IBEX

www.ibex-lib.org

Gilles Chabert
Ecole des Mines de Nantes

Dec 2014

Outline

Introduction

Contractors

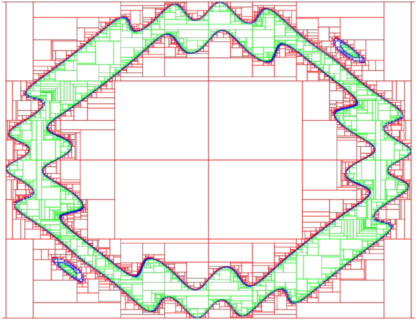
- Forward-backward
- XTaylor
- Q-intersection

Getting started

Conclusion

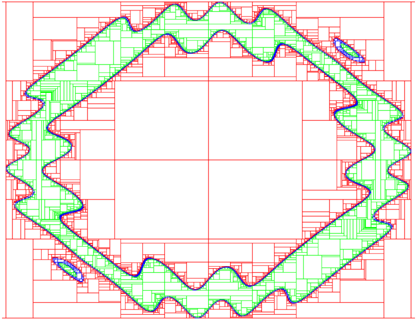
Introduction

IBEX is an open-source C++ library for **set computation**, based on interval arithmetic.



Introduction

IBEX is an open-source C++ library for **set computation**, based on interval arithmetic.



A **toolbox philosophy**: IBEX provides composable software modules (or "blocks") to allow high-level programming.

Introduction

The library also includes two end-user programs:

Introduction

The library also includes two end-user programs:

- a **solver** for systems of nonlinear equations/inequalities

Introduction

The library also includes two end-user programs:

- a **solver** for systems of nonlinear equations/inequalities
- a global **optimizer**

Introduction

The library also includes two end-user programs:

- a **solver** for systems of nonlinear equations/inequalities
- a global **optimizer**

Both come in three flavours:

Introduction

The library also includes two end-user programs:

- a **solver** for systems of nonlinear equations/inequalities
- a global **optimizer**

Both come in three flavours:

- *black-box*: model the problem and press “enter”.

Introduction

The library also includes two end-user programs:

- a **solver** for systems of nonlinear equations/inequalities
- a global **optimizer**

Both come in three flavours:

- *black-box*: model the problem and press “enter”.
- *gray-box*: set some advanced parameters and chose one of the built-in strategies

Introduction

The library also includes two end-user programs:

- a **solver** for systems of nonlinear equations/inequalities
- a global **optimizer**

Both come in three flavours:

- *black-box*: model the problem and press “enter”.
- *gray-box*: set some advanced parameters and chose one of the built-in strategies
- *generic*: run the program with your own modules

Introduction

Type of problems usually solved

- classical non-linear programs (system with possibly objective function)

Introduction

Type of problems usually solved

- classical non-linear programs (system with possibly objective function)
- heterogenous constraint systems (algebraic, differential, combinatorial, data processing, ...)

Introduction

Type of problems usually solved

- classical non-linear programs (system with possibly objective function)
- heterogenous constraint systems (algebraic, differential, combinatorial, data processing, ...)
- set characterization (set inversion, set image, set projection, ...)

Introduction

Type of problems usually solved

- classical non-linear programs (system with possibly objective function)
- heterogenous constraint systems (algebraic, differential, combinatorial, data processing, . . .)
- set characterization (set inversion, set image, set projection, . . .)
- parameter/state estimation

Common denominator

Introduction

Common denominator

- importance of handling **rigorously** uncertainties (computer-aided proofs)

Introduction

Common denominator

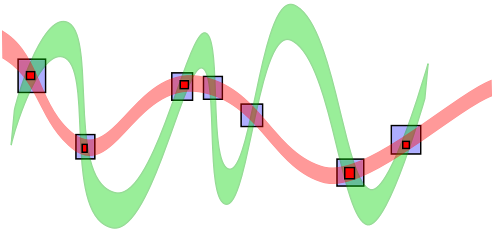
- importance of handling **rigorously** uncertainties (computer-aided proofs)
- uncertainty represented with **bounded errors**

Introduction

Common denominator

- importance of handling **rigorously** uncertainties (computer-aided proofs)
- uncertainty represented with **bounded errors**

Example



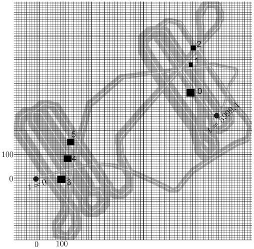
Let $f_p : \mathbb{R}^n \to \mathbb{R}^m$, $p \in [p]$.

