Computing Best Possible Pseudo-Solutions to Interval Linear Systems of Equations[*]

Anatoly V. Panyukov
South Ural State University, Chelyabinsk, Russia
a_panyukov@mail.ru

Valentin A. Golodov
South Ural State University, Chelyabinsk, Russia
avaksa@gmail.com

Abstract

In the paper, we consider interval linear algebraic systems of equations $\boldsymbol{A}x = \boldsymbol{b}$, with an interval matrix $\boldsymbol{A}$ and interval right-hand side vector $\boldsymbol{b}$, as a model of imprecise systems of linear algebraic equations of the same form. We propose a new regularization procedure that reduces the solution of the imprecise linear system to computing a point from the tolerable solution set for the interval linear system with a widened right-hand side.

The points from the tolerable solution set to the widened interval linear system are called *pseudo-solutions*, while the *best pseudo-solutions* are those corresponding to the minimal extension of the right-hand side that produces a nonempty tolerable solution set. We prove the existence of the best pseudo-solutions and propose a method for their computation, as a solution to a linear programming problem. Since the auxiliary linear programming problem may become nearly degenerate, it is necessary to perform computations with a precision that substantially exceeds that of the standard floating point data types. A simplex method with errorless rational computations gives an effective solution to the problem. Coarse-grained parallelism for distributed computer systems using MPI and the software for errorless rational calculations using CUDA C small-grained parallelism are the main instruments of our suitable implementation.

Keywords: system of linear equations, pseudo-solution, regularization, interval linear system, tolerable solution set, linear programming problem, exact computations

AMS subject classifications: 65F22, 65G40

[*]Submitted: December 19, 2012; Revised: July 22, 2013; Accepted: December 19, 2013.

1 Introduction and Main Idea

A system of linear algebraic equations is a fundamental object that arises in the solution of many problems. Sometimes, the coefficients of such systems cannot be determined precisely, but we know intervals that contain them. Under such interval uncertainty, the definition of a solution must be specified.

In our paper, we consider the interval linear system of equations as a class of equivalent point linear systems specified to within a prescribed accuracy set by the interval elements in the matrix and right-hand side vector. We are going to find a single point solution to such an interval system that represents a solution of the "approximate" linear system best of all.

We first give a short survey of various concepts of solutions and solution sets to interval systems of equations. Different interpretations of the interval uncertainty have been systematized and its classified in [17, 18]. In particular, one has to distinguish between two interval uncertainty types, A-uncertainty and E-uncertainty, that correspond to the logical quantifiers "$\forall$" and "$\exists$", respectively. In accordance with this classification, there exist many solution sets to interval systems of equations and inequalities that differ in the logical quantifiers applied to the interval parameters. The interval E-uncertainty is also called *weak*, while the A-uncertainty is called *strong.*

The most popular solution sets to the interval systems of equations are those formed by $\exists\exists$-solutions. Such solution sets often are called *united solution sets* or sets of *weak solutions* (as far as they correspond to the "weak" interval uncertainties both in the matrix and right-hand side). Given a square system of interval linear algebraic equations of the form $\boldsymbol{A}x = \boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{a_{ij}} = ([\underline{a}_{ij}, \overline{a}_{ij}])$ and $\boldsymbol{b} = (\boldsymbol{b_j}) = ([\underline{b}_j, \overline{b}_j])$, $\underline{a}_{ij}$, $\overline{a}_{ij}$, $\underline{b}_i$, $\overline{b}_i \in \mathbb{R}$ for $i, j = 1, 2, \dots, n$, the united solution set or the set of weak solutions is formally defined as

$$\varXi_{uni}(\boldsymbol{A}, \boldsymbol{b}) = \left\{ x \;\middle|\; (\forall i, j = 1, 2, \dots, n)\, (\exists a_{ij} \in \boldsymbol{a}_{ij})\, (\exists b_i \in \boldsymbol{b}_i) \left(\sum_{j=1}^{n} a_{ij} x_j = b_i \right) \right\},$$

(see details in [2, 7, 19]), which is equivalent to

$$\varXi_{uni}(\boldsymbol{A}, \boldsymbol{b}) = \left\{ x \in \mathbb{R}^n \;\middle|\; (\exists A \in \boldsymbol{A})\, (\exists b \in \boldsymbol{b})\, (Ax = b) \right\}.$$

Another solution set frequently encountered in various practical problems is the *tolerable solution set*

$$\varXi_{tol}(\boldsymbol{A}, \boldsymbol{b}) = \left\{ x \;\middle|\; (\forall i, j = 1, 2, \dots, n)\, (\forall a_{ij} \in \boldsymbol{a}_{ij})\, (\exists b_i \in \boldsymbol{b}_i) \left(\sum_{j=1}^{n} a_{ij} x_j = b_i \right) \right\},$$

usually written as

$$\varXi_{tol}(\boldsymbol{A}, \boldsymbol{b}) = \left\{ x \in \mathbb{R}^n \;\middle|\; (\forall A \in \boldsymbol{A})\, (\exists b \in \boldsymbol{b})\, (Ax = b) \right\}.$$

It is formed by all points x such that the product Ax falls into the right-hand side interval bounds $\boldsymbol{b}$ for every $A \in \boldsymbol{A}$ [17, 18]. These two solution sets have much in common, but their behavior differ greatly as the interval matrix of the system varies. When the interval matrix widens, the united solution set grows in size, while the tolerable solution set shrinks.

