acsystème®
analyse et contrôle des systèmes

User Guide

acsystème intervaltoolbox
version 1.2

Acsystème (siège social)
8 rue des Tisserands
35830 Betton
France
phone: +33 2 99 55 18 11

Acsystème Paris
215 rue Jean-Jacques Rousseau
92136 Issy-les-Moulineaux Cedex
France
phone: +33 1 47 36 51 80

web: www.acsysteme.com
e-mail: info@acsysteme.com

fax: +33 2 99 55 19 53 (Betton)
fax: +33 1 47 36 51 89 (Issy)

SA au capital de 166 100 euros
RCS Rennes B 441 000 775
Siret 441 000 775 00011
VAT: FR34 441 000 775
APE/NAF 742C

Table of contents

1. Limitation of liability 4
2. Acknowledgments 5
3. What is Acsystème Interval Toolbox? 6
4. Tutorial 7
   4.1 Introduction 7
   4.2 Getting started 8
      4.2.1 Interval object 8
      4.2.2 Interval Display 10
      4.2.3 Rigorous computing 11
      4.2.4 Matrix computations 12
      4.2.5 Contractors 13
   4.3 Examples using Acsystème Interval Toolbox 14
      4.3.1 Equation solving 14
      4.3.2 Inequation solving 16
      4.3.3 Finding stability domain – Efficient way 18
      4.3.4 Finding stability domain – Easy way 21
      4.3.5 Compute Image of a set 23
      4.3.6 Unconstrained global minimization 24
      4.3.7 Unconstrained global minimization with contractor 26
      4.3.8 Constrained global minimization 29
5. How INTOPTIMIZE works 32
   5.1 A global optimization algorithm 32
      5.1.1 Principles 32
      5.1.2 Results 32
      5.1.3 How to decrease the upper bound c_{sup}? 32
      5.1.4 When reject or bisect an interval? 32
      5.1.5 When stop the algorithm? 33
   5.2 Implementation 33
      5.2.1 Initialization: 33
      5.2.2 Algorithm: 33
   5.3 Example 34
   5.4 Constraints 34
6. Function Reference 35
   6.1 SIVIA 35
   6.2 SIVIAC 37
   6.3 INTOPTIMIZE 39
   6.4 INTOPTIMIZEC 41
   6.5 IMAGESET 44
   6.6 INTVISU1D 45
   6.7 INTVISU2D 46
   6.8 INTOPTVISU1D 47
   6.9 INTOPTVISU2D 48
   6.10 INTCONNEX 49
   6.11 HORNER 50

6.12 REALROOTS.. 51
6.13 INTBISECT.. 52
6.14 INTCONV.. 53
6.15 INTHASPOLYIMAGROOT.. 54
6.16 INTROUTH.. 55
6.17 INTTSTROUTH.. 56
6.18 INTTSTZERO.. 57
6.19 INTBOUNDPOLYDROOTS.. 58
6.20 MERGEINNEROUTER.. 59
6.21 MINCE.. 60
6.22 POLY2REIM.. 61

7. References.. 62

1. Limitation of liability

THE PROGRAMS SHOULD NOT BE RELIED ON AS THE SOLE BASIS TO SOLVE A PROBLEM WHOSE INCORRECT SOLUTION COULD RESULT IN INJURY TO PERSON OR PROPERTY. IF A PROGRAM IS EMPLOYED IN SUCH A MANNER, IT IS AT THE LICENCEE'S OWN RISK AND ACSYSTÈME EXPLICITY DISCLAIMS ALL LIABILITY FOR SUCH MISUSE TO THE EXTENT ALLOWED BY LAW.

ACSYSTÈME SHALL HAVE NO LIABILITY FOR ANY INDIRECT CONSEQUENTIAL LOSS (WHETHER FORESEEABLE OR OTHERWISE AND INCLUDING LOSS OF PROFITS, LOSS OF BUSINESS, LOSS OF OPPORTUNITY, AND LOSS OF USE OF ANY COMPUTER HARDWARE OR SOFTWARE).

2. Acknowledgments

We would like to acknowledge the authors of the book that we used to write this toolbox:

Luc JAULIN

Michel KIEFFER

Olivier DIDRIT

Éric WALTER

They have done a great job:

L. JAULIN, M. KIEFFER, O. DIDRIT and E. WALTER, *Applied Interval Analysis*, Springer 2001.

3. What is Acsystème Interval Toolbox?

Acsystème Interval Toolbox is a collection of functions that extends the capability of the Matlab numeric computing environment. The toolbox includes routines for many types of problems including:

- Unconstrained global non linear minimization
- Constrained global non linear minimization
- Guaranteed Set Inversion
- Guaranteed Equation/Inequation solving
- Guaranteed imaging of a set by a non linear function

All the toolbox functions are Matlab M-files (P-files for evaluation version), made up of Matlab statements that implements specialized algorithms. You can view the Matlab code for these functions using the statement:

```
type function_name
```

You can extend the capability of Acsystème Interval Toolbox by writing your own M-files, or by using the toolbox in combination with other toolboxes, or with Matlab, or Simulink.

Acsystème Interval Toolbox is able to work in combination with Optimization toolbox for Matlab to solve optimization problem (see INTOPTIMIZE routine), but it is not necessary to have the Optimization Toolbox installed on your computer to use Acsystème Interval Toolbox.

4. Tutorial

4.1 Introduction

Interval methods can provide guaranteed solutions to difficult nonlinear problems, such as the global optimization of non-convex criteria or the characterization of sets defined by nonlinear inequalities.

The toolbox provides two main algorithms:

- Set Inversion Via Interval Analysis :
  - SIVIA: classic version
  - SIVIAC: version with contractor
- Global Optimization Via Interval Analysis:
  - INTOPTIMIZE: classic version
  - INTOPTIMIZEC: version with contractor

SIVIA or SIVIAC allow you to find the solution of the Set Inversion problem:

Find the set ***X*** such for all *x* in ***X***, *f*(*x*) is in ***Y***

SIVIA or SIVIAC provide two sets:

- The *Inner* set, that is included in solution set ***X***
- The *Outer* set, such that ***X*** is included in union of Inner and Outer sets.

SIVIA or SIVIAC should be used to solve problems like:

- Equation finding problem, that should be written as:

  Find the set ***X*** such for all *x* in ***X***, *f*(*x*) is in 0
- Inequation solving problem, that should be written as:

  Find the set ***X*** such for all *x* in ***X***, *f*(*x*) is in [*ymin*, *ymax*]

INTOPTIMIZE or INTOPTIMIZEC should be used to solve problems like:

- Constrained/Unconstrained optimization
- Equation finding problem, that should be written as:

  Find the set ***X*** that minimizes abs(*f*(*x*))
- Curve fitting problem, that should be written as:

  Find the set ***X*** that minimizes sum(($f(x_i, x)-y_i)^2$)

4.2 Getting started

4.2.1 Interval object

Acsysteme Interval Toolbox provides definition of interval object. An interval is composed of a lower bound (lb) and an upper bound (ub).

Elementary operations (+, /, ×…) have then been defined for this object. Most of the current functions (sin, exp…) have also been extended to the interval object. At last, specific operations are provided (intersection, inclusion…).

4.2.1.1 Constructor

To create an interval object, you can use one of these constructors:

```
intX = interval(x);

intX = interval(lb, ub);
```

Where x, lb, ub are real scalars, real vector, real matrixes or strings. Use strings in order to use rigorous constructor.

Type `help interval` for more details.

4.2.1.2 Arithmetic

You can now use intervals quite as simply as other objects:

`intS=intX+intY;` %compute interval that **contains** the set $S = \{x + y | x \in \mathrm{intX}, y \in \mathrm{intY}\}$

`intS=intX*intY;` %compute interval that **contains** the set $S = \{x * y | x \in \mathrm{intX}, y \in \mathrm{intY}\}$

…

Note: as outward rounding has been implemented for arithmetic computations, resulting interval always **contains** the true set that is the result of the required operation.

It is not guaranteed to be equal to the true set because the true set may not have an exact representation from IEEE numbers.

> **Example**: let `x = interval(0,2);` and compute `y=x/3;`. As 0 and 2 are truly represented in IEEE double format, and as 2/3 is not a truly represented number, the resulting interval will only contain the true set.

4.2.1.3 Basic functions

Basic functions have been extended to interval object, so that you can now use intervals quite as simply as other objects:

`intS = exp(intX);` %compute an interval that matches the set $S = \left\{ \exp(x) \middle| x \in \mathrm{intX} \right\}$

`intS = sin(intX);` %compute an interval that matches the set $S = \left\{ \sin(x) \middle| x \in \mathrm{intX} \right\}$

…

Note: as outward rounding has **not** been implemented for basic functions, resulting interval is only a close approximation of the true set that is the result of the required operation. The precision of this approximation is only affected by rounding errors.

4.2.1.4 Specific functions

Some specific operations have been defined for interval objects:

Compute interval that contains the intersection set $S = \left\{ s \middle| s \in \mathrm{intX} \wedge s \in \mathrm{intY} \right\}$:

```
intS = intersect(intX,intY);
```

Compute interval that contains the convex union set $S = \left\{ s \middle| s \in \mathrm{intX} \vee s \in \mathrm{intY} \right\}$:

```
intS = union(intX,intY);
```

Extracts lower bound of interval `intX`: $\mathrm{xinf} = \left\{ x \in \mathrm{intX} \middle| \forall a \in \mathrm{intX}, \mathrm{x} \leq \mathrm{a} \right\}$

```
xlb = lb(intX); %xinf is a no longer an interval
```

Extracts upper bound of interval `intX`: $\mathrm{xsup} = \left\{ x \in \mathrm{intX} \middle| \forall a \in \mathrm{intX}, \mathrm{x} \geq \mathrm{a} \right\}$

```
xub = ub(intX); %xinf is a no longer an interval
```

Extracts central value of `intX`:

```
xmid = mid(intX); %xmid is a no longer an interval
```

Test if an element or an interval is included in an interval:

```
boo = in(elementX, intX);
```

Test if an element is in inferior or equal to all values of a interval:

```
boo = (elementX <= intX);
```

…

4.2.2 Interval Display

Scalar intervals are displayed as follows:

```
[low bound, upper bound]
```

Matrix intervals are displayed as follows:

```
[lb(a11), ub(a11)]  …  [lb(a1N), ub(a1N)]

      …                      …

[lb(aM1), ub(aM1)]  …  [lb(aMN), ub(aMN)]
```

with `lb=lower bound` and `ub=upper bound`

Acsysteme Interval Toolbox supports most of the display formats available with *Matlab* command `format`:

```
format short
format short g
format short e
format long
format long g
format long e
format compact
format loose
```

Type `help format` in the *Matlab* command window for more information about the `format` command.

Important note: using `long` option will produce **rigorous** display (cf. 4.2.3).

4.2.3 Rigorous computing

4.2.3.1 Rigorous constructor

Computers can only handle a (large) set of real numbers that is defined by the IEEE 754 standard. For floating point representation (double), only real numbers that can be represented by the following formula are truly represented:

$$\pm\left(1 + d_1\beta^{-1} + d_2\beta^{-2} + ... + d_{p-1}\beta^{-(p-1)}\right)\times \beta^e$$

with $\beta = 2$, $p = 53$ and $-1022 \le e \le 1023$.

If you create an interval object from a real number, it creates in fact an interval from the nearest truly represented real of the one you have chosen by writing its using decimals.

