IBEX

Formation du 18 déc 2012

G. Chabert, L. Jaulin, F. Le Bars, J. Ninin

ENSTA Bretagne

Plan de l'exposé

1 Introduction
2 Exemples de problèmes
3 TP n˚1
4 TP n˚2
5 TP n˚3

But de cette formation

Vous aider à prendre en main IBEX à travers trois TPs :

- **approximation extérieure de l'image d'une fonction**
  $\longrightarrow$ *création de boîtes, création de fonctions, évaluation*
- **inversion ensembliste** (approximation intérieure et extérieure),
  $\longrightarrow$ *bissection, différence ensembliste, backward*
- **localisation robuste**
  $\longrightarrow$ *programmation par contracteurs, Q-intersection, fonctions de projection*

Introduction

IBEX est un outil de **calcul ensembliste**.

Le cacul ensembliste peut être défini comme la combinaison de trois types d'opérations sur les ensembles :

1. logiques ($\cap$,$\cup$,. . .)
2. arihmétiques ($+$,$\times$,sin,. . .)
3. géométriques (milieu, diamètre, bissection, . . .)

La structure de base utilisée est celle d'**intervalle.**

Un produit cartésien d'intervalles forme une **boîte.**

En combinant ces opérations, IBEX peut répondre à une classe de problèmes très large que l'on peut résumer ainsi :

Objectif du calcul ensembliste

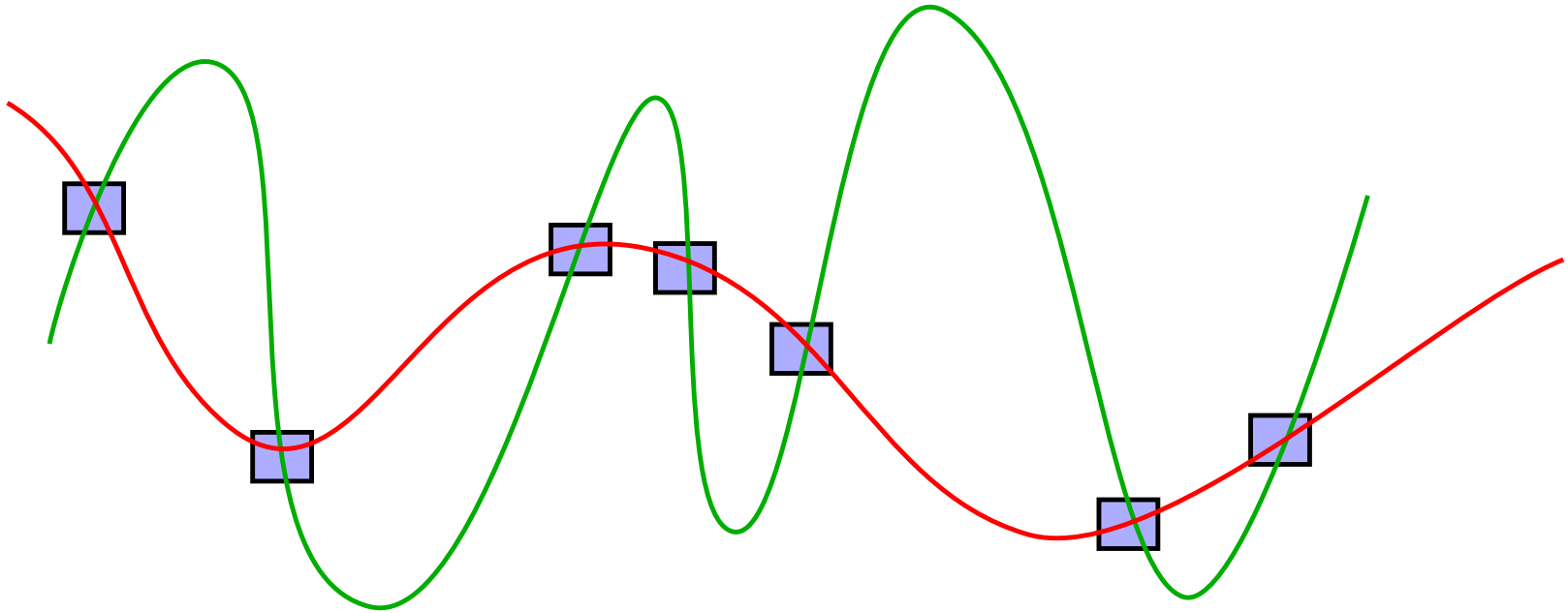
Caractériser de façon garantie, par des boîtes, un ou plusieurs ensembles $\mathcal{S}, \ldots,$ définis implicitement par des relations (ou contraintes).

