Finding and Verifying All Solutions of a System of Nonlinear Equations Using Public Domain Software

Max E. Jerrell
Northern Arizona University
Flagstaff, AZ 86011-5066
e–mail max.jerrell@nau.edu

Wendy A. Campione
Northern Arizona Univeristy
e–mailwendy.campione@nau.edu

Abstract

Economic models are often stated as systems of nonlinear equations, for example general equilibrium models, game theory models, and macroeconomic models. A critical issue is wether the model has a solution and is the solution unique. Interval arithmetic is an arithmetic that operates on intervals rather than point values. It can be used to find all solutions of a system of nonlinear equations over a specified region. We present arguments demonstrating that this arithmetic is capable of determining existence and uniqueness. We then use a public domain software package on several simple economic examples.

1 Introduction

Hansen [6] cites an example given by Dennis and Schnabel [4] concerning the function $f_1(x) = x^4 - 12x^3 + 47x^2 - 60x$ where they state that "It would be wonderful if we had a general purpose computer routine that would tell us: 'the roots of $f_1(x)$ are $x = 0, 3, 4,$ and $5; \cdots$'. It is unlikely that there will ever be such a routine. In general, the questions of existence and uniqueness–does a given problem have a solution and is it unique?–are beyond the capabilities one can expect of algorithms that solve nonlinear problems.."

Interval arithmetic is a mathematical method that can be used to produce the results Dennis and Schnabel desire. The solution to this problem using interval arithmetic is shown in Sec 6. *Globsol* is a public domain package written in Fortran 90 that includes an interval as a user defined data type and the software methods needed to implement the rules of interval arithmetic. This software is part of the code made available by the *Globsol project* headed by R. Baker Kearfott (University of Louisiana at Lafayette), George Corliss (Marquette University), Chenyi Hu (University of Houston–Downtown), Michael Schulte (Lehigh Univeristy) and Mark A. Stadherr (Notre Dame University). *Globsol* was designed primarily to be a global optimization package, but can be used to solve systems of nonlinear equations as well. A URL that points to the code for *Globsol* as well as other interval arithmetic packages is `http://cs.utep.edu/interval-comp/main.html`. The *Globsol* site can be found at `http://studsys.mscs.mu.edu/ globsol/`. In addition to the *Globsol* code, the authors also provide makefiles for the Sun Fortran compiler, the NAG compiler for the Sun operating systems, SGI with the MIPS compiler, the Compaq Fortran compiler for machines using Microsoft operating systems, and the NAG/Salford Fortran compiler for Microsoft operating systems.

Two of the working papers available at the *Globsol* web site are applications of the program to financial problems. Working note No. 6 is portfolio optimization problem and working note No. 7 is an application of portfolio management subject to currency rate risk. These papers provide substantial detail illustrating much of the the development of the theory of the problem and then showing how it coded for *Globsol*. An explanation of the theory of the *Globsol* routines can be found in Kearfott [8] .

Sec 2 presents the rules of interval arithmetic, Sec 3 gives some useful results of interval analysis and Sec 4 outlines an algorithm that can used for locating and verifying the uniqueness of the solutions of a system of nonlinear equations. Sec 5 is a discussion of the capabilities and use of *Globsol*. Sec 6 applies this algorithm to the problem mentioned by Dennis and Schnabel. Sects 7 and 8 apply the algorithm to multivariate examples. Sec 9 applies the method to a small classical macroeconomic model. Sec 10 applies the algorithm to a three consumer, three good general equilibrium model and Sec 11 considers a general equilibrium model with multiple roots. Sec 12 is devoted to conclusions.

2 The rules of interval arithmetic

The modern form of interval arithmetic was introduced by R. E. Moore [10]. This section presents some of the concepts of Moore's arithmetic and introduces the notation used in the remainder of this work.

Upper case letters, such as X, indicate intervals while lower case letters, such as x, represent point values. An interval is defined to be the real compact interval $X = [a, b]$ where a and b indicate the lower and upper endpoints. If $X = [a, b]$ and $Y = [c, d]$ are intervals, the binary operations for interval arithmetic are

$$\begin{aligned}
X + Y &= [a + c, b + d], \\
X - Y &= [a - d, b - c], \\
X \cdot Y &= [\min(ac, ad, bc, bd), \\
&\quad \max(ac, ad, bc, bd)], \\
X/Y &= [a, b] \cdot [1/d, 1/c] \text{ if } 0 \notin Y.
\end{aligned}$$

If $0 \in Y$ then the result for division requires an *extended interval arithmetic*. So if $0 \in Y$

