МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2002
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2002
Proceedings of Thirty First Spring Conference of
the Union of Bulgarian Mathematicians
Borovets, April 3–6, 2002

SOLVING PARAMETRIC INTERVAL LINEAR SYSTEMS BY *MATHEMATICA*[*]

Evgenija D. Popova

A *Mathematica* package for solving general and parametric linear systems with imprecise data is presented. What we gain by the interaction between computer algebra and validated computing is discussed.

1. The Problem. Consider a system of linear interval equations $[A] \cdot x = [b]$, i.e. the system of the type

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad (i = 1, \ldots, n), \tag{1}$$

where a_{ij}, b_i can take arbitrary values from given intervals $[a_{ij}]$, $[b_i]$. These systems are common in practice, when due to measurement imprecision, approximation, or uncertainty, the exact values of the coefficients are not known. By a *solution set* of such system we mean $\Sigma^g = \Sigma([A],[b]) := \{x \in \mathbb{R}^n \mid \exists A \in [A], \exists b \in [b] : Ax = b\}$. This description of the solution set makes sense if the uncertainties in all coefficients are independent (the values of a_{ij} and b_i are not a priori related), which is not the case for the most practical situations. The simplest example of dependencies is when the matrix is symmetric or skew-symmetric. In most engineering design problems, models in operational research, linear prediction problems, etc. [1], [2], [5], [7], 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. Let us denote all the parameters that influence the coefficients by p_ν, $\nu = 1, \ldots, k$. Then the coefficients a_{ij}, b_i depend on these parameters

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

so that we have a parametric linear system $A(p) \cdot x = b(p)$ with affine-linear dependencies in the coefficients. $\lambda_{ij}, \beta_i \in \mathbb{R}^{k+1}$ $(i, j = 1, \ldots, n)$ are numerical vectors and the parameters p_ν can take arbitrary values from the given intervals $[p_\nu]$, $(\nu = 1, \ldots, k)$. The parametric solution set (PSS)

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

[*]This work was supported by the Bulgarian National Science Fund grant No. I-903/99 and the Swiss NSF Co-operation Agreement No. 7IP 65642.

is usually a subset of the symmetric solution set and the general non-parametric solution set (GSS). Since the solution sets have a complicated structure which is difficult to find, we look for the interval hull $\square\Sigma := [\inf \Sigma,\ \sup \Sigma]$, whenever Σ is a nonempty bounded subset of $\mathbb{R}^n$, or for an interval enclosure of $\square\Sigma$.

Below we present a *Mathematica* [8] package `IntervalComputations'LinearSolve` which offers a variety of functions for solving parametric and non-parametric interval linear systems. Supposing that the package is loaded, some simple examples will illustrate the usage of these functions. The impact of computer algebra on the solution of interval parametric problems will be discussed.

2. Exact Bounds on the Solutions. `HullSolutionSet[A, b]` gives the exact interval hull of the GSS to a non-parametric interval system with numerical interval matrix `A` and numerical interval vector `b`. The computational procedure is based on an algorithm by J. Rohn [5] solving only 2^{2n} point linear systems. `AllPoints` is a symbol, used as last optional argument for the function `HullSolutionSet`. `AllPoints` specifies a full computational procedure and an output in the form $\{\{\text{hull}\}, \{\text{points}\}\}$, where {hull} is the interval hull and {points} is a list of the solutions to all 2^{n^2} vertex linear systems. Matrix and/or vector entries can be either intervals or elements from the domain `Real`. *Mathematica* [8] is the only environment that supports exact and variable precision interval arithmetic, so that all functions will produce exact interval results on exactly specified arguments.

Rohn's algorithm can detect a singular interval matrix.

```
In[2] := A =
{{Interval[{1,2}], Interval[{3,4}]}, {Interval[{5,6}], Interval[{7,8}]}};
       b = {Interval[{0,1}], Interval[{0,1}]};
In[3] := HullSolutionSet[A, b];
HullSolutionSet::sing: Singular matrix encountered.
```

While the algorithm, based on all possible combinations of the end-points, cannot do so in the general case, unless some of the end-point matrices is singular.

```
In[4] := HullSolutionSet[A, b, AllPoints][[1]]
Out[4] = {Interval[{-7,8}], Interval[{-5,5}]}
```

`HullSolutionSet[Ap, bp, rls]` computes the exact hull of the PSS to a parametric system with matrix `Ap`, right-hand side `bp` and parameters varying within given numerical intervals specified by a list of transformation rules `rls`. The computing method 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. `AllPoints` can be used as optional argument for parametric systems, too.

