Parametric Interval Linear Solver

Evgenija D. Popova*
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
bl. 8, Acad. G.Bonchev str., 1113 Sofia, Bulgaria
e-mail: epopova@bio.bas.bg

Abstract. `IntervalComputations‘LinearSystems‘` is a *Mathematica* package supporting tools for solving parametric and non-parametric linear systems involving uncertainties. It includes a variety of functions, implementing different interval techniques, that help in producing sharp and rigorous results in validated interval arithmetic. The package is designed to be easy to use, versatile, to provide a necessary background for further exploration, comparisons and prototyping, and to provide some indispensable tools for solving parametric interval linear systems.

This paper presents the functionality, provided by the current version of the package, and briefly discusses the underlying methodology. A new hybrid approach for sharp parametric enclosures, that combines parametric residual iteration, exact bounds, based on monotonicity properties, and refinement by interval subdivision, is outlined.

Keywords: parametric linear systems, validated interval software.

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 operational research, linear prediction problems, etc. [2], [4], [5], [6] usually there are complicated dependencies between coefficients. The main reason for this dependency is that the errors in several different coefficients may be caused by the same factor.

Consider a parametric linear system

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

with affine-linear dependencies in the coefficients

$$a_{ij}(p) := \lambda_{ij0} + \sum_{\nu=1}^{k} \lambda_{ij\nu} p_\nu, \qquad b_i(p) := \beta_{i0} + \sum_{\nu=1}^{k} \beta_{i\nu} p_\nu, \tag{2}$$

$\lambda_{ij}, \beta_i \in \mathbb{R}^{k+1}$ $(i,j = 1,\ldots,n)$ are numerical vectors and the parameters p_ν can take arbitrary values from given intervals $[p_\nu]$, $(\nu = 1,\ldots,k)$. The parametric solution set (PSS)

$$\Sigma^p = \Sigma\left(A(p), b(p), [p]\right) \;\; := \;\; \{x \in \mathbb{R}^n \mid A(p)\cdot x = b(p) \text{ for some } p \in [p]\} \tag{3}$$

is usually a subset of the corresponding non-parametric solution set $\Sigma^g = \Sigma([A],[b]) := \{x \in \mathbb{R}^n \mid Ax = b \text{ for some } A \in [A] = A([p]),\ b \in [b] = b([p])\}$ and has much smaller volume than

*This work was supported by the Bulgarian National Science Fund under grant No. I-903/99 and by the NATO CLG 979541. The presentation at SCAN 2002 was supported by the Swiss NSF Co-operation Agreement No. 7 IP 65642.

the latter. The simplest example of dependencies is when the matrix is symmetric or skew-symmetric. Since the solution sets have a complicated structure (see e.g. [1]) which is difficult to find, we look for the interval hull $\Box\Sigma := [\inf \Sigma,\ \sup \Sigma]$, whenever Σ is a nonempty bounded subset of $\mathbb{R}^n$, or for an interval enclosure of $\Box\Sigma$.

At present, there is an increasing interest in interval applications to various practical problems involving the solution of parameter dependent linear systems. However, no official (public or commercial) software for solving such systems is available.

Although designed quite long ago, the only available iterative method for solving parametric interval linear systems [15] seems to be not known to the application scientists, or at least it is not applied. Except for [8], by now there is no other work that would apply and study the results of this method. Remarkably, the parametric Rump's method was recently reinvented in slightly different notations [2]. A significant effort has been devoted to eliminate the sources of overestimation by several construction methods that reduce the number of parameters in the system to be solved [4], [5], [6]. 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 quest for narrow and rigorous results, as well as for performance efficiency, requires carefully designed software based on an expertise and state-of-the-art in validated interval computations. In this paper we present a *Mathematica* package `IntervalComputations‘LinearSystems‘` which tries to fill in the existing gap. The package is initially designed for studying the performance and cost-efficiency of different algorithms and approaches for solving parametric linear systems. The focus is on the iterative residual methods (parametric and nonparametric) and on methods for proving monotonicity of the solution components. Since the environment is a computer algebra system, tools for finding the exact solution hull are also involved, and the impact of computer algebra on interval computations is outlined. A fruitful synergism between different methods leads to a new hybrid approach, presented in Section 2.4, for sharp parametric solution enclosure. A second goal is to provide the application scientists with carefully designed mathematical software, providing validated solutions and sharp bounds, so that they focus on developing applications rather than debugging low-level code.