$X/Y =$

$$\begin{cases} [b/c, +\infty) & \text{if } b \le 0 \text{ and } d = 0, \\ (-\infty, b/d] \cup [b/c, +\infty) & \text{if } b \le 0 \\ & \text{and } c < 0 < d, \\ (-\infty, b/d] & \text{if } b \le 0 \text{ and } c = 0, \\ (-\infty, a/c] & \text{if } a \ge 0 \text{ and } d = 0, \\ (-\infty, a/c] \cup [a/d, +\infty) & \text{if } a \ge 0 \\ & \text{and } c < 0 < d, \\ [a/d, +\infty) & \text{if } a \ge 0 \text{ and } c = 0, \\ (-\infty, +\infty) & \text{if } a < 0 \text{ and } b > 0, \end{cases}$$

and $X/0 = (-\infty, +\infty)$. The details of computing with infinite or semi-infinite intervals can be found in [13, 6, 8].

Some definitions that will be needed later are the *width*, $w(X) = b - a$ of an interval $X = [a, b]$ and the absolute value $\mid X \mid = max\{\mid a \mid, \mid b \mid\}$. Also let the midpoint of X be $m(X) = (b + a)/2$.

Interval evaluation of monotonic elementary functions is straightforward,

$$f(X) = [f(a), f(b)],$$

where $f(X)$ is a function evaluated over an interval $X = [a, b]$. For example,

$$\ln(X) = [\ln(a), \ln(b)].$$

Non–monotonic elementary functions (sine, cosine, etc.) are more difficult and will not be considered here. Non–monotonic functions that are constructed using monotonic elementary functions can be computed using the binary rules and the rule for elementary monotonic functions without any special treatment, as in

$$f(X) = \ln(\exp(X^2 - X)).$$

A *box* is a vector of intervals such as $X = [X_1, X_2, \ldots, X_n]$. The above rules can be applied element by element to boxes.

3 Two Useful Results

Moore [9] established the following two important results. One of these is that interval arithmetic is *inclusion monotonic*,

$$X \subset Y \text{ implies } f(X) \subset f(Y). \tag{1}$$

Rall [12] calls the second useful result the fundamental theorem of interval analysis.

$$x \in X \text{ implies } f(x) \in f(X). \tag{2}$$

This result says that the range of a function, $f(x)$, evaluated over some interval X, will always be a subset of the interval computation $f(X)$. The fundamental theorem implies that if $0 \notin f(X)$ then X cannot contain a zero of f.

4 An Interval Newton Method

Moore [9] also derived an interval Newton method for the solution of a system of nonlinear equations based on the mean value theorem. Moore's method is presented here for a function of a single variable for simplicity; the results can be extended to the multivariate case. The mean value theorem is

$$f(x) - f(x^*) = (x - x^*)f'(\xi),$$

where $x \leq \xi \leq x^*$. Suppose $f(x^*) = 0$, then

$$x^* = x - \frac{f(x)}{f'(\xi)}.$$

Let X be an interval containing x and x^*, then because $x \leq \xi \leq x^*$, $\xi \in X$. Using the fundamental theorem (2), $f'(\xi) \in f'(X)$ and

$$x^* = x - \frac{f(x)}{f'(\xi)} \in x - \frac{f(x)}{f'(X)},$$

so $x^* \in N(x, X)$ where

$$N(x, X) = x - \frac{f(x)}{f'(X)}.$$

Assume that for the moment that $0 \notin f'(X)$ so the difficulties of an extended interval arithmetic can be avoided. Now any zero of f in X is also in $N(x, X)$ and hence is in the intersection $N(x, X) \cap X$. $N(x, X)$ has the following properties:

- **(a)** Every zero $x^* \in X$ of f satisfies $x^* \in N(x, X)$,
- **(b)** If $N(x, X) \cap X = \emptyset$ then no zero of F exists in X, and
- **(c)** If $N(x, X) \subset X$, then a unique zero of f exists in X and hence in $N(x, X)$.

See [9, 6, 11] for proofs. If items (b) or (c) in the above list cannot be satisfied, f may have two or more zeros in X. The interval Newton method (for $n = 0, 1, 2 \ldots n$) is:

$$\begin{aligned} x_n &= m(X_n), \\ N(x_n, X_n) &= x_n - \tfrac{f(x_n)}{f'(X_n)}, \\ X_{n+1} &= X_n \cap N(x_n, X_n). \end{aligned} \tag{3}$$

If $0 \in f'(X_n)$ then $N(x_n, X_n)$ results in one or two semi-infinite intervals. The intersection $N(x_n, X_n) \cap X_n$, however, results in a single interval, the union of two finite intervals, or the empty set.