Example 1: the real 0.8 has no exact representation in binary IEEE double format (you can not obtain 0.8 from the above formula). So the constructor `interval(0.8)` returns:

```
>> interval(0.8)
interval ans =

[   0.80000000000000004,    0.80000000000000004]
```

Example 2: the real 0.5 has an exact representation in binary IEEE double format. So the constructor `interval(0.5)` returns:

```
>> interval(0.5)
interval ans =
[   0.5,   0.5]
```

We provide a **rigorous constructor**, that enable you to obtain an interval that **contains** the number you describe using its decimals. To use rigorous constructor, you must use **strings** for constructor arguments:

Example 3: the real 0.8 has no exact representation in binary IEEE double format (you can not obtain 0.8 from the above formula). So the constructor `interval('0.8')` returns:

```
>> interval('0.8')

interval ans =

   [   0.79999999999999993,    0.80000000000000004]
```

Example 4: the real 0.5 has an exact representation in binary IEEE double format. So the constructor `interval('0.5')` returns:

```
>> interval('0.5')
interval ans =
[   0.5,   0.5]
```

Therefore, working on interval with AIT can lead to guaranteed computations.

4.2.3.2 What should be guaranteed and what should not?

Outward rounding has been implemented in order to provide guaranteed **arithmetic** computations: computations that use only +, -, *, / should always lead to guaranteed results. The computations using ^ should do the same, although we are not sure of what is done by the *Matlab* ^ function about the rounding errors.

Outward rounding consists in switching the rounding mode of the processor between lower and upper bounds computations in order to obtain a resulting interval that should always contain the true set. This technique takes into account the rounding errors that affect each operation on a computer.

Outward rounding has also been implemented for `sqrt`, but the Matlab `sqrt` function seems to be not correctly rounded for other rounding modes than the default mode, so the results are not guaranteed.

Outward rounding has **NOT** been implemented for other basic functions (sin, exp…): the results are very close from the true set (very little errors could occur due to rounding errors), but are not guaranteed to contain the true set.

Outward rounding is not necessary for set operations (`in`, `intersect`…) because no computations on doubles are done. Set operations should be guaranteed.

Outward rounding has been implemented for displaying intervals, but this is not fully guaranteed because we do not know what the Matlab function `sprintf` does exactly.

4.2.3.3 Conclusion

Even if results of some functions are not truly guaranteed, rounding errors should be extremely small on almost all problems.

Global optimization and Set Inversion algorithms will work properly even in case of rounding errors and will lead to “nearly guaranteed” results:

- global optimization algorithm will still find the global optimizers
- set inversion algorithm will still find the solution set

So, don’t be afraid of the “guaranteed or not” aspect of computations.

4.2.4 Matrix computations

Guaranteed matrix computations are now available: +, -, *, inv, /

Note that A\B is also available if A is square.

4.2.5 Contractors

Warning: we will only provide a very short presentation of contractors. For more information about contractors, the reader is urged to consult this book: *Applied Interval Analysis* ([1]).

A **contractor** is a specific algorithm that can reduce the domain on a variable *x* using some specific constraints that *x* should satisfy.

Example: if x=interval(-0.8, 10) and x satisfies the constraint $x^2 \leq 1$, it is obvious that the specific constraint $x^2 \in [-1, 1]$ leads to reduce the domain of x to [-0.8 1]. This example can be implemented quickly with the contractors available with the toolbox:

```
>> x = interval(-0.8, 10);
>> [junk, x]=psqr(interval(-1,1),x)

interval x =

   [-0.8, 1]
```

Basic contractors are available for scalar operators: +,-,*,/,cos, sin, exp…

You can use the **constraint propagation** theory to build quickly complex contractors for your specific problem (one again, the reader is urged to consult this book for more information: *Applied Interval Analysis* [1]).

Then you can provide your contractor to SIVIAC or INTOPTIMIZEC to increase speed computation. Note that using contractors can sometimes lead you to increase the computation speed by a factor greater than 1000!

4.3 Examples using Acsystème Interval Toolbox

4.3.1 Equation solving

Consider the problem of finding the set of ALL values [x_1, x_2] that solves:

$$f(x_1, x_2) = (x_1^2 - 4)x_1^2 + 4x_2^2 = 0$$

To solve this two-dimensional problem, write an M-file that returns the function value. The, invoke SIVIA using the `IntTstZero` function.

The `'IntTstZero'` function allows you to write an M-file that simply returns the function value.

Step 1: Write an M-file `FcnTestSivia1.m`

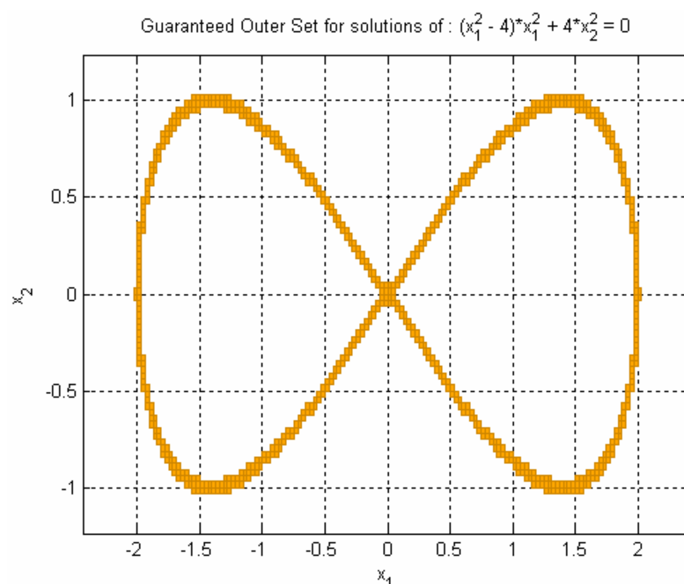
```
function y = FcnTestSivia1(x)
y = (x(1)^2 - 4)* x(1)^2 + 4*x(2)^2;
```

Step 2: Invoke SIVIA

```
fcname = 'FcnTestSivia1'; %function to be called by IntTstZero
dmax=0.05;  %precision
xN = [];    %normalization factors([]->let SIVIA compute them)
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
[InnerQ, OuterQ]=sivia('IntTstZero',intXo, dmax, xN, fcname);
```

You can plot Inner and Outer sets using routine `INTVISU2D`:

```
[hf, ha] = intvisu2D(InnerQ, OuterQ); %Display results
smartaxis(ha); %smart axis parameters
xlabel('x_1'); ylabel('x_2'); %set labels
title('Guaranteed  Outer  Set  for  solutions  of:  (x_1^2  -
4)*x_1^2 + 4*x_2^2 = 0'); %set title
```



Inner Set is plotted using blue boxes and Outer set is plotted using orange boxes. In this case, Inner set is empty, and there is no blue box.

Note: InnerQ is often empty for equation problems, as solution set generally has no interior.

It is important to note that initial domain could be set to $x_1 \in [-\infty \quad \infty]$ and $x_2 \in [-\infty \quad \infty]$:

```
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
```

Therefore, all the solutions of $f(x_1, x_2) = 0$ are in the outer set drawn on the figure.

The parameter `dmax` characterizes the size of resulting boxes: all of them have a normalized maximum width equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)} \left(\frac{width(x_i)}{x_N(i)} \right) \leq d_{\max} \quad \text{Rq: } \dim(x) = 2 \text{ in this example}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`.

The normalization factors `xN` can be used to increase speed in case of ill conditioned initial domain.

Example of ill conditioned initial domain: $x_1 \in \left[-10^{-8} \quad 10^{-8}\right]$, $x_2 \in \left[-10^{8} \quad 10^{8}\right]$

You can let the normalization factors to `[]` so that SIVIA compute them automatically. In case of infinite initial domain, there are set to 1. For finite initial domain, `xN` is set to `width(x)`.

4.3.2 Inequation solving

Consider the problem of finding the set of ALL values [x_1, x_2] that solves:

$$1 \le f(x_1, x_2) = x_1^2 + x_2^2 \le 2$$

To solve this two-dimensional problem, write an M-file that returns the test function value. Then, invoke SIVIA.

Step 1: Write an M-file `FcnTestSivia2.m`

```
function ibboo = FcnTestSivia2(x)
%constants
IBTRUE  = 1;
IBFALSE = 0;
IBINDET = 0.5;
%compute f(x)
z = x(1)^2 + x(2)^2;
%inclusion set
r=interval(1, 2);
%inclusion test
if in(z, r)
    %z is in r
    ibboo = IBTRUE;
elseif isempty(intersect(z, r))
    %z and r do not intersect
    ibboo = IBFALSE;
else
    %z and r intersect
    ibboo = IBINDET;
end
```

The first part of `FcnTestSivia2` just code f(*x*) and put result into variable `z`. Then, you have to compute yourself the logical condition that solution set satisfies. It is very simple to do: you just have to deal with interval objects instead of real values.

First, create an interval object for inclusion set:

```
%inclusion set
r=interval(1, 2);
```

Then, compute logical test:

If $f(x) \subset [1 \quad 2]$, then x is valid:

```
if in(z, r)
    ibboo = IBTRUE; %z is in r
```

else if $f(x) \cap [1 \quad 2] = \varnothing$, then x is invalid

```
elseif isempty(intersect(z, r))
    ibboo = IBFALSE; %z and r do not intersect
```

otherwise, nothing can be said

```
else
    ibboo = IBINDET; %z and r intersect
end
```

Note that we had to introduce the INDETERMINATE value for logical test, in case of x can not be declared completely valid nor completely invalid. This value must always be set to `IBINDET = 0.5`.

Step 2: Invoke SIVIA

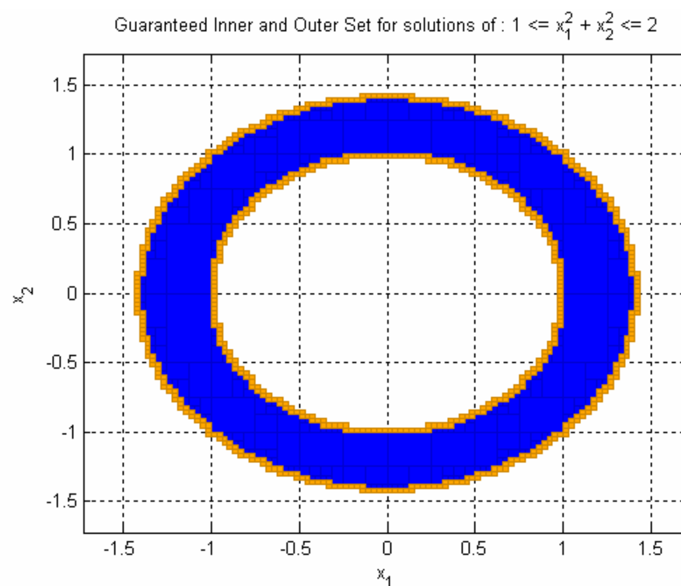
```
dmax=0.05;  %precision
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
[InnerQ, OuterQ] = sivia('FcnTestSivia2', intXo, dmax);
```

Save the results in order to reuse them with `IMAGESET` (see §4.3.5).

```
save('FcnTestSivia2-result.mat','InnerQ','OuterQ');
```

You can plot Inner and Outer sets using routine `INTVISU2D`:

```
[hf, ha] = intvisu2D(InnerQ, OuterQ); %Display results
smartaxis(ha); %smart axis parameters
xlabel('x_1'); ylabel('x_2'); %set labels
title('Guaranteed Inner and Outer Set for solutions of: 1 <=
x_1^2 + x_2^2 <= 2'); %set title
```



Inner Set is plotted using blue boxes and Outer set is plotted using orange boxes.