Various methods for the application of FEM and for composing the parametric system to be solved usually lead to different parametric systems with different properties. Although the number of parameters may be reduced, or the different parameters can be grouped and isolated, the linear system that should be finally solved usually involves more dependencies than in a symmetric matrix. The methods, implemented in this package, and the corresponding software tools are general in the sense that they are applicable to any parametric system independent of the way a particular system has been obtained.

This paper aims at presenting mainly the functionality of the package and briefly discusses the underlining methodology. A comparison of the algorithms, although depending very much on the problem to be solved, deserves extensive presentation in a separate paper.

2 *Mathematica* Package

Since version 2.2 *Mathematica* supports the object `Interval` into the kernel of the system [16]. Basic interval arithmetic is automatic, performed with machine or user-specified precision and a posteriori outward rounding. *Mathematica* is maybe the only environment that supports exact interval arithmetic and combined rigorous usage of approximate numbers with exact numbers, mathematical constants and exact singletons at the interval end-points. Based on the exact or validated interval arithmetic of the *Mathematica* kernel, all the functions involved in the package `IntervalComputations‘LinearSystems‘` provide rigorous results.

It is important for all the enclosure methods/tools to be able to estimate the degree of overestimation they provide. A measure for sharpness of the outer enclosure is based on an

inner estimation of $\square\Sigma$ [7], [14]. Inner estimations can be computed by using interval operations with inward rounding but the overloading concept of some programming environments hampers the convenient implementation of these operations. That is why most of the interval packages/environments do not support inwardly rounded interval arithmetic and thus the possibility to estimate the degree of sharpness of the enclosures or to compute a minimum set of the solutions instead of an enclosure. The arithmetic of proper and improper intervals possesses properties due to which inwardly rounded interval arithmetic can be applied at no additional cost [10]. Based on the methodology developed in [8] and the arithmetic supported by the package `IntervalComputations‘GeneralizedIntervals‘` [10], all the iterative solvers, supported by the package `IntervalComputations‘LinearSystems‘`, provide rigorous inner estimations for the solution. The presented package is maybe the only that provides this feature.

The computer algebra environment of *Mathematica* provides tools for analytic solution, symbolic differentiation, and algebraic simplification which, combined with exact interval arithmetic and visualization facilities, present excellent tools for experimentation, exploration, and prototyping. Furthermore, the users have the advantage to input and manipulate their parametric data using convenient mathematical (symbolic) notations. Due to the computer algebra facilities, our package involves functions which can handle also parametric linear systems involving nonlinear dependencies between parameters.

2.1 Exact Bounds of the Solution Sets.

The existing methods for hull computation of the solution set to parametric and non-parametric interval linear systems are generally of exponential complexity and/or restricted scope of application. However, due to their simplicity and easy implementation, these methods are often used as reference algorithms during numerical experiments with other enclosure methods. That is why, the *Mathematica* package `IntervalComputations‘LinearSystems‘` involves some functions computing the exact hull of the solution set to parametric and non-parametric interval linear systems. The implemented methods are rigorous in exact arithmetic and not rigorous in approximate floating-point arithmetic. Since *Mathematica* [16] is an environment that supports also exact interval arithmetic, all functions from this class produce exact and rigorous interval results on exactly specified arguments. Matrix and/or vector entries can be either intervals or elements from the domain `Real`.