Hansen [6] gives the following algorithm (modified here to avoid some subtle technical issues of using floating point arithmetic):

1. Put the initial interval X_0 into a list L.
2. If L is empty, stop. Otherwise remove the first interval from L, call it X.
3. if $w(X) < \epsilon_X$ and $| f(X) | < \epsilon_F$ store X on a final list F.
4. Let $x = m(X)$. Evaluate $f'(X)$. If $0 \notin f'(X)$ go to 5, else go to 6.
5. Do one step of the interval Newton method. If the result is empty, then no zero exists in X, go to 2. If X is unchanged then X contains two or more zeros. In this case bisect X into two parts, X_1 and X_2 such that $X = X_1 \cup X_2$ and $X_1 \cap X_2 = \emptyset$. Store X_1 and X_2 in L then go to 2. Otherwise go to 3.
6. Do one step of the interval Newton method. The results will be the empty set, a single interval, or two disjoint intervals. If the result is empty go to 2. If the result is a single interval, call it X and go to 3. If the result is two disjoint intervals, store one of them in L, call the other one X, and go to 3.

In the algorithm ϵ_X and ϵ_f are stopping criteria. The intervals stored in the list F contain the zeros of f.

5 Using *Globsol*

Globsol uses automatic (computational) differentiation, interval arithmetic and numerical techniques that have been adapted to use these concepts, such as the interval Newton method mentioned above. The authors have written the software in such a fashion as to shield users from most of the details of the software.

The chief tasks for a user is to (a) write a main program that represents the system of equations to be solved and (b) to modify a configuration file that provides parameters to the program that control its execution. The latter file specifies such things as the length of time to allow the program to execute, how much printed output is to be generated, and what solution techniques are to be used. Further details regarding the configuration file will not be discussed here.

The main program uses a Fortran90 interface (called *overload* in the following code examples)to the definition of the interval arithmetic user defined type and to the interval arithmetic routines. The main program consists of the system of equations to be solved (as well as any equality or inequality constraints in optimization problems). The output of the main program is an internal representation of the problem called a *codelist* [8]. The codelist is then processed by a second program that uses automatic differentiation to produce a new codelist that contains derivative information. This second program is part of *Globsol* and the user has only has to submit the first codelist to this program.

The second codelist is then processed by a third program that finds the roots of the system of equations (or the global minimum for optimization problems). Again the user does not have to do anything other than give this third program the second codelist as input. In summary the steps are

Step 1. Write a program representing the problem to be solved. The output of this program will be an internal representation of the problem called a codelist.

Step 2. Process this codelist with a program that produces a new codelist that contains derivative information.

Step 3. Process the second codelist with a program that finds all solutions of the system of equations.

Examples of program representation of problems are presented later.

The interval Newton method requires interval derivative information. The authors of *Globsol* have used automatic differentiation techniques adapted to interval methods to compute this derivative information. Without this users would have to find analytical expressions for the derivatives and then code the appropriate interval expressions which would be a very laborious and error prone process. The authors have gone a step further and also provided for an automatic interval slope computations. Recall that the fundamental theorem implies that interval calculations will tend to lead to produce interval results that are wider than the range of a particular function. Hansen [5] was able to show that interval slopes tended to be narrower than interval derivatives and still enclose the range of a derivative. So replacing derivative information with slope information would tend to lead to sharper results with, say, the interval Newton method.

A large number of other specialized interval routines are contained in the *Globsol* package that cannot be discussed here. Details can be found at the previously mentioned web sites or in Kearfott's [8] book. Some examples of problems solved using *Globsol* are considered next.

6 Example: $f(x) = x^4 - 12x^3 + 47x^2 - 60x$

The problem cited by Dennis and Schnabel in Sec 1 is to find the zeros is to find the zeros in $f(x) = x^4 - 12x^3 + 47x^2 - 60x$. Recall that the zeros are located at $x = 0, 3, 4, 5$ all of which are found by

Globsol. The program for this problem is shown next followed by the programs output. The search space is $X = [-10^{20}, 10^{20}]$.

```
Program Hansen1
 use overload      !contains the definition of an interval
                   !and provides an interface to interval routines
 type(cdlvar), dimension(1) ::x
 type(cdllhs), dimension(1) ::f

 output_file_name = 'Hansen1.cdl'
 call initialize_codelist(x)
 f(1) = x(1)**4 - 12.0d0*x(1)**3 + 47.0d0*x(1)**2 - 60.0d0*x(1)
 call finish_codelist

End Program Hansen1
```

The output for the program follows. The first item shown is the search space for the problem. This is then followed by the CPU time required for solution. This research used the Compaq Fortran compiler and a 1.6 MHz Pentium computer running under Windows XP. Note that the computer time for the problem is quite small considering the size of the search space.

