Graded Homework 1 Foundations of Computational Math 1 Fall 2021

The solutions must be submitted by 11:59PM on October 1, 2021using the Canvas course page.

Written Exercises

Problem 1.1

Let $f(\xi_1,\ \xi_2, \ldots, \xi_k)$ be a function of k real parameters ξ_i, $1 \leq i \leq k$. Recall, the relative condition number of f with respect to ξ_1 can be expressed

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

where $0 \leq c(\xi_1, \xi_2, \ldots, \xi_k)$ is a value that indicates the sensitivity of f to small relative perturbations to ξ_1 as a function of the parameters ξ_i, $1 \leq i \leq k$. If $c(\xi_1, \xi_2, \ldots, \xi_k) \leq 1$ 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 ξ_1.

Let $n \geq 2$ 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 β.

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

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

Problem 1.2

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

$$\sigma_j^{(0)} = \xi_j,\ 0 \le j \le 3$$
$$\sigma_0^{(1)} = \xi_0 + \xi_1, \quad \sigma_1^{(1)} = \xi_2 + \xi_3, \quad \sigma_2^{(1)} = \xi_4 + \xi_5, \quad \sigma_3^{(1)} = \xi_6 + \xi_7$$

$$\sigma_0^{(2)} = \sigma_0^{(1)} + \sigma_1^{(1)}, \qquad \sigma_1^{(2)} = \sigma_2^{(1)} + \sigma_3^{(1)}$$
$$\sigma = \sigma_0^{(3)} = \sigma_0^{(2)} + \sigma_1^{(2)}$$

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.

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

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

1.2.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 1.3

1.3.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 one of them:

1. the recursive algorithm of the notes (simple accumulation)
    - IEEE Single Precision version
    - IEEE Double Precision version
    - A t-decimal digit via simple simulation version.
2. the binary fan-in tree algorithm of the notes
    - IEEE Single Precision version
    - IEEE Double Precision version
    - A t-decimal digit via simple simulation version.
3. double precision accumulation (described below)

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

1.3.b Implementation Requirements

Data for the codes to be tested

All algorithms and their versions should be implemented in a compiled and typed language. The goal is to evaluate the single-precision of the three algorithms (with double-precision used very specifically as described below in the third algorithm). The t-decimal-digit version is to be used on only a few examples.

In 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. Similarly, for the t-decimal-digit tests the elements of XI_DP(1:n) should be copied into a second double precision array XI_TD(1:n) where XI_TD(i)= XI_TD(i)$\times(1+\epsilon_i)$ where the relative change of $-1 < \epsilon_i < 1$ with a randomly selected magnitude satisfying $|\epsilon_i| < 0.5\ 10^{-t}$. The sign should also be selected randomly.

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. All three must work for an arbitrary value of n. This will require some generalization of the fan-in as presented in the notes and homework.

Using the single precision input data XI_SP(1:n) all operations for the basic accumulation algorithm and the fan-in algorithm should be single precision or t-decimal-digit simulation. 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) using single precision IEEE arithmetic
- double precision accumulation algorithm (Note that the input is still the coerced single precision data array):
  - S_SP single precision, S_DP double precision, XI_SP(1:n) single precision
  - accumulation statement S_DP = S_DP + double(XI_SP(I)) using double precision IEEE arithmetic.
  - 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.

A fully double precision version of the basic accumulation algorithm should also be implemented as a simple modification of the double precision accumulation algorithm, i.e.,

- S_TRUE double precision,
- XI_DP(1:n) original double precision input data
- accumulation statement S_TRUE = S_TRUE + XI_DP(I) using double precision IEEE arithmetic

BE SURE YOU UNDERSTAND THE DIFFERENCE BETWEEN THE FULLY DOUBLE PRECISION ALGORITHM AND THE DOUBLE PRECISION ACCUMULATION ALGORITHM.

Similarly, you should implement a fully double precision fan-in algorithm that sums the elements of XI_DP(1:n) original double precision input data to give S_TRUE in double precision.

The t-decimal-digit versions of the basic accumulation algorithm and fan-in algorithm should be based on the fully double precision versions just described. However, it operates on the perturbed data t-decimal-digit input data array XI_TD(i) and every double precision operation in the fully double precision versions should be modified to include a t-decimal-digit perturbation before storing it in the accumulation variable. For example, the double precision basic summation has its accumulation statement modified to be S_TD = (S_TD + XI_TD(I))$\times(1+\epsilon)$ computed in double precision with each ϵ randomly generated as above.

1.3.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 may use the fully double precision version of either of the two basic algorithms (basic summation and fan-in) 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.

1.3.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 δ_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.

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

In addition to the tasks above, use the t-decimal-digit version of the basic summation and fan-in algorithms on a small number of examples to show that the conclusions you reach about the IEEE versions can be applied to a less accurate floating point system with the parameters in the conclusions altered appropriately.

1.3.e Organization of Summation 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.

1.3.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

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

ω_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 ω_i for $1 \le i \le N-2$ are random values from the set $\{-3, -2.9, \cdots, -0.1, 0, 0.1, \cdots, 2.9, 3\}$.

1.3.g Other Examples from Notes

Several examples of cancellation and algorithmic alterations to avoid it are given in the notes and study questions. Write code to implement some of them to verify the statements about the effect of cancellation in computations other than summations and how they may be fixed. You should include the use of the quadratic equation and its improved algorithm as one of the examples presented. Note that most of the work for this assignment is concerned with the summation algorithms above. This section need not be that involved but should be enough to be evidence of your understanding of the problem and the fixes discussed.