The tolerable solution set can be represented as

$$\varXi_{tol}(\boldsymbol{A}, \boldsymbol{b}) = \bigcap_{A \in \boldsymbol{A}} \left\{ x \in \mathbb{R}^n \;\middle|\; (\exists b \in \boldsymbol{b})\, (Ax = b) \right\}, \tag{1}$$

while for the united solution set, the dual representation is valid:

$$\Xi_{uni}(\boldsymbol{A},\boldsymbol{b}) = \bigcup_{A\in\boldsymbol{A}} \left\{ \, x \in \mathbb{R}^n \;\middle|\; (\exists b \in \boldsymbol{b})\,(Ax = b) \right\}.$$

In the above formulas, $\left\{ \, x \in \mathbb{R}^n \;\middle|\; (\exists b \in \boldsymbol{b})(Ax = b) \right\}$ is the solution set to the interval system $Ax = \boldsymbol{b}$ with the interval uncertainty concentrated only in the right-hand side vector.

The presence of the universal quantifier $\forall A \in \boldsymbol{A}$ in the definition (1) of the tolerable solution set is equivalent to the set-theoretical intersection. Then, it follows from (1) that the tolerable solution set is least sensitive, among all the solution sets, to the change in the interval matrix of the system $\boldsymbol{A}x = \boldsymbol{b}$, since (1) is not greater than the solution set $\left\{ \, x \in \mathbb{R}^n \;\middle|\; (\exists b \in \boldsymbol{b})\,(Ax = b) \right\}$ determined by the "best" and "most robust" matrix $A \in \boldsymbol{A}$. Although some matrices from $\boldsymbol{A}$ may be "bad" and even singular, the contribution of "good" matrices to (1) causes the tolerable solution set to be bounded and well-estimated. Consequently, we can involve the tolerable solution set as a kind of "regularizing tool" in the solution of sensitive and ill-conditioned problems, when one needs to somehow "smooth" the effect of variation in the solution of the problem caused by imprecise data or any disturbances. Our paper exploits and elaborates this idea that may be called *interval regularization*.

In many practical situations, the interval linear system of equations may be ill-conditioned or even inconsistent, that is, represent ill-conditioned or incompatible point equations. For such cases, it makes sense to introduce a concept of a *pseudo-solution* by analogy with the work [5, 22].

J. Rohn obtained in 1985 a result on a characterization of the points from the tolerable solution set [15] (called 'inner solutions' at that time). Rohn's Theorem claims that any point $x \in \Xi_{tol}(\boldsymbol{A},\boldsymbol{b})$ may be represented in the form $x = x^+ - x^-$, where x^+ and x^- are solutions to the inequality system

$$\begin{aligned}
\sum_{j=1}^{n} \left(\underline{a}_{ij} x_j^+ - \overline{a}_{ij} x_j^- \right) \;&\geq\; \underline{b}_i, \qquad i = 1, 2, \ldots, n.,\\
\sum_{j=1}^{n} \left(\overline{a}_{ij} x_j^+ - \underline{a}_{ij} x_j^- \right) \;&\leq\; \overline{b}_i, \qquad i = 1, 2, \ldots, n,\\
x^+, x^- \;&\geq\; 0.
\end{aligned}$$

Consequently, finding tolerable solutions to interval linear systems has polynomial complexity.

Estimation of the tolerable solution set for the case $\Xi_{tol}(\boldsymbol{A},\boldsymbol{b}) \neq \varnothing$ is considered in detail, e.g., in [17] and other work. A very powerful tool for the investigation of the tolerable solution set is developed in Novosibirsk [17, 20] as the "method of recognizing functional". It draws a conclusion on the solvability of the problem (i.e., about emptyness / nonemptyness of the solution set) after computing an unconstrained maximum of a special (nonsmooth and concave) functional, named "recognizing". Maximization of the recognizing functional may be performed practically with the use of nonsmooth optimization methods (e.g., those developed in the Institute of Cybernetics NAS of Ukraine by P.I. Stetsyuk are very popular and thoroughly tested [21]). Additionally, the information obtained during the seach of the maximum of the recognizing functional makes it possible to correct the initial tolerance problem in a desirable way. Computer codes for investigation of solvability of the interval linear tolerance problem (emptiness / nonemptyness of solutions set, etc.), developed by S.P. Shary and P.I. Stetsyuk are publicly available at [25]. These programs are implemented using INTLAB, an interval extension of MATLAB, and Int4Sci, an interval extension of Scilab.

This paper presents in a more detailed form the results previously announced in [8, 9, 10], that relate to pseudo-solutions for systems of linear equations under interval uncertainty and numerical methods for computation of the best possible pseudo-solution for such systems.