The next portion of the output reports the approximate location of the roots. The routine first reports a relatively small box that has been verified to contain a unique root. *Globsol* finds that there is a unique root in Box no. 1 which is the box $[0.3998D + 01, 0.4002D + 01]$. An approximate root is computed and the box further reduced until a "small" box known to include the root is computed. The statement `THERE WERE NO UNRESOLVED BOXES` means that no other roots exist in the search space.

```
Initial box coordinates:
[   -0.1000D-19,   0.1000D+21 ]
CPU time:   0.3125D-01

The following boxes have been verified to contain unique roots:

Box no.:          1
```

```
Box coordinates:
[    0.3998D+01,   0.4002D+01 ]
Level:           0
Box contains the following approximate root:
   0.4000D+01
Small box in which the root must lie:
[    0.4000D+01,   0.4000D+01 ]
Interval residuals over the small box:
[   -0.4210D-11,   0.4210D-11 ]
-------------------------------------------------


Box no.:           2
Box coordinates:
[    0.4987D+01,   0.5013D+01 ]
Level:           0
Box contains the following approximate root:
   0.5000D+01
Small box in which the root must lie:
[    0.5000D+01,   0.5000D+01 ]
Interval residuals over the small box:
[   -0.7734D-11,   0.7800D-11 ]
-------------------------------------------------


Box no.:           3
Box coordinates:
[    0.2995D+01,   0.3005D+01 ]
Level:           0
Box contains the following approximate root:
   0.3000D+01
Small box in which the root must lie:
[    0.3000D+01,   0.3000D+01 ]
```

```
Interval residuals over the small box:
[   -0.1867D-11,   0.1836D-11 ]
-------------------------------------------------


Box no.:           4
Box coordinates:
[   -0.3940D-23,   0.1316D-23 ]
Level:            0
An approximate root has not been computed.
Small box in which the root must lie:
[   -0.3940D-23,   0.1316D-23 ]
Interval residuals over the small box:
[    0.0000D+00,   0.0000D+00 ]
-------------------------------------------------


THERE WERE NO UNRESOLVED BOXES
```

7 A Multivariate Example

Burden and Faires [1] consider a problem where two species compete for the same food supply. The number of each species alive at time t is $x_1(t)$ and $x_2(t)$. The population of each species is described by

$$\dot{x}_1(t) = x_1(t)[4 - 0.0003x_1(t) - 0.0004x_2(t)]$$

and

$$\dot{x}_2(t) = x_2(t)[2 - 0.0002x_1(t) - 0.0001x_2(t)].$$

The two populations will be at equilibrium when

$$\dot{x}_1(t) = \dot{x}_2(t) = 0.$$

Four solutions are shown below. The last three solutions arise when one or both of the species are extinct. The first solution gives equilibrium between the two species where neither is extinct. All solutions for the problem have been found Note the large domain, $[0, 10^{10}]$, covered by each variable.

Glosbol could not find small boxes on the first run for all roots. It was necessary to take the initial results (fairly large boxes in some case) and these as starting boxes in a second run. The second run did produce acceptably small boxes containing unique roots.

The *Globsol* program is

```
Program Main


! This program solves the nonlinear system of equations from
! Burden and Faires where two species compete for the same food supply
! The variables x1(t) and x2(t)  speficy the number of each species alive at time t.
! The system will be in equlibilrium when dx1(t)/dt = dx2(t)/dt = 0.
! The derivatives are presented here


   use overload
   type(cdlvar), dimension(2) :: x
   type(cdllhs), dimension(2) :: f
   output_file_name = 'Burden.cdl'
   call initialize_codelist(x)


   f(1) = x(1)*(4 - 0.0003d0*x(1) - 0.0004d0*x(2)) ! dx1(t)/dt
   f(2) = x(2)*(2 - 0.0002d0*x(1) - 0.0001d0*x(2)) ! dx2(t)/dt
   call finish_codelist


End Program Main
```

Results

```
 Initial box coordinates:
 [    0.0000D+00,   0.1000D+11 ] [    0.0000D+00,   0.1000D+11 ]



 The following boxes have been verified to contain unique roots:
```

```
Box coordinates:
[    0.8000D+04,   0.8000D+04 ] [    0.4000D+04,   0.4000D+04 ]
Level:            1
Box contains the following approximate root:
   0.8000D+04   0.4000D+04
Small box in which the root must lie:
[    0.8000D+04,   0.8000D+04 ] [    0.4000D+04,   0.4000D+04 ]
Interval residuals over the small box:
[   -0.9600D-05,   0.9600D-05 ] [   -0.3200D-05,   0.3200D-05 ]
------------------------------------------------

Box coordinates:
[   -0.5000D-09,   0.5000D-09 ] [    0.1278D+05,   0.2722D+05 ]
Level:            0
Box contains the following approximate root:
   0.0000D+00   0.2000D+05
Small box in which the root must lie:
[   -0.5000D-09,   0.5000D-09 ] [    0.2000D+05,   0.2000D+05 ]
Interval residuals over the small box:
[   -0.2000D-08,   0.2000D-08 ] [   -0.2086D-08,   0.2086D-08 ]
------------------------------------------------

Box coordinates:
[    0.1332D+05,   0.1334D+05 ] [   -0.1000D+01,   0.1000D+01 ]
Level:            0
Box contains the following approximate root:
   0.1333D+05   0.0000D+00
Small box in which the root must lie:
[    0.1333D+05,   0.1333D+05 ] [   -0.5563-307,   0.5563-307 ]
Interval residuals over the small box:
[   -0.2667D-04,   0.2667D-04 ] [   -0.8159-307,   0.8159-307 ]
```

```
------------------------------------------------


Box coordinates:
[   -0.1000D+01,   0.1000D+01 ] [   -0.1000D+01,   0.1000D+01 ]
Level:            1
Box contains the following approximate root:
   0.0000D+00   0.0000D+00
Small box in which the root must lie:
[   -0.5000D-09,   0.5000D-09 ] [   -0.4450-307,   0.4450-307 ]
Interval residuals over the small box:
[   -0.2000D-08,   0.2000D-08 ] [   -0.1335-306,   0.1335-306 ]
```