Monotonicity can also be used even when the solution is not monotonic provided its behavior is sufficiently well known. We have proven in [3] some sufficient conditions for parametric solution set having the same *quality* ($\square\Sigma^p = \square\Sigma^g$) as the solution set to the corresponding non-parametric system $A([p]) \cdot x = b([p])$.

`ExactBounds[Ap, bp, rls]` checks which bounds of the PSS coincide with the bounds of the corresponding GSS. The output is in the form {{{`inf-bds`}, {`sup-bds`}}, {`hull`}}, wherein {`inf-bds`} and {`sup-bds`} are lists with the numbers of the coinciding bounds. The input of the parametric vectors and matrices is in convenient mathematical notations. Fig. 1 represents the parametric (gray region) and the corresponding general (dashed line) solution set for the linear system of `In[5]`.

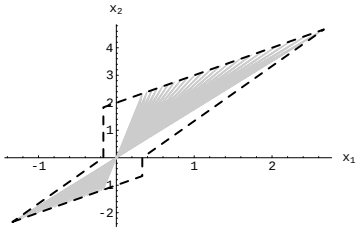

Fig. 1

```
In[5] := m = {{-1, p1}, {p1+1, p2}};  v = {p3, 1/3  p3};
tr = {p1->Interval[{1,2}], p2->Interval[{-1,0}], p3->Interval[{-1,2}]};
In[6] := ExactBounds[m, v, tr]
Out[6]= {{{1,2}, {1,2}}, {Interval[{-4/3, 8/3}], Interval[{-7/3, 14/3}]}}
```

3. Computer Algebra in Proving Monotonicity. Computer algebra saves manual amounts, decreases the probability of blunders, and the necessity of expertise. At present, simplification is the "killer application" of computer algebra (application that everyone wants but only one can). Although symbolic differentiation leads to lengthly formulas for the derivatives, the length of these formulas can be greatly reduced by algebraic manipulations. Our next example shows how the algebraic approach can yield sharper results, by reducing the dependency problem, in proving monotonicity of the analytic solution $x(p) = A(p)^{-1}b(p)$ to parametric interval linear systems.

Consider a system arising in the analysis of a resistive electrical circuit [3]

$$\begin{pmatrix} g_1+g_6 & -g_6 & 0 & 0 & 0 \\ -g_6 & g_2+g_6+g_7 & -g_7 & 0 & 0 \\ 0 & -g_7 & g_3+g_7+g_8 & -g_8 & 0 \\ 0 & 0 & -g_8 & g_4+g_8+g_9 & -g_9 \\ 0 & 0 & 0 & -g_9 & g_5+g_9 \end{pmatrix} \cdot x = \begin{pmatrix} 10 \\ 0 \\ 10 \\ 0 \\ 0 \end{pmatrix},$$

where the parameters are subject to tolerances $[g_i] = [1-\delta,\ 1+\delta],\ i = 1, ..., 9$. We shall prove monotonicity of the solution for different values of the tolerances δ varying from 1 to 10% of the nominal value. Let `m` denote the parametric matrix, `b` denote the right-hand side vector; `tr1`, `tr5` and `tr10` denote the *Mathematica* transformation rules specifying the parameter values corresponding to tolerances of $1, 5$ and 10%, resp. Then

```
In[10] := x= Inverse[m]^(-1) b;  dx= Table[D[x[[j]], g[i]], {i,5},{j,9}];
```

Evaluating the derivatives for 1% tolerance intervals, `Map[IntervalMemberQ[#, 0]&, dx /.  tr1, {2}]` shows that most of the derivatives contain zero even for small tolerances. *Mathematica* provides a variety of functions for converting expressions from one form to another. Since `x` contains quotients, we have chosen to use a function making a sum of terms into a single rational function. `Together[expr]` puts terms in a sum over a common denominator, and cancels factors in the result.

```
In[11] := dx = Together[dx];
```