In our paper, we use the standard notation of interval analysis [6].

2 Pseudo-Solutions to Interval Linear Systems

Let an interval linear system of equations $\boldsymbol{A}x = \boldsymbol{b}$ be given, where elements of the matrices $\boldsymbol{A}$ and $\boldsymbol{b}$ are intervals $\boldsymbol{a}_{ij} = \left[\underline{a}_{ij}, \overline{a}_{ij}\right]$, $\boldsymbol{b}_j = \left[\underline{b}_j, \overline{b}_j\right]$, $i, j = 1, 2, \ldots, n$. According to the ideas presented in the previous section, we wish to define pseudo-solutions for $\boldsymbol{A}x = \boldsymbol{b}$ as points from the tolerable solution set $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b})$. However, the main problem with this construction is that the tolerable solution set $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b})$ may be empty even for common interval data $\boldsymbol{A}$ and $\boldsymbol{b}$ (see e.g., [17, 19]).

A natural way out of the difficulty is to artificially extend ("inflate") the right-hand side $\boldsymbol{b}$ of the interval system to get a nonempty tolerable solution set $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b})$. The details of the construction are as follows. We embed the given system of equations $\boldsymbol{A}x = \boldsymbol{b}$ into a parameterized family of systems of the equations of the form $\boldsymbol{A}x = \boldsymbol{b}(z)$ with a modified right-hand side vector $\boldsymbol{b}(z) = \left[\underline{b} - z\,|\underline{b}|, \overline{b} + z\,|\overline{b}|\right]$, $z \geq 0$. Then the parameter z is chosen to ensure the nonemptyness of $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b})$.

Let z be such that $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b}(z)) \neq \varnothing$. We will refer to points of the tolerable set $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b}(z))$ as pseudo-solution of the original system $\boldsymbol{A}x = \boldsymbol{b}$. A pseudo-solution that corresponds to the minimal extension of the right-hand side of the system of equations, i.e., to $z^* = \inf\{\, z \mid \Xi_{tol}(\boldsymbol{A}, \boldsymbol{b}(z)) \neq \varnothing\}$, is the *best possible pseudo-solution.*

The following theorem states the existence of the introduced objects.

Theorem 2.1 *For any interval linear system of equations $\boldsymbol{A}x = \boldsymbol{b}$ and for all $z \geq 1$, the tolerable set $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b}(z))$ is nonempty.*

Proof: It follows from Rohn's theorem that the condition $\Xi_{tol}(\boldsymbol{A}, \boldsymbol{b}(z)) \neq \varnothing$ is equivalent to the consistency of the inequality system

$$\sum_{j=1}^{n} \left(\underline{a}_{ij} x_j^+ - \overline{a}_{ij} x_j^-\right) \;\geq\; \underline{b}_i - z\,|\underline{b}_i|\,, \quad i = 1, 2, \ldots, n., \tag{2}$$

$$\sum_{j=1}^{n} \left(\overline{a}_{ij} x_j^+ - \underline{a}_{ij} x_j^-\right) \;\leq\; \overline{b}_i + z\,|\overline{b}_i|\,, \quad i = 1, 2, \ldots, n, \tag{3}$$

$$x^+, x^- \;\geq\; 0. \tag{4}$$

Put in (2) – (4) $x^+ = x^- = 0$, this yields

$$0 \geq \underline{b}_i - z|\underline{b}_i|, \quad 0 \leq \overline{b}_i + z|\overline{b}_i|, \quad i = 1, 2, \ldots, n.$$

Thus, for all $z \geq 1$, the inclusion $0 \in \Xi_{tol}(\boldsymbol{A}, \boldsymbol{b}(z))$ is true, completing the proof. ■

Traditional linear systems of equations are a special case of interval linear systems of equations, so the pseudo-solution is not unique, e.g., when rank $A < n$ for the square $n \times n$-matrix. The case $\varXi_{tol}(\boldsymbol{A}, \boldsymbol{b}) = \varnothing$, i.e., $z > 0$ has its own features.

Another concept that arises in connection with interval linear systems and relates to our pseudo-solutions: is that these are *universal solutions*, as introduced in [1]. Let ε be a nonnegative vector from $\mathbb{R}^n$. Then the set of ε-solutions to the system of linear equations $Ax = b$ is

$$X_\varepsilon = \big\{\, x \in \mathbb{R}^n \;\big|\; |Ax - b| \le \varepsilon \,\big\}.$$

For an interval linear system $\boldsymbol{A}x = \boldsymbol{b}$, the concept of ε-solutions requires that the inequality $|Ax - b| \le \varepsilon$ holds for all the admissible $A \in \boldsymbol{A}$ and $b \in \boldsymbol{b}$. That is, the *set of ε-solutions* to $\boldsymbol{A}x = \boldsymbol{b}$ should be

$$X_\varepsilon = \Big\{\, x \in \mathbb{R}^n \;\Big|\; \max_{A \in \boldsymbol{A},\, b \in \boldsymbol{b}} |Ax - b| \le \varepsilon \,\Big\}.$$

Then we set a norm $\|\cdot\|$ and find minimal, with respect to this norm, vector $\tilde{\varepsilon}$ such that the set $X_{\tilde{\varepsilon}}$ of ε-solutions is still nonempty. The "universal solutions" to the interval linear system under study are defined as points from such a set $X_{\tilde{\varepsilon}}$. In other words, the universal solution x^{uni} to the interval linear system of equations $\boldsymbol{A}x = \boldsymbol{b}$ is

$$x^{uni} \in \operatorname{Arg} \min_{X_\varepsilon \neq \varnothing} \|\varepsilon\|,$$

where a large letter in Arg means that the argument of min may be set-valued.