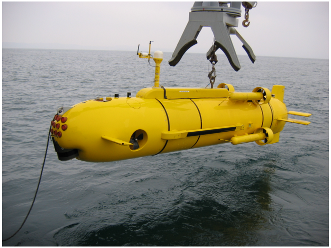
$$\mathcal{S}_1 \supseteq \{x, \exists p \in [p],\ f_p(x) = 0\} \supseteq \mathcal{S}_2$$

Introduction

Application fields

- artificial intelligence, autonomous robotics

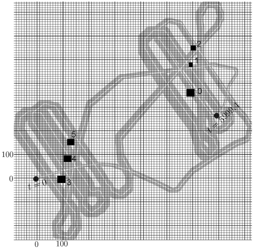


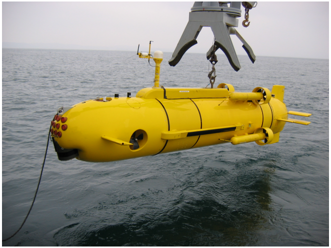


Introduction

Application fields

- artificial intelligence, autonomous robotics
- automation

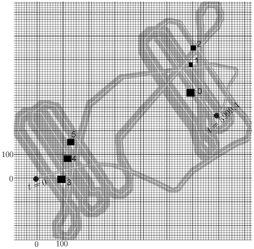


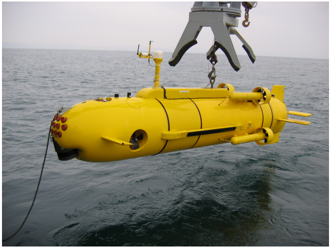


Introduction

Application fields

- artificial intelligence, autonomous robotics
- automation
- geometry, CAD

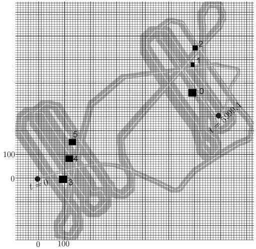


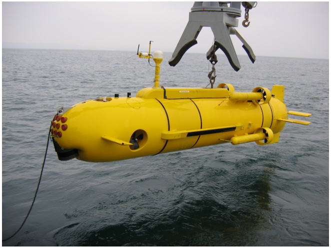


Introduction

Application fields

- artificial intelligence, autonomous robotics
- automation
- geometry, CAD
- material science





Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias

Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias
- LP solvers comptability: Cplex, Soplex, CLP

Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias
- LP solvers comptability: Cplex, Soplex, CLP
- OS comptability: Linux, MacOS, Windows (MinGW/Visual)

Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias
- LP solvers comptability: Cplex, Soplex, CLP
- OS comptability: Linux, MacOS, Windows (MinGW/Visual)
- Platforms: 32/64 bits

Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias
- LP solvers comptability: Cplex, Soplex, CLP
- OS comptability: Linux, MacOS, Windows (MinGW/Visual)
- Platforms: 32/64 bits
- Compilation mode: static/dynamic

Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias
- LP solvers comptability: Cplex, Soplex, CLP
- OS comptability: Linux, MacOS, Windows (MinGW/Visual)
- Platforms: 32/64 bits
- Compilation mode: static/dynamic
- Installation mode: local/system

Introduction

Installation features

- Interval libraries comptability: Gaol, Filib, Profil/Bias
- LP solvers comptability: Cplex, Soplex, CLP
- OS comptability: Linux, MacOS, Windows (MinGW/Visual)
- Platforms: 32/64 bits
- Compilation mode: static/dynamic
- Installation mode: local/system
- Bridge with a Java library for discrete optimization (CHOCO)

Introduction

Resources

- Web site: `ibex-lib.org`

Introduction

Resources

- Web site: `ibex-lib.org`
- Documentation: `ibex-lib.org/doc/`

Introduction

Resources

- Web site: `ibex-lib.org`
- Documentation: `ibex-lib.org/doc/`
- Source code: `github.com/ibex-team/ibex-lib`

Introduction

Resources

- Web site: `ibex-lib.org`
- Documentation: `ibex-lib.org/doc/`
- Source code: `github.com/ibex-team/ibex-lib`
- Forum: `ibex-lib.org/forum`

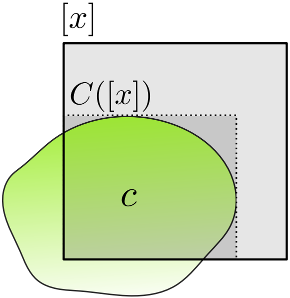
Outline

Contractors

IBEX is a *contractor-oriented* library.

A contractor is the main type of block in IBEX.

Given a set S on $\mathbb{IR}^n$, we call *contractor* with respect to S an algorithm C such that,



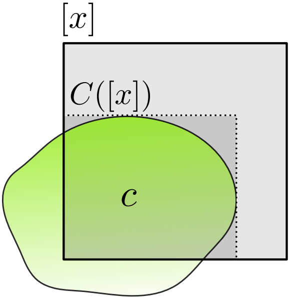
Contractors

IBEX is a *contractor-oriented* library.

A contractor is the main type of block in IBEX.

Given a set S on $\mathbb{IR}^n$, we call *contractor* with respect to S an algorithm C such that,

$$\forall [x] \in \mathbb{IR}^n \quad C([x]) \supseteq \{x \in [x] \cap S\}.$$



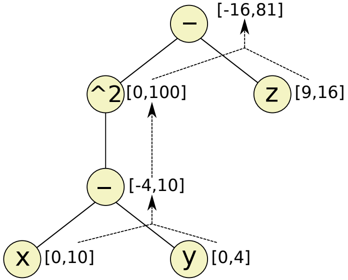
Contractors

We give now 3 examples of contractors that are implemented in IBEX.