Plan de l'exposé

Résolution d'équations

Soit $f : \mathbb{R}^n \to \mathbb{R}^m$.

$$\mathcal{S} \supseteq \{x \in \mathbb{R}^n,\ f(x) = 0\}$$

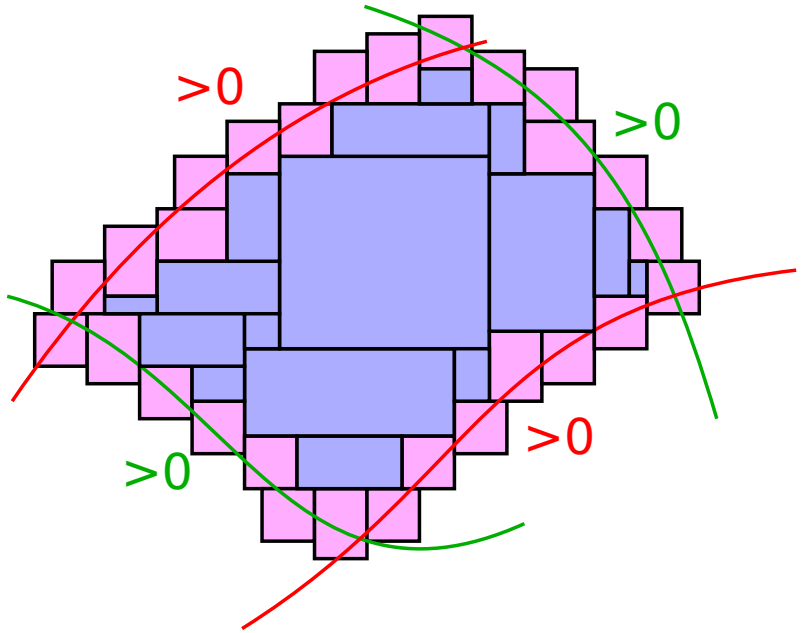


Exemples de problèmes

Inégalités

Soit $f : \mathbb{R}^n \to \mathbb{R}^m$.

$$\mathcal{S}_1 \cup \mathcal{S}_2 \supseteq \{x \in \mathbb{R}^n,\ \forall i,\ 1 \leq i \leq m,\ f_i(x) \leq 0\} \supseteq \mathcal{S}_1$$