The concept of universal solutions to interval systems of equations proves useful in some practical situations [1], but it differs from our pseudo-solutions intended primarily for regularization of ill-conditioned problems.

3 Computing the Best Pseudo-Solution

Our computational approach to finding the best possible pseudo-solutions to the system of equations $\boldsymbol{A}x = \boldsymbol{b}$ is based on the following result.

Theorem 3.1 *There exists a solution x^{+^*} and $x^{-^*} \in \mathbb{R}^n$, $z^* \in \mathbb{R}$ to the linear programming problem*

$$z \;\to\; \min_{x^+,\ x^-,\ z}\;, \tag{5}$$

$$\sum_{j=1}^{n} (\underline{a}_{ij} x_j^+ - \overline{a}_{ij} x_j^-) \;\ge\; \underline{b}_i - z\,|\underline{b}_i|, \qquad i = 1, 2, \ldots, n, \tag{6}$$

$$\sum_{j=1}^{n} (\overline{a}_{ij} x_j^+ - \underline{a}_{ij} x_j^-) \;\le\; \overline{b}_i + z\,|\overline{b}_i|, \qquad i = 1, 2, \ldots, n, \tag{7}$$

$$x_j^+, x_j^-, z \;\ge\; 0, \qquad j = 1, 2, \ldots, n. \tag{8}$$

In addition, the vector $x^* = x^{+^*} - x^{-^*}$ is the best possible pseudo-solution to $\boldsymbol{A}x = \boldsymbol{b}$.

Proof: First, we prove the existence of the optimal solution x^{+^*} and $x^{-^*} \in \mathbb{R}^n$, $z^* \in \mathbb{R}$ to the linear programming problem (5)–(8). From Theorem 2.1 and Rohn's theorem, it follows that the tolerable solution set of the considered problem is not empty. The corresponding dual problem is

$$\sum_{i=1}^{n} \underline{b}_i y_{1i} - \sum_{i=1}^{n} \overline{b}_i y_{2i} \quad \to \quad \max_{y_{1i}, y_{2i}}, \tag{9}$$

$$\sum_{i=1}^{n} \underline{a}_{ji} y_{1i} - \sum_{i=1}^{n} \overline{a}_{ji} y_{2i} \quad \leq \quad 0, \quad j = 1, 2, \ldots, n, \tag{10}$$

$$-\sum_{i=1}^{n} \overline{a}_{ji} y_{1i} + \sum_{i=1}^{n} \underline{a}_{ji} y_{2i} \quad \leq \quad 0, \quad j = 1, 2, \ldots, n, \tag{11}$$

$$\sum_{i=1}^{n} |\underline{b}_i| y_{1i} + \sum_{i=1}^{n} |\overline{b}_i| y_{2i} \quad \leq \quad 1, \tag{12}$$

$$y_{1i}, y_{2i} \quad \geq \quad 0, \quad i = 1, 2, \ldots, n. \tag{13}$$

The zero solution $y_{1i}, y_{2i} = 0,\ i = 1, 2, \ldots, n$ is a suitable solution to the problem (9)–(13). Hence, the primal linear programming problem (5)–(8) and the dual (9)–(13) are solvable. Therefore, the problem (5)–(8) really has optimum. ■

It is readily seen that the above results, both Theorems 2.1 and 3.1, can be generalized to the case when we widen the right-hand-side in the form

$$\big([\underline{b}_1 - p_1 z, \overline{b}_1 + q_1 z], [\underline{b}_2 - p_2 z, \overline{b}_2 + q_2 z], \ldots, [\underline{b}_n - p_n z, \overline{b}_n + q_n z]\big)^\top,$$

that is, with real positive factors p_i and q_i, instead of $|\underline{b}_i|$ and $|\overline{b}_i|$. This gives rise to a new interpretation of the above results and a new concept of pseudo-solutions closely related to that introduced in the preceding section. At the same time, one can ask a natural question on how to choose these factors in an optimal manner. We will consider these issues in our future research.

Both the problem (5)–(8) and the problem (9)–(13) are nearly degenerate. Therefore, an implementation of the numerical solution requires special attention. First, it is necessary to provide high precision of the calculations to prevent the simplex-method from cycling. Such anti-cycling techniques for the simplex-method are described, e.g., [16]. Second, calculations should be sufficiently fast to solve the problems within comfortable time limits. Third, the linear programming problem under solution should be decomposed to exploit modern multicore/multithread processors and the capabilities of cluster architectures.