This rearrangement was sufficient to prove monotonicity of the solution for 1% tolerances. For 5% tolerances, only the derivatives with respect to $g[7]$ contain zero, and for 10% tolerances – the derivatives with respect to $g[6]$ and $g[7]$ contain zero.

```
In[12] := SameQ[Sequence @@ Denominator /@ dx[[7]]]
Out[12] = True
```

`Out[12]` shows that all the components of `dx[[7]]` have the same denominator. A simple view shows that the expression of the denominator involves only summation of nonzero intervals with equal signs. That is why, it is necessary to rearrange and evaluate only the numerators. `Simplify` tries to find the simplest form of an expression by applying various standard algebraic transformations.

```
In[13] := Sign /@ Simplify[Numerator /@ dx[[7]]] /. tr5
Out[13] = {1,1,-1,-1,-1}
In[14] := Sign /@ Simplify[Numerator /@ dx[[6]]] /. tr10
Out[14] = {-1,1,1,1,1}
```

Applying `Simplify` was not enough to prove monotonicity of the solution with respect to `g[7]` for 10% tolerances. Getting expressions into the form we want is something of an art. In most cases, it is best simply to experiment, trying different transformations until you get what you want.

```
In[18] := t = Simplify[Numerator /@ dx[[7]]];
In[19] := t[[1]] === t[[1,1]] t[[1,2]] t[[1,3]](t[[1,4,1]] + t[[1,4,2]])
Out[19] = True
In[20] := (t[[1,1]]  t[[1,2]]  t[[1,3]]
      (t[[1,4,1]] + Apart[Expand[t[[1,4,2]]], g[8]])) /. tr10
Out[20] = Interval[{21.1054, 1653.77}]
```

`Simplify` helps us to reveal the structure of an expression (`In[19]`) and to identify terms involving summands with different signs (`t[[1,4,2]]`). In expressions with several variables, we can use `Apart[expr, var]` to do partial fraction decompositions with respect to different variables. `Apart` with respect to `g[8]` turned out to be sufficient to prove the monotonicity.

The above example demonstrates the advantage of symbolic differentiation plus algebraic manipulation versus automatic differentiation (AD) in proving monotonicity. It is possible to reduce the dependency problem by minimizing the occurrences of variables. There is no need to evaluate the range of the function (as in AD) but only the derivatives. Sometimes, it is even sufficient to evaluate only a part of the derivatives expression. Beside getting sharper results, this approach is more flexible. It could also generate effective codes with respect to runtime and memory cost.

2. Iterative Solvers. `GeneralSolve[A, b, opts]` and `DependencySolve[Ap, bp, rls, opts]` are functions for iterative enclosure of the GSS, resp. the PSS. Both functions are based on the iterative methods by Rump [8]. Several options (`opts`) can be used to control the computational process. `WorkingPrecision` specifies the number of digits to use in internal computations and can be `Exact`, `$MachinePrecision` (default),

or user specified. `EnclosureRefinement` requires an additional refinement of the enclosure. `SharpnessEstimation` requires computing of component-wise inner approximation of the solution set in order to estimate the degree of sharpness of the outer enclosure.

`SharpnessEstimation` is unique for our implementation and requires loading of a *Mathematica* package for generalized interval arithmetic [5]. 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. Due to the algebraic properties of the arithmetic on proper and improper intervals, [5] supports inwardly rounded interval arithmetic at no additional cost.

A key feature of the algorithm for enclosing the PSS is a sharp enclosure of $Z = R^{-1} \cdot (b(p) - A(p)\tilde{x})$ for $p \in [p]$, where R is the mid-point of $A([p])$ and $\tilde{x}$ is the solution of the mid-point system. A sharp enclosure can be obtained if every parameter occurs at most once in every component of Z, that is provided by the formula

$$[Z]_i \quad := \quad (\sum_{j,l=1}^{n} \{R_{ij} \cdot (\beta_j - \tilde{x}_l \cdot \lambda_{jl})\}^{\top}) \cdot [p], \qquad i = 1, \ldots, n. \tag{4}$$

The formula can be derived either by hand for each particular parametric system, or automatically by algebraic simplification in a suitable computer algebra environment. Even we have obtained the computing formula (4) somehow, the implementation of the corresponding iterative method in an environment not supporting symbolic entries would require input of $n(k+1)(n+1)$ numerical values corresponding to the numerical vectors λ_{ij}, β_i (see (2)). The latter would be an exhaustive human effort especially for large systems and complicated dependencies. An advantage of the symbolic manipulation environment of *Mathematica* is the ability to use symbolic mathematical notations to input the parametric matrix and the right-hand side, and to derive the necessary computing formula of the algorithm automatically, regardless of the particular parametric problem.