It is important to note that initial domain could be set to $x_1 \in [-\infty \quad \infty]$ and $x_2 \in [-\infty \quad \infty]$:

```
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
```

Therefore, all the solutions of $1 \le x_1^2 + x_2^2 \le 2$ are in the union of the inner and outer sets drawn on the figure. You can verify it is true…

4.3.3 Finding stability domain – Efficient way

Consider the problem of finding the set of ALL values [p_1, p_2] within $p_1 \in [-3 \quad 7]$, $p_2 \in [-1.4 \quad 2]$ for which the following polynomial has roots with negative real parts:

$$P(s, p_1, p_2) = s^3 + \left(p_1 + p_2^3 + 2\right)s^2 + \left(p_1 + p_2^3 + 2\right)s + 2p_1p_2^3 + 6p_1 + 6p_2^3 + 2 + 0.75^2$$

To solve this two-dimensional problem, it is very efficient to write an M-file that computes directly the Routh table and logical tests on it. Then, invoke SIVIA.

Step 1: Write an M-file `FcnTestSivia4.m`

The Routh table of polynomial $P(s) = a_n s^n + a_{n-1}s^{n-1} + ... + a_1 s + a_0$ is given by:

$$R = \begin{bmatrix} a_n & a_{n-2} & a_{n-4} & ... & 0 \\ a_{n-1} & a_{n-3} & a_{n-5} & ... & 0 \\ b_1 & b_2 & b_3 & ... & 0 \\ c_1 & c_2 & c_3 & ... & 0 \\ ... & ... & ... & ... & ... \\ g_1 & 0 & & & \\ h_1 & & & & \end{bmatrix}$$

with:

$$b_1 = \frac{a_{n-1}a_{n-2} - a_n a_{n-3}}{a_{n-1}}, \quad b_2 = \frac{a_{n-1}a_{n-4} - a_n a_{n-5}}{a_{n-1}}, \quad ....$$

$$c_1 = \frac{b_1 a_{n-3} - a_{n-1}b_2}{b_1}, \quad c_2 = \frac{b_1 a_{n-5} - a_{n-1}b_3}{b_1}, \quad ....$$

...

We note $r = (a_{n-1}, b_1, ..., h_1)^T$ the first column of the Routh table (a_n is assumed to be 1 and is omitted).

P is stable if and only $r > 0$ and unstable (i.e. not asymptotically stable) if and only if there exists a component r_i such that $r_i \leq 0$.

In our case, the first column of the Routh table is:

$$r = \begin{bmatrix} p_1 + p_2^3 + 1 \\ (p_1 - 1)^2 + \left(p_2^3 - 1\right)^2 - 0.75^2 \\ 2(p_1 + 3)\left(p_2^3 + 3\right) - 16 + 0.75^2 \end{bmatrix}$$

You can see that a simplification occurred, *i.e.* c_1 has no denominator. This is a typical case for which coding the Routh table is very powerful.

Therefore, the code for `FcnTestSivia4` should be:

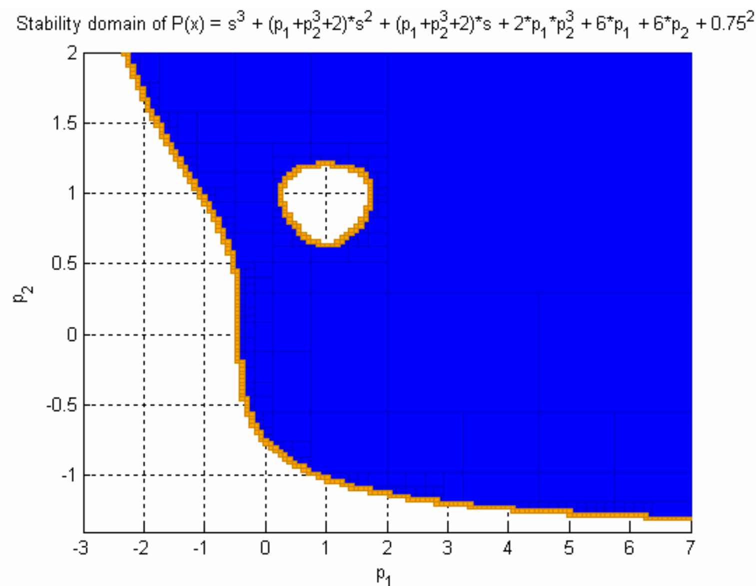
```
function ibboo = FcnTestSivia4(x)
%constants
IBTRUE  = 1;
IBFALSE = 0;
IBINDET = 0.5;

%get parameters
p1 = x(1);
p2 = x(2);
%compute p2^3
p23 = p2^3;

%compute Routh table (with use of the simplification)
RouthTable = [p1+p23+2; (p1-1)^2+(p23-1)^2-(0.75)^2;
2*(p1+3)*(p23+3) - 16 + (0.75)^2];

%check stability
if any(RouthTable<=0)
    %P(x) is never stable
    ibboo = IBFALSE;
elseif all(RouthTable>0)
    %P(x) is always stable
    ibboo = IBTRUE;
else
    %nothing can be said
    ibboo = IBINDET;
end
```

Step 2: Invoke SIVIA

```
dmax=0.01;  %precision
intXo=interval([-3;-1.4],[7;2]);    %initial    domain    for
parameters
[InnerQ,OuterQ]=sivia('FcnTestSivia4', intXo, dmax);
```

You can plot Inner and Outer sets using routine `INTVISU2D`:

```
[hf, ha] = intvisu2D(InnerQ, OuterQ); %Display results
axis([-3 7 -1.4 2]); % axis parameters
xlabel('p_1'); ylabel('p_2'); %labels
title('Stability domain of P(x) = s^3 + (p_1+p_2^3+2)*s^2 +
(p_1+p_2^3+2)*s + 2*p_1*p_2^3 + 6*p_1 + 6*p_2 + 0.75^2');
```



Inner Set is plotted using blue boxes and Outer set is plotted using orange boxes.

Inner set (in blue) is included in guaranteed stability domain. Stability domain is included in union of Inner set (in blue) and Outer set (in Orange).

Note: white area is included in guaranteed instability domain

4.3.4 Finding stability domain – Easy way

Consider again the problem of finding the set of ALL values [p_1, p_2] within $p_1 \in [-3 \quad 7]$, $p_2 \in [-1.4 \quad 2]$ for which the following polynomial has roots with negative real parts:

$$P(s, p_1, p_2) = s^3 + \left(p_1 + p_2^3 + 2\right)s^2 + \left(p_1 + p_2^3 + 2\right)s + 2p_1p_2^3 + 6p_1 + 6p_2^3 + 2 + 0.75^2$$

To solve this two-dimensional problem, you can write an M-file that returns only the polynomial coefficients. Then, invoke SIVIA using the `'IntTstRouth'` function.

The `'IntTstRouth'` function allows you to write an M-file that simply returns the polynomial coefficients.

Step 1: Write an M-file `FcnTestSivia5.m`

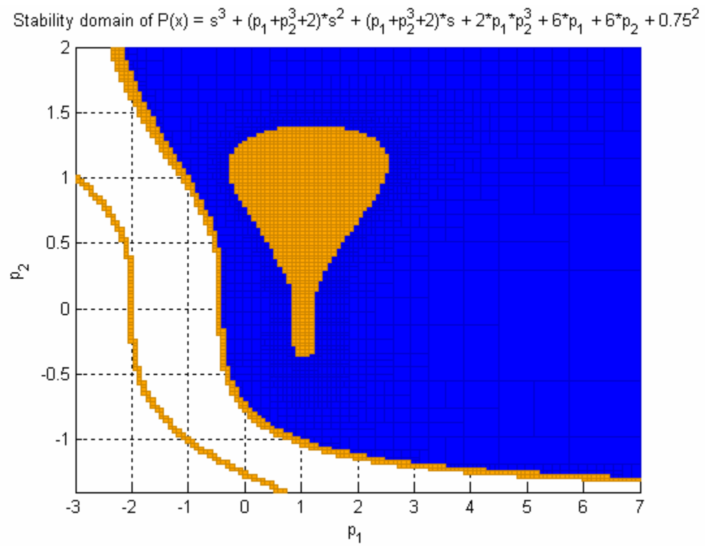
```
function y = FcnTestSivia5(x)
%get parameters
p1 = x(1);
p2 = x(2);
%compute p2^3
p23 = p2^3;
%compute coefficients of polynomial P(x)
P = [1, p1+p23+2, p1+p23+2, 2*p1*p23+6*p1+6*p23+2+0.75^2];
```

Step 2: Invoke SIVIA

```
fcname='FcnTestSivia5'; %function to be called by IntTstRouth
dmax=0.01;  %precision
xN = [];    %normalization factors([]->let SIVIA compute them)
intXo=interval([-3;-1.4],[7;2]);    %initial    domain    for
parameters
[InnerQ,OuterQ]=sivia('IntTstRouth', intXo, dmax, xN, fcname);
```

You can plot Inner and Outer sets using routine `INTVISU2D`:

```
[hf, ha] = intvisu2D(InnerQ, OuterQ); %Display results
axis([-3 7 -1.4 2]); % axis parameters
xlabel('p_1'); ylabel('p_2'); %labels
title('Stability domain of P(x) = s^3 + (p_1+p_2^3+2)*s^2 +
(p_1+p_2^3+2)*s + 2*p_1*p_2^3 + 6*p_1 + 6*p_2 + 0.75^2');
```



Inner Set is plotted using blue boxes and Outer set is plotted using orange boxes.

Inner set (in blue) is included in guaranteed stability domain. Stability domain is included in union of Inner set (in blue) and Outer set (in Orange).

Note: white area is included in guaranteed instability domain.

You can see that the Outer set is much larger than on previous example and is obtained with much more computing effort. This is due to a simplification in the Routh table that could not be taken into account using this simple procedure. See previous example for a more efficient way to solve this problem.

4.3.5 Compute Image of a set

Consider problem of finding a guaranteed outer set of values [y_1, y_2] that characterize the image of the set defined by example 2 (see §4.3.2) by the following non linear function:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = f(x_1, x_2) = \begin{bmatrix} (x_1 - 1)^2 - 1 + x_2 \\ (x_2 - 1)^2 - x_1^2 \end{bmatrix}$$

with x_1, x_2 defined by $1 \le x_1^2 + x_2^2 \le 2$.

To solve this two-dimensional problem, write an M-file that returns the function value. Then invoke IMAGESET with the result of example 2.

Step 1: Write an M-file `FcnTestImageSet.m`

```
function y = FcnTestImageSet(x)
y = [(x(1)-1)^2 - 1 + x(2); -x(1)^2+ (x(2)-1)^2];
```

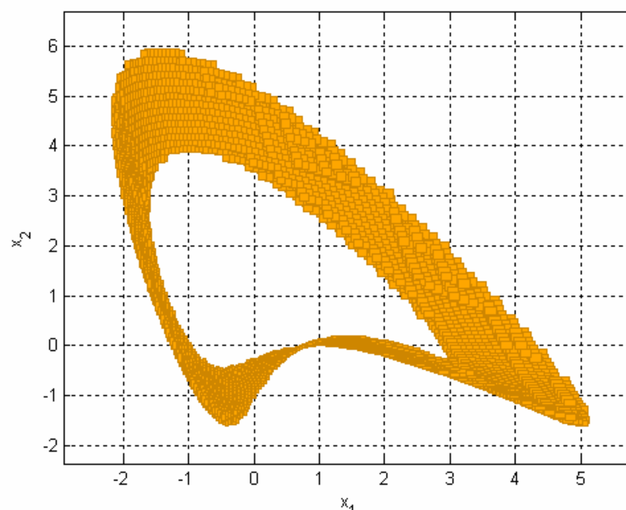
Step 2: Invoke IMAGESET

```
%load results of example #2:
load('FcnTestSivia2-result.mat','InnerQ','OuterQ');
dmax=0.05;  %precision
%Merge Inner and Outer Set
mergeSet = mergeInnerOuter(InnerQ, OuterQ);
%Compute Image Set
OuterQ = imageset('FcnTestImageSet', mergeSet);
```

You can plot results using routine `INTVISU2D`:

```
[hf, ha] = intvisu2D([],OuterQ);
smartaxis; %smart axis parameters
xlabel('x_1'); ylabel('x_2'); %set labels
title('Guaranteed Outer set of Image of solution set for 1 <=
x_1^2 + x_2^2 <= 2 by a non linear function');
```

Guaranteed Outer set of Image of solution set for 1 <= $x_1^2 + x_2^2$ <= 2 by a non linear function



Guaranteed image set is included in Outer set (in Orange). Inner Set is not yet computed by IMAGESET.

