Bergische Universität
Wuppertal

Parametric Fixed-Point Iteration Implemented in C-XSC

Evgenija D. Popova and Walter Krämer

Preprint 2003/3

Wissenschaftliches Rechnen/
Softwaretechnologie

This work was partially supported by DFG and the Bulgarian National Science Fund under grants No I-903/99 and MM-1301/2003.

Impressum

Herausgeber:	Prof. Dr. W. Krämer, Dr. W. Hofschuster Wissenschaftliches Rechnen/Softwaretechnologie Fachbereich 7 (Mathematik) Bergische Universität Wuppertal Gaußstr. 20 D-42097 Wuppertal

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über die World Wide Web Seiten

`http://www.math.uni-wuppertal.de/wrswt/literatur.html`

Autoren-Kontaktadresse

Evgenija D. Popova
Institute of Mathematics & Informatics
Bulgarian Academy of Sciences
Acad. G. Bonchev str., block 8
BG-1113 Sofia, Bulgaria

E-mail: `epopova@bio.bas.bg`

Walter Krämer
Bergische Universität Wuppertal
Gaußstr. 20
D-42097 Wuppertal

E-mail: `kraemer@math.uni-wuppertal.de`

Parametric Fixed-Point Iteration Implemented in C-XSC

Evgenija Popova, Walter Krämer

Abstract. Consider linear systems whose matrix and right-hand side vector depend affine-linearly on parameters varying within prescribed intervals. We present a C-XSC [9] implementation of a parametric fixed-point iteration method for the verified enclosure of the parametric solution set. Input data for the system have an entirely numerical representation by dense two-dimensional matrices. Some specific features of the corresponding algorithm concerning sharp enclosure of the contracting matrix, epsilon inflation, and inner approximation of the solution enclosure are discussed. Numerical examples illustrate the application of the presented software and the discussed specific features of the algorithm.

Our software seems to be the first open source software delivering verified results in the field of parametric linear systems (inner and outer estimations for the hull of the solution set). The source code is available at http://www.math.uni-wuppertal.de/wrswt/xsc/cxsc_software.html. The software will be developed further.

Keywords: parametric linear systems, validated interval software, C-XSC, inner estimation, open source.

MSC (2000): 65F10, 65G20, 65Y15

1 Introduction

Solving parametric linear systems involving uncertainties in the parameters is an important part of the solution to many scientific and engineering problems. In most engineering design problems, models in operation research, linear prediction problems, etc. usually there are complicated dependencies between coefficients [2], [17]. The main reason for this dependency is that the errors in several different coefficients may be caused by the same factor.

Consider the linear system

$$A(p) \cdot x \;=\; b(p) \tag{1}$$

where $A(p) \in I\!R^{n\times n}$ and $b(p) \in I\!R^n$ depend affine-linearly on a parameter vector $p \in I\!R^k$. When p varies within a range $[p] \in I I\!R^k$, the set of solutions to all $A(p) \cdot x = b(p)$, $p \in [p]$, called parametric solution set, is

$$\Sigma^p = \Sigma\left(A(p), b(p), [p]\right) \;\;:=\;\; \{x \in I\!R^n \mid A(p) \cdot x = b(p) \text{ for some } p \in [p]\}\,. \tag{2}$$

Since the solution set has a complicated structure which is difficult to find, one looks for the interval hull $\Diamond(\Sigma)$ whenever Σ is a nonempty bounded subset of $I\!R^n$. For $\Sigma \subseteq I\!R^n$, define $\Diamond : PI\!R^n \to II\!R^n$ by

$$\Diamond(\Sigma) := [\inf \Sigma,\ \sup \Sigma] \ = \ \cap\{[x] \in II\!R^n \mid \Sigma \subseteq [x]\}.$$

The calculation of $\Diamond(\Sigma)$ is also quite expensive, so a more realistic task is to find a tight interval enclosure of it. Here we discuss the computation of $[y] \in II\!R^n$ such that $[y] \supseteq \Diamond(\Sigma^p) \supseteq \Sigma^p$.

A general iteration method for verified enclosure of Σ^p, that accounts for arbitrary affine-linear dependencies in the matrix and the right hand side vector, is proposed by S. Rump in [21] and generalized in [15]. At present the parametric fixed-point iteration and the parametric Gauss-Seidel iteration [14] are the only general purpose methods for verified enclosure of the parametric solution set. The parametric fixed-point iteration possesses excellent performance whenever the parameters vary within relatively small tolerances [14], [21]. This method is also an indispensable tool for proving monotonicity properties of the parametric solution set [16] and for the parametric Gauss-Seidel iteration in finding an initial approximation of the parametric solution set.

Despite of the many attempts (mainly from an application point of view) to treat the parametric problem, there is a lack of a suitable software basis for comparison of different methods and studying their efficiency. The parametric fixed-point iteration has only a few applications [14], [17] and a *Mathematica* implementation [16]. The application scientists neither apply, nor compare their own methods to the parametric fixed-point solver. Remarkably, the method was recently reinvented in slightly different notations [2]. Since various practical problems involve solution of parameter dependent linear systems, there is an increasing interest in rigorous efficient methods and software tools for solving such systems.

The goal of this work is to provide a free, open-source software for the verified enclosure of the parametric solution set in the environment of C-XSC [8], [9]. The software tool, we describe here, implements a fixed-point iteration method for parametric linear systems, and has our expertise and experience built in. In Section 2 we recall Rump's parametric method and its generalization. In Section 3 we present an algorithm which uses a two-dimensional numerical representation of the parametric system. Some specific features of the corresponding C-XSC implementation concerning a sharp enclosure of the iteration matrix, epsilon inflation, and inner approximation of the outer enclosure are discussed. Numerical examples, given in Section 4, illustrate the application of the presented software and the discussed specific features.

Because it is free and open-source, this software could be used as a benchmark for the next methodology and performance investigations related to parametric interval linear systems.

We use the following notations. $I\!R^n, I\!R^{n\times m}$ denote the set of real vectors with n components and the set of real $n \times m$ matrices, respectively. By normal (*proper*) interval we mean a real compact interval

$$[a] := [a^-, a^+] := \{a \in I\!R \mid a^- \le a \le a^+\}.$$

By $I\,I\!R^n, I\,I\!R^{n\times m}$ we denote interval n-vectors and interval $n \times m$ matrices. The end-point functionals $(\cdot)^-, (\cdot)^+$, the mid-point function $\mathrm{mid}(\cdot)$, where $\mathrm{mid}([a^-, a^+]) := \dfrac{a^- + a^+}{2}$, and the diameter function $\mathrm{diam}(\cdot)$, where $\mathrm{diam}([a^-, a^+]) := a^+ - a^-$, are applied to interval vectors and matrices componentwise. $\varrho(A)$ is the spectral radius of a matrix A.

Denote by

$$A([p]) := \Diamond\{A(p) \in I\!R^{n\times n} \mid p \in [p]\}, \qquad b([p]) := \Diamond\{b(p) \in I\!R^{n} \mid p \in [p]\}$$

the non-parametric interval matrix, resp. vector, that correspond and are obtained from the parametric ones. Hence, $A([p]) \cdot x = b([p])$ is the non-parametric system corresponding to the parametric one, and

$$\Sigma^g = \Sigma\left(A([p]), b([p])\right) \;:=\; \{x \in I\!R^n \mid A \cdot x = b \text{ for some } A \in A([p]), b \in b([p])\}$$

is the non-parametric solution set corresponding to the parametric one. The parametric solution set is a subset of the corresponding non-parametric solution set and has much smaller volume than the latter.

2 Theoretical Background

In this section we give a brief summary of the theory of the enclosure methods for our problem. A more detailed presentation can be found in [21] and [15].

