Numerical Challenge to Ill-posed Problems by Fast Multiple-Precision System

Hiroshi Fujiwara and Yuusuke Iso

Graduate School of Informatics, Kyoto University, Kyoto

We discuss an application of a multiple-precision system to numerical computation for ill-conditioned problems. We introduce the Fast Multiple-precision System (F-system) constructed by the first author and show some numerical results applied to numerical analysis for an integral equation of the 1st kind.

1 Introduction

The aim of the present paper is to show utility of the Fast Multiple-precision System (F-system) applied to numerical analysis for ill-posed problems. Development of recent technology requires us to deal with numerical treatments of ill-posed problems including various kinds of inverse problems, but we cannot expect stable numerical processes for the ill-posed problems in the sense of Hadamard. Ill-posedness is the opposite concept to well-posedness which implies stability of numerical processes in their discretizations, and some device should be necessary in numerical computations for them.

It is almost impossible to construct exact numerical solutions to discretizations for functional equations, e.g. PDE’s, by the usual numerical computation, because we must admit rounding errors in the expression of real numbers on computers. The influence of the rounding errors does not matter in numerically stable processes, but it is crucial in numerically unstable processes which should arise in discretizations of ill-posed problems. Stabilization for numerically unstable problems, which are called ill-conditioned problems, is so important that we lose reliability of their numerical solutions without it.

We propose an application of multiple-precision systems to ill-conditioned problems in order to construct reliable numerical solutions, and the authors have succeeded in some cases. We consider a fatal blow in numerically unstable processes to be rapid increase of the rounding errors, and the control of the rounding errors is expected to lead us to reasonable numerical solutions. Hence we introduce a multiple-precision system, which is one of the extended formats (see [5]) in the floating point arithmetic, and we develop and implement it in order to apply it to numerical analysis for ill-conditioned problems.
One of the defects of the application of multiple-precision systems is consumption of vast computing time, because most of the systems are not aimed for numerical computations of the ill-conditioned problems. We need large amount of memories in numerical simulations for differential equations and integral equations, and we should tune up the systems to apply them to the simulations. It is quite desirable to design and implement a fast multiple-precision system in order to realize our proposal, and the first author has succeeded in implementation of the F-system (Fujiwara [1]). We shall show details of the F-system in the next section.

We shall show in §3 some numerical results of the F-system applied to a numerical computation of the 1st kind integral equation with an analytic kernel. This problem is one of the most typical examples of ill-posed problems, and we give numerical solutions to the problem by use of the F-system.

2 Fast Multiple-Precision System

In this section, we shall introduce the Fast Multiple-precision System (F-system) implemented by the first author. Refer to Fujiwara [1] for details.

2.1 Principles of Design and the Feature

Our aim is the numerical treatment of ill-conditioned problems under a control of the rounding errors, and we need a computer system in which we can represent and operate real numbers without rounding errors. To this end, we choose a strategy that we fix an arbitrary (finite) precision to represent and to calculate approximated real numbers with guarantee of this precision. We call it a multiple-precision arithmetic system, which is considered one of the extended formats. Though we cannot remove the rounding errors even by the strategy, we virtually carry out exact numerical computations on the floating point arithmetic. The term 'virtual' means that we control significant digits to avoid the influence of the rounding errors as much as we wish, and we obtain numerical results as precisely as we wish.

The principles of the design for the F-system are;

- fast computing,
- saving of memory, which enables us to deal with large scale problems,
- fully designed in 64bit base; possibility of high performance with resources of the typical next generation computer environments,
- equipment of interfaces for easy access of low end users.

The most important features in the construction of a fast multiple-precision system are data structure and implementation of improved arithmetic algorithms.

We firstly explain a format of multiple-precision numbers. Though several ways have been proposed to represent multiple-precision numbers, we adopt an original extension of the floating point format designed in IEEE754 (see [5]). The following figure (Fig. 1) shows our format. Each real number is normalized in the form \((-1)^s \times 2^e \times 1.F\), where a sign part \(s\) has one bit, an exponent part \(e\) has 63 bits, and a fraction part \(F = f_1f_2\cdots f_n\) has \(64 \times n\) bits. (Each \(f_i\) has 64bits.) This type of floating point multiple-precision numbers are stored in memory using the
Fig. 1: Format of a floating point multiple-precision number

64-bit unsigned integer array, and accuracy of the numbers is $\log_{10} 2^{64n+1} (\approx 19.26 \times n)$ digits. Hence a real number of the format shown in Fig. 1 is

$$(-1)^s \times 2^{e_b} \times \left( 1 + \sum_{i=1}^{n} f_i \times 2^{-64i} \right),$$

where $b = 2^{62} - 1$.