`ExactHull[A, b]` computes the exact interval hull of the solution set to a non-parametric interval linear system with numerical interval matrix `A` and numerical interval vector `b`. The computational procedure is based on a J. Rohn's sign-accord algorithm that requires solving 2^{2n} point linear systems [11].

`ExactHull[Ap, bp, parLst]` gives the exact hull of the solution set to a parametric linear system with matrix `Ap`, right-hand side `bp` and parameters varying within given numerical intervals specified by a list of *Mathematica* transformation rules[1] `parLst`. A fast algorithm, based on the signs of the partial derivatives, is used as a default computing method. The latter assumes that the components of the analytic solution $x(p) = A(p)^{-1}b(p)$ are monotone functions with respect to each parameter. The function itself does not check the monotonicity, the package provides another function for this purpose.

For the sake of completeness and experimentation purposes, the package contains functions `ExactHull[A, b, Combinatorial]` and `ExactHull[Ap, bp, parLst, Combinatorial]` that find the exact hull of the corresponding interval linear system with regular matrix by solving point linear systems, obtained as all possible combinations of the interval end-points. `AllPoints` is a symbol, which can be used as last optional argument for these functions applying combinatorial computing method. `AllPoints` implies output of the solutions to all point linear systems

[1] Rules of the form `name -> value`.

solved. In the interactive environment of *Mathematica* this output could be suitably used for visualization.

Monotonicity can also be used even when the solution is not monotonic provided its behavior is sufficiently well known. Some sufficient conditions for PSS having the same *quality* ($\square\Sigma^p = \square\Sigma^g$) as the solution set to the corresponding non-parametric system $A([p]) \cdot x = b([p])$ are proven in [9]. Based on this research, `HullCoincidence[Ap, bp, parLst]` checks which bounds of the solution set to a parametric system `Ap.x = bp`, where the parameters and their interval values are specified by a list of rules `parLst`, coincide with the bounds of the corresponding non-parametric solution set. The output is in the form {{{`inf-bds`}, {`sup-bds`}}, {`hull`}}, wherein {`inf-bds`} and {`sup-bds`} are lists with indexes of the coinciding bounds.

2.2 Iterative Solvers.

The *Mathematica* package `IntervalComputations‘LinearSystems‘` contains a collection of functions which compute guaranteed inclusions for the solution set of a square interval linear system. These functions, called iterative solvers, implement residual iteration methods, based on the fixed point theorem, that lead to guaranteed interval enclosures quite fast. The general methodology is due to S. Rump [13], [15]. The particular solvers differ upon the type of the linear system to be solved — general nonparametric system, system with symmetric matrix, or parametric linear system — implementing residual iteration methods, designed to be efficient for the specific interval problem [13], [3], [15].

`ILinearSolve[A, b]` computes guaranteed bounds for the solution set $\Sigma([A], [b])$ of a square interval linear system, where all elements vary independently in their intervals. The input elements can be either numerical intervals or elements from the domain `Real`.

Linear systems with symmetric matrix are the simplest special case of a parametric linear system. The fixed point iteration algorithm for this special case is proposed by C. Jansson [3]. Due to the simplicity of its implementation, this algorithm is often used in the applications, although they may involve more dependencies than in a symmetric matrix [6]. `SymmetricSolve[A, b]` computes guaranteed bounds for the symmetric solution set of a square linear system with symmetric matrix `A` and independent interval vector `b`. For linear systems with symmetric matrix and independent right hand side vector `bp`, involving dependencies in the elements, the function `SymmetricSolve[A, bp, parLst]` can be used, where the list of transformation rules `parLst` specifies the parameters and their interval values.