Consider the parametric linear system (1). $A(p) \in I\!R^{n\times n}$ and $b(p) \in I\!R^n$ depend affine-linearly on a parameter vector $p \in I\!R^{k+1}$, which first component is equal to one, means that, with the notations of Rump in [21], there are vectors

$$\begin{array}{l} w(i,j) \in I\!R^{k+1} \quad \text{for} \quad 0 \le i \le n,\ 1 \le j \le n \quad \text{with} \\ \{A(p)\}_{ij} = w(i,j)^\top \cdot p \quad \text{and} \quad \{b(p)\}_j = w(0,j)^\top \cdot p. \end{array} \tag{3}$$

This way $A(p)$ can be represented as a three dimensional matrix

$$\begin{pmatrix} w(1,1) & \cdots & w(1,n) \\ & \cdots & \\ w(n,1) & \cdots & w(n,n) \end{pmatrix} \in I\!R^{n\times n\times(k+1)}.$$

In order to avoid the three dimensional numeric representation of the parametric matrix we shall use another equivalent representation which will facilitate the algorithmic formulation. Each individual component of $A(p)$, $b(p)$ is an affine-linear combination of the k parameters

$$a_{ij}(p) := a_{ij}^{(0)} + \sum_{\nu=1}^{k} p_\nu a_{ij}^{(\nu)}, \qquad b_i(p) := b_i^{(0)} + \sum_{\nu=1}^{k} p_\nu b_i^{(\nu)}. \tag{4}$$

Denote the $k+1$ numerical matrices

$$A^{(0)} := \left(a_{ij}^{(0)}\right),\ A^{(1)} := \left(a_{ij}^{(1)}\right), \ldots, A^{(k)} := \left(a_{ij}^{(k)}\right) \in I\!R^{n\times n}$$

and the corresponding numerical vectors $b^{(0)} := (b_i^{(0)}), b^{(1)} := (b_i^{(1)}), \ldots, b^{(k)} := (b_i^{(k)}) \in I\!R^n$. Hence, the parametric matrix and the right-hand side vector can be represented by

$$A(p) = A^{(0)} + \sum_{\nu=1}^{k} p_\nu A^{(\nu)}, \qquad b(p) = b^{(0)} + \sum_{\nu=1}^{k} p_\nu b^{(\nu)}. \tag{5}$$

and the parametric system (1) can be rewritten in the following form

$$\left(A^{(0)} + \sum_{\nu=1}^{k} p_\nu A^{(\nu)}\right) x \;=\; b^{(0)} + \sum_{\nu=1}^{k} p_\nu b^{(\nu)}, \tag{6}$$

where the parameter vector p varies within the range $[p] \in I\mathbb{R}^k$.

2.1 Existence and Uniqueness of the Solution

Rather than computing an enclosure for the parametric solution directly, we will try to enclose the difference $Y := X - \tilde{x}$ w.r.t. an approximate solution $\tilde{x}$. For an arbitrary nonsingular matrix $R \in \mathbb{R}^{n\times n}$ and arbitrary $\tilde{x} \in \mathbb{R}^n$, (1) is equivalent to the fixed-point equation

$$Y = R\,(b(p) - A(p)\cdot\tilde{x}) + (I - R\cdot A(p))\cdot Y,$$

where $I \in \mathbb{R}^{n\times n}$ denotes the identity matrix. Rump's parametric fixed-point method for enclosing the solution of (1) is contained in the following theorem.

Theorem 1 ([21]) *Let $A(p)\cdot x = b(p)$ with $A(p) \in \mathbb{R}^{n\times n}$, $b(p) \in \mathbb{R}^n$, $p \in \mathbb{R}^{k+1}$ with $p \in [p] \in I\,\mathbb{R}^{k+1}$ be a parametrised linear system, where $A(p), b(p)$ are given by (3). Let $R \in \mathbb{R}^{n\times n}$, $[Y] \in I\mathbb{R}^n$, $\tilde{x} \in \mathbb{R}^n$ and define $[Z] \in I\mathbb{R}^n$, $[C] \in I\mathbb{R}^{n\times n}$ by*

$$[Z]_i \;:=\; \Big(\sum_{j,\nu=1}^{n} \{R_{ij}\,(w(0,j) - \tilde{x}_\nu w(j,\nu))\}^\top\Big)\cdot[p], \tag{7}$$

$$[C] \;:=\; I - R\cdot A([p]). \tag{8}$$

Define $[V] \in I\mathbb{R}^n$ by means of the following Einzelschrittverfahren

$$1 \le i \le n \;:\; V_i := \{[Z] + [C]\cdot[U]\}_i, \quad [U] := (V_1, ..., V_{i-1}, Y_i, ..., Y_n)^\top. \tag{9}$$

If $[V] \subsetneqq [Y]$, then R and every matrix $A(p), p \in [p]$ are regular, and for every $p \in [p]$ the unique solution $\widehat{x} = A^{-1}(p)b(p)$ of (1) satisfies $\widehat{x} \in \tilde{x} + [V]$.

With $[D] := [C]\cdot[V] \in I\,\mathbb{R}^n$ and the solution set Σ^p, defined by (2), the following inner estimation holds true

$$\left[\tilde{x} + [Z]^- + [D]^+,\; \tilde{x} + [Z]^+ + [D]^-\right] \subseteq \left[\inf(\Sigma^p),\; \sup(\Sigma^p)\right]. \tag{10}$$

The first part of Theorem 1 establishes existence and uniqueness of the solution. The existence follows from Brower's fixed-point theorem for $g(p,x) := x - R\cdot f(p,x)$, where $f(p,x) := A(p)x - b(p)$ has the expansion $f(p,x) = f(p,\tilde{x}) + A(p)\cdot(x-\tilde{x})$. The important point in obtaining an enclosure of the *parametric* solution set is to obtain sharp bounds for

$$Z \;:=\; -R\cdot f(p,\tilde{x};[p]) := \diamond\,(R\cdot\{b(p) - A(p)\cdot\tilde{x} \mid p \in [p]\})$$

because a straightforward evaluation $R \cdot (b([p]) - A([p])\tilde{x})$ causes overestimation. $[Z]$, defined in (7), provides a sharp estimation. Next, with the notations (5), we give another equivalent representation of (7)

$$\begin{aligned}
[Z] &:= \Diamond\left\{R\,(b(p) - A(p)\tilde{x}) \mid p \in [p]\right\} \\
&= \Diamond\left\{R\left(b^{(0)} + \sum_{\nu=1}^{k} p_\nu b^{(\nu)} - \left(A^{(0)} + \sum_{\nu=1}^{k} p_\nu A^{(\nu)}\right)\tilde{x}\right) \mid p \in [p]\right\} \\
&= \Diamond\left\{R(b^{(0)} - A^{(0)}\tilde{x}) + \sum_{\nu=1}^{k} \left(p_\nu(Rb^{(\nu)} - RA^{(\nu)}\tilde{x})\right) \mid p \in [p]\right\} \\
&= R \cdot (b^{(0)} - A^{(0)}\tilde{x}) + \sum_{\nu=1}^{k} [p_\nu](R \cdot b^{(\nu)} - R \cdot A^{(\nu)} \cdot \tilde{x}).
\end{aligned}$$

As it is proven in [21], the inclusion $[V] \subsetneqq [Y]$ together with (7)–(9) implies $\varrho\,(|[C]|) < 1$, consequently non-singularity of R and every $A(p)$, $p \in [p]$, thus the uniqueness of the solution of (1). However, it is shown in [15] that for some parametric matrices verifying $\varrho\,(|[C]|) < 1$ fails, while R and every $A(p)$, $p \in [p]$, are regular. It is proven therein that to have a better sufficient condition for the regularity of every $A(p)$, $p \in [p]$, one has to compute instead of (8)

$$\begin{aligned}
[C(p)] &:= \Diamond\{I - R \cdot A(p) \mid p \in [p]\} \\
&= I - R \cdot A^{(0)} - \sum_{\nu=1}^{k} [p_\nu](R \cdot A^{(\nu)}).
\end{aligned}$$

This way, the following theorem expands the scope of applicability of the parametric fixed-point iteration.

Theorem 2 ([15]) *Let $A(p) \cdot x = b(p)$ with $A(p) \in I\!R^{n\times n}$, $b(p) \in I\!R^n$, $p \in I\!R^k$ be a parametrised linear system, where $A(p), b(p)$ are given by (5). Let $R \in I\!R^{n\times n}$, $[Y] \in I\,I\!R^n$, $\tilde{x} \in I\!R^n$ and define $[Z] \in I\,I\!R^n$, $[C(p)] \in I\,I\!R^{n\times n}$ by*

$$[Z] := R \cdot (b^{(0)} - A^{(0)}\tilde{x}) + \sum_{\nu=1}^{k} [p_\nu](R \cdot b^{(\nu)} - R \cdot A^{(\nu)} \cdot \tilde{x}), \tag{11}$$

$$[C(p)] := I - R \cdot A^{(0)} - \sum_{\nu=1}^{k} [p_\nu](R \cdot A^{(\nu)}). \tag{12}$$

Define $[V] \in I\,I\!R^n$ by means of the following Einzelschrittverfahren

$$1 \le i \le n \;:\; V_i := \{[Z] + [C(p)] \cdot [U]\}_i, \quad [U] := (V_1, ..., V_{i-1}, Y_i, ..., Y_n)^\top.$$

If $[V] \subsetneqq [Y]$, then R and every matrix $A(p), p \in [p]$ are regular, and for every $p \in [p]$ the unique solution $\widehat{x} = A^{-1}(p)b(p)$ of (1) satisfies $\widehat{x} \in \tilde{x} + [V]$.