Outline

Forward-backward

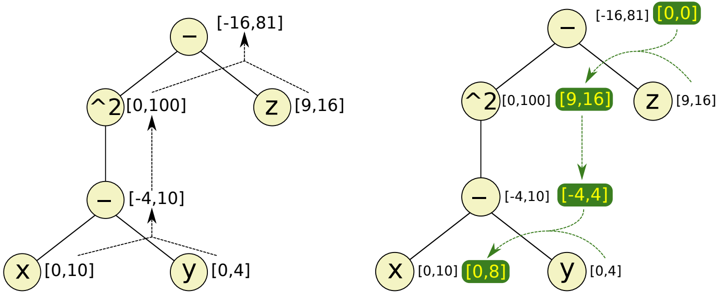
Example with $S = \{(x, y), \quad (x - y)^2 - z = 0\}$:



forward step

Forward-backward

Example with $S = \{(x,y), \quad (x-y)^2 - z = 0\}$:

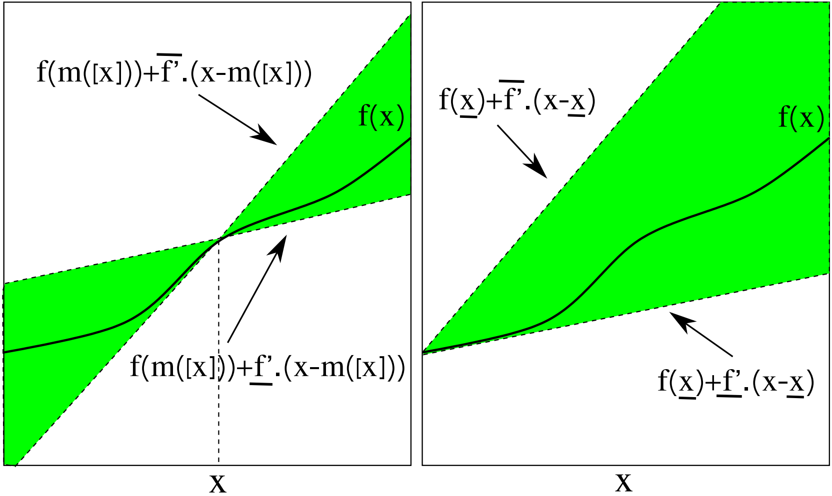


forward step *backward step*

Outline

Contractors

Example 2: **XTaylor**



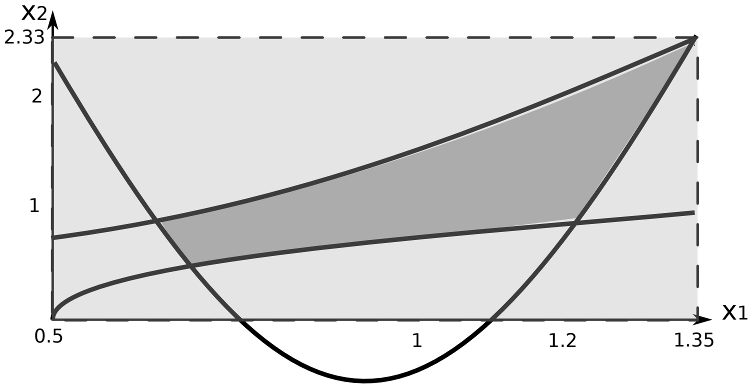
XTaylor

$$x_1^2 - x_2 + 0.5 \leq 0$$

$$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$$

$$sqrt(x_1 - 0.5) - x_2 \leq 0$$

$$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$$



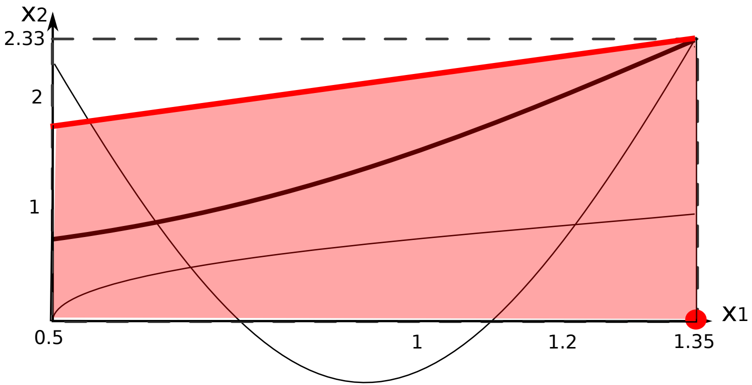
XTaylor

$$x_1^2 - x_2 + 0.5 \leq 0$$

$$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$$

$$sqrt(x_1 - 0.5) - x_2 \leq 0$$

$$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$$



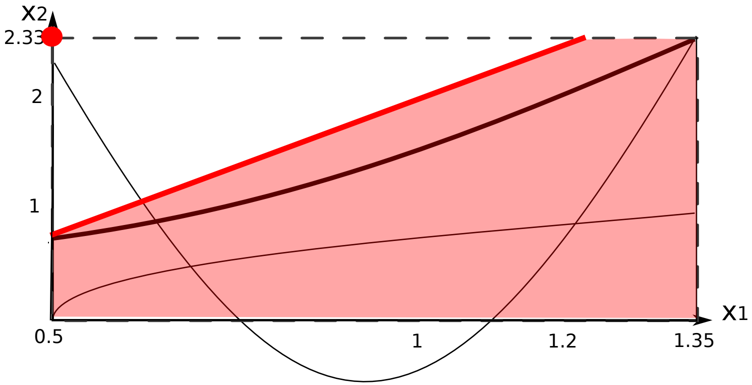
XTaylor

$$x_1^2 - x_2 + 0.5 \leq 0$$

$$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$$

$$sqrt(x_1 - 0.5) - x_2 \leq 0$$

$$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$$



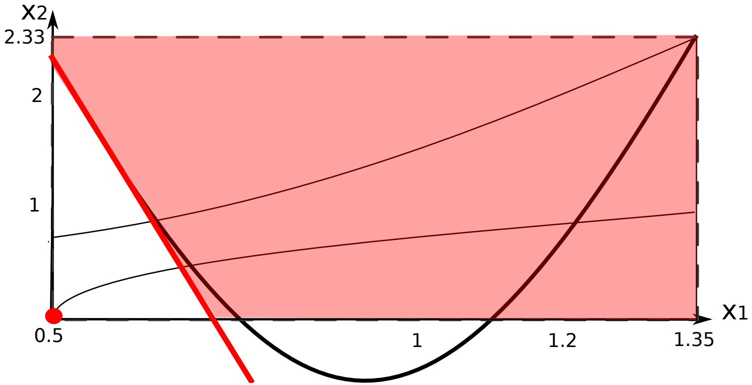
XTaylor