4 Necessary Precision of Computation

The software tools necessary for exact computation have been developed earlier in South Ural State University (Chelyabinsk, Russia) as a C++ class library "Exact Computation 2.0" [4] based on reimplemented `overlong` and `rational` classes [4, 13, 14]. They prove to be almost as useful as the standard C++ data types, but without their range and precision limitations.

The objects of the class `rational` are common fractions p/q, where p and q are `overlong` integer numbers. The class `overlong` provides integer number from the range $]-2^{2^{32}}, 2^{2^{32}}[$ for 32-bit operating systems, or from the range $]-2^{2^{64}}, 2^{2^{64}}[$ for 64-bit operating systems, while the minimal "sampling intervals" of the `rational` objects are $2^{-2^{32}-1}$ and $2^{-2^{64}-1}$, respectively. The data types provides all the basic arithmetic operations and relations as C++ standard data types do. Such computation functionality is provided by the GMP library, an open source project that consists of C-types and many mathematical functions developed to work with high precision numbers. The GMP library is optimized for many modern processors using assembler code blocks [24], but GMP does **not** provide its types with the opportunity to send/receive its objects over communication networks in a distributed computing environment, e.g., a cluster. The library "Exact Computation 2.0" **provides** such functionality.

Internal form of the integer numbers inside the `overlong` object is a numerical notation with base equal to 2^{32}. This base requires some overhead to input/output numbers to/from inner notation, but inner calculations may be well optimized using standard C++ tools. Memory operations are optimized too, since there is no automatic garbage collector in the C++ runtime environment: frequent memory reallocations may lead to its fragmentation and to decreased application performance.

A brief review of the current version of the `overlong` classes and `rational` is given in [3]. Technical details of all the memory operations are encapsulated in special memory handle class `MemHandle`. On the other hand, all the basic arithmetic operations are encapsulated in the class `ArifRealization` (see Listing 1).

```
class overlong {
    private: static ArifRealization realization;
    private: MemHandle mhandle;
      . . .
    public: inline int32 size() const {return leng;}//length
    public: inline int32 sign() const {return sgn;} //sign
      . . .
    //addition
    template<typename Type>friend const overlong operator+
        (const overlong &num,Type v)
               {overlong rez(num); return (rez+=v);}
    friend const overlong operator+
                 (const overlong&,const overlong&);
      . . .
}
```

Listing 1: Fragment of the `overlong` class

An `overlong` object contains a `MemHandle` object for memory handling and `static ArifRealization`, a link to basic arithmetic operation realization. All the basic arithmetic operation are implemented as corresponding methods of the `ArifRealization` class. A sample code for the addition operation is demonstrated in the listing, see Figure 2. These techniques abstract the actual storage locations of the numbers and the number presentation notation by using the interface of `MemHandle` for memory handling calls and the interface of `ArifRealization` for arithmetic operations calls.

4.1 Necessary Productivity Supply

The decomposition "interface–memory–arithmetic" described above allows flexible usage of the computer system resources. Productivity of the simplex-method algorithm is based on the efficiency of the exact calculations. The basic class `overlong` has the

```
void overlong::add(const overlong &alpha, const overlong& beta){
    d_t carry;
    const overlong& a=(alpha.size()>=beta.size())? alpha:beta;
    const overlong& b=(alpha.size()>=beta.size())? beta:alpha;
    int32 LA=a.size(),LB=b.size(),sg=alpha.sgn,newleng;//LA>=LB
    ArifRealization::add(a.mhandle.getptr(), LA,
                         b.mhandle.getptr(), LB,
                         mhandle.providetmpptr(LA,1), newleng, carry);
    mhandle.settmpasptr();
    if(carry) mhandle.safesetvalue(LA, carry);
    leng=newleng;
    sgn=sg;
}
```

Listing 2: Addition operation

ability to use modern parallel accelerators such as Nvidia(R) GPUs using CUDA C [23] or other accelerators including AMD GPUs, Intel Xeon Phi using OpenCL [26].

Algorithms for parallel execution of the basic arithmetic operations and their implementation in heterogeneous computer environment is described e.g., [3, 12]. Storage optimization depends on the device where calculation are performed. For CPU calculations, operating memory of the compute node is the best choice, while GPU calculations use global memory (see [26] for an explanation of the term) of the device.

If a computer system contains an Nvidia(R) GPU, then all the data corresponding to digits of the numbers are stored in the global memory of the device which performs calculations with it, leading to substantial reduction of data transfer over the PCI bus.

4.2 Fine-grained Parallelism

Parallel computations on the GPU require a revision of the basic arithmetic operations in accordance to the device architecture. In the rest of the paper, we assume an Nvidia(R) GPU accelerator as one of the most popular GPGPU devices. A discussion of some operations in CUDA C follows.