A general algorithm, that accounts for arbitrary affine-linear dependencies in the matrix and the right hand side vector, is proposed by S. Rump in [15]. The *Mathematica* functions `ParametricNSolve[Ap, bp, parLst]` and `ParametricSSolve[Ap, bp, parLst]` implement the **parametric** Rump's method[2] and compute guaranteed bounds for the PSS (3) to the linear system `Ap.x = bp`. The parameters and their interval values should be specified by a list of rules `parLst`. The environment of *Mathematica* [16] allows a convenient mathematical description of the parametric matrix and the right hand side vector.

```
In[2] := m = {{3, p1}, {p1, 3}};   b = {p2, p2};
     tr = {p1 -> Interval[{1, 2}], p2 -> Interval[{10, 10.5}]};
     ParametricNSolve[m, b, tr]
Out[4] = {Interval[{1.81481, 2.74074}], Interval[{1.81481, 2.74074}]}
```

All the iterative solvers can take options affecting the computational process and/or the output of the particular function[3]. The three options, associated with each of the iterative solvers

[2] should be distinguished from the nonparametric Rump's algorithm.

[3] Options in *Mathematica* are set by giving rules of the form `name -> value`. Each rule must appear after all the other arguments of a function. Rules for different options can be given in any order. If no explicit rule is given for a particular option, a default setting for that option is used.

are `InnerEstimation`, `Refinement`, and `Statistics`. `InnerEstimation`, when set to `True`, specifies the computing of component-wise inner approximation of the solution set in addition to the outer enclosure. Inner estimations allow to obtain the very important measure for the degree of sharpness of an outer solution set enclosure [14]. Computing inner approximations by the iterative solvers is based on generalized interval arithmetic (see [8]) and requires the package `IntervalComputations‘GeneralizedIntervals‘` [10] which, if available, is loaded automatically. `Refinement` is an option that, when set to `True`, implies the application of an iterative refinement procedure for the outer solution set approximation. As mentioned in [13], [15], this procedure usually brings quite little improvement on the solution set enclosure. `Statistics` is an option that, when set to `True`, implies output of some intermediate results like the number of iterations and the relative improvement by the refinement procedure. The three options `InnerEstimation`, `Refinement`, and `Statistics` are set to `False` by default.

```
In[5] := SymmetricSolve[m/. tr , b, tr, Refinement->True,
      InnerEstimation->True, Statistics->True]

  SymmetricSolve::stat1: Initial Iterations = 2.
  SymmetricSolve::stat2: Refinement Iterations = 23;
       Max relative improvement  2.7999999999762992‘.
Out[5] = {{Interval[{1.87037, 2.68519}], Interval[{1.87037, 2.68519}]},
      {Interval[{1.05556, 3.5}], Interval[{1.05556, 3.5}]}}
```

The two parametric solvers differ upon the way they minimize the number of parameter occurrences. `ParametricNSolve` transforms the parametric input data into numerical $n \times n \times (k+1)$-, $n \times (k+1)$-matrices and does a fast numerical simplification. `ParametricSSolve` is based on algebraic simplification of the parametric (symbolic) data.

The inclusion theory, developed in [15], can be applied directly even when $A(p)$ and $b(p)$ involve **nonlinear dependencies between parameters**. The key issue is to obtain sharp bounds for the difference between the true solution and the approximate solution, and sharp bounds for the contracting matrix. While linear dependencies in $A(p)$, $b(p)$ allow easy computation of sharp enclosures, the nonlinear dependencies hamper the solution, reducing the overall problem to a problem for sharp range estimation. Fortunately, the computer algebra environment of *Mathematica* provides tools for algebraic simplification of expressions, reducing thus the number of parameter occurrences causing overestimation. All iterative solvers, based on algebraic simplification (`SymmetricSolve[A, bp, parLst]` and `ParametricSSolve[Ap, bp, parLst]`) can be applied to linear systems involving either affine-linear, or nonlinear dependencies between parameters. An enhancement of the *Mathematica* interval tools by various methods for sharp range estimation will provide a further improvement in the corresponding parametric solvers towards sharper inclusion of the PSS in case of nonlinear dependencies.

The development of quality mathematical software for solving parametric linear systems, and our functions in particular, involve several other specific issues, like rigorous interval bounding, sharpness in contracting matrix enclosure, proper choice of epsilon inflation etc., that reflect the state-of-the-art in the implementation of verification algorithms.