$x_1^2 - x_2 + 0.5 \leq 0$

$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$

$sqrt(x_1 - 0.5) - x_2 \leq 0$

$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$



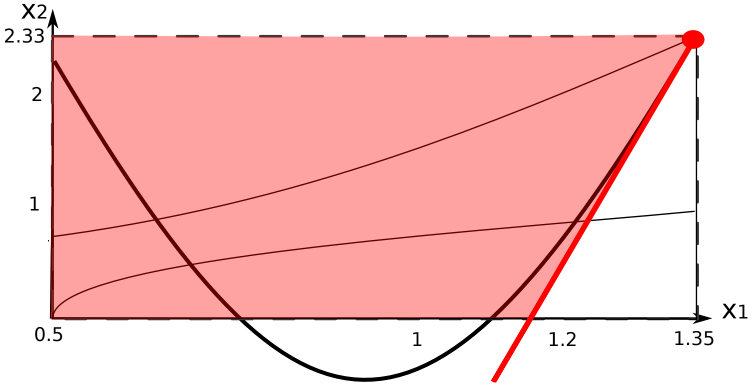
XTaylor

$$x_1^2 - x_2 + 0.5 \leq 0$$

$$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$$

$$sqrt(x_1 - 0.5) - x_2 \leq 0$$

$$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$$



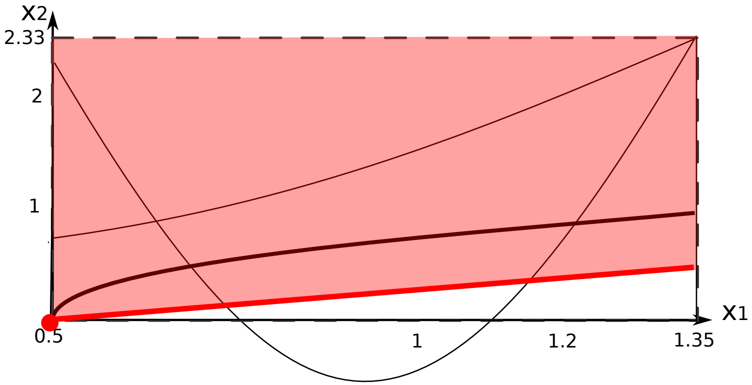
XTaylor

$$x_1^2 - x_2 + 0.5 \leq 0$$

$$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$$

$$sqrt(x_1 - 0.5) - x_2 \leq 0$$

$$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$$



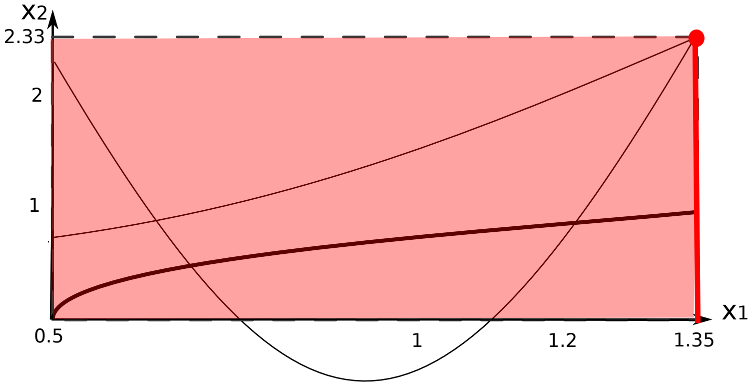
XTaylor

$x_1^2 - x_2 + 0.5 \leq 0$

$-2.5\,sin(4x_1 + 1) + x_2 \leq 2$

$sqrt(x_1 - 0.5) - x_2 \leq 0$

$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$



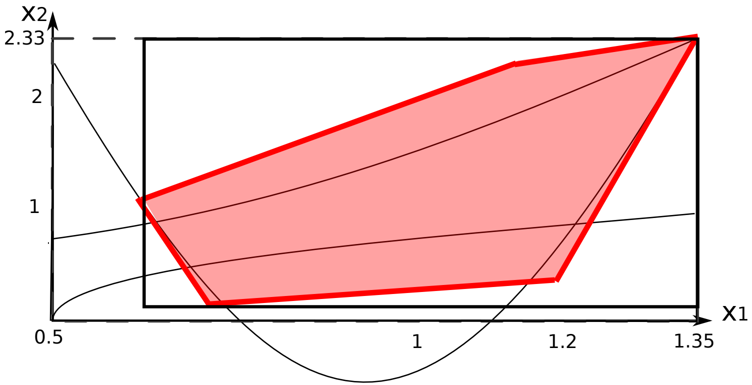
XTaylor

$$x_1^2 - x_2 + 0.5 \leq 0$$

$$-2.5\, sin(4x_1 + 1) + x_2 \leq 2$$

$$sqrt(x_1 - 0.5) - x_2 \leq 0$$

$$\text{Domain} = [0.5, 1.35] \times [0, 2.33]$$



Outline

Q-intersection