One of the advantages of the design is cost performance in the use of memory. The ALU (Arithmetic Logical Unit) of 64-bit CPU's is more advantageous in accuracy than the FPU (Floating Point Unit), and we can use all bits in the integer array which stores a fraction part of the multiple-precision format shown in Fig. 1, although we can use at most about 40% of memory resource in the design of a floating-point number array.

Hence we are enough to use less memory than in the case of IEEE754 double format number array, and our system becomes possible to deal with larger scale problems. At the same time, since a number of necessary elements of the array decreases, and since the memory access and branch operators are less issued, we succeed in speeding up in computation. We remark that our 64-bit integer base representation and operation should be a remarkable feature of the design and have an advantage.

We secondly mention improvement and implementation of arithmetic algorithms. In our system, the arithmetic for multiple-precision numbers is designed as external routines of the programming language C (LP 64 model). We choose the Alpha system as a 64-bit computing environment in the present research. The four fundamental rules for multiple-precision arithmetic, multiplication by integer, and division by integer are implemented in the Alpha assembly language and are optimized for its architecture. The classical algorithms are used in addition and in subtraction, and we adopt the classical algorithm ($O(n^2)$) and Karatsuba-Offman's algorithm ($O(n^{\log_2 3})$) in multiplication (see [4]). In division, we implement an improved Ozawa's algorithm (see [7]); we adopt the original algorithm in the quotient predication for minimizing the number of error correction.

2.2 Interface

Considering usability, we should design multiple-precision systems as a module of an existing programming language, and we implement our system as external routines of the programming language C.

The language C has a disadvantage in vulnerability of the argument types etc, and users should be requested to parse formulas in their programming. These processes are burdens for users, and we design the interface of the F-system by use of polymorphism supplied by the programming language C++.
In our system, operators of four basic arithmetic rules, comparison, and printing multiple-precision numbers to an output stream in radix-10 are polymorphic, and manipulators also work well to specify a printing format. For example, using operator overloading for realizing
\[ S = \frac{(a + b)h}{2}, \]
users are simply requested to type
\[ S = (a+b)*h/2; \]
and in case of its change to
\[ S = \frac{(a + b)h}{1.5}, \]
they are sufficient to type
\[ S = (a+b)*h/1.5; \]
The compiler automatically parses the statements and binds appropriate functions.

The features of our interface using polymorphism are;

- saving users from a burden,
- security for type consistency,
- portability of user programs written in the language C,
- robustness for changing programs.

3 Numerical Results

We shall show some numerical results of the application of the F-system to numerical computations of an integral equation of the 1st kind, of which the discretization is an ill-conditioned problem.

Consider an integral equation on the interval \(0 \leq x \leq 1\),
\[
\int_0^1 e^{xy} u(y) \, dy = \frac{1}{x} \{ (1-e^x + 1) + (x-1)e^x \}. \tag{3.1}
\]
We should remark that the exact solution is \(u(x) = x \) (\(0 \leq x \leq 1\)). For a discretization by the collocation method, let \(N\) be an integer and set \(h = 1/N\), and \(x_j = jh\) (\(0 \leq j \leq N\)). We denote our aimed solution \(u_h(x)\) by
\[
u_h(x) = \sum_{j=0}^N u_j \varphi_j(x), \tag{3.2}\]
where \(\{ \varphi_j \}_{j=0}^N\) a set of piecewise linear functions on [0,1] with \(\varphi_j(x_i) = \delta_{ij} \) (\(0 \leq i,j \leq N\)).
We apply the collocation method to calculate
\[ a_{ij} := \int_0^1 e^{xy} \varphi_j(y) \, dy, \]
and we obtain a discretized problem

\[(a_{ij})(u_i) = (f(x_i)), \quad (3.3)\]

where \(f(x) = x^{-2}(x - 1)e^x + 1\).

We solve a system of linear equation (3.3) on floating-point arithmetics and show the graphs of numerical solutions in Fig. 2 and Fig. 3.

![Fig. 2: Result computed by the double precision](image1)

![Fig. 3: Result computed by the F-system with 190 digits in radix-10](image2)

We calculate it both by the double precision system (Fig. 2) and by the F-system with 190 digits in radix-10 (Fig. 3). The dotted line denotes the exact solution to the equation (3.1) and square dots denote computed values. We note that the results by the F-system coincide with the exact solution. We have the same kind of results in its application to numerical analysis for an inverse scattering problem [3], [6].

4 Conclusion

A simple example in §3 implies nice utility of the F-system applied to numerical computations of ill-conditioned problems, but we should pay attention to the connection between the F-system and the Tikhonov regularization (e.g. [2]) for ill-posed problems arising in engineering and physics. We remark that the F-system is one of the new tools to treat ill-conditioned problems, but its mere application does not seem enough for numerical analysis of practical ill-posed problems. For details of cost performance of the F-system, refer to Fujiwara [1].

References