4.2.1 Addition

Addition on the GPU is performed in several stages: parallel bitwise addition, synchronization, and parallel carry propagation from all digit positions. A feature of the GPUs groups all parallel threads into blocks. Threads of the block are fully concurrent, but there are no tools to synchronize threads from different blocks. Hence, it is necessary to store "border" carries into a temporary array (see Listing 3), and propagate them later (see Listing 4). The average length of the carry chains is not more than two digits, but in the worst case, the carry length is equal to the maximum length of the summands.

The code for setup of the kernel parameters and initiation of the kernel execution are given in the Listing 5. This code is executed on the CPU, also called *host side* of the heterogeneous CPU+GPU computer system.

```
__global__ void DNumAdd_part1 (d_t *A,int32 LA, d_t *B, int32 LB,
                               d_t *C, d_t *bGCarry, int32 *f) {
     int32 gId=blockDim.x*blockIdx.x + threadIdx.x;
     int64 tmp=0;
     if(gId >= LA) return; // bound check
     if(gId >= LB) C[gId]=A[gId];
     else {
         tmp=(int64)A[gId]+(int64)B[gId];
         C[gId]=tmp&MAX_DIGIT;
     }
     __syncthreads();  // carry propagation in the block
     int32 lId=threadIdx.x+1,i=gId+1, gS=blockDim.x;
     for(tmp>>=BIT_IN_DIGIT; tmp && i<LA; lId++, i++) {
         if(lId == gS) {
             bGCarry[blockIdx.x]=tmp; *f=1;
             return;
         }
         tmp+=(int64)C[i];
         C[i]=tmp&MAX_DIGIT;
         tmp>>=BIT_IN_DIGIT;
     }
    if(i==LA && tmp) bGCarry[Lcarry]=1;
}
```

Listing 3: Bitwise addition

```
__global__ void DNumAdd_part2
        (d_t *C, d_t *bGCarry, int32 gS, uint32 Lcarry, uint32 LA) {
     // carry propagation between blocks
     int gId = blockDim.x*blockIdx.x + threadIdx.x,i=(gId+1)*gS;
     if (gId >= Lcarry) return;
     uint64 tmp=(uint64)bGCarry[gId];
     for(; tmp && i<LA; i++) {
         tmp+=(uint64)C[i];
         C[i]=tmp&MAX_DIGIT;
         tmp>>=BIT_IN_DIGIT;
     }
     if (i==LA && tmp) bGCarry[Lcarry]=1;
}
```

Listing 4: Parallel carry propagation

```
void ArifRealization::add(const d_t *A,int32 LA,const d_t *B,int32 LB,
                          d_t *C, int32 &NL,d_t &Carry) {
     int tPerBlock = 128,bPerGrid = (LA + tPerBlock - 1) / tPerBlock;
     d_t *bGCarry_d=NULL,*ansCarry_h=new d_t[1];
     int32 *F_d=NULL,*F_h=new int32[1];
     cudaMalloc((void**)&bGCarry_d, sizeof(d_t)*(bPerGrid+1));
     cudaMemset((void*)bGCarry_d, 0, sizeof(d_t)*(bPerGrid+1));
     cudaMalloc((void**)&F_d, sizeof(int32));
     cudaMemset((void*)cF_d, 0, sizeof(int32));
     DNumAdd_part1 <<< bPerGrid, tPerBlock >>>
          (const_cast<d_t*>(A), LA, const_cast<d_t*>(B), LB,
                          C, bGCarry_d, CF_d);
     cudaMemcpy(cF_h, cF_d, sizeof(int32), cudaMemcpyDeviceToHost);
     if(*cF_h) {
        int gS = tPerBlock, LCarry = bPerGrid;
        bPerGrid = (bPerGrid + tPerBlock - 1) / tPerBlock;
        DNumAdd_part2 <<< bPerGrid, tPerBlock>>>
                (d_buffC, bGC_d, gS, LCarry, LA);
     }
     cudaMemcpy(ansCarry_h, &bGCarry_d[bPerGrid],
                sizeof(d_t), cudaMemcpyDeviceToHost);
     NL= (carry = *ansCarry_h)? LA+1: LA;
     delete[] ansCarry_h; cudaFree(bGCarry_d);  cudaFree(cF_d);
}
```

Listing 5: Kernel call at the GPU

4.2.2 Multiplication

The multiplication is one of the most time-consuming operations. The multiplication algorithm uses fast on-chip memory shared between threads inside the block of threads to accelerate temporary calculations. Its implementation essentially uses an Nvidia(R) GPU architecture feature that all the instructions inside one warp of the threads (16/32 threads) are performed simultaneously, freeing threads inside a warp from extra synchronization. The source code of the device side (GPU) is given in the Listing 6. The final result is formed sequentially by transforming 64-bit integers `rez[j]` to 32-bit digits and a further 32-bit carry digit.

4.2.3 Division

The fully sequential nature of the standard long division algorithm makes it a poor choice for parallel GPU architecture. We use more efficient iterative methods, e.g., [12].