8 Another Multivariate Example

Chow [2] gives as a problem finding the steady state solution of

$$
\begin{aligned}
y_{1t} &= .8y_{1,t-1} + .4\exp(-.05y_{2,t-1}) + .2 + u_{1t},\\
y_{2t} &= .1\exp(.2y_{1,t-1}) + .75y_{2,t-1} + u_{2t},
\end{aligned}
$$

where u_{1t} and u_{2t} are error terms. The program and output follow.

```
Program Main
  use overload
  type(cdlvar), dimension(2)   ::x
  type(cdllhs), dimension(2)   ::f

  output_file_name = 'chow.cdl'
  call initialize_codelist(x)
  f(1) = x(1) - (0.8d0*x(1) + 0.4d0*exp(-0.05d0*x(2)) + 0.2d0)
  f(2) = x(2) - (0.1d0*exp(0.2d0*x(1)) +0.75d0*x(2))
  call finish_codelist

End Program Main
```

```
Initial box coordinates:

[   -0.1000D+04,   0.1000D+04 ] [   -0.1000D+04,   0.1000D+04 ]


CPU time:   0.9375D-01


The following boxes have been verified to contain unique roots:


Box no.:           1

Box coordinates:

[    0.2057D+01,   0.3802D+01 ] [   -0.1298D+01,   0.2735D+01 ]

Level:            7

Box contains the following approximate root:

   0.2929D+01   0.7186D+00

Small box in which the root must lie:

[    0.2929D+01,   0.2929D+01 ] [    0.7186D+00,   0.7186D+00 ]

Interval residuals over the small box:

[   -0.2929D-09,   0.2929D-09 ] [   -0.5263D-10,   0.5263D-10 ]

-------------------------------------------------


THERE WERE NO UNRESOLVED BOXES
```

9 A Classical Macroeconomic Model

In this section a small classical macroeconomic model is solved. The model is

$$\begin{array}{rll} l & = \alpha p y / w & \text{labor demand} \\ l & = N_0 + N_d(w/p) & \text{labor supply} \\ y & = l^{\alpha} & \text{production function} \\ M_d & = kpy & \text{Cambridge equation} \\ M_s & = M_d & \text{Money market equilibrium} \end{array}$$

where l is the amount of labor used, y is real output, and p is the price level, M_s and M_d represent money supply and money demand. The *Globsol* program and the output from the program are shown next. Note that the initial box is very large as usual. Further note that Globsol has determined that only one solution to the model over the search space.

```
Program Main
  use overload
  type(cdlvar), dimension(4) :: x
  type(cdllhs), dimension(4) :: f
  type(cdlvar)               :: w  !nominal wage rate
  type(cdlvar)               :: p  !price level
  type(cdlvar)               :: l  !labor
  type(cdlvar)               :: y  !output
  real(kind=8), parameter    :: No = 10.0d0    !labor supply parameter
  real(kind=8), parameter    :: Ns = 10.0d0    !labor supply parameter
  real(kind=8), parameter    :: a = 0.2d0      !production function parameter
  real(kind=8), parameter    :: Ms = 20.0d0    !Money supply
  real(kind=8), parameter    :: k = 0.20d0     !Cambridge k
  output_file_name = 'classical.cdl'

  call initialize_codelist(x)

  w = x(1)
  l = x(2)
  y = x(3)
  p = x(4)

  f(1) = l - (No + Ns*(w/p) )  !labor supply
  f(2) = l - a*p*y/w           !labor demand
  f(3) = y - l**(a)            !production function
  f(4) = Ms - k*p*y            !Money market equilibrium



  call finish_codelist
End Program Main
```