2.3 Exploiting Monotonicity of the Parametric Solution.

It is well known that we can easily find the exact range of a function which is monotone w.r.t. all parameters [7]. That is why, exploiting the monotonicity of the solution $x(p) = A(p)^{-1}b(p)$ to (1) is favorable by some authors in finding the exact bounds of the parametric solution set [4], [5]. To make use of this property, however, it should be rigorously proven.

Computer algebra environments possess facilities for analytic solution, symbolic differentiation and algebraic simplification, that combined with interval arithmetic evaluation present

excellent tools for proving monotonicity of the parametric solution $x(p) = A(p)^{-1}b(p)$. We have developed a function `Monotonicity[Ap, bp, parLst]` which uses the computer algebra tools and interval arithmetic of *Mathematica* to give the monotonicity of the solution to a square parametric linear system `Ap.x = bp`, where the parameters and their interval values are specified by a list of transformation rules `parLst`. The output is an $(n \times k)$ matrix (n – number of the solution components; k – number of parameters) with elements from the set $\{-1, 0, 1\}$ denoting the sign of the corresponding partial derivative. 0 means that the monotonicity type cannot be determined. This function can be applied to parametric linear systems involving either affine-linear dependencies or nonlinear dependencies between parameters. For some problems, this function finds the monotonicity faster than the numerical tools, presented below.

It seems that J. Rohn is the first who has tried to derive numerical proof for the monotonicity of the parametric solution [12]. All published attempts to exploit the monotonicity of the parametric solution components consider the special case when all the components are monotone w.r.t. all the parameters [4], [5]. Some practical problems, however, may have solution components that are monotone w.r.t. some of the parameters and non-monotone with respect to others. Monotonicity properties of the parametric solution can be used to reduce the number of parameters and thus to sharpen the solution set enclosure.

Consider the parametric system (1), factorized by the parameters

$$\left(A_0 + \sum_{\nu=1}^{k} A_\nu p_\nu\right) x \;=\; b_0 + \sum_{\nu=1}^{q} b_\nu p_\nu, \qquad p_\nu \in [p_\nu] = [p_\nu^-, p_\nu^+] \tag{4}$$

where $A_\nu = (\lambda_{ij\nu}) \in \mathbb{R}^{n\times n}$ and $b_\nu = (b_{i\nu}) \in \mathbb{R}^n$, $(\nu = 0, \ldots, k)$. The goal is prove monotonicity of x_i, $1 \le i \le n$, with respect to every p_ν, $1 \le \nu \le k$. For every fixed $\nu \in \{1, \ldots, k\}$, by taking partial derivatives $\dfrac{\partial}{\partial p_\nu}$ on both sides of (1), we come to the equation

$$A(p)\frac{\partial x}{\partial p_\nu} \;=\; \frac{\partial b(p)}{\partial p_\nu} \;-\; \frac{\partial A(p)}{\partial p_\nu} \cdot [x^*], \tag{5}$$

where $[x^*] \supseteq \Sigma^p$ is an enclosure of the parametric solution set. Solving (5) we can rigorously enclose the corresponding partial derivative of the solution components, and thus, through their signs, to prove the monotonicity type of the parametric solution components with respect to all parameters. Let us suppose that for fixed i, $1 \le i \le n$

$$L_- = \{\nu \mid \text{Sign}\left[\frac{\partial x_i}{\partial p_\nu}\right] = 1\}, \qquad\qquad L_+ = \{\nu \mid \text{Sign}\left[\frac{\partial x_i}{\partial p_\nu}\right] = -1\}.$$