5 Coarse-Grained Parallelism

Details of the parallel implementation techniques for the simplex method can be found in [11]. Threads on the CPU may be produced by any available tool, for example, by MPI, OpenMP, or std::threads of C++ 0x11.

```
__global__ void DNumMult(d_t *A, int32 LA, d_t *B, int32 LB, d_t *rez) {
     int32 lId=threadIdx.x,gId=blockDim.x*blockIdx.x + lId;
     if(gId >= LB) return;
     int32 cBS=(LA+blockDim.x-1) / blockDim.x;
     __shared__ uint64 sha[],shrez[];
     for(int i=lId*cBS; i<(lId+1)*cBS && i<LA; i++) {
         sha[i]=A[i]; shrez[i]=0;
     }
     shrez[LA+lId]=0;
     uint64 digit=(uint64)B[gId], t=0UL;
     for(int i=0; i<LA; i++) {
         t+=sha[i]*digit;
         shrez[i+lId]+=t&MAX_DIGIT;
         t>>=BIT_IN_DIGIT;
     }
     shrez[LA+lId]+=t&MAX_DIGIT;
     cBS = (LA+blockDim.x+blockDim.x-1) / blockDim.x;
     for(int i=lId*cBS; i<(lId+1)*cBS && i<LA+blockDim.x; i++) {
         AtomicAdd(rez[i+gId],shrez[i]);
     }
}
```

Listing 6: Multiplication on the GPU

5.1 A Computational Experiment

A computational test has been performed on a computer with an Intel(R) Core(R) i7-950 processor at 3.06GHz with 6 GB RAM and an Nvidia(R) GTX-460 with 1Gb GDDR5. The code was compiled with the Visual C++ 2011 compiler. The test problem was to solve a system of interval linear equations $\boldsymbol{A}x = \boldsymbol{b}$ with an "intervalized" Hilbert matrix $\boldsymbol{A} = (\boldsymbol{a}_{ij})$ and point (non-interval) right-hand side $\boldsymbol{b}$ such that

$$\begin{aligned} \boldsymbol{a}_{ij} &= \left[\frac{i(1-\delta)}{i+j-1}, \frac{i(1+\delta)}{i+j-1} \right], \\ \boldsymbol{b} &= (1, 1/2, \dots, 1/(n-1), 1/n)^{\top}. \end{aligned}$$

The minimal extension z^* of the right-hand side corresponding to the pseudo-solution depending on the parameter δ with fixed $n = 20$ is given in Table 1. Table 2 contains the results for various sizes of the model problem with fixed parameter δ.

Table 1: Minimal values of the right-hand side extension parameter

δ	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
z^*	0.81	0.389	0.1	0.025	0.0062	0.0017

Table 2: Computation time

Matrix (n)	10	20	50	100
Time	0.46 s	7.73 s	7.39 m	15.1 h

6 Conclusion

The paper presents the concept of a pseudo-solution to an interval linear algebraic system of equations, a new and promising regularization approach for sensitive and ill-conditioned problems. We prove that pseudo-solutions exist and develop a numerical approach for their computation based on the solution of a related linear programming problem. Since the new approach may produce linear programming problems that are very close to degenerate ones, we require special means for their numerical solution since the standard data types cannot cope with the accuracy losses.

Software that implements the new approach is developed for CUDA C. It is based on errorless rational-fractional computations and small-grained parallelism, which enables sufficiently high performance. The simplex method, coupled with errorless rational-fractional computation, gives an effective solution to the linear programming problem that arises in the new approach. Coarse-grained parallelism for distributed computer systems with MPI is the suitable instrument for their implementation.

References

[1] L.T. Ashhepkov and D.V. Davydov, *Universal Solutions of Interval Optimization and Control Problems,* Moscow, Nauka, 2006. (in Russian)

[2] M. Fiedler, J. Nedoma, J. Ramik, J. Rohn, and K. Zimmermann. *Linear Optimization Problems with Inexact Data.* New York, Springer, 2006.

[3] V.A. Golodov, Distibuted symbolic fractional-rational calculations with the x86 and x64 processors. In *Parallel Computational Technologies (PaCT'2012): Proceedings of International Scientific Conference, Novosibirsk, March 26–30, 2012.* Chelyabinsk, SUSU Publishing Office, page 719. (in Russian)

[4] V.A. Golodov and A.V. Panyukov, Library of classes "Exact Computation 2.0". State. reg. 201361818, March 14, 2013. *Official Bulletin of Russian Agency for Patents and Trademarks*, Federal Service for Intellectual Property, 2013, No. 2. Series "Programs for Computers, Databases, Topology of VLSI". (in Russian)

[5] V.K. Ivanov. About linear ill-conditioned problems, *Doklady AN SSSR*, vol. 145 (2) (1962), pp. 270–272. (in Russian)

[6] R.B. Kearfott, M.T. Nakao, A. Neumaier, S.M. Rump, S.P. Shary, and P. van Hentenryck. Standardized notation in interval analysis, *Computational Technologies*, vol. 15 (2010) No. 1, pp. 7–13. Available at `http://www.ict.nsc.ru/jct/getfile.php?id=1345`.