Initial box coordinates:

```
[    0.1000D-09,   0.1000D+21 ] [    0.1000D-09,   0.1000D+21 ]
[    0.1000D-09,   0.1000D+21 ] [    0.1000D-09,   0.1000D+21 ]


CPU time:   0.3267D+02
The following boxes have been verified to contain unique roots:


Box no.:           1
Box coordinates:
[    0.1713D+01,   0.2167D+01 ] [    0.9104D+01,   0.1151D+02 ]
[    0.1408D+01,   0.1781D+01 ] [    0.6203D+02,   0.6340D+02 ]
Level:          128
Box contains the following approximate root:
   0.1940D+01   0.1031D+02   0.1595D+01   0.6271D+02


Small box in which the root must lie:
[    0.1940D+01,   0.1940D+01 ] [    0.1031D+02,   0.1031D+02 ]
[    0.1595D+01,   0.1595D+01 ] [    0.6271D+02,   0.6271D+02 ]
Interval residuals over the small box:
[   -0.5309D-08,   0.5309D-08 ] [   -0.1546D-07,   0.1546D-07 ]
[   -0.9568D-09,   0.9568D-09 ] [   -0.1000D-07,   0.1000D-07 ]
-------------------------------------------------


THERE WERE NO UNRESOLVED BOXES
```

10 General equilibrium

Cornwall [3] gives a model with the following excess demand equations for a three consumer, three good world:

$$\begin{aligned} z_1(p) &= \tfrac{-r_1p_2}{p_1+p_2} + \tfrac{r_2p_3}{p_1+p_3}, \\ z_2(p) &= \tfrac{r_1p_1}{p_1+p_2} - \tfrac{r_2p_2}{p_2+p_3}, \\ z_3p &= \tfrac{r_2p_2}{p_2+p_3} - \tfrac{p_1r_3}{p_1+p_3}, \end{aligned}$$

where p_i are prices and r_i is the endowment that consumer i has of good i. Suppose that the initial endowments are $r_1 = r_2 = r3 = 1$, then the only equilibrium prices occur where $p_1 = p_2 = p_3 > 0$ [3]. Choosing p_1 as a numéraire and applying Walras' law gives the following *Globsol* program:

```
Program Main

  use overload

  type(cdlvar), dimension(2)  :: p
  type(cdllhs), dimension(2)  :: f

  real(kind=8), dimension(3)  ::  r
  real(kind=8)                :: num

  num =  1.0d0
  r(1) = 1.0d0
  r(2) = 1.0d0
  r(3) = 1.0d0

  output_file_name = 'GE1.cdl'

  call initialize_codelist(p)

  f(1) = -r(1)*num/(num + p(1)) + p(2)*r(3)/(num+p(2))
  f(2) = num*r(1)/(num+p(1))    - r(2)*p(2)/(p(1)+p(2))



  call finish_codelist
End Program Main
```

```
CPU time:   0.4531D+00
```

```
The following boxes have been verified to contain unique roots:


Box no.:           1
Box coordinates:
[    0.9817D+00,   0.1018D+01 ] [    0.9474D+00,   0.1053D+01 ]
Level:          134
Box contains the following approximate root:
   0.1000D+01   0.1000D+01
Small box in which the root must lie:
[    0.1000D+01,   0.1000D+01 ] [    0.1000D+01,   0.1000D+01 ]
Interval residuals over the small box:
[   -0.1250D-09,   0.1250D-09 ] [   -0.2942D-14,   0.2859D-14 ]
------------------------------------------------


THERE WERE NO UNRESOLVED BOXES
```

11 Another general equilibrium problem

Judd [7] presents a general equilibrium problem that would create difficulty for Newton's method. Judd considers an exhcange economy with two agents ($i = 1, 2$) and two goods ($j = 1, 2$). The agents have utility functions

$$u_i(x_1, x_2) = \frac{a_1^i x_1^{(\gamma_i+1)}}{\gamma_i + 1} + \frac{a_2^i x_2^{(\gamma_i+1)}}{\gamma_i + 1}.$$

Agent i is assumed to have endowment $e^i = (e_1^i, e_2^i)$. Let p_1 and p_2 designate the price of the two goods. Agent i has a demand function $d_j^i(p) = \theta_j^i I^i p_j^{-\eta_i}$ where $I^i = p_1 e_1^i + p_2 e_2^i$ and $\theta_j^i = (a_j^i)/\sum_{l=1}^{2}(a_l^i)^{\eta_i} p_l^{(1-\eta_i)}$. Judd assumes the following values for the problem: $a_1^1 = a_2^2 = 1024$, $a_2^1 = a_1^2 = 1$, $e_1^1 = e_2^2 = 12$, and $\eta_1 = \eta_2 = 0.2$ where $\eta_i \equiv -1/\gamma_i$.