4.3.6 Unconstrained global minimization

Consider problem of finding the set of ALL the values $[x_1 \quad x_2]$ that solves:

$$\underset{x}{\text{minimize}}\, f(x) = \left(R - \left(x_1^2 + x_2^2\right)\right)^2$$

To solve this two-dimensional problem, write an M-file that returns the function value. Then, invoke INTOPTIMIZE.

Step 1: Write an M-file `FcnTestOptimize1.m`

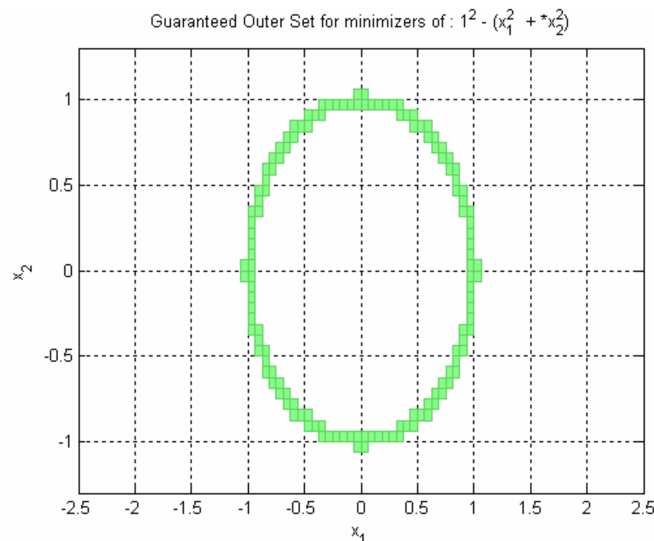
```
function f = FcnTestOptimize1(x,R)
f = (R^2 - (x(1)^2 + x(2)^2))^2;
```

Step 2: Invoke INTOPTIMIZE

```
fcname = 'FcnTestOptimize1'; %name of test function
%precision
dmax=[0.1]; cmax=inf;
%Fixed parameter: radius
R = 1;
%initial domain for parameters
intXo = interval([-inf;-inf],[inf;inf]);
%Global Optimization Via Interval Analysis
[OuterQ,ListB]=intoptimize('FcnTestOptimize1',intXo,[dmax
cmax]);
```

You can plot Inner and Outer sets using routine `INTOPTVISU2D`:

```
[hf, ha] = intoptvisu2D(OuterQ, ListB); %Display results
smartaxis(ha); %smart axis parameters
xlabel('x_1'); ylabel('x_2'); %set labels
title(['Guaranteed Outer Set for minimizers of: ' num2str(R)
'^2 - (x_1^2  + *x_2^2)']); %set title
```



Outer is plotted using boxes whose color is function of their own lower bound. If the lower bounds are all equal to the same value, all boxes are green. Otherwise, a colorbar is shown. Note that all boxes are chained: no box has a lower bound superior to the upper bound of another one.

It is important to note that initial domain could be set to $x_1 \in [-\infty \quad \infty]$ and $x_2 \in [-\infty \quad \infty]$:

```
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
```

Therefore, all the global minimizers of f(x) are in the outer set drawn on the figure.

The parameter `dmax` characterizes the size of resulting boxes: all of them have a normalized maximum width equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)}\left(\frac{width(x_i)}{x_N(i)}\right) \le d_{\max} \quad \text{Rq: } \dim(x)=2 \text{ in this example}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`.

The normalization factors `xN` can be used to increase speed in case of ill conditioned initial domain.

Example of ill conditioned initial domain: $x_1 \in \left[-10^{-8} \quad 10^{-8}\right],\ x_2 \in \left[-10^{8} \quad 10^{8}\right]$

You can let the normalization factors to `[]` so that INTOPTIMIZE compute them automatically. In case of infinite initial domain, there are set to 1. For finite initial domain, `xN` is set to `width(x)` .

The parameter `cmax` characterizes the distance between lower and upper bounds of each resulting boxes: all of them have an upper bound and an lower bound such that:

$$ub(x) - lb(x) \le c_{\max}$$

4.3.7 Unconstrained global minimization with contractor

Consider problem of finding the set of ALL the values $[x_1 \quad x_2]$ that solves:

$$\underset{x}{\text{minimize}}\, f(x) = \left(R - \left(x_1^2 + x_2^2\right)\right)^2$$

To solve this two-dimensional problem, you can:

- write an M-file that returns the function value, then invoke INTOPTIMIZE (cf. §4.3.6
- write an M-file that returns the function value AND the associated contractor, then invoke INTOPTIMIZEC (this paragraph).

Step 1: Write an M-file `FcnTestOptimize1.m` that returns the function value

```
function f = FcnTestOptimize1(x,R)
f = (R^2 - (x(1)^2 + x(2)^2))^2;
```

Step 2: Write an M-file `ContractorFcnTestOptimize1.m` that implements a forward/backard constraint propagation contractor associated to `FcnTestOptimize1`:

```
function x = ContractorFcnTestOptimize1(x, c, R);
%CONTRACTORFCNTESTOPTIMIZE1 contractor function associated to FCNTESTOPTIMIZE1
%
% USES:
%
%   x = ContractorFcnTestPptimize1(x, c, R);
%
% INPUTS:
%
%   x   :   interval
%   c   :   upper bound for the global minimum
%   R   :   radius
%
% OUTPUTS:
%
%   x   :   contracted interval

%get parameters
x1 = x(1);
x2 = x(2);

%R^2
R2 = interval(R^2);

%initialize variables
a1 = interval(-inf, inf);
a2 = interval(-inf, inf);
a3 = interval(-inf, inf);
a4 = interval(-inf, inf);
y  = interval(-inf, inf);

% FORWARD/BACKWARD CONTRACTOR
VolP0 = width(x1)*width(x2);
done=0;
cpt=0;
while ~done
    % update loop index
    cpt=cpt+1;

    %
    % FORWARD  : y = (R^2 - (x1^2 + x2^2))^2
    %
    %a1 = x1^2;
    a1 = psqr(a1, x1);
    %a2 = x2^2;
    a2 = psqr(a2, x2);
    %a3 = x1^2+x2^2;
    a3 = pplus(a3, a1, a2);
    %a4 = R2 - a3 = R^2 - (x1^2 + x2^2);
```

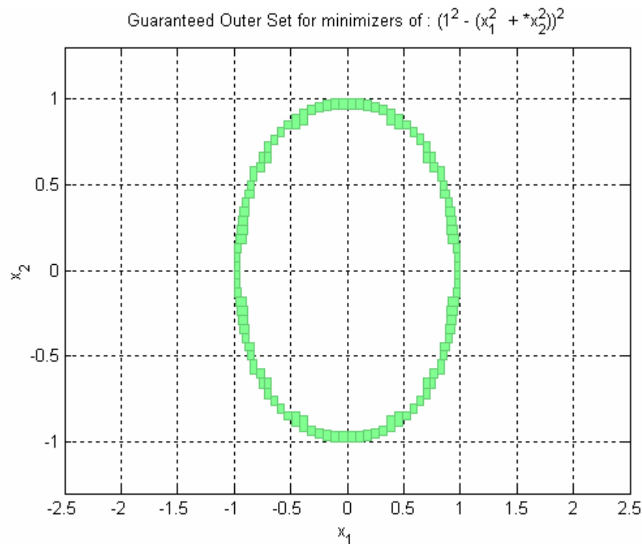
```
    a4 = pminus(a4, R2, a3);
    %y = a4^2;
    y  = psqr(y, a4);
    %
    % CONTRACTION
    %
    y = intersect(y, interval(-inf,c));
    %
    % BACKWARD
    %
    %y  = psqr(a4);
    [y, a4]  = psqr(y, a4);
    %a4 = R2 - a3;
    [a4, junk, a3] = pminus(a4, R2, a3);
    %a3 = pplus(a3, a1, a2);
    [a3, a1, a2] = pplus(a3, a1, a2);
    %a2 = psqr(a2, x2);
    [a2, x2] = psqr(a2, x2);
    %a1 = psqr(a1, x1);
    [a1, x1] = psqr(a1, x1);

    %compute new volume
    VolP = width(x1)*width(x2);
    if VolP>=0.9*VolP0 | cpt>=10
        done=1;
    else
        VolP0=VolP;
    end
end

%output
x(1) = x1;
x(2) = x2;
```

Step 3: Invoke INTOPTIMIZEC

```
fcname = 'FcnTestOptimize1'; %name of test function
contractorfcn   =   'ContractorFcnTestOptimize1';   %name   of
contractor function

%precision
dmax=[0.1]; cmax=inf;
%Fixed parameter: radius
R = 1;
%initial domain for parameters
intXo = interval([-inf;-inf],[inf;inf]);
%Global Optimization Via Interval Analysis
[OuterQ,ListB]=intoptimizec('FcnTestOptimize1',intXo,[dmax
cmax] , [], 'none', '', contractorfcn, R);
```

You can plot Inner and Outer sets using routine `INTOPTVISU2D`:

```
[hf, ha] = intoptvisu2D(OuterQ, ListB); %Display results
smartaxis(ha); %smart axis parameters
xlabel('x_1'); ylabel('x_2'); %set labels
title(['Guaranteed Outer Set for minimizers of: ' num2str(R)
'^2 - (x_1^2  + *x_2^2)']); %set title
```



Outer is plotted using boxes whose color is function of their own lower bound. If the lower bounds are all equal to the same value, all boxes are green. Otherwise, a colorbar is shown. Note that all boxes are chained: no box has a lower bound superior to the upper bound of another one.

It is important to note that initial domain could be set to $x_1 \in [-\infty \quad \infty]$ and $x_2 \in [-\infty \quad \infty]$:

```
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
```

Therefore, all the global minimizers of f(x) are in the outer set drawn on the figure.

The parameter `dmax` characterizes the size of resulting boxes: all of them have a normalized maximum width equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)}\left(\frac{width(x_i)}{x_N(i)}\right) \le d_{\max} \qquad \text{Rq: } \dim(x) = 2 \text{ in this example}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`.

The normalization factors `xN` can be used to increase speed in case of ill conditioned initial domain.

Example of ill conditioned initial domain: $x_1 \in \left[-10^{-8} \quad 10^{-8}\right],\ x_2 \in \left[-10^{8} \quad 10^{8}\right]$

You can let the normalization factors to `[]` so that INTOPTIMIZE compute them automatically. In case of infinite initial domain, there are set to 1. For finite initial domain, `xN` is set to `width(x)` .

The parameter `cmax` characterizes the distance between lower and upper bounds of each resulting boxes: all of them have an upper bound and an lower bound such that:

$$ub(x) - lb(x) \le c_{\max}$$

4.3.8 Constrained global minimization

Consider problem of finding the set of ALL the values $[x_1 \quad x_2]$ that solves:

$$\underset{x}{\text{minimize}}\, f(x) = \left(R - \left(x_1^2 + x_2^2\right)\right)^2$$

under the constraint:

$$g(x_1, x_2) = \left(x_1^2 - 4\right)x_1^2 + 4x_2^2 = 0$$ (See example 1, §4.3.1)

To solve this two-dimensional problem, write:

- an M-file that returns the function value
- an M-file that returns the constraint value

Then, invoke INTOPTIMIZE.