With $[D] := [C(p)] \cdot [V] \in I\,I\!R^n$ and the solution set Σ^p, defined by (2), the following inner estimation holds true

$$\left[\tilde{x} + [Z]^- + [D]^+,\ \tilde{x} + [Z]^+ + [D]^-\right] \subseteq \left[\inf(\Sigma^p),\ \sup(\Sigma^p)\right].$$

In the implementation of Theorem 2, resp. Theorem 1 we choose $R \approx A^{-1}(p_m)$ and $\tilde{x} \approx A^{-1}(p_m) \cdot b(p_m)$, where $p_m = \text{mid}([p])$. To force $[V] \subsetneqq [Y]$, the concept of ε-*inflation* is introduced. Epsilon inflation blows up the intervals somewhat in order to "catch" a nearby fixed-point. For a real interval $[w]$, we denote ε-*inflation* with the functional notation $\text{blow}([w], \varepsilon)$, where

$$\text{blow}([w], \varepsilon) = \begin{cases} [w] + \text{diam}([w])[-\varepsilon,\ \varepsilon], & \text{if } \text{diam}([w]) > 0 \\ [\text{pred}(w),\ \text{succ}(w)], & \text{if } \text{diam}([w]) = 0, \end{cases}$$

where $\text{pred}(w)$, $\text{succ}(w)$ are the predecessor and successor of a floating-point number w in the floating-point screen. The ε-*inflation* is applied to interval vectors componentwise. It is a numerical experience that the method of "trial and error" yields to $[V] \subsetneqq [Y]$ in the algorithm after a few iterations only, provided ε is chosen appropriately. Usually, $\varepsilon = 0.1$ or $\varepsilon = 0.2$ are reasonable values. For interval linear systems one may vary the value of ε in order to get tighter solution enclosures (see the examples in Section 4).

2.2 Inner Approximation of the Hull of the Solution Set

For linear systems with parameters varying within certain tolerances there is a whole set of solutions (2) and we compute an outer enclosure, i.e. an interval vector which is verified to contain the exact hull of the parametric solution set, by the first part of Theorem 2, resp. Theorem 1. However, it is important to know how much this inclusion overestimates the exact hull of the parametric solution set, in other words: what is the quality of the outer enclosure. The amount of overestimation can be estimated by an inner inclusion which is a componentwise inner estimation of the exact hull [20]. $[x] \in I\mathbb{R}^n$ is called componentwise inner approximation for some set $\Sigma \in \mathbb{R}^n$ if

$$\inf_{\sigma\in\Sigma} \sigma_i \leq x_i^- \quad \text{and} \quad x_i^+ \leq \sup_{\sigma\in\Sigma} \sigma_i, \qquad \text{for every } 1 \leq i \leq n.$$

It should be noted that $[x] \subseteq [\inf(\Sigma), \sup(\Sigma)]$ but $[x] \nsubseteq \Sigma$. The second part of Theorem 1, resp. Theorem 2, establishes how to compute the componentwise inner estimation. In order to have a guaranteed inner inclusion all the computations should be done in computer arithmetic with directed roundings.

Let $\mathbb{F} \subset \mathbb{R}$ denote the set of floating-point numbers on a computer (floating-point screen). Denote by $\triangledown, \triangle : \mathbb{R} \longrightarrow \mathbb{F}$ the floating-point directed roundings toward $-\infty$, resp. $+\infty$, defined by the IEEE floating-point standard [1], [11]. For intervals $[a] = [a^-, a^+] \in I\mathbb{R}$, outward ($\diamond$) and inward ($\bigcirc$) roundings $\diamond, \bigcirc : I\mathbb{R} \longrightarrow I\mathbb{F}$ are defined as

$$\diamond([a]) := [\triangledown(a^-),\ \triangle(a^+)] \supseteq [a], \qquad \bigcirc[a] := [\triangle(a^-),\ \triangledown(a^+)] \subseteq [a]. \tag{13}$$

If $\circ \in \{+, -, \times, /\}$ is an arithmetic operation and $[a], [b] \in I\mathbb{F}$, the corresponding computer operations $\diamond\!\!\!\circ\,, \odot : I\mathbb{F} \times I\mathbb{F} \longrightarrow I\mathbb{F}$ are defined by

$$[a] \diamond\!\!\!\circ [b] := \diamond([a] \circ [b]) = [\ \triangledown(([a] \circ [b])^-),\ \ \triangle(([a] \circ [b])^+)] \supseteq [a] \circ [b], \tag{14}$$

$$[a] \odot [b] := \bigcirc([a] \circ [b]) = [\ \triangle\ (([a] \circ [b])^-),\ \ \triangledown(([a] \circ [b])^+)] \subseteq [a] \circ [b]. \tag{15}$$

The next theorem follows from a Theorem by S. Rump [20] and specifies the computation of guaranteed outer and inner estimations for the hull of a parametric solution set on the computer.

Theorem 3 *Let a parametric linear system (1) be given with $p \in [p] \in I\mathbb{F}^k$. Let $R \in \mathbb{F}^{n\times n}$, $[Y] \in I\mathbb{F}^n$, $\tilde{x} \in \mathbb{F}^n$. $[Z] \in I\mathbb{R}^n$, $[C(p)] \in I\mathbb{R}^{n\times n}$ are defined by (11), resp. (12). Let $\bigcirc([Z]), \Diamond([Z]) \in I\mathbb{F}^n$ be the corresponding inward and outward inclusions of $[Z]$, $\bigcirc([Z]) \subseteq [Z] \subseteq \Diamond([Z])$, and $\Diamond([C(p)]) \in I\mathbb{F}^{n\times n}$ be an outward inclusion of $[C(p)]$. Define $[V] \in I\mathbb{F}^n$ by*[1]

$$1 \le i \le n \;:\; V_i := \{\Diamond([Z]) \mathbin{\Diamond\!\!\!\!+} \Diamond([C(p)]) \mathbin{\Diamond\!\!\!\!\cdot} [U]\}_i, \quad [U] := (V_1, ..., V_{i-1}, Y_i, ..., Y_n)^\top.$$

If $[V] \subsetneq [Y]$, then R and every matrix $A(p), p \in [p]$ are regular, and the solution set (2) satisfies $[\inf(\Sigma^p), \sup(\Sigma^p)] \subseteq \tilde{x} \mathbin{\Diamond\!\!\!\!+} [V]$.

With $[D] := \Diamond([C(p)]) \mathbin{\Diamond\!\!\!\!\cdot} [V] \in I\mathbb{F}^n$ the following inner estimation holds true

$$\left[\tilde{x} \mathbin{\triangle\!\!\!\!\!+} (\bigcirc[Z])^- \mathbin{\triangle\!\!\!\!\!+} ([D])^+,\ \ \tilde{x} \mathbin{\triangledown\!\!\!\!\!+} (\bigcirc[Z])^+ \mathbin{\triangledown\!\!\!\!\!+} ([D])^-\right] \subseteq [\inf(\Sigma^p),\ \sup(\Sigma^p)]\,. \tag{16}$$

Obtaining inner approximations on a computer in conventional interval arithmetic is possible only if the four interval arithmetic operations are implemented with inward rounding $\odot$ in addition to the four $\Diamond\!\!\!\!\cdot$ operations. The overloading concepts of some programming environments do not allow the operators to be distinguished by their result type, which imposes the implementation of inwardly rounded interval operations as functions or subroutines. Most of the wide-spread interval packages do not support inwardly rounded interval arithmetic.

Unfortunately, C-XSC also does not provide the necessary tools to compute $\bigcirc([Z])$. If we denote

$$[z^{(\nu)}] := \bigcirc\left(R(b^{(\nu)} - A^{(\nu)}\tilde{x})\right), \qquad \nu = 0, 1, \ldots, k,$$

then

$$\bigcirc([Z]) \;=\; [z^{(0)}] \oplus [p_1] \otimes [z^{(1)}] \;\oplus \ldots \oplus\; [p_k] \otimes [z^{(k)}].$$

This expression is of type interval scalar product expression and can be computed without overestimation by using an interval dotproduct accumulator applying only one final rounding. However, C-XSC does not support neither inward rounding of the interval dotproduct accumulators nor interval operations with inward rounding. Below we give an alternative computational technique based on the properties of an algebraic extension of the conventional interval arithmetic. Componentwise inner estimation of $\Diamond(\Sigma^p)$ in terms of the arithmetic of proper and improper intervals was first discussed in [14]. Here, we present a detailed implementation scheme based on the properties of the arithmetic on proper and improper intervals.