Our package provides also a function `ExpressionToMatrix[data, pars]` converting a symbolic matrix (or vector) `data` which components are affine-linear expressions in given parameters `pars` into a 3-dimensional (resp. 2-dim) numerical matrix of the coefficients $\alpha_{ij\nu}$. The latter could be then exported into a data file to be used by any other programming environment.

```
In[22] := m = {{2, g[1] + g[2]}, {Sqrt[2] - t + 1, g[2]}};
      ExpressionToMatrix[m, {t, g[1], g[2]}]
Out[22] = {{{2,0,0,0}, {0,0,1,1}}, {{1+Sqrt[2],-1,0,0}, {0,0,0,1}}}
In[23] := Export["data.dat", Flatten[N[Out[22]], 1]]
```

Interval Gauss-Seidel iteration for parametric linear systems is introduced in [2] for improving the enclosures, obtained by the iterative method, whenever they are not good enough. Functions `GaussSeidelIteration` performing the corresponding iterative process for general non-parametric and for parametric interval linear systems, are provided by the package. In case of parametric systems, generalized interval arithmetic [4] should be used to eliminate the dependency problem in generating the expressions for the parametric Gauss-Seidel operator. Some numerical experiments, performed by the above

functions, are presented in [2].

5. Conclusion. Approaching to linear systems with uncertain parameters, the integration of symbolic and self-validating numerical computation is shown. Key features of symbolic-algebraic computations are used for convenient input and handling of parametric problems. The power of *Mathematica* to support rigorous exact and/or variable precision interval computations, the functionality of a generalized interval arithmetic package for computing inner approximations and reducing the dependency problem, as well as the tools provided by the presented interval problem solving package, make a suitable environment for efficient solving of real-life parametric problems with uncertainty.

We gave only a short overview of the possibilities, which follow from coupling various techniques of computation. A complete description of the package with many numerical examples illustrating its usage will be put on the Web soon.

REFERENCES

[1] B.R. Barmish. New Tools for Robustness of Linear Systems. McMillan, N.Y., 1994.
[2] E. Popova. On the Solution of Parametrised Linear Systems. In: W. Kraemer et al. (Eds.): Scientific Computing, Validated Numerics, Interval Methods, Kluwer Acad. Pub., 2001, 127-138.
[3] E. Popova. Quality of the Solution Sets of Parameter-Dependent Interval Linear Systems. *ZAMM* (to appear).
[4] E. Popova, Ch. Ullrich. Directed Interval Arithmetic in Mathematica: Implementation and Applications. TR 96-3, U. Basel, 1996, 1–56. (www.math.bas.bg/~epopova/directed.html)
[5] S.S. Rao, L. Berke. Analysis of Uncertain Structural Systems Using Interval Analysis. *AIAA Journal*, Vol. 35, No. 4, 1997, 727–735.
[6] J. Rohn. Systems of Linear Interval Equations. *LAA* **126** (1989), 39–78.
[7] S. Rump. Verification Methods for Dense and Sparse Systems of Equations. In: J. Herzberger (Ed.): Topics in Validated Computations. Elsevier Science B. V., 1994, 63–135.
[8] S. Wolfram. The Mathematica Book, 4th ed., Wolfram Media/Cambridge U. Press, 1999.

Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Acad. G. Bonchev Str., bl. 8
113 Sofia, Bulgaria
e-mail: epopova@iph.bio.bas.bg

РЕШАВАНЕ НА ПАРАМЕТРИЧНИ ИНТЕРВАЛНИ ЛИНЕЙНИ СИСТЕМИ С *MATHEMATICA*

Евгения Д. Попова

Представен е пакет за решаване на параметрични и общи интервални линейни системи в системата *Mathematica*. Разглеждат се преимуществата от комбинирането на компютърно-алгебрични с интервално-аритметични методи.