If $L_- \cup L_+ = \{1, \ldots, k\}$, the exact bounds x_i^-, x_i^+, $1 \le i \le n$, of the PSS can be obtained as

$$\begin{aligned} x_i^- &= (A_0 + \sum_{\nu\in L_-} A_\nu p_\nu^- + \sum_{\nu\in L_+} A_\nu p_\nu^+)^{-1} \cdot (b_0 + \sum_{\nu\in L_-} b_\nu p_\nu^- + \sum_{\nu\in L_+} b_\nu p_\nu^+) \\ x_i^+ &= (A_0 + \sum_{\nu\in L_-} A_\nu p_\nu^+ + \sum_{\nu\in L_+} A_\nu p_\nu^-)^{-1} \cdot (b_0 + \sum_{\nu\in L_-} b_\nu p_\nu^+ + \sum_{\nu\in L_+} b_\nu p_\nu^-) \end{aligned} \tag{6}$$

If $L_- \cup L_+ \neq \{1, \ldots, k\}$ and $L_- \cup L_+ \neq \emptyset$, the monotonic parameters can be adjusted, so that we should solve two parametric linear systems with reduced number of parameters for every i, $1 \le i \le n$ in order to get a sharper enclosure for the parametric solution set.

To prove monotonicity properties of a parametric solution is often more difficult than finding an enclosure to the PSS itself. Here we briefly discuss some implementation problems and present our concept, based on some comparisons of the performance and cost-efficiency of various techniques. The method for numerical proof of solution monotonicity requires an enclosure of the parametric solution set (see (5)). The recommended, by now, method for finding such

an enclosure is the (preconditioned) Gaussian algorithm. Although the interval Gaussian elimination is, may be, the fastest method, it has two drawbacks. First, the interval Gaussian algorithm is not always feasible and collapses in double precision for almost any matrix of dimension greater than 70. The major drawback is that the enclosure that this method (and every other non-parametric method) provides is too rough since it encloses the corresponding nonparametric solution set. We propose to find enclosures of the PSS and of the derivatives (5) by parametric residual iteration based on numerical simplification. Since we usually want to prove solution monotonicity for problems involving comparatively large tolerances for the parameters, the method will not succeed in finding the monotonicity type at once. The only way to get $L_- \cup L_+ \neq \emptyset$ (or $L_- \cup L_+ = \{1, \ldots, k\}$ for entirely monotone solutions) for large intervals is by iterative subdivision of the intervals for the parameters.

We have developed the following functions for finding monotonicity type of the solution components to a parametric linear system `Ap.x = bp`, where the parameters and their interval values are specified by a list of transformation rules `parLst`.

```
LUMonotonicity[Ap, bp, parLst, Subdivision]   LUMonotonicity[Ap, bp, parLst]
PMonotonicity[Ap, bp, parLst, Subdivision]     PMonotonicity[Ap, bp, parLst]
NMonotonicity[Ap, bp, parLst, Subdivision]     NMonotonicity[Ap, bp, parLst]
```

All the functions have the same output as the function `Monotonicity` described above. Function `LUMonotonicity` is based on interval Gaussian elimination. By analogy with the iterative solvers, `PMonotonicity` is based on symbolic differentiation, algebraic simplification and the parametric Rump's residual iteration to be applied to parametric linear systems involving either affine-linear dependencies or nonlinear dependencies between parameters. Function `NMonotonicity` does entirely numerical computations.

The symbol `Subdivision`, used as fourth argument of the functions `LUMonotonicity`, `PMonotonicity`, and `NMonotonicity`, specifies finding the solution monotonicity of a parametric linear system by successive subdivision of the input intervals for the parameters. Two options are associated with these functions involving `Subdivision` argument. `SubdivisionLimit` specifies the maximum number of subdivisions, that can be applied to the input interval value for each parameter, in finding solution monotonicity. `Components` specifies which of the solution components to be considered. `Components`-value should be a nonempty list of numbers. The default setting for these options is {`SubdivisionLimit -> 100, Components -> All`}.