The set of proper intervals $I\mathbb{R}$ is extended in [10] by the set $\{[a^-, a^+] \mid a^-, a^+ \in \mathbb{R}, a^- \ge a^+\}$ of *improper* intervals obtaining thus the set $I\,\mathbb{R}^* = \{[a^-, a^+] \mid a^-, a^+ \in \mathbb{R}\}$ of all ordered couples of real numbers called here generalised intervals. Normal (proper) intervals are a special case of generalised intervals and the conventional interval arithmetic can be considered as a projection of generalised interval arithmetic on $I\,\mathbb{R}$. The conventional (arithmetic and lattice) operations, order relations and other functions are isomorphically extended onto the whole set of proper and improper intervals [10]. The same is done for the inward and outward roundings so that formulae (13)–(15) are valid for generalised intervals, too. Here we present only those basic facts from generalised interval arithmetic which are necessary to use it as an intermediate computational tool for handling proper interval problems. For more details on

[1] $\Diamond\!\!\!\!\cdot$ denotes a dot product computed with outward rounding.

the theory, implementation and applications of generalised interval arithmetic consult [3], [10], [18], [22].

"$Dual$" is an important monadic operator that reverses the end-points of the intervals and expresses an element-to-element symmetry between proper and improper intervals in $I\,I\!R^*$. For $[a] = [a^-, a^+] \in I\,I\!R^*$, its dual is defined by $Dual([a]) = [a^+, a^-]$. $Dual$ is applied componentwise to vectors and matrices. For $[a], [b] \in I\!I\!R^*$ and $\circ \in \{+, -, \times, /\}$,

$$\begin{aligned} Dual(Dual([a])) &= [a], & (17) \\ Dual([a] \circ [b]) &= Dual([a]) \circ Dual([b]). & (18) \end{aligned}$$

The following properties of the rounded generalised interval arithmetic are of major importance for obtaining inner numerical approximations at no additional cost and show that the latter can be obtained only by means of outward directed rounding and the $Dual$ operator in $I\!F^*$ [3].

$$\text{For } [a] \in I\!R^*, \qquad \bigcirc([a]) = Dual(\Diamond(Dual([a]))). \qquad (19)$$

$$\text{For } [a], [b] \in I\!F^*,\ \circ \in \{+, -, \times, /\}, \qquad [a] \odot [b] = Dual(Dual([a]) \mathbin{\Diamond} Dual([b])). \qquad (20)$$

We will apply the above properties to obtain inner estimations of proper interval problems in a computing environment not supporting generalised interval arithmetic.

For a point interval $[a] = [a, a] \in I\!R$

$$[\triangle(a),\ \bigtriangledown(a)] = \bigcirc([a]) \subseteq [a] \subseteq \Diamond([a]) = [\bigtriangledown(a),\ \triangle(a)].$$

If $a \notin I\!F$, $\bigcirc([a]) \in I\!F^*$ is an improper interval since $\triangle(a) > \bigtriangledown(a)$. The following inclusion relation holds true in $I\!R^*$

$$\bigcirc([a]) \subseteq [a] \subseteq \Diamond([a]).$$

It can be easily verified that

$$\text{for } [a] = [a, a], \qquad \bigcirc([a]) = Dual\left(\Diamond([a])\right). \qquad (21)$$

Now, we consider the computation of $\bigcirc([Z])$. With the notations of Theorem 2, let

$$z^{(\nu)} := R(b^{(\nu)} - A^{(\nu)}\tilde{x}), \qquad \nu = 0, 1, \ldots, k.$$

Then $[Z] = z^{(0)} + \sum_{\nu=1}^{k} [p_\nu] z^{(\nu)}$. Using the well-known inclusion properties of interval arithmetic we obtain

$$[Z] = z^{(0)} + \sum_{\nu=1}^{k} [p_\nu] z^{(\nu)} \subseteq \Diamond\left(\Diamond(z^{(0)}) + \sum_{\nu=1}^{k} [p_\nu] \times \Diamond(z^{(\nu)})\right),$$

and

$$\bigcirc\left(\bigcirc(z^{(0)}) + \sum_{\nu=1}^{k} [p_\nu] \times \bigcirc(z^{(\nu)})\right) \subseteq [Z].$$

Applying properties (19) and (18) to the last inclusion in $I\!R^{*n}$, we get

$$Dual(\Diamond\left(Dual(\bigcirc(z^{(0)})) + \sum_{\nu=1}^{k} Dual([p_\nu]) \times Dual(\bigcirc(z^{(\nu)}))\right)) \subseteq [Z]. \qquad (22)$$

For fixed $\nu = 0, 1, \ldots, k$, let $d^{(\nu)} \in \mathbb{F}^n$ be a floating-point approximation of $b^{(\nu)} - A^{(\nu)}\tilde{x}$

$$d^{(\nu)} \approx b^{(\nu)} - A^{(\nu)}\tilde{x}.$$

The error $e^{(\nu)}$ of this approximation is $e^{(\nu)} = b^{(\nu)} - A^{(\nu)}\tilde{x} - d^{(\nu)}$.
Hence, $\bigcirc(e^{(\nu)}) := \bigcirc\left(b^{(\nu)} - A^{(\nu)}\tilde{x} - d^{(\nu)}\right)$ is an inner approximation of the error, while $\diamond(e^{(\nu)}) := \diamond\left(b^{(\nu)} - A^{(\nu)}\tilde{x} - d^{(\nu)}\right)$ is an outer one

$$\bigcirc(e^{(\nu)}) \subseteq b^{(\nu)} - A^{(\nu)}\tilde{x} - d^{(\nu)} \subseteq \diamond(e^{(\nu)}).$$

Multiplying both sides above by R and applying the inclusion properties of interval operations, we get

$$\bigcirc(z^{(\nu)}) := \bigcirc\left(R \cdot \bigcirc(e^{(\nu)}) + R \cdot d^{(\nu)}\right) \subseteq z^{(\nu)} \subseteq \diamond\left(R \cdot \diamond(e^{(\nu)}) + R \cdot d^{(\nu)}\right) =: \diamond(z^{(\nu)}). \tag{23}$$

If computed by a real dotproduct accumulator, $\bigcirc(e^{(\nu)}) = Dual\left(\diamond(e^{(\nu)})\right)$ due to (21), and thus left-hand side inclusion in (23) is equivalent to

$$\bigcirc\left(R \cdot Dual(\diamond(e^{(\nu)})) + R \cdot d^{(\nu)}\right) \subseteq z^{(\nu)}.$$

Applying (19) and (18) we get

$$Dual\left(\diamond\left(R \cdot \diamond(e^{(\nu)}) + R \cdot d^{(\nu)}\right)\right) \subseteq z^{(\nu)},$$

that is

$$\bigcirc(z^{(\nu)}) = Dual\left(\diamond\left(R \cdot \diamond(e^{(\nu)}) + R \cdot d^{(\nu)}\right)\right) = Dual(\diamond(z^{(\nu)})).$$

Substituting the last expression into (22) and applying (17), we obtain for $\bigcirc([Z])$

$$\bigcirc([Z]) = Dual(\diamond\left(\diamond(z^{(0)}) + \sum_{\nu=1}^{k} Dual([p_\nu]) \times \diamond(z^{(\nu)})\right)).$$

This way we represented the inward approximation of $[Z]$ only by outwardly rounded interval operations between proper and improper intervals. The only thing that remains to be considered is the product $Dual([p_\nu]) \times \diamond(z^{(\nu)})$ for $\nu = 0, 1, \ldots, k$, where $\diamond(z^{(\nu)})$ is a proper interval vector and $Dual([p_\nu])$ is an improper interval. Although interval data types in C-XSC allow storing of improper intervals, the conventional multiplication operation cannot be applied to the product $Dual([p_\nu]) \times \diamond(z^{(\nu)})$ [13]. Table 1 defines the product of a proper and an improper interval. The latter is a special case of the multiplication of generalised intervals considered in [10].

In environments not supporting generalised interval arithmetic the left and the right endpoints of (16) can be computed in two real dotproduct accumulators with the corresponding roundings.

Above we have proven the following

Theorem 4 *Let $A(p)x = b(p)$ be a parametric linear system with $A(p) \in \mathbb{R}^{n\times n}$, $b(p) \in \mathbb{R}^n$ given by (5) and a parameter vector $p \in \mathbb{R}^k$ varying within given intervals $p \in [p] \in I\mathbb{F}^k$. Let $R \in \mathbb{F}^{n\times n}$, $[Y] \in I\mathbb{F}^n$ and $\tilde{x} \in \mathbb{F}^n$. For $\nu = 0, 1, \ldots, k$, define $d^{(\nu)} :\approx b^{(\nu)} - A^{(\nu)}\tilde{x}$, $\diamond(e^{(\nu)}) :=$*

$Dual([a]) \times [b]$	$b^- \geq 0$	$b^+ \leq 0$	$b^- < 0 < b^+$
$a^- \geq 0$	$[a^+b^-,\ a^-b^+]$	$[a^-b^-,\ a^+b^+]$	$[a^-b^-,\ a^-b^+]$
$a^+ \leq 0$	$[a^+b^+,\ a^-b^-]$	$[a^-b^+,\ a^+b^-]$	$[a^+b^+,\ a^+b^-]$
$a^- < 0 < a^+$	$[a^+b^-,\ a^-b^-]$	$[a^-b^+,\ a^+b^+]$	$[0,\ 0]$

Table 1: Multiplication $Dual([a^-, a^+]) \times [b^-, b^+]$ for $[a], [b] \in I\mathbb{R}$.