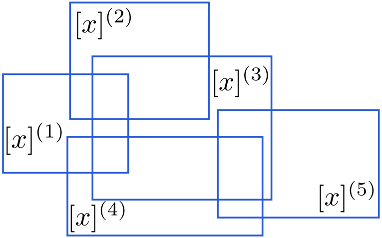
Definition

The q-intersection of a set of boxes S is the smallest box encompassing all the points that belong to at least q boxes of S.

Q-intersection

Definition

The q-intersection of a set of boxes S is the smallest box encompassing all the points that belong to at least q boxes of S.

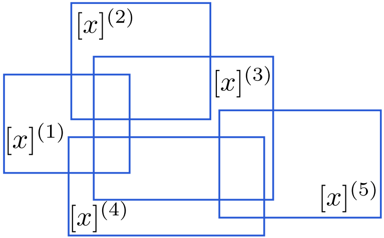


Boxes (S)

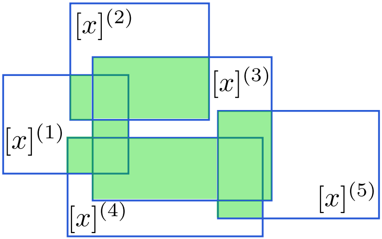
Q-intersection

Definition

The q-intersection of a set of boxes S is the smallest box encompassing all the points that belong to at least q boxes of S.



Boxes (S)

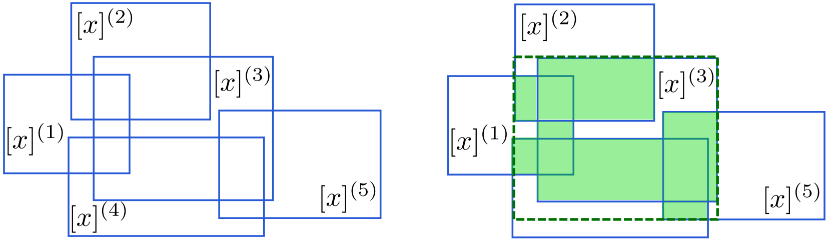


Points that belong to 2 boxes

Q-intersection

Definition

The q-intersection of a set of boxes *S* is the smallest box encompassing all the points that belong to at least *q* boxes of *S*.

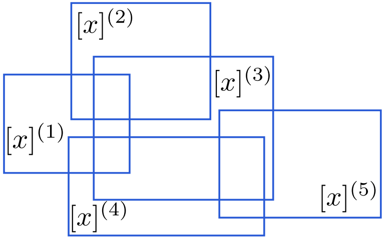


Boxes (S) *2-intersection*

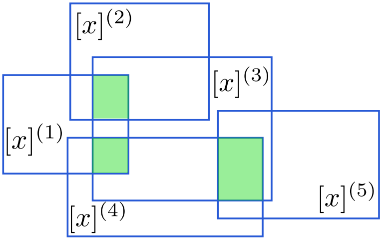
Q-intersection

Definition

The q-intersection of a set of boxes S is the smallest box encompassing all the points that belong to at least q boxes of S.