2.4 A General Hybrid Approach

It was shown that the parametric Rump's residual iteration possesses an excellent performance for parameters with small tolerances and does not give very sharp enclosures for parameters with large tolerances [8], [15]. On another side, some practical problems require very sharp solution set enclosure. The latter is crucial for dynamical problems described by parametric linear systems involving uncertainties. That is why we propose the following general hybrid approach for sharp solution enclosures.

1. Try to prove monotonicity of the solution components w.r.t. each of the parameters.

   1.1 If every x_i, $(1 \leq i \leq n)$ is monotone w.r.t. all p_ν, $(1 \leq \nu \leq k)$,
   then find rigorously, and componentwise by (6)

   - the exact $\square\, \Sigma^p$ in exact arithmetic; or
   - very sharp $[Y] \supseteq \square\, \Sigma^p$ by non-parametric residual iterations in floating point arithmetic;

   Stop.

1.2 If $\exists\ x_i$, $(1 \le i \le n)$, which is monotone w.r.t. some p_ν, $(1 \le \nu \le k)$, then
– adjust the monotone parameters componentwise; Goto 2.

1.3 If all x_i, $(1 \le i \le n)$ are non-monotone w.r.t. all p_ν, $(1 \le \nu \le k)$, then Goto 3.

2. Apply parametric residual iteration componentwise; Fix a global sharpness ε, $0 \le \varepsilon \le .99$; Use inner estimation to measure the component sharpness.
– If sharpness $\geqq \varepsilon$ for $\forall\ x_i$, $1 \le i \le n$, then Stop; else Goto 3.

3. Refinement by Subdivision. Fix a sharpness ε, $0 \le \varepsilon \le .99$;

3.1 Subdivide the parameter box into subboxes with fixed tolerance, so that $[p_\nu] = \cup_{j_\nu}[p_{\nu j_\nu}]$ with tolerance$([p_{\nu j_\nu}]) \leqq t^{(i)}\%$ (say $t^{(1)} = 1\%$)

3.2 Compute new enclosure $[Y^{(i)}]$ (with inner estimation) as union of the enclosures on every subbox.
– If $[Y^{(i)}] \subseteq [Y^{(i-1)}]$ and sharpness$([Y^{(i)}]) \geqq \varepsilon$, then Stop;
– elseIf IterationLimit, then Stop;
else Goto 3.1 with tolerance $t^{(i+1)}) < t^{(i)}\%$.

The proposed hybrid approach provides sharp bounds for the solution set to parametric systems whose solution is monotone w.r.t. some/all parameters. If the parametric solution is monotone w.r.t. all the parameters, we practically compute the exact hull and the floating-point enclosure has the sharpness of the verification procedures for the solution of the corresponding point linear systems. The sharpness of the enclosure, provided by the parametric fixed-point iteration, depends on the number of parameters and the value of the interval tolerances. If the parametric solution is monotone w.r.t. only some of the parameters, adjusting the latter provides sharper solution set enclosures. Subdivision of the parameter intervals is an universal method for getting sharper enclosure for parametric solution which is not monotone w.r.t. the parameters.

The first two steps of the general hybrid approach are implemented in a function `MonotonicPISolve`, which is part of the package under consideration. `MonotonicPISolve[Ap, bp, parLst, sgns]` *rigorously* computes sharp bounds for the solution set of the parametric linear system `Ap.x = bp`, where the parameters and their intervals are specified by the list of transformation rules `parLst`, assuming that the solution is monotone w.r.t. some parameters and the type of monotonicity is specified componentwise by the matrix `sgns`. Naturally, the monotonicity matrix `sgns` can be the output of some of the functions `NMonotonicity, PMonotonicity, LUMonotonicity`, or `Monotonicity`. The three options `InnerEstimation, Refinement`, and `Statistics`, associated to the iterative solvers, are associated to `MonotonicPISolve` too. One more option `Method`, with values `Verified` (set by default) and `Rational`, is associated to `MonotonicPISolve` to guide the computing method. `Method -> Rational` requires solving point linear systems by the kernel function `LinearSolve` which does exact computations on exact data. If some of the input data are not exact, the results will not be rigorous. By default, all point linear systems are solved by the enclosure non-parametric `ILinearSolve` function. Even set to `True`, `InnerEstimation` is active only if a parametric linear system has to be solved, that is `sgns` involves zero. The monotonic solver acts as an expert system automatically choosing between the available linear system solvers, basing on the monotonicity matrix `sgns` and the value of the option `Method`. At present, the monotonic solver works on parametric linear systems involving only affine-linear dependencies. An additional option, specifying the type of the dependencies, could be associated to the monotonic solver.