$\Diamond\left(b^{(\nu)} - A^{(\nu)}\tilde{x} - d^{(\nu)}\right)$ *and* $\Diamond(z^{(\nu)}) := \Diamond\left(R \cdot \Diamond(e^{(\nu)}) + R \cdot d^{(\nu)}\right)$. *Define* $\Diamond([Z]) \in I\mathbb{F}^n$, $[C(p)] \in I\mathbb{F}^{n\times n}$ *by*

$$\begin{aligned} \Diamond([Z]) &:= \Diamond\left(\Diamond(z^{(0)}) + \sum_{\nu=1}^{k}[p_\nu] \times \Diamond(z^{(\nu)})\right), \\ [C(p)] &:= \Diamond\left(I - R\cdot A^{(0)} - \sum_{\nu=1}^{k}[p_\nu] \times (R\cdot A^{(\nu)})\right). \end{aligned}$$

Define $[V] \in I\mathbb{F}^n$ *by*

$$1 \leq i \leq n \ :\ V_i := \{\Diamond([Z]) \,\diamondplus\, [C(p)] \,\diamonddot\, [U]\}_i, \quad [U] := (V_1, ..., V_{i-1}, Y_i, ..., Y_n)^\top.$$

If $[V] \subsetneqq [Y]$, *then* R *and every matrix* $A(p), p \in [p]$ *are regular, and the solution set (2) satisfies* $[\inf(\Sigma^p), \sup(\Sigma^p)] \subseteq \tilde{x} \,\diamondplus\, [V]$.

With $\bigcirc([Z]) := Dual(\Diamond\left(\Diamond(z^{(0)}) + \sum_{\nu=1}^{k} Dual([p_\nu]) \times \Diamond(z^{(\nu)})\right))$ *and* $[D] := [C(p)] \diamonddot [V]$ *the following inner estimation holds true*

$$Dual(\tilde{x} \,\diamondplus\, \Diamond\left(\Diamond(z^{(0)}) + \sum_{\nu=1}^{k} Dual([p_\nu]) \times \Diamond(z^{(\nu)})\right) \,\diamondplus\, [D]) \subseteq [\inf(\Sigma^p),\ \sup(\Sigma^p)]\,, \quad (24)$$

where the intervals and the operations are considered as generalised.

The second part of the above theorem specifies how to compute a verified inner estimation for the hull of the parametric solution set by intermediate computations with proper and improper intervals. The results of the application of this theorem, that is the inner estimation (24) should be interpreted in terms of conventional interval arithmetic (proper intervals). Components of the inner estimations (24) may be improper intervals which are considered as empty in conventional interval arithmetic. In this case no inner estimation for these components of the hull of the solution set can be given [14].

3 Algorithms and Implementation

Based on the above theoretical considerations, in this section we present an algorithm for solving parametric linear systems that is intended to work with entirely numeric representation of the data for the parametric system. Let us consider the parametric linear system (1) in a representation (6) factorized by the parameters

$$\left(A^{(0)} + \sum_{\nu=1}^{k} p_\nu A^{(\nu)}\right) x \;=\; \left(b^{(0)} + \sum_{\nu=1}^{k} p_\nu b^{(\nu)}\right),$$

where $p_\nu \in [p_\nu] \in I\mathbb{R}$, $\nu = 1, \ldots, k$. We shall assume that:

1. The interval vector $[ip] \in I\mathbb{F}^{k+1}$ contains the interval values for the k parameters involved in the system (1), resp. (6) and that the first component of this vector is the point interval $[1, 1]$, that is
$$[ip] \;=\; ([1], [p_1], \ldots, [p_k])^\top .$$

2. The parametric matrix of the system (1) is represented by a two-dimensional numerical matrix $Ap \in \mathbb{F}^{(nk+n)\times n}$ having the following block structure
$$\begin{pmatrix} A^{(0)} \\ A^{(1)} \\ \vdots \\ A^{(k)} \end{pmatrix}, \tag{25}$$
where $A^{(\nu)} \in \mathbb{F}^{n\times n}$, $\nu = 0, 1, \ldots, k$, are the numerical matrices (5) from the factorized representation (6). Each numerical matrix $A^{(\nu)}$, $\nu = 0, 1, \ldots, k$, corresponds to a component of the interval vector $[ip]$ and involves the numerical coefficients in front of the corresponding parameter in $A(p)$.

3. The parametric right-hand side vector $b(p)$ is represented by a numerical matrix $bp \in \mathbb{F}^{n\times(k+1)}$ having the following block structure:
$$bp = \left(b^{(0)}, b^{(1)}, \ldots, b^{(k)}\right),$$
where each vector-column $b^{(\nu)} \in \mathbb{R}^{n\times 1},\ \nu = 0, 1, \ldots, k$, corresponds to a component of the interval vector $[ip]$.

The algorithm for solving square parametric linear systems, presented below, is an extension of the well-known fixed-point iteration algorithm for non-parametric linear systems, which is presented in more details in [19] and [5]. In what follows we shall use the notations from [5] and some of the algorithms presented therein.

The procedure for solving a parametric interval linear system (1), resp. (6) has two principal steps: First step, compute an approximate solution $\tilde{x}$ to the mid-point system $A(\check{p}) \cdot x = b(\check{p})$, where $\check{p} = \mathrm{mid}([ip])$. Second, try to find an interval enclosure for the error of this approximation. The first step involves initialization of the mid-point vectors and the matrix; computation of an approximate mid-point solution; and real residual iteration for the approximate solution. An algorithm for computing the approximate inverse of a point matrix and its implementation as a C-XSC function `MatInv()` is precisely presented in [5], Chapter 10. The CToolbox functions `CheckForZeros()` and `Accurate()` are used as they are defined in [5], Chapter 10, in order to facilitate obtaining a good approximation of the mid-point solution.

Second step, try to find an interval enclosure for the error of the mid-point approximation. An enclosure of the parametric solution set is provided by accounting for the data dependencies. To this end and to prepare the verification step, a (rough or sharp) enclosure of the iteration matrix $[C]$ and a sharp enclosure of $[Z]$ are to be computed by using the exact scalar product tools of C-XSC. The following notations are used in the algorithms:

$\Box(expr)$	–	for real scalar product expressions rounded to the nearest;
$\Diamond(expr)$	–	for real scalar product expressions, or for interval scalar product expressions with outward rounding to the enclosing interval;
$\bigtriangledown(expr)$	–	for real scalar product expressions rounded toward $-\infty$;
$\bigtriangleup(expr)$	–	for real scalar product expressions rounded toward $+\infty$.

The following algorithm extends the classical `VerificationStep()` for nonparametric linear systems by an Einzelschrittferfahren which accelerates the iterations.

Algorithm: VerificationStep ([x], [z], [C], Epsilon, IsVerified)

1. {Initialization}
   $p_{\max} := 10;\ p = 0;\ [x] := [z];$

2. {Verification}

   repeat
   $[x] := [x] \bowtie Epsilon;$ $\{\varepsilon$-Inflation$\}$
   $[y] := [x];$
   for $i = 1$ to n **do** {Einzelschrittferfahren}
   $[x_i] := \Diamond\,([z_i] + [C_i] \cdot [x]);$
   $IsVerified := ([x] \subsetneqq [y]);$
   $p := p + 1;$
   until $IsVerified$ **or** $(p \geq p_{\max});$

3. {Return enclosure $[x]$ and flag $IsVerified$}

   return $[x]$, $IsVerified$;

In order to get solution set enclosures with better quality for large interval tolerances, the user is given the opportunity to specify the epsilon-inflation constant. Instead of the relation $\subsetneqq$ between $[x]$ and $[y]$, the current implementation uses the built-in function "inclusion in the interior".

The local function `Dual()` reverses the end-points of intervals. If the verification step is successful, an inner estimation of the outer enclosure could be computed. Except for the multiplication between proper and improper interval vectors, which is simulated, the inner estimation is obtained efficiently and at no additional cost since it uses the intermediate quantities necessary for computing the outer enclosure.

Below we give the complete algorithm for computing a verified enclosure of the solution set of a parametric linear system of equations as well as an inner estimation of this enclosure.