Equilibrium is given by

$$\sum_{i=1}^{2} d_1^i(p) = \sum_{i=1}^{2} e_1^i, \qquad p_1 + p_2 = 1.$$

The market for good two is not needed because of Walras's law. There are three equilibria for this problem: $p = (0.5, 0.5)$, $p = (0.1129, 0.8871)$ and $p = (0.8871, p = 0.1129)$. Judd simplifies the problem by imposing the solution $p_2 = 1 = p_1$, leading to an equation with only one unknown. This substitution creates numerical difficulties for Newton's method. Newton's method in this case can lead to searching in regions with negative prices. This is not a problem for the interval Newton method.

Two versions of the problem will be considered here. The first version uses the substitution $p_2 = 1 - p_1$ while the second does not. The *Globsol* program for the first version is

```
Program Main

    use overload
    real(kind=8), dimension(2,2)           :: e           !format e(person, good)
    real(kind=8), dimension(2,2)           :: a
    real(kind=8), dimension(2)             :: eta
    integer                               :: i, j
    type(cdlvar), dimension(1)            :: p
    type(cdlvar), dimension(2)            :: Index
    type(cdlvar), dimension(2,2)          :: theta
    type(cdlvar), dimension(2)            :: d
    type(cdllhs), dimension(1)            :: f

    call initialize_codelist(p)
    output_file_name = 'Judd3.cdl'

    e(1,1) = 12.0d0
    e(1,2) = 1.0d0
    e(2,1) = 1.0d0
    e(2,2) = 12.0d0
    a(1,1) = 1024.0d0
    a(1,2) = 1.0d0
    a(2,1) = 1.0d0
    a(2,2) = 1024.0d0
    eta(1) = 0.2d0
    eta(2) = 0.2d0
    Index(1) = p(1)*e(1,1) + (1-p(1))*e(1,2)
```

```
    Index(2) = p(1)*e(2,1) + (1-p(1))*e(2,2)
    Do i = 1,2
      Do j =1,2
        theta(i,j) = (a(i,j)**eta(i))/( (a(i,1)**eta(i))*(p(1)**(1-eta(i)) )&
                                       + (a(i,2)**eta(i))*((1-p(1))**(1-eta(i)) ) )
      end do  ! j
    end do   ! i
    d(1) =  theta(1,1)*Index(1)*(p(1)**(-eta(1) ) )
    d(2) =  theta(2,1)*Index(2)*(p(1)**(-eta(2) ) )
    f(1) = e(1,1) + e(2,1) - d(1) - d(2)
    call finish_codelist
End Program Main
```

```
Initial box coordinates:
[    0.1000D-10,   0.1000D+01 ]


CPU time:   0.3438D+00


The following boxes have been verified to contain unique roots:


Box no.:           1
Box coordinates:
[    0.8813D+00,   0.8929D+00 ]
Level:            1
Box contains the following approximate root:
   0.8871D+00
Small box in which the root must lie:
[    0.8871D+00,   0.8871D+00 ]
Interval residuals over the small box:
[   -0.2483D-12,   0.2560D-12 ]
--------------------------------------------------
```

```
Box no.:          2

Box coordinates:

[    0.4988D+00,   0.5012D+00 ]

Level:           1

Box contains the following approximate root:

   0.5000D+00

Small box in which the root must lie:

[    0.5000D+00,   0.5000D+00 ]

Interval residuals over the small box:

[   -0.3904D-12,   0.3855D-12 ]

-------------------------------------------------


Box no.:          3

Box coordinates:

[    0.1116D+00,   0.1142D+00 ]

Level:           1

Box contains the following approximate root:

   0.1129D+00

Small box in which the root must lie:

[    0.1129D+00,   0.1129D+00 ]

Interval residuals over the small box:

[   -0.4647D-12,   0.4628D-12 ]


THERE WERE NO UNRESOLVED BOXES
```