Step 1: Write M-files

1) Write an M-file `FcnTestOptimize1`

```
function f = FcnTestOptimize1(x,R)
f = (R^2 - (x(1)^2 + x(2)^2))^2;
```

2) Write an M-file `constrfcntestoptimize1`

```
function [C, Ceq] = constrfcntestoptimize1(x,R)
% inequality constraints: C(x) <= 0
C  = [];
% equality constraints: Ceq(X) = 0
Ceq = (x(1)^2 - 4)* x(1)^2 + 4*x(2)^2;
```

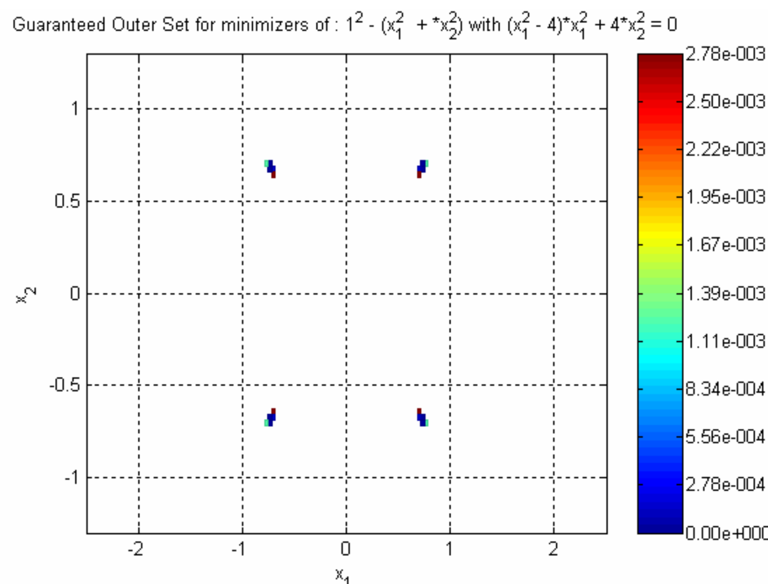
Write both of this functions as if you wanted to use them with the `fmincon` function of the Optimization Toolbox for Matlab.

Step 2: Invoke INTOPTIMIZE

```
fcname = 'FcnTestOptimize1'; %name of test function
%Fixed parameter: radius
R = 1;
%precision
dmax=[0.05]; cmax=inf;
xN = [];    %normalization factors([]->let SIVIA compute them)
%initial domain for parameters
intXo = interval([-inf;-inf],[inf;inf]);
%Global Optimization Via Interval Analysis
[OuterQ,  ListB]  =  intoptimize(fcname,intXo,[dmax  cmax],  xN,
'none', 'constrfcntestoptimize1',R);
```

You can plot Inner and Outer sets using routine `INTOPTVISU2D`:

```
[hf, ha] = intoptvisu2D(OuterQ, ListB); %Display results
axis([-2.5 2.5 -1.3 1.3]); % axis parameters
xlabel('x_1'); ylabel('x_2'); %set labels
title(['Guaranteed Outer Set for minimizers of: ' num2str(R)
'^2 - (x_1^2  + *x_2^2) with (x_1^2 - 4)*x_1^2 + 4*x_2^2 =
0']); %set title
```



Outer is plotted using boxes whose color is function of their own lower bound. If the lower bounds are all equal to the same value, all boxes are green. Otherwise, a colorbar is shown. Note that all boxes are chained: no box has a lower bound superior to the upper bound of another one.

It is important to note that initial domain could be set to $x_1 \in [-\infty \quad \infty]$ and $x_2 \in [-\infty \quad \infty]$:

```
intXo = interval([-inf;-inf],[inf;inf]); %initial domain
```

Therefore, all the global minimizers of f(x) under the constraints g(x) are in the outer set drawn on the figure.

You can see that this outer set is the intersection of resulting outer sets from previous example (See §4.3.6) and example 1 (See §4.3.1).

You can now **split** the list of outer intervals into 4 connex lists by using INTCONNEX (cf. §6.10):

```
[ListX, TabSizeX, UnionX] = intconnex(OuterQ);
```

and then display the outer intervals of each solution:

```
%union of each branches
fprintf('\n\nThere are at most %i different solutions :\n',
length(UnionX));
for idx=1:length(UnionX)
      fprintf('Outer interval for solution %i for [x1;x2] :\n',idx);
      eval(sprintf('S%i = UnionX{%i}',idx,idx));
end
```

These lines produce the following display:

```
There are at most 4 different solutions :
Outer interval for solution 1 for [x1;x2] :
interval S1 =
  [0.6875, 0.78125]
  [ 0.625, 0.71875]
Outer interval for solution 2 for [x1;x2] :
interval S2 =
  [  0.6875, 0.78125]
  [-0.71875,  -0.625]
Outer interval for solution 3 for [x1;x2] :
interval S3 =
  [-0.78125, -0.6875]
  [   0.625, 0.71875]
Outer interval for solution 4 for [x1;x2] :
interval S4 =
  [-0.78125, -0.6875]
  [-0.71875,  -0.625]
```

5. How INTOPTIMIZE works

Global optimization problem:

$$\min_{x \in X} c(x)$$

- $c(x)$ is the cost function to minimize, defined by a user's program
- initial search domain is defined by the interval X

5.1 A global optimization algorithm

5.1.1 Principles

It is a loop between two main steps:

- try to decrease an upper bound c_{sup} of the global minimum c_{opt} of the cost function
- reject or bisect candidates interval to enclose the values x_{opt} where c_{opt} is obtained

5.1.2 Results

- A list of intervals that enclose all the x_{opt}
- A list of associated intervals that enclose the values of the cost function of the first list

5.1.3 How to decrease the upper bound c_{sup}?

The global minimum c_{opt} is lower than any value you can obtain for $x \in X$. So, given an interval [x], you may decrease c_{sup} by one of the following ways:

- Take any $x \in [x]$ (for example, choose the center of [x]). Then apply:

$$c_{\sup} \leftarrow \min(c_{\sup}, c(x))$$

- Do a local minimization on [x]. You obtain x_i and the "minimum" cost c_{supi}, that is in fact a upper bound of the global minimum c_{opt}. Then apply:

$$c_{\sup} \leftarrow \min(c_{\sup}, c_{\sup i})$$

5.1.4 When reject or bisect an interval?

Given an interval [x]:

1) Compute an outer estimation of the image of the cost function through [x]: $c_{out}([x])$. To obtain c_{out}, we just need to call the program c(x) that the user must have defined.

2) If the lower bound of the interval $c_{out}([x])$ is strictly greater than c_{sup}, i.e. if a guaranteed lower bound of c([x]) is strictly greater then a guaranteed upper bound of c_{opt}, it is sure that $x_{opt} \notin [x]$, and we can reject [x]
3) Otherwise bisect [x] into two smaller intervals $[x]_1$ and $[x]_2$, that will replace [x] in the list of candidate intervals

When you bisect [x], you may help to algorithm to:

- Decrease c_{sup} by adding new points : $c_{\sup} \leftarrow \min(c_{\sup}, c(mid([x]_1)), c(mid([x]_2)))$
- Reduce the outer estimation of the interval [x]: $c_{out}([x]) \supset c_{out}([x]_1) \cup c_{out}([x]_2)$

5.1.5 When stop the algorithm?

We may stop bisect [x] if this interval is small enough:

If width([x])<ε:

- Don't bisect [x]
- Put [x] in the list of resulting intervals
- Suppress [x] from the list of candidate intervals

5.2 Implementation

5.2.1 Initialization:

$$c_{\sup} = +\infty$$
$$Q = \{([X], -\infty)\}$$
$$L = \varnothing$$

Q: list of couples (candidate interval [x], lower bound of c([x]))

L: list of resulting intervals that contain all the global minimizers

5.2.2 Algorithm:

Repeat
    Pop out the first interval [x] of Q
    Try to improve c_{sup} using [x]
    Suppress from Q any couple for which the lower bound is strictly greater than c_{sup}
    If width([x]) < ε
        Add [x] to L
    else
        Bisect [x] into $[x]_1$ and $[x]_2$
        Compute the lower bounds lb_1 and lb_2 for $[x]_1$ and $[x]_2$
        Add ($[x]_1$, lb_1) if lb_1<=c_{sup} and ($[x]_2$, lb_2) if lb_2<=c_{sup} to the list Q (*)
Until Q is empty
End task: suppress from Q any couple for which the lower bound is strictly greater than c_{sup}

(*) this test is not necessary: in a simple version of the algorithm, you can put add ($[x]_1$, lb_1) and ($[x]_2$, lb_2) to Q without any test, it works anyway due to the suppress step done at each loop.

5.3 Example

$$\min_{x \in \Re} c(x) = (x+2)^2$$

State of the list Q at the end of each loop:

Q = {([- ∞, +∞], -∞)} c_{sup} = +∞
Q = { ([-∞, 0], 0), ([0, +∞], 4) } c_{sup} = 4
Q = { ([-∞, -1], 0), ([-1, 0], 1)} c_{sup} = 1
Q = { ([-∞, -2], 0), ([-2 -1], 0)} c_{sup} = 0
Q = {([-2 -1], 0), ([-3, -2], 0)} c_{sup} = 0
Q = {([-3, -2], 0), ([-2 -1.5], 0)} c_{sup} = 0
Q = {([-2 -1.5], 0), ([-2.5, -2], 0)} c_{sup} = 0
…

→ The widths of the intervals in Q converge to 0. The estimation of the minimal cost is equal to 0 after only 4 steps.

When the width becomes lower than ε, the intervals of Q are moved into L and the algorithm stops.

5.4 Constraints

It is possible to introduce equality and inequality constraints into the algorithm. You have then to reject an interval no part of it satisfies the constraints. Otherwise, you have to take care that the $x \in [x]$ you choose to decrease c_{sup} satisfies the constraints.

6. Function Reference

This section describes only the major functions of Acsystème Interval Toolbox.

Type `help <function_cname>` for help on other functions.

6.1 SIVIA

Purpose

Find the set X defined by:

$$X = \{x \in X_0 | f(x) \in Y\}$$

where X_0 is the initial set, *Y* is the destination set, *f* is a function from $\Re^{\dim(X_0)}$ to $\Re^{\dim(Y)}$.

This is solved using the equivalent formulation:

$$X = \{x \in X_0 | f_Y(x) = 1\}$$

where f_Y is a test function from $[\Re]^{\dim(X_0)}$ to $\Re$ that returns:

- 0 if $f([x]) \cap Y = \varnothing$
- 1 if $f([x]) \subset Y$
- 0.5 otherwise

where $[x]$ is an interval from the set $[\Re]^{\dim(X_0)}$ of intervals with bounds in $\Re^{\dim(X_0)}$.

Syntax

```
[InnerQ, OuterQ] = sivia(fcname, ListX);
[InnerQ, OuterQ] = sivia(fcname, ListX, dmax);
[InnerQ, OuterQ] = sivia(fcname, ListX, dmax, xN);
[InnerQ, OuterQ] = sivia(fcname, ListX, dmax, xN, P1, ..., PN);
```

Description

SIVIA finds inner and outer approximations of *X* with respects to the following conditions:

$$X_{inner} \subset X \subset (X_{inner} \cup X_{outer})$$

This is generally referred to as *set inversion problem*.

`[InnerQ, OuterQ] = sivia(fcname, ListX);` solves the set inversion problem with f_Y defined by `fcname` and X_0 defined by `ListX`. `ListX` is a cell array of intervals whose union defines the set X_0.