Algorithm: ParLinSolve (Ap, bp, [ip], SharpC, Epsilon, Inner, [x], [y], Err)

1. {Initialization}
   Compute the parametric mid-point vector $\check{p}$, the corresponding mid-point right-hand side and the mid-point matrix
   $\check{p} := \textbf{mid}([ip]);\quad \check{b} := bp \cdot \check{p};\quad \check{A} := \sum_{\nu=1}^{k+1} \left(\check{p}_\nu * Ap^{(\nu)}\right).$

2. {Computation of an approximate mid-point solution}
   MatInv$(\check{A}, R, Err)$;
   if $(Err \neq$ "No error") **then**
   return $Err :=$ "Matrix is probably singular!"

3. {Real residual iteration for an approximate solution}
   $k_{\max} := 10;\ k := 0;\ \widetilde{x}^{(0)} := R \cdot \check{b};$
   repeat
   $d := \Box(\check{b} - \check{A} \cdot \widetilde{x}^{(k)});$
   $\widetilde{x}^{(k+1)} := \Box(\widetilde{x}^{(k)} + R \cdot d);$
   CheckForZeros$(\widetilde{x}^{(k)}, \widetilde{x}^{(k+1)});$
   $Success :=$ **Accurate**$(\widetilde{x}^{(k)}, \widetilde{x}^{(k+1)});$
   $k := k + 1;$
   until $Success$ **or** $(k \geq k_{\max});$
   $\widetilde{x} := \widetilde{x}^{(k)};$

4. {Computation of an enclosure $[C]$ for the set $\{I - R \cdot A(p) \mid p \in [ip]\}$}
   if $(SharpC)$ **then** {sharp enclosure}
   for $i = 1$ **to** n **do**
   for $j = 1$ **to** n **do**
   $[T_1] := \Diamond \left(I_{ij} - R_i \cdot (Ap^{(0)})_j^\top\right);$
   $[T_\nu] := \Diamond \left(-R_i \cdot (Ap^{(\nu)})_j^\top\right);\quad (\nu = 2, \ldots, k+1)$
   $[C_{ij}] := \Diamond\,([T] \cdot [ip]);$
   else {rough enclosure }
   $[AA] := \Diamond \left(\sum_{\nu=1}^{k+1} [ip_\nu] * Ap^{(\nu)}\right);$
   $[C] := \Diamond\,(I - R \cdot [AA]);$

5. {Computation of an enclosure $[z]$ for the set $\{R \cdot (b(p) - A(p) \cdot \widetilde{x}) \mid p \in [ip]\}\}$
   for $\nu = 1$ **to** $k + 1$ **do**
   $d := \Box \left(b^{(\nu)} - A^{(\nu)} \cdot \widetilde{x}\right);$
   $[e] := \Diamond \left(b^{(\nu)} - A^{(\nu)} \cdot \widetilde{x} - d\right);$
   $[T_k^\top] := \Diamond\,(R \cdot d + R \cdot [e]));$
   $[z] := \Diamond\,([T] \cdot [ip]);$

6. {Verification step and Inner estimation}

   VerificationStep$([x], [z], [C], Epsilon, IsVerified)$;
   if (**not** $IsVerified$) **then**
      Err := "Verification failed, the system is probably ill-conditioned";
      return Err;
   else
      if ($Inner$) **then**
         for $i = 1$ **to** n **do**
            Adjust element by element the end-points of $[ip]$ and the i-th row of $[T]$, $[T_i]$, according to Table 1, so that the product of the left (right) vector end-points gives the result left (right) end-point;
            $\text{inf}([z_i]) = \bigtriangledown\,(\text{inf}([ip]) \cdot \text{inf}([T_i]))$;
            $\text{sup}([z_i]) = \bigtriangleup\,(\text{sup}([ip]) \cdot \text{sup}([T_i]))$;
            $[y_i] = \Diamond\,(\tilde{x}_i + [z_i] + [C_i] \cdot [x])$;
      $[y]$:=**Dual**$([y])$;
      $[x] := \tilde{x} + [x]$;   { Mid-point approximation plus enclosing interval}

7. {Return outer enclosure $[x]$, inner estimation $[y]$ and error code Err}

   return $[x]$, $[y]$, Err;

Implementation:

The distribution of the C-XSC module *parlinsys*, which is intended to extend the C++ Numerical Toolbox [5] with tools for solving parametric interval linear systems, involves the following files:

- a header file *parlinsys.hpp*,
- the source code of the module *parlinsys.cpp*,
- a sample *Makefile*,
- the source code of a sample program *parlinsys_ex.cpp* for console data,
- the source code of a sample program *parlinsys_exf.cpp* for file data,
- a file *README.txt*.

The header file of the module *parlinsys* provides interfaces of the functions `ParLinSolve()` and `ParLinSolveErrMsg()`. The function `ParLinSolveErrMsg()` can be used to get an explicit error message for the integer error code returned by the function `ParLinSolve()`. The function `ParLinSolve()` is an implementation of the above Algorithm *ParLinSolve (Ap, bp, [ip], SharpC, Epsilon, Inner, [x], [y], Err)*.

The current distribution of the module *parlinsys* can be obtained from

`http://www.math.uni-wuppertal.de/wrswt/xsc/cxsc_software.html`

Since our implementation is intended also for education and experimentation purposes, the function involves arguments for: switching on/off the sharp enclosure of the iteration matrix, specifying a value for the constant of epsilon inflation, and switching on/off the computation of an inner approximation for the outer enclosure.

4 Examples

Here, we demonstrate how to use the modules defined in the previous section. A sample program contained in the distribution file `parlinsys_ex.cpp` reads input data from the console. It is suitable for interactive solution of small parametric problems. Another sample program from the distribution file `parlinsys_exf.cpp` is presented below. It reads input data from a file which should have the following structure:

'n'	–	integer > 0 specifying the dimension of the system;
'k'	–	integer > 0 specifying the number of the parameters;
'$SharpC$'	–	0 or 1; 1 for computing sharp enclosure of the contracting matrix C; 0 – otherwise;
'Eps'	–	constant for the epsilon inflation;
'$Inner$'	–	0 or 1; 1 for computing inner estimation;
Ap	–	matrix of the system in factorized representation (25);
bp	–	right-hand side of the system in a factorized matrix representation $bp = (b^{(0)}, b^{(1)}, \ldots, b^{(k)})$;
p	–	vector of the interval values for the k parameters.

The dynamic arrays needed to store the numerical input data, and of the solution set, are allocated dynamically depending on n, k and $Inner$.

A local function `Sharpness([x], [y])` is involved to compute a measure for the quality of a solution enclosure $[x] \supseteq \Diamond(\Sigma^p)$ based on an inner estimation $[y] \subseteq \Diamond(\Sigma^p)$.

$$\text{Sharpness}([x],[y]) := \begin{cases} 1 & \text{if } \textbf{diam}([x]) = 0, \\ -100 & \text{if } [y] \not\subseteq [x], \\ 0 & \text{if } [y] = \emptyset, \\ \textbf{diam}([y])/\textbf{diam}([x]) & \text{otherwise.} \end{cases}$$

After reading all input data, the program tries to enclose the solution by calling `ParLin Solve()`. If the algorithm succeeds, the enclosure of the solution goes to the output, and if an inner estimation has been required the program sends the latter to the output, and also computes the sharpness of the enclosure by calling the function `Sharpness()`. If the algorithm is not successful, the program sends a corresponding error message to the output by calling the function `ParLinSolveErrMsg()`.

```
#include <iostream>
#include <parlinsys.hpp>  // Parametric Linear System Solver
#include <ivector.hpp>    // Interval vector arithmetic

using namespace std;
```

```
using namespace cxsc;

static real Sharpness(interval& xx, interval& yy)
{  real sh;
   if (diam(xx) == 0.0) sh = 1.0;
   else if (Inf(yy) > Sup(yy)) sh = 0.0;
        else
          if (!(Inf(xx) <= Inf(yy) && Sup(yy) <= Sup(xx)))
            sh = -100;
          else sh = diam(yy)/diam(xx);
   return sh;
}

int main()
{ int n, k, i, j, Err, SharpC, Inner;
  real Eps;

  cout << SetPrecision(10, 12) << Scientific << endl; //Output format
  cin >> n;
  cin >> k;
  cin >> SharpC;
  cin >> Eps;
  cin >> Inner;

  rmatrix Ap(n*(k+1), n), bp(n, k+1), A(n, n);
  ivector p(k+1), xx, yy;

  for(i = 1, j = 0; i <= k+1; ++i,j = n*(i-1))
  { cin >> A;
    Ap(j+1, i*n, 1, n) = A;
  }
  cin >> bp;
  p[1] = interval(1);
  for(i = 2; i <= k+1; i++)  cin >> p[i];

  cout << "n = " << n << "  ";
  if (SharpC) cout << "Sharp  eps =  " << Eps << endl;
   else cout << "Rough  eps = " << Eps << endl;

  ParLinSolve(Ap, bp, p, SharpC, Eps, Inner, xx, yy, Err);

  if(!Err)
  { cout << "Verified solution enclosure found in:"
    << endl << xx <<  endl;
    if (Inner)
    { rvector sharpness(n);
      cout << "Verified Inner Estimation:" << endl;
```

```
      for (i=1; i<=n; i++)
        if (Inf(yy[i]) > Sup(yy[i]))
        { cout << "In[" << i << "] = Empty" << endl;
          sharpness[i] = Sharpness(xx[i], yy[i]);
        }
        else
       { cout << yy[i] << endl;
         sharpness[i] = Sharpness(xx[i], yy[i]);
       };
       cout << endl << "Componentwise sharpness of the outer enclosure:"
       << endl << SetPrecision(5, 2) << sharpness << endl;
    } // end Inner
  }
  else
    cout << ParLinSolveErrMsg(Err) << endl;

  cout << endl;
  return 0;
}
```

