is usually done automatically when one of the operands and the result are double precision.
  - final sum is S_SP = single(S_DP) that converts the accumulated double precision value to single precision. This is done automatically when assigning a double precision variable to a single precision variable.

## 1.4.c The Exact Data and Sum

As noted earlier, the input data will be read or generated in double precision and stored in XI_DP(1:n) and will represent the "exact" data to be summed. These data could be input or computed.

Similarly, the "exact" sum $\sigma_{exact} = \sum_{i=1}^{n} \xi_i$ will be either a double precision computation of a known closed form, e.g.,

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

or it will be computed from the double precision data XI_DP(1:n) using a double precision version of the accumulation algorithm or the fan-in algorithm.

Therefore, you must also implement a fully double precision version of either of the two basic algorithms in which all operations are double precision and the final sum is double precision S_DP. This value will be used as the exact sum $f(d)$ in all conditioning and stability bounds. You are encouraged to implement both basic algorithms and to provide evidence supporting your choice of which you use as the exact sum generating algorithm.

## 1.4.d Tasks

1. Correctness: Design, execute and summarize tests to demonstrate that your code is correct. This should use sums with known closed form totals, random data, sums with particular structure to the data (think about the comments in the stability and conditioning lectures/notes). A variety of lengths of sums, $n$, should be used. The double precision evaluation of the closed form or the double precision algorithm you choose should be used to generate $f(d) = \sigma_{exact} \approx$ S_DP. Your algorithms will provide the "computed" answers $g(d)$ for the stability bound assessment.

2. Conditioning: Design, execute and summarize tests to assess the conditioning bound on summation. This requires the use of $f(d)$ and $f(d+e)$, i.e., the "exact" closed forms or double precision algorithm. The data $d$ will therefore be double precision and $d+e$ can be generated by perturbing the elements of XI_DP(1:n) with relative perturbations $\delta_i$ for elements that are not small, i.e., not subnormal. Be sure to organize and discuss your hypotheses, design, and observations clearly.

3. Stability: Design, execute and summarize tests to assess the numerical stability of the three algorithms. This requires the use of $f(d)$ and $g(d)$ as well as the stability bounds in the notes and the bound for the fan-in algorithm you derive in the written exercises (for which your experiments will provide evidence for or against your written problem answer). For the double precision accumulation algorithm, consider how you expect the single precision accumulation algorithm stability bound to change and evaluate that hypothesis with these experiments.

## 1.4.e Organization of Results

The main goal of this assignment is for you to learn how to organize and present your design, execution and analysis of observations. You may use graphics, statistical, and algorithmic support from established libraries and environments such as Matlab. The algorithmic support, of course, may not be used as a substitute for your codes but it can be used for problem generation, i.e., random number generation, computation of sums for comparison, and computation of various values useful for presentation of results. Make use of specific tests, i.e., particular sums of a particular length, and its results to make points, e.g., verifying very well or very ill conditioned examples as well as large groups of sums that are some how related to make a point, e.g., mean and variance of accuracy of the computed sum. Histograms, graphs and appropriate tables should be used to compress data and support conclusions. Simply running a few sums and comparing the answers to the exact sums is not acceptable and will not prepare you for more complicated algorithm evaluation later in the two semesters or your research.

### 1.4.f Some Test Sums

In addition to sums you may develop for testing based on the notes and your readings, these might be useful

$$S_1 = \sum_{i=1}^{N} (-1)^i \frac{1}{i^p}, \quad p > 0$$

$$S_2 = \sum_{i=1}^{N} i \sin\left[\left(\frac{i}{2} + \frac{1}{i}\right)\pi\right]$$

$$S_3 = \sum_{i=1}^{N} i \sin\left(\frac{i}{2} + \frac{1}{i}\right)$$

$$S_4 = \sum_{i=1}^{N} a_i$$

where

$$\{a_i\} = \{1,\ \omega_1 M, \ldots, \omega_{N-1} M\},$$

$\omega_i$ for $1 \le i \le N-2$ are random integers from $[-3, 3]$,

$$w_{N-1} = -\sum_{n=1}^{N-2} w_n, \ \text{ i.e., } \sum_{i=1}^{N-1} w_i = 0$$

and $M > 10^6$ is a big number compared to 1.

Finally,

$$S_5 = \sum_{i=1}^{N} a_i$$

where the $\{a_i\}$ are as in $S_4$ except the $\omega_i$ for $1 \le i \le N-2$ are random values from the set $\{-3, -2.9, \cdots, -0.1, 0, 0.1, \cdots, 2.9, 3\}$.