Boxes (S)

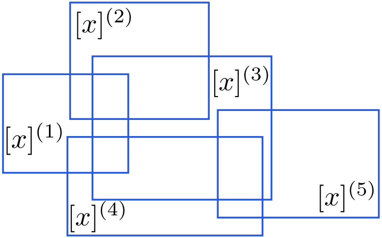


Points that belong to 3 boxes

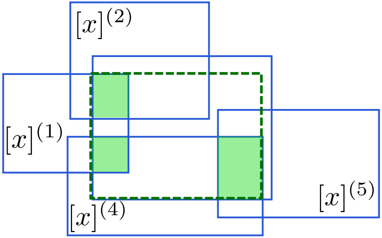
Q-intersection

Definition

The q-intersection of a set of boxes S is the smallest box encompassing all the points that belong to at least q boxes of S.



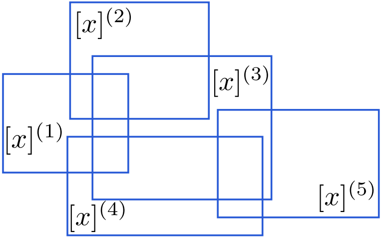
Boxes (S)



3-intersection

Q-intersection

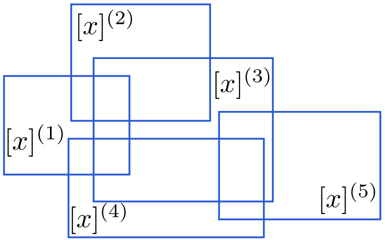
The q-intersection can be calculated from intersection graph.



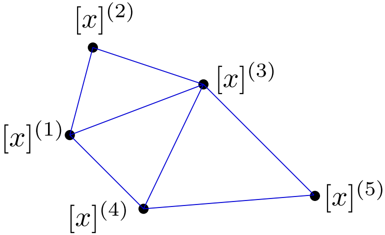
Boxes (S)

Q-intersection

The q-intersection can be calculated from intersection graph.



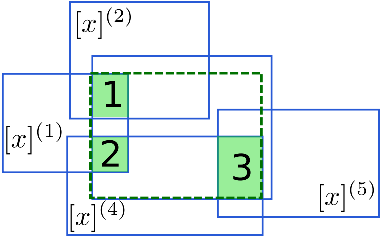
Boxes (S)



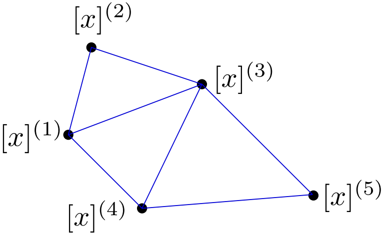
Intersection graph (G)

Q-intersection

The q-intersection can be calculated from intersection graph.



3-intersection

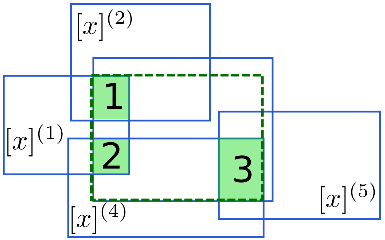


Intersection graph (G)

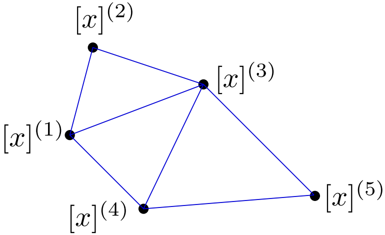
Q-intersection

The q-intersection can be calculated from intersection graph.

➡ A point does not belong to the green area if it does not belong to a *q*-clique:



3-intersection

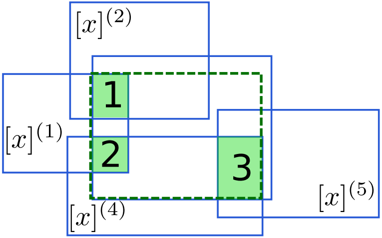


Intersection graph (G)

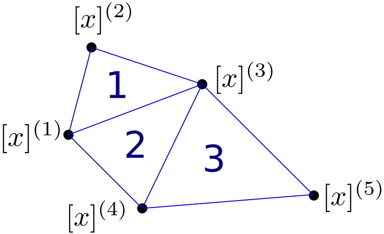
Q-intersection

The q-intersection can be calculated from intersection graph.

➡ A point does not belong to the green area if it does not belong to a *q*-clique:



3-intersection



Corresponding 3-cliques

Q-intersection

We have presented the q-intersection as an operator on **sets of boxes.**

Q-intersection

We have presented the q-intersection as an operator on **sets of boxes.**

This operator is also extended to an operator on **sets of contractors**:

Q-intersection

We have presented the q-intersection as an operator on **sets of boxes.**

This operator is also extended to an operator on **sets of contractors**:

Definition (Q-intersection operator)

The q-intersection is an operator that maps n contractors $C_1, \ldots, C_n$ and $q \in \mathbb{N}$ to the following contractor:

$$\text{qinter}(q, C_1, \ldots, C_n) : [x] \mapsto \cap^q \Big(C_1([x]), \ldots, C_n([x]) \Big).$$

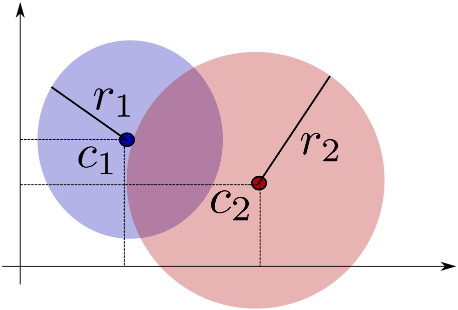
Outline

Getting started

The goal of this example is to show:

- how to use some generic contractors supplied in IBEX.
- how to implement a specific contractor and make it collaborate with the generic ones.