`[InnerQ, OuterQ] = sivia(fcname, ListX, dmax);` uses `dmax` for normalized maximum diameter of resulting boxes instead of default value. The parameter `dmax`

characterizes the size of resulting boxes: all of them have a normalized maximum diameter equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)}\left(\frac{width(x_i)}{x_N(i)}\right) \le d_{\max}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`. By default, `dmax` is set to:

$$d_{\max} = 0.01 \times \max_{i=1..length(ListX)}\left(\max_{j=1..\dim(ListX\{i\})}\left(\frac{width\left(ListX\{i\}_j\right)}{x_N(j)}\right)\right)$$

Where x_N is the vector of normalization factors. Set `dmax` to `[]` in order to use default value.

`[InnerQ, OuterQ] = sivia(fcname, ListX, dmax, xN);` uses xN for normalization factors. The x_N vector can be used to increase speed in case of ill conditioned initial domain. You can set the normalization factors to `[]` in order to use the default value. In case of infinite initial domain, there are set to 1 for each dimension for which domain is infinite. For finite initial domain, `xN` is set to:

$$x_N(j) = \max_{i=1..length(ListX)}\left(width\left(ListX\{i\}_j\right)\right), \quad j = 1..\dim(X)$$

`[InnerQ, OuterQ] = sivia(fcname, ListX, dmax, xN, P1, ..., PN);` passes the problem-dependent parameters P1, ..., PN directly to the function `fcname`. Pass empty matrixes for `dmax` and `xN` if you want to use defaults values for these arguments.

Arguments

Inputs arguments

`fcname`	The test function f_Y that define appurtenance to destination set *Y*
`ListX`	The list of intervals that define the initial set X_0
`dmax`	The maximum normalized width of resulting boxes
`xN`	The normalization factors
`P1, ...`	Other parameters to be passed to `fcname`

Outputs arguments

`InnerQ`	The list of intervals that define the inner approximation of the solution set *X*
`OuterQ`	The list of intervals that define the outer approximation of the solution set *X*

6.2 SIVIAC

Purpose

Find the set X defined by:

$$X = \{x \in X_0 | f(x) \in Y\}$$

where X_0 is the initial set, *Y* is the destination set, *f* is a function from $\Re^{\dim(X_0)}$ to $\Re^{\dim(Y)}$.

This is solved using the equivalent formulation:

$$X = \{x \in X_0 | f_Y(x) = 1\}$$

where f_Y is a test function from $[\Re]^{\dim(X_0)}$ to $\Re$ that returns:

- 0 if $f([x]) \cap Y = \varnothing$
- 1 if $f([x]) \subset Y$
- 0.5 otherwise

where $[x]$ is an interval from the set $[\Re]^{\dim(X_0)}$ of intervals with bounds in $\Re^{\dim(X_0)}$.

This routine can use a user supplied contractor function $g(x)$ that enables the algorithm to contract the domain more efficiently. This contractor function $g(x)$ should implement a contractor associated to the specific inversion problem and return the contracted set x.

Syntax

```
[InnerQ, OuterQ] = siviac(fcname, ListX);
[InnerQ, OuterQ] = siviac(fcname, ListX, dmax);
[InnerQ, OuterQ] = siviac(fcname, ListX, dmax, xN);
[InnerQ, OuterQ] = siviac(fcname, ListX, dmax, xN, contractorfcn);
[InnerQ, OuterQ] = siviac(fcname, ListX, dmax, xN, contractorfcn,
                          P1, P2, ..., PN);
```

Description

SIVIA finds inner and outer approximations of *X* with respects to the following conditions:

$$X_{inner} \subset X \subset (X_{inner} \cup X_{outer})$$

This is generally referred to as *set inversion problem*.

`[InnerQ, OuterQ] = siviac(fcname, ListX);` solves the set inversion problem with f_Y defined by `fcname` and X_0 defined by `ListX.` `ListX` is a cell array of intervals whose union defines the set X_0.

`[InnerQ, OuterQ] = siviac(fcname, ListX, dmax);` uses `dmax` for normalized maximum diameter of resulting boxes instead of default value. The parameter `dmax` characterizes the size of resulting boxes: all of them have a normalized maximum diameter equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)}\left(\frac{width(x_i)}{x_N(i)}\right)\le d_{\max}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`. By default, `dmax` is set to:

$$d_{\max}=0.01\times\max_{i=1..length(ListX)}\left(\max_{j=1..\dim(ListX\{i\})}\left(\frac{width\left(ListX\{i\}_j\right)}{x_N(j)}\right)\right)$$

Where x_N is the vector of normalization factors. Set `dmax` to `[]` in order to use default value.

`[InnerQ, OuterQ] = siviac(fcname, ListX, dmax, xN);` uses xN for normalization factors. The x_N vector can be used to increase speed in case of ill conditioned initial domain. You can set the normalization factors to `[]` in order to use the default value. In case of infinite initial domain, there are set to 1 for each dimension for which domain is infinite. For finite initial domain, `xN` is set to:

$$x_N(j)=\max_{i=1..length(ListX)}\left(width\left(ListX\{i\}_j\right)\right),\quad j=1..\dim(X)$$

`[InnerQ, OuterQ] = siviac(fcname, ListX, dmax, xN, contractorfcn);` passes the problem-dependent contractor function `contractorfcn`:

```
x = contractorfcn(x);
```

`[InnerQ, OuterQ] = siviac(fcname, ListX, dmax, xN, contractorfcn, P1, ..., PN);` passes the problem-dependent parameters P1, ..., PN directly to the function `fcname` and `contractorfcn`. Pass empty matrixes for `dmax` and `xN` if you want to use defaults values for these arguments.

Arguments

Inputs arguments

`fcname`	The test function f_Y that define appurtenance to destination set *Y*
`ListX`	The list of intervals that define the initial set X_0
`dmax`	The maximum normalized width of resulting boxes
`xN`	The normalization factors
`P1, ...`	Other parameters to be passed to `fcname`

Outputs arguments

`InnerQ`	The list of intervals that define the inner approximation of the solution set *X*
`OuterQ`	The list of intervals that define the outer approximation of the solution set *X*

6.3 INTOPTIMIZE

Purpose

Find ALL the minima of a constrained non linear multivariable function

$$\min_{x \in X_0} f(x) \qquad \text{Global Optimization problem}$$

under the constraints:

$$c(x) \leq 0$$
$$ceq(x) = 0$$

where X_0 is the initial set, f , *c* and *ceq* are functions from $\Re^{\dim(X_0)}$ to $\Re$.

Syntax

```
[OuterQ, ListB] = intoptimize(fcname, ListX);
[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax);
[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN);
[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN, options);
[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN, options,
constrfcn);
[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN, options,
constrfcn, P1, P2, ..., PN;
```

Description

INTOPTIMIZE finds outer approximations of the set *X* of all the global minimizers with respects to the following conditions:

$$X = \left\{ x \in X_0 \middle| f(x) = \min_{x \in X_{à}} f(x) \right\} \subset X_{outer}$$

This is generally referred to as *Global Optimization Problem*.

`[OuterQ, ListB] = intoptimize(fcname, ListX);` solves the global unconstrained optimization problem with f defined by `fcname` and X_0 defined by `ListX`. `ListX` is a cell array of intervals whose union defines the set X_0 .

`[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax);` uses `dcmax` for precision parameters instead of default value. `dcmax = [dmax cmax]`, where `dmax` is the normalized maximum diameter of parameters. The parameter `dmax` characterizes the size of resulting boxes: all of them have a normalized maximum diameter equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)} \left(\frac{width(x_i)}{x_N(i)} \right) \leq d_{\max}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`. By default, `dmax` is set to:

$$d_{\max} = 0.01 \times \max_{i=1..length(ListX)} \left(\max_{j=1..\dim(ListX\{i\})} \left(\frac{width(ListX\{i\}_j)}{x_N(j)} \right) \right)$$

Where x_N is the vector of normalization factors.

The parameter `cmax` characterizes the distance between lower and upper bounds of each resulting boxes: all of them have an upper bound and a lower bound such that:

$$ub(x) - lb(x) \le c_{\max}$$

Default value for `cmax` is $+\infty$.

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax` and/or for `cmax`. Set `dcmax` to `[]` in order to use its default value.

`[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN);` uses xN for normalization factors. The x_N vector can be used to increase speed in case of ill conditioned initial domain. You can set the normalization factors to `[]` in order to use the default value. In case of infinite initial domain, there are set to 1 for each dimension for which domain is infinite. For finite initial domain, `xN` is set to:

$$x_N(j) = \max_{i=1..length(ListX)} \left(width(ListX\{i\}_j) \right), \quad j = 1..\dim(X)$$

`[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN, options);` passes `options` to the `fmincon` function of the Optimization Toolbox for Matlab. Create `options` using `optimset`, or set it to `'none'` in order to avoid local minimization (this is the default case). Set options to [] in order to use default local optimizations options:

```
options = optimset('Display','off','largescale','off',
'GradObj','off','MaxIter',20);
```

`[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN, options, constrfcn);` solves the global constrained optimization problem where the constraints are defined by `constrfcn`.

```
[C, Ceq] = constrfcn(x, P1, ..., PN);
```

where *C* is the inequality constraints vector and C_{eq} is the equality constraints vector.

`[OuterQ, ListB] = intoptimize(fcname, ListX, dcmax, xN, options, constrfcn, P1, ..., PN);` passes the problem-dependent parameters P1, ..., PN directly to the function `fcname and constrfcn`.

Arguments

Inputs arguments

`fcname`	The test function f_Y that define appurtenance to destination set *Y*
`ListX`	The list of intervals that define the initial set X_0
`dmax`	The maximum normalized diameter of resulting boxes
`xN`	The normalization factors
`P1, ...`	Other parameters to be passed to `fcname` and `constrfcn`

Outputs arguments

`OuterQ`	The list of intervals that define the outer approximation of the solution set *X*
`ListB`	The list of intervals that define the outer approximation of the cost criteria on each box of OuterQ

6.4 INTOPTIMIZEC

Purpose

Find ALL the minima of a constrained non linear multivariable function

$\min_{x \in X_0} f(x)$ Global Optimization problem

under the constraints:

$$c(x) \leq 0$$
$$ceq(x) = 0$$

where X_0 is the initial set, f , *c* and *ceq* are functions from $\Re^{\dim(X_0)}$ to $\Re$.

This routine can use a user supplied contractor function $g(x, c_{\sup})$that enables the algorithm to contract the domain more efficiently regarding to the current estimation $c_{\sup}$ of the minimal cost. This contractor function $g(x, c_{\sup})$should implement a contractor associated to the specific inversion problem and return the contracted set x .

Syntax

```
[OuterQ, ListB] = intoptimizec(fcname, ListX);
[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax);
[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN);
[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options);
[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options,
constrfcn);
 [OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options,
constrfcn, contractorfcn);
[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options,
constrfcn, contractorfcn, P1, P2, ..., PN;
```

Description

INTOPTIMIZEC finds outer approximations of the set *X* of all the global minimizers with respects to the following conditions:

$$X = \left\{ x \in X_0 \middle| f(x) = \min_{x \in X_0} f(x) \right\} \subset X_{outer}$$

This is generally referred to as *Global Optimization Problem*.

`[OuterQ, ListB] = intoptimizec(fcname, ListX);` solves the global unconstrained optimization problem with f defined by `fcname` and X_0 defined by `ListX`. `ListX` is a cell array of intervals whose union defines the set X_0 .

`[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax);` uses `dcmax` for precision parameters instead of default value. `dcmax = [dmax cmax]`, where `dmax` is the normalized maximum diameter of parameters. The parameter `dmax` characterizes the size of resulting boxes: all of them have a normalized maximum diameter equal of inferior to `dmax`:

$$\max_{i=1\ldots\dim(x)}\left(\frac{width(x_i)}{x_N(i)}\right)\leq d_{\max}$$

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`. By default, `dmax` is set to:

$$d_{\max}=0.01\times\max_{i=1..length(ListX)}\left(\max_{j=1..\dim(ListX\{i\})}\left(\frac{width(ListX\{i\}_j)}{x_N(j)}\right)\right)$$

Where x_N is the vector of normalization factors.