Version 2 of the solution

```
Program Main


    use overload
```

```
real(kind=8), dimension(2,2)  :: e           !format e(person, good)
real(kind=8), dimension(2,2)  :: a
real(kind=8), dimension(2)    :: eta
integer                       :: i, j
type(cdlvar), dimension(2)    :: p
type(cdlvar), dimension(2)    :: Index
type(cdlvar), dimension(2,2)  :: theta
type(cdlvar), dimension(2)    :: d
type(cdllhs), dimension(2)    :: f

call initialize_codelist(p)
output_file_name = 'Judd2.cdl'

e(1,1) = 12.0d0
e(1,2) = 1.0d0
e(2,1) = 1.0d0
e(2,2) = 12.0d0
a(1,1) = 1024.0d0
a(1,2) = 1.0d0
a(2,1) = 1.0d0
a(2,2) = 1024.0d0
eta(1) = 0.2d0
eta(2) = 0.2d0
Index(1) = p(1)*e(1,1) + p(2)*e(1,2)
Index(2) = p(1)*e(2,1) + p(2)*e(2,2)
Do i = 1,2
  Do j =1,2
    theta(i,j) = (a(i,j)**eta(i))/( (a(i,1)**eta(i))*(p(1)**(1-eta(i)) )&
                                + (a(i,2)**eta(i))*(p(2)**(1-eta(i)) ) )
  end do  ! j
end do   ! i
```

```
   d(1) =  theta(1,1)*Index(1)*(p(1)**(-eta(1) ) )
   d(2) =  theta(2,1)*Index(2)*(p(1)**(-eta(2) ) )
   f(1) = e(1,1) + e(2,1) - d(1) - d(2)
   f(2) = 1 - p(1) - p(2)
   call finish_codelist
End Program Main
```

```
 Initial box coordinates:
 [    0.1000D-09,   0.1000D+01 ] [    0.1000D-09,   0.1000D+01 ]


 CPU time:   0.1734D+01


 The following boxes have been verified to contain unique roots:


 Box no.:           1
 Box coordinates:
 [    0.1116D+00,   0.1143D+00 ] [    0.8830D+00,   0.8911D+00 ]
 Level:           10
 Box contains the following approximate root:
    0.1129D+00   0.8871D+00
 Small box in which the root must lie:
 [    0.1129D+00,   0.1129D+00 ] [    0.8871D+00,   0.8871D+00 ]
 Interval residuals over the small box:
 [   -0.2212D-06,   0.2212D-06 ] [   -0.5000D-07,   0.5000D-07 ]
 -------------------------------------------------


 Box no.:           2
 Box coordinates:
 [    0.8857D+00,   0.8884D+00 ] [    0.1092D+00,   0.1167D+00 ]
 Level:           10
 Box contains the following approximate root:
    0.8871D+00   0.1129D+00
```

```
Small box in which the root must lie:
[    0.8871D+00,   0.8871D+00 ] [    0.1129D+00,   0.1129D+00 ]
Interval residuals over the small box:
[   -0.3585D-08,   0.3586D-08 ] [   -0.5001D-07,   0.5000D-07 ]
-------------------------------------------------

Box no.:           3
Box coordinates:
[    0.4968D+00,   0.5032D+00 ] [    0.4981D+00,   0.5019D+00 ]
Level:           10
Box contains the following approximate root:
   0.5000D+00   0.5000D+00
Small box in which the root must lie:
[    0.5000D+00,   0.5000D+00 ] [    0.5000D+00,   0.5000D+00 ]
Interval residuals over the small box:
[   -0.1565D-07,   0.4631D-07 ] [   -0.6111D-07,   0.1809D-06 ]
-------------------------------------------------

THERE WERE NO UNRESOLVED BOXES
```

12 Conclusions

Interval arithmetic is a method of determining all roots of a system of nonlinear equations within a specified space. We have examined a public domain software package that incorporates interval arithmetic, automatic differentiation arithmetic and slope arithmetic, and an interval Newton method. We were able to use this package, *Globsol*, to find all the roots of a number of test cases. We will apply *Globsol* to much larger systems of equations in the near future.

References

[1] R. L. Burden and J. D. Faires. *Numerical Analysis.* Prindle, Weber & Schmidt, Boston, 3rd edition, 1985.

[2] Gregory C. Chow. *Analysis and Control of Dynamic Economic Systems.* John Wiley & Sons, New York, 1975.

[3] Richard R. Cornwall. *Introduction to the Use of General Equilibrium Analysis.* Advanced Textbooks in Economics. North–Holland, Amsterdam, 1984.

[4] J. E. Dennis, Jr. and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice–Hall, Englewood Clifts, New Jersey, 1983.

[5] E. R. Hansen. Interval forms of newton's method. *Computing*, 20:153–163, 1978.

[6] E. R. Hansen. *Global Optimization Using Interval Analysis.* Marcel Dekker, Inc., New York, 1992.

[7] Kenneth L. Judd. *Numerical Methods in Economics.* The MIT Press, Cambridge, Massachusetts, 1998.

[8] R. B. Kearfott. *Rigorous Global Search: Continuous Problems.* Kluwer Academic Publishers, Dordrecht, The Netherlands, 1966.

[9] R. E. Moore. *Interval Analysis.* Prentice Hall, Englewood Cliffs, NJ, 1966.

[10] Ramon E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing.* PhD thesis, Stanford University, Stanford, CA 94305, October 1962.

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

[12] L. B. Rall. *Computational Solution of Nonlinear Operator Equations.* John Wiley & Sons, New York, 1969.

[13] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* John Wiley & Sons, Inc., New York, 1988.