This will be based on the simple example of the intersection of two circles.



Getting started

Create the parameters of the problem

```
/* parameters of the first circle */
IntervalVector c1(2);
c1[0]=0; c1[1]=0;
double r1=1;

/* parameters of the second circle */
IntervalVector c2(2);
c2[0]=2; c2[1]=1;
double r2=2;
```

Getting started

Model the problem as an equation $f(x) = 0$.

```
/* create the distance function with 2 arguments */
Variable x(2);
Variable y(2);
Function dist(x,y,sqrt(sqr(x[0]-y[0])+sqr(x[1]-y[1])));

/* model of the problem (f must be equal to 0) */
Function f(x,Return(dist(x,c1)-r1, dist(x,c2)-r2));
```

Create a forward-backward contractor

```
CtcFwdBwd ctc1(f);
```

Getting started

Run the contractor. Example:

$$c_1 = (0,0),\ r_1 = 1,\ c_2 = (2,1),\ r_2 = 2.$$

Solutions:

$$\{(0.8, -0.6),\ (0,1)\}.$$

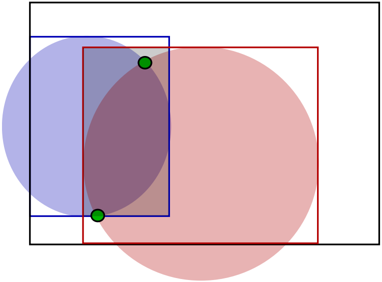
```
IntervalVector box(2,Interval(-3,3));
cout << "box before=" << box << endl;
ctc.contract(box);
cout << "box after=" << box << endl;
```

The program gives:

```
box before=([-3, 3] ; [-3, 3])
box after=([0, 1] ; [-0.732051, 1])
```

Getting started

The forward-backward contractor



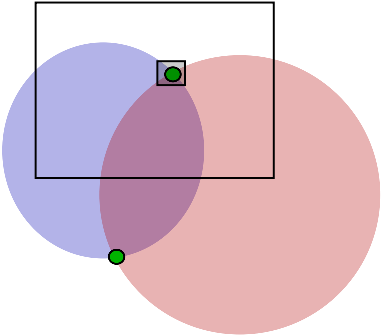
Contract in almost any cases and optimaly with respect to each circle... but not with respect to the intersection.

Including if there is only one solution:

```
box before=([0.799, 0.801] ; [-0.601, -0.599])
box after=([0.799249, 0.800749] ; [-0.600561, -0.599436])
```

Getting started

The interval Newton contractor



If the box is enclosing one solution, the interval Newton can give an optimal contraction.

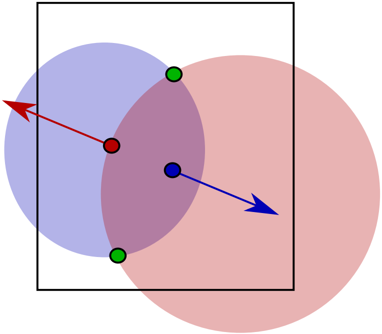
Getting started

```
CtcNewton ctc2(f);

IntervalVector sol(2);
sol[0]=0.8;
sol[1]=-0.6;
IntervalVector box=sol.inflate(1e-3);

cout << "box before=" << box << endl;
ctc2.contract(box);
cout << "box after=" << box << endl;
```

```
box before=([0.799, 0.801] ; [-0.601, -0.599])
box after=([0.8, 0.8] ; [-0.6, -0.6])
```

Getting started

The interval Newton contractor



But it may not contract at all if a singularity occurs (especially in the case of a box enclosing two solutions).

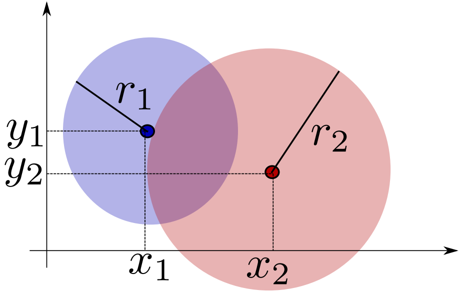
```
box before=([-3, 3] ; [-3, 3])
box after=([-3, 3] ; [-3, 3])
```

Getting started

A dedicated contractor

The two intersection point of two circles can actually be formally obtained.

Consider two circles with origin points (x_1, y_1) and (x_2, y_2) and radii r_1 and r_2.