The parameter `cmax` characterizes the distance between lower and upper bounds of each resulting boxes: all of them have an upper bound and a lower bound such that:

$$ub(x)-lb(x)\leq c_{\max}$$

Default value for `cmax` is $+\infty$.

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax` and/or for `cmax`. Set `dcmax` to `[]` in order to use its default value.

`[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN);` uses xN for normalization factors. The x_N vector can be used to increase speed in case of ill conditioned initial domain. You can set the normalization factors to `[]` in order to use the default value. In case of infinite initial domain, there are set to 1 for each dimension for which domain is infinite. For finite initial domain, `xN` is set to:

$$x_N(j)=\max_{i=1..length(ListX)}\left(width(ListX\{i\}_j)\right),\quad j=1..\dim(X)$$

`[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options);` passes `options` to the `fmincon` function of the Optimization Toolbox for Matlab. Create `options` using `optimset`, or set it to `'none'` in order to avoid local minimization (this is the default case). Set options to [] in order to use default local optimizations options:

```
options = optimset('Display','off','largescale','off',
'GradObj','off','MaxIter',20);
```

`[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options, constrfcn);` solves the global constrained optimization problem where the constraints are defined by `constrfcn`.

```
[C, Ceq] = constrfcn(x, P1, ..., PN);
```

`[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options, constrfcn, constractorfcn);` use the specific user supplied contractor `contractorfcn`.

```
x = contractorfcn(x, csup, P1, ..., PN);
```

where `csup` is the current estimation $c_{\sup}$ of the minimal cost.

`[OuterQ, ListB] = intoptimizec(fcname, ListX, dcmax, xN, options, constrfcn, constractorfcn, P1, ..., PN);` passes the problem-dependent parameters P1, ..., PN directly to the function `fcname`, `constrfcn` and `contractorfcn`.

Arguments

Inputs arguments

`fcname` The test function f_Y that define appurtenance to destination set *Y*
`ListX` The list of intervals that define the initial set X_0
`dmax` The maximum normalized diameter of resulting boxes
`xN` The normalization factors
`P1, ...` Other parameters to be passed to `fcname` and `constrfcn`

Outputs arguments

`OuterQ` The list of intervals that define the outer approximation of the solution set *X*
`ListB` The list of intervals that define the outer approximation of the cost criteria on each box of OuterQ

6.5 IMAGESET

Purpose

Find the image of a set by a non linear multivariable function

$$\text{find } Y = \left\{ y \in \Re^p \middle| \exists x \in X_0, y = f(x) \right\}$$

where *p* is the dimension of *Y*, X_0 is the initial set and f is a function from $\Re^{\dim(X_0)}$ to $\Re^p$.

Syntax

```
OuterQ = imageset(fcname, ListX);
OuterQ = imageset(fcname, ListX, dmax);
OuterQ = imageset(fcname, ListX, dmax, P1, P2, ..., PN);
```

Description

IMAGESET finds and outer approximation Y_{outer} of the set *Y* of all the global minimizers with respects to the following condition:

$$Y = \left\{ y \in \Re^p \middle| \exists x \in X_0, y = f(x) \right\} \subset Y_{outer}$$

`OuterQ = imageset(fcname, ListX);` solves the set inversion problem with f_Y defined by `fcname` and X_0 defined by `ListX`. `ListX` is a cell array of intervals whose union defines the set X_0.

`OuterQ = imageset(fcname, ListX);` uses `dmax` for maximum diameter of resulting boxes instead of default value. The parameter `dmax` characterizes the size of resulting boxes: all of them have a maximum diameter equal of inferior to `dmax.`:

You can increase precision (that is, obtain a lower width for outer set boxes) by choosing a lower value for `dmax`. By default, `dmax` is set to:

$$d_{\max} = \min_{i=1..length(ListX)} \left(\min_{j=1..\dim(ListX\{i\})} \left(width\left(ListX\{i\}_j \right) \right) \right)$$

`OuterQ = imageset(fcname, ListX, P1, ..., PN);` passes the problem-dependent parameters P1, ..., PN directly to the function `fcname`.

Arguments

Inputs arguments

`fcname`	The name of the function of which you want to compute image from X_0
`ListX`	The list of intervals that define the initial set X_0
`dmax`	The maximum diameter of resulting boxes
`P1, ...`	Other parameters to be passed to `fcname`

Outputs arguments

`OuterQ`	The list of intervals that define the outer approximation of the image set Y

6.6 INTVISU1D

Purpose

Display the 1D projection of an Inner and Outer Approximation of a set.

Syntax

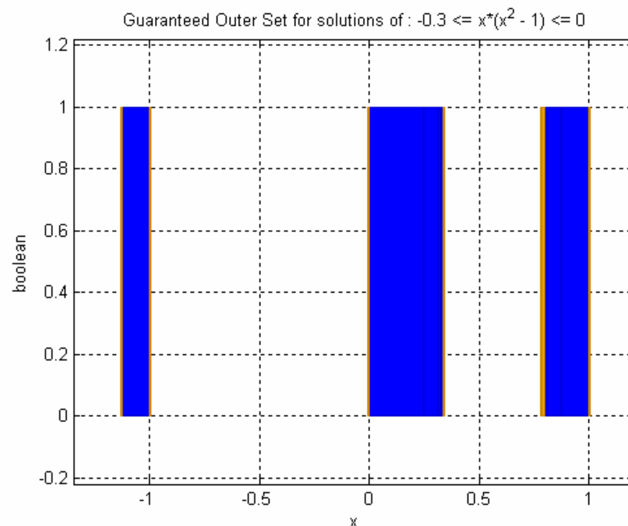
```
[hf, ha] = intvisu1D(InnerQ, OuterQ);
[hf, ha] = intvisu1D(InnerQ, OuterQ, idxX);
```

Description

`[hf, ha] = intvisu1D(InnerQ, OuterQ);` displays the projection of the first dimension of inner and outer sets in a figure.

`[hf, ha] = intvisu1D(InnerQ, OuterQ, idxX);` displays the projection of dimension of indice `idxX`.

An example of result is shown in the following figure:



The Inner set is plotted using blue boxes, and the outer set is plotted using orange boxes.

The abscissa corresponds to the selected dimension. The second dimension is just here to recall the Boolean aspect of the sets: as inner set is certain, it is drawn in blue, that means that boolean value is always 1 on this set, and as outer set may contains some parts that are not in the true set, boolean value may be 0 or 1, and we use orange color.

Arguments

Inputs arguments

`InnerQ`	The list of intervals that define the inner approximation of the set X
`OuterQ`	The list of intervals that define the outer approximation of the set X
`idxX`	The dimension to plot

Outputs arguments

`hf`	The handle of the created figure
`ha`	The handle of the created axe

6.7 INTVISU2D

Purpose

Display the 2D projection of an Inner and Outer Approximation of a set.

Syntax

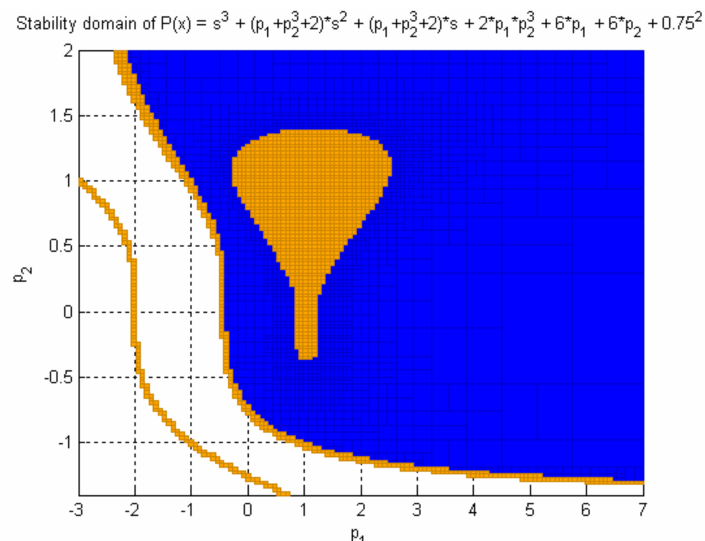
```
[hf, ha] = intvisu2D(InnerQ, OuterQ);
[hf, ha] = intvisu2D(InnerQ, OuterQ, idxX, idxY);
```

Description

`[hf, ha] = intvisu2D(InnerQ, OuterQ);` displays the projection of the two first dimensions of inner and outer sets in a figure.

`[hf, ha] = intvisu2D(InnerQ, OuterQ, idxX, idxY);` displays the projection of dimensions of indices `idxX` and `idxY`.

An example of result is shown in the following figure:



The Inner set is plotted using blue boxes, and the outer set is plotted using orange boxes.

The abscissa corresponds to the first selected dimension and the ordinate to the second dimension.

Arguments

Inputs arguments

`InnerQ`	The list of intervals that define the inner approximation of the set X
`OuterQ`	The list of intervals that define the outer approximation of the set X
`idxX`	The dimension to plot on abscissa
`idxY`	The dimension to plot on ordinate

Outputs arguments

`hf`	The handle of the created figure
`ha`	The handle of the created axe

6.8 INTOPTVISU1D

Purpose

Display the 1D projection of an Outer Approximation of a set, with representation of lower bound of criteria on this set.

Syntax

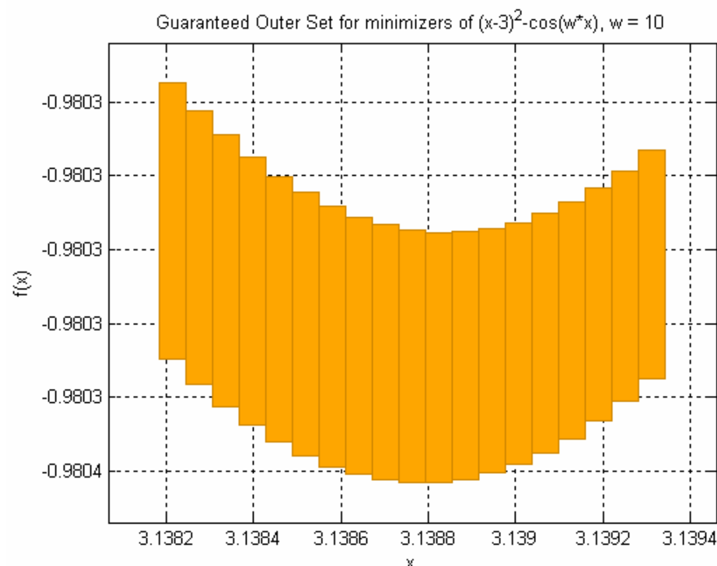
```
[hf, ha] = intoptvisu1D(OuterQ, ListB);
[hf, ha] = intoptvisu1D(OuterQ, ListB, idxX);
```

Description

`[hf, ha] = intoptvisu1D(OuterQ, ListB);` displays the projection of the first dimension of the outer set in a figure.

`[hf, ha] = intoptvisu1D(OuterQ, ListB, idxX);` displays the projection of dimension of indice `idxX` of the outer set.

An example of result is shown in the following figure:



The outer set is plotted using orange boxes. The abscissa corresponds to the values of parameter, while the second dimension corresponds to the lower and upper bounds for values of criteria on theses intervals.

Arguments

Inputs arguments

`OuterQ`	The list of intervals that define the outer approximation of the set X
`ListB`	The list of intervals that define the corresponding values of a criteria on X
`idxX`	The dimension of OuterQ to plot

Outputs arguments

`hf`	The handle of the created figure
`ha`	The handle of the created axe

6.9 INTOPTVISU2D

Purpose

Display the 2D projection of an Outer Approximation of a set, using colors with regards to lower bounds on these boxes.