[7] A. Neumaier. *Interval Methods for Systems of Equations*, Cambridge, Cambridge University Press, Cambridge, 1990.

[8] A.V. Panyukov and E.S. Chechulina, Pseudosolutions of systems of linear equations with interval uncertainty of coefficients (Psevdoresheniya sistem

lineynyh algebraicheskih uravneniy pri intervalnoy neopredelennosti koyefficientov.), In *Proceedings of XIII Baikal International School-Seminar "Optimization Methods and Their Applications", July 2–8, 2005, Irkutsk, Baikal*, vol. 4 "Interval Analysis", Irkutsk, Melentiev Energy Systems Institute SB RAS, 2005, pp. 62–65. (in Russian)

[9] A. V. Panyukov and E. S. Chechulina, Solutions of systems of linear equations with interval uncertainty of coefficients (Resheniya sistem lineynyh algebraicheskih uravneniy pri intervalnoy neopredelennosti koyefficientov.), In *"Algorithmic Analysis of Unstable Problems": Abstracts of the International Conference, Ekaterinburg, February 2 – 6, 2004*, Ekaterinburg, Publishing House of Ural University, 2004, pp. 291–292. (in Russian)

[10] A.V. Panyukov and V.A. Golodov, Calculation of pseudosolutions of systems of linear equations with interval uncertainty of coefficients. In *Sixth International Conference "Algorithmic Analysis of Unstable Problems", Ekaterinburg, Russia: October 31 – November 5, 2011*, Ekaterinburg, Institute of Mathematics and Mechanics, Ural Branch of the RAS, 2011, pp. 261–262. (in Russian)

[11] A.V. Panyukov and V.V. Gorbik, Parallel implementation of symplex-method for the exact solution of linear programming problems. *Vestnik of SUSU. Series: "Mathematical Modeling and Programming"*, vol. 9 (2011) No. 25 (242), pp. 107–118. (in Russian)

[12] A.V. Panyukov and S.Yu. Lesovoi, Implementation of basic arithmetic operations in heterogeneous systems. In *Parallel Computational Technologies (PCT'2012): Proceedings of International Scientific Conference, Novosibirsk, March 26 – 30, 2012*, Chelyabinsk, SUSU Publishing Office, 2012, pp. 634–637. (in Russian)

[13] A.V. Panyukov and M.M. Silaev, `overlong` class. Registration number 990489. In *Official Journal of Federal Service for Intellectual Property. Computer Programs, Databases, Integrated Circuit Patterns*, Moscow: FIPS, vol. 4 (1999), No. 29.

[14] A.V. Panyukov, M.M. Silaev, and M.I. Germanenko, `rational` class. Registration number 990607, In *Official Journal of Federal Service for Intellectual Property. Computer Programs, Databases, Integrated Circuit Patterns*, Moscow: FIPS, vol. 4 (1999), No. 29.

[15] J. Rohn, Inner solutions of linear interval systems, In *Interval Mathematics 1985*, Berlin-Heidelberg-New York-Tokyo, Springer Verlag, 1986, pp. 157–158.

[16] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley & Sons, Chichester, 1986.

[17] S.P. Shary, Solving the linear interval tolerance problem, *Mathematics and Computers in Simulation*, vol. 39 (1996), No. 1–2, pp. 53–85. DOI: 10.1016/0378-4754(95)00135-K.

[18] S.P. Shary, A new technique in systems analysis under interval uncertainty and ambiguity, *Reliable Computing*, vol. 8 (2002), No. 5, pp. 321–418.
Available at `http://www.nsc.ru/interval/shary/Papers/ANewTech.pdf`.

[19] S.P. Shary. *Finite-Dimensional Interval Analysis.* Electronic Book. Novosibirsk, XYZ, 2013. Available at `http://www.nsc.ru/interval/Library/InteBooks/SharyBook.pdf`. (in Russian)

[20] S.P. Shary, An interval linear tolerance problem, *Automation and Remote Control*, vol. 65 (2004), No. 10, pp. 1653–1666. DOI: 10.1023/B:AURC.0000044274.25098.da.

[21] P.I. Stetsyuk. Acceleration of Polyak's subgradient method. In *Theory of Optimal Decisions*, Kiev, V.M. Glushkov Institute of Cybernetics, 2012, No. 11, pp. 151–160.

[22] A.N. Tihonov and V.Ya. Arsenin, *Methods for the Solution of Ill-Posed Problems*, Moscow, Nauka, 1979. (in Russian)

[23] Parallel programming and computing platform — CUDA — Nvidia, Available at `https://developer.nvidia.com/category/zone/cuda-zone`.

[24] The GNU MP Bignum Library, Available at `http://gmplib.org/`.

[25] Interval analysis and its applications. (Intervalnyi analiz i ego prilozheniya). Web-portal `http://www.nsc.ru/interval`.

[26] OpenCL — The open standard for parallel programming on heterogeneous systems, Available at `http://www.khronos.org/opencl/` and `http://www.khronos.org/opencl/`.