Getting started

A dedicated contractor

Let d be the distance between the origins of the two circles.

- If $d > r_1 + r_2$ then the two circles are distant from each others (no intersection)
- If $d < r_1 - r_2$ or $d < r_2 - r_1$, then one circle is included in the other (no intersection)
- Otherwise, the two solutions are given by:

$$x = \frac{x_2 + x_1}{2} + \frac{(x_2 - x_1)(r_1^2 - r_2^2)}{2d^2} \pm \frac{y_2 - y_1}{2d^2} \sqrt{((r_1 + r_2)^2 - d^2)(d^2 - (r_1 - r_2)^2)}$$

$$y = \frac{y_2 + y_1}{2} + \frac{(y_2 - y_1)(r_1^2 - r_2^2)}{2d^2} \pm \frac{x_2 - x_1}{2d^2} \sqrt{((r_1 + r_2)^2 - d^2)(d^2 - (r_1 - r_2)^2)}$$

Getting started

A dedicated contractor

The last formula characterizes explicitely the solutions but involves conditional operators (`if`) and disjunctions ($\pm$).

So it cannot be easily encoded as a usual function. It is rather an algorithm.

Fortunately, a contractor is a numerical object (it is a function on the set of intervals) and can embbed any algorithm. So we can create a contractor with our formula.

Getting started

```
class Ctc2Circles : public Ctc {

  Interval x1,x2,y1,y2;
  double r1,r2;

public:
  Ctc2Circles(const IntervalVector& c1, const IntervalVector& c2
      , double r1, double r2) :
    Ctc(2), x1(c1[0]), y1(c1[1]), x2(c2[0]), y2(c2[1]), r1(r1),
        r2(r2) { }

    void contract(IntervalVector& box);
}
```

Getting started

```
void Ctc2Circles::contract(IntervalVector& box) {

  Interval d=sqrt(sqr(x2-x1)+sqr(y2-y1));

  if (d.lb()>r1+r2 || d.ub()<r1-r2 || d.ub()<r2-r1) {
    box.set_empty();
    return;
  }

  IntervalVector l(2), r(2);

  l[0] = (x2+x1)/2+(x2-x1)*(sqr(r1)-sqr(r2))/(2*sqr(d));
  r[0] = (y2-y1)/(2*sqr(d))*sqrt((sqr(r1+r2)-sqr(d))*(sqr(d)-sqr
      (r1-r2)));
  l[1] = (y2+y1)/2+(y2-y1)*(sqr(r1)-sqr(r2))/(2*sqr(d));
  r[1] = -(x2-x1)/(2*sqr(d))*sqrt((sqr(r1+r2)-sqr(d))*(sqr(d)-
      sqr(r1-r2)));

  if (box.intersects(l-r))
    if (box.intersects(l+r)) box &= (l-r | l+r);
    else                     box &= (l-r);
  else
    if (box.intersects(l+r)) box &= (l+r);
    else                     box.set_empty();
}
```

Getting started

The formula is exact so the resulting contractor is optimal for any box if every parameter (circle position or radius) is fixed.

```
Ctc2Circles ctc3(c1,c2,r1,r2);

cout << "box before=" << box << endl;
ctc3.contract(box);
cout << "box after=" << box << endl;
```

```
box before=([-3, 3] ; [-3, 3])
box after=([0, 0.8] ; [-0.6, 1])
```

Getting started

However, the multi-occurrence of the parameters make the contractor pessimistic in case of uncertainty.

```
double eps=1e-2;
c1.inflate(eps);
c2.inflate(eps);
```

With `CtcFwdBwd`:

```
box before=([-3, 3] ; [-3, 3])
box after=([-0.01, 1.01] ; [-0.753445, 1.01])
```

With `Ctc2Circles`:

```
box before=([-3, 3] ; [-3, 3])
box after=([-0.0613031, 0.860516] ; [-0.676038, 1.0755])
```

Getting started

This is a situation where cooperation of contractors is typically needed.

Combining contractors is very easy in Ibex which has been designed for this purpose.

Simple composition:

```
CtcCompo ctc4(ctc1,ctc3);
```

Result:

```
box before=([-3, 3] ; [-3, 3])
box after=([-0.01, 0.860516] ; [-0.676038, 1.01])
```

Composition is the simplest way to combine contractors. There exists many other composition rules.

Outline

Conclusion

IBEX is:

- an open-source C++ library for set computation

Conclusion

IBEX is:

- an open-source C++ library for set computation
- compatible with existing tools

Conclusion

IBEX is:

- an open-source C++ library for set computation
- compatible with existing tools
- under active development ($\sim$ 10 commits / weak)

Conclusion

IBEX is:

- an open-source C++ library for set computation
- compatible with existing tools
- under active development ($\sim$ 10 commits / weak)
- based on the contractor programming paradigm

Conclusion

IBEX is:

- an open-source C++ library for set computation
- compatible with existing tools
- under active development ($\sim$ 10 commits / weak)
- based on the contractor programming paradigm
- where sophisticated strategies are built from simple composition of built-in / ad-hoc blocks