Syntax

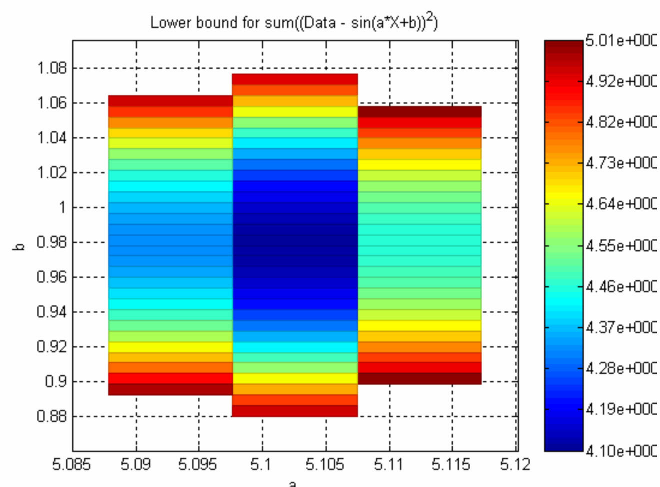
```
[hf, ha] = intoptvisu2D(OuterQ, ListB);
[hf, ha] = intoptvisu2D(OuterQ, ListB, idxX, idxY);
```

Description

`[hf, ha] = intoptvisu2D(OuterQ, ListB);` displays the projection of the two first dimensions of the outer set in a figure, using colors with regards to lower bounds on these boxes.

`[hf, ha] = intoptvisu2D(OuterQ, ListB, idxX, idxY);` displays the projection of dimensions of indices `idxX` and `idxY`.

An example of result is shown in the following figure:



The abscissa corresponds to the first selected dimension and the ordinate to the second dimension. A colorbar presents the relation between the color and the value of the lower bound of criteria on the displayed boxes. If all lower bounds are equal, the color used is the central color of the colormap. The colormap used is the Matlab default colormap.

Arguments

Inputs arguments

`OuterQ`	The list of intervals that define the outer approximation of the set X
`ListB`	The list of intervals that define the corresponding values of a criteria on X
`idxX`	The first dimension of OuterQ to plot
`idxY`	The second dimension of OuterQ to plot

Outputs arguments

`hf`	The handle of the created figure
`ha`	The handle of the created axe

6.10 INTCONNEX

Purpose

Split a list of intervals into connex branches.

Syntax

```
ListX = intconnex(ListQ);

[ListX, TabSizeX, UnionX] = intconnex(ListQ);
```

Description

`ListX = intconnex(ListQ)` compute the interval result using the formula.

Arguments

Inputs arguments

`ListQ` cell array that contains intervals

Outputs arguments

`ListX` cell array that contains the connex branches

- ListX has N rows, where N is the number of connex branches
- each row of ListX contains all the intervals of the same connex branch

`TabSizeX` vector of the length of each connex branch

`UnionX` cell array of union of interval of each connex branch

Warning

This algorithm is about n^3 complexity (n=length(ListQ)).

It should take about 1 minute with n=100.

Note it may be **very** slow if n is greater than 100.

6.11 HORNER

Purpose

Use Horner's method to compute polynomial image.

Syntax

```
out = horner(P, x)
```

Description

`out = horner(P, x)` compute the interval result using the formula

$$\sum_{i=0}^{n} a_i x^i = x(x(x(xa_n + a_{n-1}) + a_{n-2})\ldots\ldots + a_1) + a_0 .$$

Arguments

Inputs arguments

`P`	The polynomial coefficients (may be a vector of intervals)
`x`	The input intervals

Outputs arguments

`out`	The result P(x)

6.12 REALROOTS

Purpose

Find the guaranteed real roots of a polynomial.

Syntax

```
r = realroots(P)

r = realroots(P, tol)
```

Description

`r = realroots(P)` compute a list of intervals that are guaranteed to contain the real roots of the polynomial P. The width of each interval should be less than the default tolerance (1e-6).

`r = realroots(P, tol)` uses the user supplied tolerance defined by tol.

Note : `realroots` calls the generic contractor `ppoly` and a specific SIVIA algorithm.

Arguments

Inputs arguments

`P`	The polynomial coefficients (may be a vector of intervals)
`tol`	Tolerance on roots (default is 1e-6)

Outputs arguments

`r`	List of intervals that contains the real roots of the polynomial P. If r is empty, that means the polynomial P is guaranteed to have no real root

6.13 INTBISECT

Purpose

Bisect an interval into lower and upper parts.

Syntax

```
[xlow, xsup] = intbisect(x, idxsplit)
```

Description

`[xlow, xsup] = intbisect(x, idxsplit)` bisect the Interval along the dimension of `idxsplit`.

When the interval is finite, the formula to find the centre is $centre = \frac{ub + lb}{2}$.

For particular case of infinite intervals, conventions are used:

- [-inf, inf] is bisected into [-inf, 0] and [0, inf]
- [0, inf] is bisected into [0, 1] and [1, inf],
- [lb>0, inf] is bisected into [lb, 2*lb] and [2*lb, inf],
- [lb<0, inf] is bisected into [lb, -lb] and [-lb, inf],
- [-inf, 0] is bisected into [-inf, -1] and [-1, 0],
- [-inf, ub >0] is bisected into [-inf, -ub] and [-ub, ub],
- [-inf, ub<0] is bisected into [-inf, 2*ub] and [2*ub, ub].

Arguments

Inputs arguments

`x`	The interval to bisect
`idxsplit`	The bisection dimension

Outputs arguments

`xlow`	The lower part interval
`xsup`	The upper part interval

6.14 INTCONV

Purpose

Product of polynomials

Syntax

```
out = intconv(A,B);
```

Description

`out = intconv(A,B);` computes the convolution of the polynomials A and B.

Arguments

Inputs arguments

`A`	The polynomial coefficients
`B`	The polynomial coefficients

Outputs arguments

`out`	The polynomial coefficient A*B

6.15 INTHASPOLYIMAGROOT

Purpose

Check if a polynomial whose coefficients are intervals has one or more imaginary root

Syntax

```
ibboo = intHasPolyImagRoot(P, w);
```

Description

`ibboo = intHasPolyImagRoot(P, w);` computes if `j*w` is a root of the polynomial `P`.

`ibboo` is TRUE (1) if all the roots are on j*w,

`ibboo` is FALSE (0) if `P` hasn't any root on j*w,

Otherwise, `ibboo` is indeterminate (0.5).

Arguments

Inputs arguments

`P`	The polynomial coefficients of P
`w`	The interval of frequency w

Outputs arguments

`ibboo`	The result

6.16 INTROUTH

Purpose

Evaluation of Routh criteria on Intervals.

Syntax

```
ibboo = routh(P);
```

Description

`ibboo = routh(P);` computes the Routh criteria and determines if the polynomial is stable or not. A polynomial P is said to be stable if and only of all the real parts of P roots are negative.

`ibboo` is TRUE (1) when P is guaranteed to be stable when coefficients belongs to P,

`ibboo` is FALSE (0) when P is guaranteed to be unstable when coefficients belongs to P,

Otherwise, `ibboo` is indeterminate (0.5).

Arguments

Inputs arguments

`P`	The polynomial coefficients of P

Outputs arguments

`ibboo`	The result

6.17 INTTSTROUTH

Purpose

Test if a polynomial is stable by evaluation of Routh criteria on Intervals.

Syntax

```
ibboo = inttstrouth(x, fcname);
```

```
ibboo = inttstrouth(x, fcname, P1, … PN);
```

Description

`ibboo = inttstrouth(x, fcname);` test if the polynomial whose coefficients are defined by `fcname` is stable with the interval `x`.

`ibboo = inttstrouth(x, fcname, P1, … PN);` `P1, … Pn` are other parameters to pass to `fcname`.

`ibboo` is TRUE (1) when the polynomial is guaranteed to be stable,

`ibboo` is FALSE (0) when the polynomial is guaranteed to be unstable,

Otherwise, `ibboo` is indeterminate (0.5).

Arguments

Inputs arguments

`x` An interval

`fcname` Function that compute polynomial coefficients

`P1, … PN` Other parameters to pass to `fcname`

Outputs arguments

`ibboo` The result of the test

6.18 INTTSTZERO

Purpose

Test if a function is zero.

Syntax

```
ibboo = inttstzero(x, fcname);
```

```
ibboo = inttstzero(x, fcname, P1, … PN);
```

Description

`ibboo = inttstzero(x, fcname);` test if `fcname` is always zero with the interval `x`.

`ibboo = inttstrouth(x, fcname, P1, … PN); P1, … Pn` are other parameters to pass to `fcname`.

`ibboo` is TRUE (1) when fcname(x, P1, …, PN) is exactly 0.

`ibboo` is FALSE (0) when fcname(x, P1, …, PN) does not contain 0,

`ibboo` is indeterminate (0.5) when fcname(x, P1, …, PN) contains 0.

Arguments

Inputs arguments

`x`	An interval
`fcname`	Function that to be tested
`P1, … PN`	Other parameters to pass to `fcname`

Outputs arguments

`ibboo`	The result of the test

6.19 INTBOUNDPOLYDROOTS

Purpose

Compute guaranteed lower and upper bounds of the module of roots of a polynomial.

Syntax

```
[lb, ub] = intboundpolyroots(P);
```

Description

`[lb, ub] = intboundpolyroots (P);` computes the lower and upper bounds of the module roots of polynomial `P` using the Marden theorem.

Arguments

Inputs arguments

`P`	Polynomial coefficients (decreasing power)

Outputs arguments

`lb`	The guaranteed lower bound
`ub`	The guaranteed upper bound

6.20 MERGEINNEROUTER

Purpose

Merge Inner and Outer lists.

Syntax

```
mergeSet = mergeInnerOuter(InnerQ, OuterQ);
```

Description

`mergeSet = mergeInnerOuter(InnerQ, OuterQ);` merge the two lists InnerQ and OuterQ in one.

Arguments

Inputs arguments

`InnerQ`	List of inner intervals (Intervals cell array)
`OuterQ`	List of outer intervals (Intervals cell array)

Outputs arguments

`MergeSet`	The merge List (Intervals cell array)

6.21 MINCE

Purpose

Mince list of intervals into a list smaller intervals.

Syntax

```
Xmince = mince(ListX);
```

```
Xmince = mince(ListX, dmax);
```

Description

`Xmince = mince(ListX);` minces the list `ListX`. By default, the maximum width of output intervals is computed to the minimum of all the width.

`Xmince = mince(ListX, dmax);` minces the list `ListX` using the parameter `dmax` to define the maximum width of output intervals;

Arguments

Inputs arguments

`ListX`	List of intervals (Intervals cell array)
`dmax`	Maximum width of output intervals

Outputs arguments

`Xmince`	The List of minced intervals that define the same set

6.22 POLY2REIM

Purpose

Split a polynomial into real and imaginary parts so that $P(j\omega) = Pr(\omega) + j \times Pi(\omega)$.

Syntax

```
[Pr, Pi] = poly2reim(P);
```

Description

`[Pr, Pi] = poly2reim(P);` splits the polynomial P into real part and imaginary part so that $P(j\omega) = Pr(\omega) + j \times Pi(\omega)$.

Arguments

Inputs arguments

`P`	Polynomial coefficients

Outputs arguments

`Pr`	Real part
`Pi`	Imaginary part

7. References

[1] *Applied Interval Analysis*, Luc Jaulin, Michel kieffer, Olivier Didrit and Eric Walter, Springer 2001

[2] *Optimization toolbox, User Guide*, The Mathworks, Version 2