Example 4.1 *As a first sample parametric problem we use the system*

$$\begin{pmatrix} 1 & p_1 \\ p_1 & p_2 \end{pmatrix} x = \begin{pmatrix} 2+p_2 \\ 2+p_2 \end{pmatrix}, \qquad p_1 \in [0.4, 0.6],\ p_2 \in [-2.2, -1.8]. \tag{26}$$

A corresponding data file containing numerical input for the program `parlinsys_exf.cpp` is

```
2
2
0
0.1
1
1 0 0 0
0 1 1 0
0 0 0 1
2 0 1
2 0 1
[0.4, 0.6]
[-2.2, -1.8]
```

Our sample program leads to the following output:

```
n = 2  Rough  eps = 0.1  MaxIter = 2

Verified solution enclosure found in:
[-2.382280164610E-001,2.382280164610E-001]
[-6.314253461363E-002,6.314253461363E-002]
```

```
Verified Inner Estimation:
[-2.085092616858E-001,2.085092616858E-001]
[-2.684069542245E-002,2.684069542245E-002]

Componentwise sharpness of the solution enclosure:
8.75E-001
4.25E-001
```

For this problem, sharp and rough estimations of the iteration matrix give same results [15].

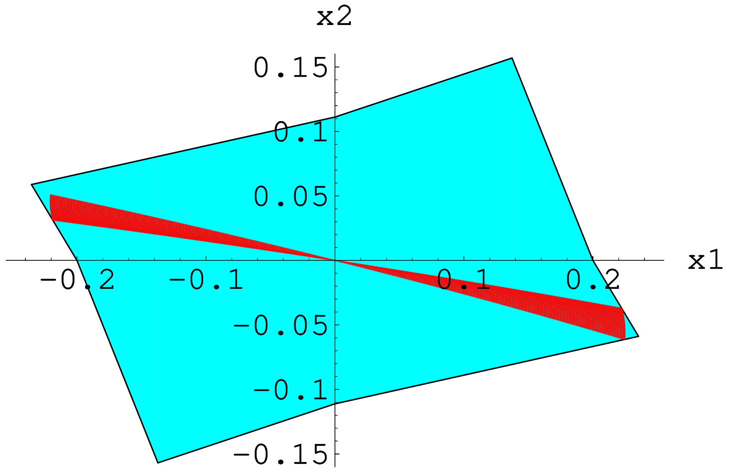

Figure 1: The parametric and the corresponding non-parametric solution sets for the problem (26).

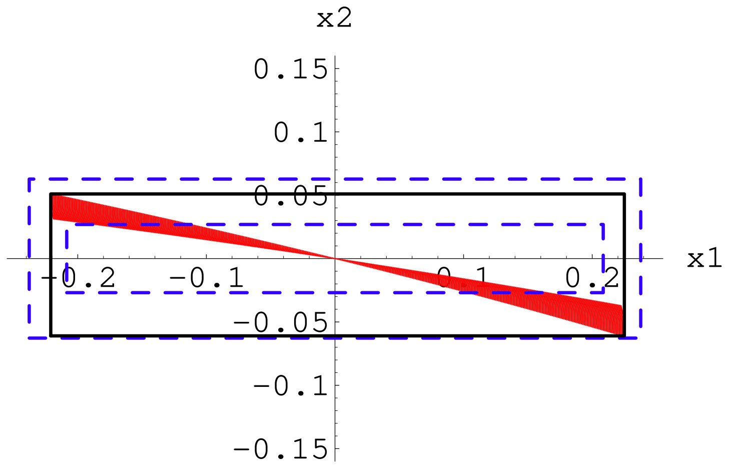

Figure 2: The parametric solution set for problem (26), its hull and the corresponding inner and outer estimations.

We can vary the epsilon constant to get a slightly better inner and outer enclosures. The two limit values for epsilon are $6 * 10^{-10}$ and 0.2:

```
n = 2  Rough  eps = 6.00E-010  MaxIter = 10

Verified solution enclosure found in:
[-2.356979405085E-001,2.356979405085E-001]
[-6.178489702601E-002,6.178489702601E-002]

Verified Inner Estimation:
[-2.087465039408E-001,2.087465039408E-001]
[-2.710399186341E-002,2.710399186341E-002]

Componentwise sharpness of the solution enclosure:
8.86E-001
4.39E-001

n = 2  Rough  eps = 2.00E-001  MaxIter = 1

Verified solution enclosure found in:
[-2.374320987655E-001,2.374320987655E-001]
[-6.191056241427E-002,6.191056241427E-002]

Verified Inner Estimation:
[-2.086912117056E-001,2.086912117056E-001]
[-2.701295534218E-002,2.701295534218E-002]

Componentwise sharpness of the solution enclosure:
8.79E-001
4.36E-001
```

Large intervals for the parameters usually result in empty inner estimations.

Example 4.2 *Consider a very ill-conditioned point linear system*

$$\begin{pmatrix} 64919121 & -159018721 \\ 41869520.5 & -102558961 \end{pmatrix} x \;=\; \begin{pmatrix} 1 \\ 0 \end{pmatrix}.$$

The function `ParLinSolve()` can solve point linear systems, too, but the input data should involve at least one dummy parameter, e.g. as in the following data file:

```
2
1
0
0.05
1
64919121 -159018721 41869520.5 -102558961
0 0 0 0
1 0
```

```
0 0
[1, 1]
```

The output of our sample program is

```
n = 2  Rough  eps = 5.0000E-002  MaxIter = 1

Verified solution enclosure found in:
[2.051179220000000E+008,2.051179220000000E+008]
[8.373904100000000E+007,8.373904100000000E+007]

Verified Inner Estimation:
[2.051179220000000E+008,2.051179220000000E+008]
[8.373904100000000E+007,8.373904100000000E+007]

Componentwise sharpness of the outer enclosure:
1.00E+000
1.00E+000
```

Although the system is very ill-conditioned, the program has found the exact solution. This is because the exact scalar product is used in the intermediate computations which results in an exact mid-point solution. Since the vector $[Z]$ is zero, and the iteration matrix $[C] = 0$ for this case, the verification step succeeds. The parametric solver fails on this example in environments that do not support exact scalar product [16].

Example 4.3 *Consider the parametric $n \times n$ linear systems $Q(2,p)x = b(p)$, where for $i, j = 1, \ldots, n$, [15]*

$$q_{ij}(2,p) := \begin{cases} p_j & \text{if } i \leqq j, \\ 0 & \text{if } i = j+2, \\ 1 & \text{otherwise}, \end{cases} \qquad b(p) = (p_1, \ldots, p_n)^\top, \qquad p_i \in [(i+1) \pm 10\%].$$

Below we present some results in case $n = 4$.

```
4
4
1
0.1
1
0 0 0 0 1 0 0 0 0 1 0 0 1 0 1 0
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1
0 1 0 0 0
0 0 1 0 0
```

```
0 0 0 1 0
0 0 0 0 1
1.8 2.2
2.7 3.3
3.6 4.4
4.5 5.5
```

For any epsilon and $SharpC$ set to 0, the program fails to find a solution because $\varrho(|[C]|) > 1$. However, if we require a sharp enclosure of the iteration matrix $[C(p)]$ (see (12)), then we get the following results:

```
n = 4  Sharp  eps =  1.0E-001  MaxIter = 2

Verified solution enclosure found in:
[-1.908320000001E+000,-9.167999999999E-002]
[-9.551360000001E-001,9.551360000001E-001]
[-1.840792533334E+000,5.074592000001E-001]
[9.119170488888E-001,1.754749617778E+000]

Verified Inner Estimation:
[-1.518336000000E+000,-4.816640000000E-001]
[-4.658976000000E-001,4.658976000000E-001]
[-1.227083306667E+000,-1.062500266666E-001]
[1.193057767111E+000,1.473608899556E+000]

Componentwise sharpness of the outer enclosure:
5.71E-001
4.88E-001
4.77E-001
3.33E-001
```