Miscellaneous. Two functions `Sharpness` and `Overestimation` are provided for an user convenience. `Sharpness[int1, int2]` is used to measure the quality of an outer interval approximation `int2`, provided that `int1` is a corresponding inner approximation. $0 \leq$ `Sharpness[int1, int2]` ≤ 1, for `int1` $\subseteq$ `int2`, and `Indeterminate` otherwise. `Overestimation[int1, int2]` gives the percentage by which the interval `int2`, overestimates the interval `int1`, provided that `int1` $\subseteq$ `int2`, and `Indeterminate` otherwise. `ParametricToNumeric[expr, parLst]` is a function that converts the symbolic representation of a parametric matrix/vector `expr`, whoose elements are affine-linear functions of some parameters, specified by `parLst`, into corresponding numerical matrices to be exported to and used by other environments and programming languages.

3 Conclusion

A *Mathematica* package supporting tools for solving parametric and non-parametric linear systems involving uncertainties is presented. It includes a variety of functions, implementing different interval techniques, that help in producing sharp and rigorous results in validated interval arithmetic. The package provides a necessary background for further exploration, comparisons and prototyping, and gives the application scientists some indispensable tools for solving parametric interval linear systems.

Our next efforts are directed towards further expanding the functionality of the package, improving the performance of some functions and deploying the available functionality over the Web. A parametric solver already can be accessed via the webComputing service at `http://cose.math.bas.bg/webComputing/`.

References

[1] G. Alefeld, V. Kreinovich, G. Mayer, The shape of the solution set for systems of interval linear equations with dependent coefficients, Math. Nachr. 192 (1998), 23-36.

[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] C. Jansson, Interval linear systems with symmetric matrices, skew-symmetric matrices and dependencies in the right hand side, Computing 46 (1991), 265-274.

[4] M. Jasiński and A. Pownuk, Modelling of heat transfer in biological tissue by interval FEM, CAMES 7 (2000), 551-558.

[5] L. Kolev, Outer solution of linear systems whose elements are affine functions of interval parameters, Reliable Computing 8 (2002) 6, 493-501.

[6] R. L. Muhanna and R. L. Mullen, Uncertainty in mechanics problems — interval-based approach, Journal of Engineering Mechanics 127 (2001) 6, 557-566.

[7] A. Neumaier, Interval Methods for Systems of Equations. Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1990.

[8] 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.

[9] E. D. Popova, Quality of the solution sets of parameter-dependent interval linear systems, ZAMM 82 (2002) 10, 723-727.

[10] E. D. Popova; Ch. Ullrich, Directed interval arithmetic in Mathematica: Implementation and applications, TR 96-3, U. Basel, 1996. (www.math.bas.bg/~epopova/papers/tr96-3.ps)

[11] J. Rohn, Systems of Linear Interval Equations, LAA 126 (1989), 39-78.

[12] J. Rohn, Interval linear equations with dependent coefficients, Reliable Computing mailing list, March 2002; personal communication, April 1999.
(http://www.ms.mff.cuni.cz/~rohn/letter/letter.ps)

[13] 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.

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

[15] S. Rump, Verification methods for dense and sparse systems of equations, in: Topics in Validated Computations, ed. J. Herzberger, Elsevier Science B. V., 1994, pp. 63-135.

[16] S. Wolfram, The Mathematica Book, 4th ed., Wolfram Media/Cambridge U. Press, 1999.