For epsilon $= 1.0 * 10^{-7}$ we get a better solution enclosure in ten iterations:

```
n = 4  Sharp  eps =  1.0E-007  MaxIter = 10

Verified solution enclosure found in:
[-1.875000025830E+000,-1.249999741700E-001]
[-9.264706180111E-001,9.264706180111E-001]
[-1.808257953782E+000,4.749246204479E-001]
[9.222389700692E-001,1.744427696598E+000]

Verified Inner Estimation:
[-1.524999994835E+000,-4.750000051659E-001]
[-4.735294047154E-001,4.735294047154E-001]
[-1.236186517124E+000,-9.714681620992E-002]
[1.188872133934E+000,1.477794532732E+000]
```

```
Componentwise sharpness of the outer enclosure:
6.00E-001
5.11E-001
4.99E-001
3.51E-001
```

Example 4.4 *Consider a parametric linear system* $A(p)x = b(p)$ *of dimension* $n = 50$ *with Milness matrix [4], where for* $i, j = 1, \dots, n$

$$a_{ij}(p) = \begin{cases} p_j & \text{if } i > j, \\ 1 & \text{otherwise}, \end{cases} \qquad b(p) = (p_1, \dots, p_n)^\top, \quad p_i \in [1/(i+1) \pm 5\%].$$

For epsilon $= 0.2$ and using the rough enclosure of the iteration matrix $[C]$ we get the following results

```
n = 50  Rough  eps = 2.00E-001  MaxIter = 2

Verified solution enclosure found in:
[2.258433333333E-001,4.408233333334E-001]
[6.808432812499E-002,1.819156718751E-001]
[2.489202677469E-002,1.084413065587E-001]
[7.235818895880E-003,7.609751443746E-002]
[-1.509469390293E-003,5.865232653315E-002]
[-6.370772402539E-003,4.803743906921E-002]
[-9.290461104292E-003,4.103649285033E-002]
[-1.114390216537E-002,3.614390216537E-002]
[-1.237017267304E-002,3.257219287506E-002]
..............
[-1.537057586849E-002,1.617089599654E-002]
[-2.900281024112E-001,-1.703640544515E-001]

Verified Inner Estimation:
[2.387078333333E-001,4.279588333334E-001]
[8.760964179687E-002,1.623903582032E-001]
[4.594738497042E-002,8.738594836292E-002]
[2.905462108855E-002,5.427871224479E-002]
[2.082070111216E-002,3.632215603070E-002]
[1.635178013702E-002,2.531488652964E-002]
[1.375548469914E-002,1.799054704689E-002]
[1.217999247961E-002,1.282000752039E-002]
In[9] = Empty
................
In[49] = Empty
[-2.692842805157E-001,-1.911078763471E-001]
```

```
Componentwise sharpness of the outer enclosure:
8.80E-001
6.57E-001
4.96E-001
3.66E-001
2.58E-001
1.65E-001
8.42E-002
1.35E-002
0.00E+000
................
0.00E+000
6.53E-001
```

With a sharp enclosure of the iteration matrix and the same value for the epsilon constant, we get the solution enclosure in one iteration, better nonempty inner estimation and correspondingly much better quality of the solution sharpness.

```
Componentwise sharpness of the outer enclosure:
0.885, 0.941, 0.961, 0.970, 0.976, 0.98, 0.983, 0.985, 0.987,
0.988, 0.989, 0.99, 0.991, 0.991, 0.992, 0.993, 0.993, 0.993,
0.994, 0.994, 0.994, 0.995, ...., 0.997, 0.998, 0.998, 0.711
```

References

[1] American National Standards Institute/Institute of Electrical and Electronics Engineers: “IEEE Standard for Binary Floating-Point Arithmetic”; ANSI/IEEE Std 754–1985, New York (1985).

[2] O. Dessombz et al., Analysis of mechanical systems using interval computations applied to finite element methods, Journal of Sound and Vibration 239 (2001) 5, 949-968.

[3] Gardeñes, E., Trepat, A.: Fundamentals of SIGLA, an Interval Computing System over the Completed Set of Intervals, *Computing* **24** (1980), pp. 161–179.

[4] Gregory, R. T., Karney, D. L., A Collection of Matrices for Testing Computational Algorithms. Wiley-Interscience, N.Y., 1969.

[5] Hammer, R.; Hocks, M.; Kulisch, U.; Ratz, D., C++ Toolbox for Verified Computing: Basic Numerical Problems. Springer-Verlag, Berlin / Heidelberg / New York, 1995.

[6] Herzberger, J. (Ed), Topics in Validated Computations. Proceedings of IMACS-GAMM International Workshop on Validated Numerics, Oldenburg, 1993. North Holland, 1994.

[7] Hölbig, C.; Krämer, W., Selfverifying Solvers for Dense Systems of Linear Equations Realized in C-XSC. Preprint 2003/1, Universität Wuppertal, 2003.

[8] Hofschuster, W.; Krämer, W.; Wedner, S.; Wiethoff, A., C-XSC 2.0 – A C++ Class Library for Extended Scientific Computing. Preprint 2001/1, Wissenschaftliches Rechnen / Softwaretechnologie, Universität Wuppertal, 2001. (http://www.math.uni-wuppertal.de/wrswt/preprints/prep_01_1.pdf)

[9] Hofschuster, W.; Krämer, W., C-XSC 2.0 – A C++ Class Library for Extended Scientific Computing. In: *Numerical Software with Result Verification*, R. Alt, A. Frommer, B. Kearfott, W. Luther (eds), Springer Lecture Notes in Computer Science **2991**, 2004, pp. 15–35.

[10] Kaucher, E.: Interval Analysis in the Extended Interval Space IR, *Computing Suppl.* **2** (1980), pp. 33–49.

[11] Klatte, R.; Kulisch, U.; Lawo, C.; Rauch, M.; Wiethoff, A.: C-XSC, A C++ Class Library for Extended Scientific Computing. Springer - Verlag, Berlin / Hiedelberg / New York, 1993.

[12] Krämer, W., Kulisch, U.,Lohner, R.: *Numerical Toolbox for Verified Computing II – Advanced Numerical Problems*. Universität Karlsruhe, 1994, *http://www.uni-karlsruhe.de/R̃udolf.Lohner/papers/tb2.ps.gz*

[13] E. D. Popova: Extended Interval Arithmetic in IEEE Floating-Point Environment. Interval Computations, No. 4, 1994, pp. 100-129.

[14] E. D. Popova, On the solution of parametrised linear systems, in: Scientific Computing, Validated Numerics, Interval Methods, eds. W. Kraemer and J. Wolff von Gudenberg, Kluwer Acad. Pub., 2001, pp. 127-138.

[15] E. D. Popova, Improved Parametric Fixed-Point Iteration, Preprint Inst. of Mathematics & Informatics, BAS, Sofia, 2003.

[16] E. D. Popova, Parametric Interval Linear Solver, to appear in Numerical Algorithms, special issue Proceedings of SCAN 2002.

[17] E. D. Popova; M. Datcheva; R. Iankov; T. Schanz, Mechanical Models with Interval Parameters. In K. Gürlebeck L. Hempel C. Könke (Eds.) IKM2003: Digital Proceedings of 16th International Conference on the Applications of Computer Science and Mathematics in Architecture and Civil Engineering, ISSN 1611-4086, Weimar, 2003. (http://euklid.bauing.uni-weimar.de/papers/36/M_36.pdf)

[18] Popova, E., Ullrich, C.: *Directed Interval Arithmetic in Mathematica: Implementation and Applications.* TR 96-3, U. Basel, 1996, pp. 1–56. (http://www.math.bas.bg/˜epopova/papers/tr96-3.ps)

[19] S. Rump, Solving algebraic problems with high accuracy, in: A New Approach in Scientific Computation, eds. U. Kulisch and W. Miranker, Academic Press, 1983, pp. 51-120.

[20] S. Rump, Rigorous sensitivity analysis for systems of linear and nonlinear equations, Mathematics of Computation 54 (1990) 190, 721-736.

[21] S. Rump, Verification methods for dense and sparse systems of equations, in [6], 1994, pp. 63-135.

[22] SIGLA/X: Modal Intervals, in Vehi, J., Sainz M. (eds.): *Applications of Interval Analysis to Systems & Control*, Proc. MISC'99, Univ. de Girona, Spain (1999), pp. 139–207.