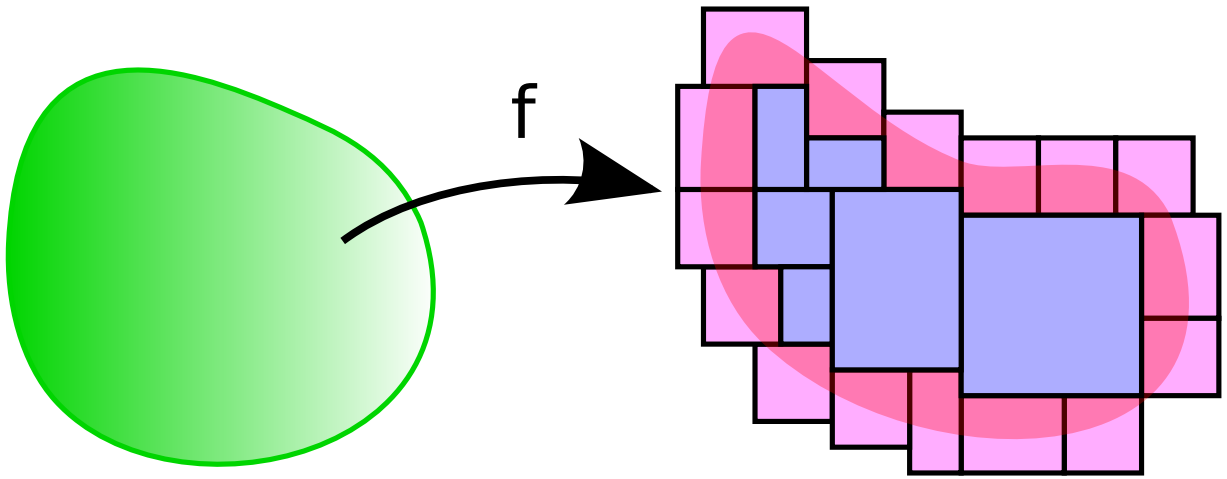


➡ inversion ensembliste

Image d'un ensemble

Soit $f : \mathbb{R}^n \to \mathbb{R}^m$ et $\mathcal{E}$ un sous-ensemble de $\mathbb{R}^n$.

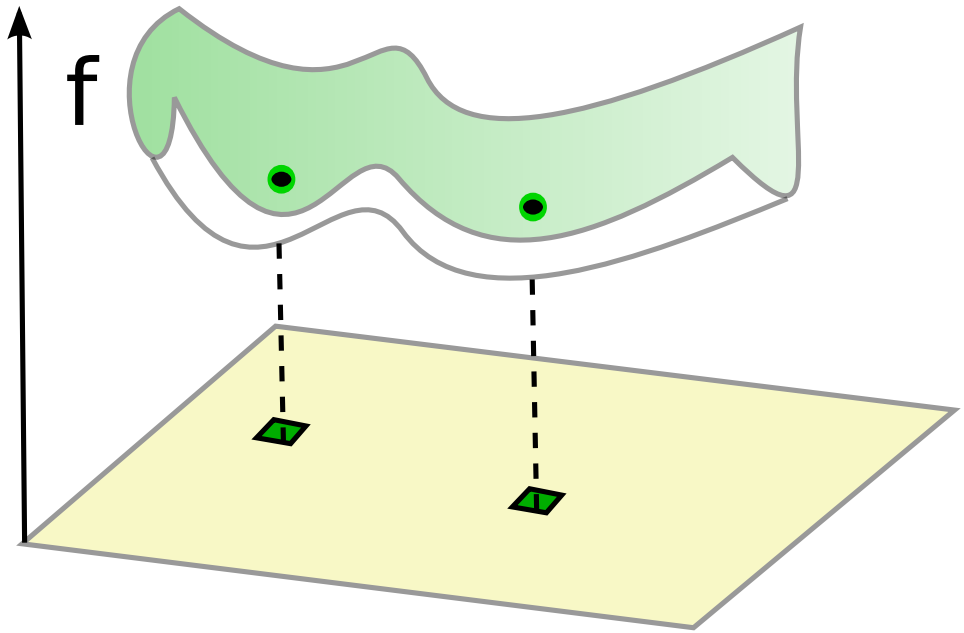
$$\mathcal{S}_1 \cup \mathcal{S}_2 \supseteq \{f(x), x \in \mathcal{E}\} \supseteq \mathcal{S}_1$$



Optimisation globale

Soit $f : \mathbb{R}^n \to \mathbb{R}$.

$$\mathcal{S} \supseteq \{x, \forall y \in \mathbb{R}^n, f(y) \geq f(x)\}$$

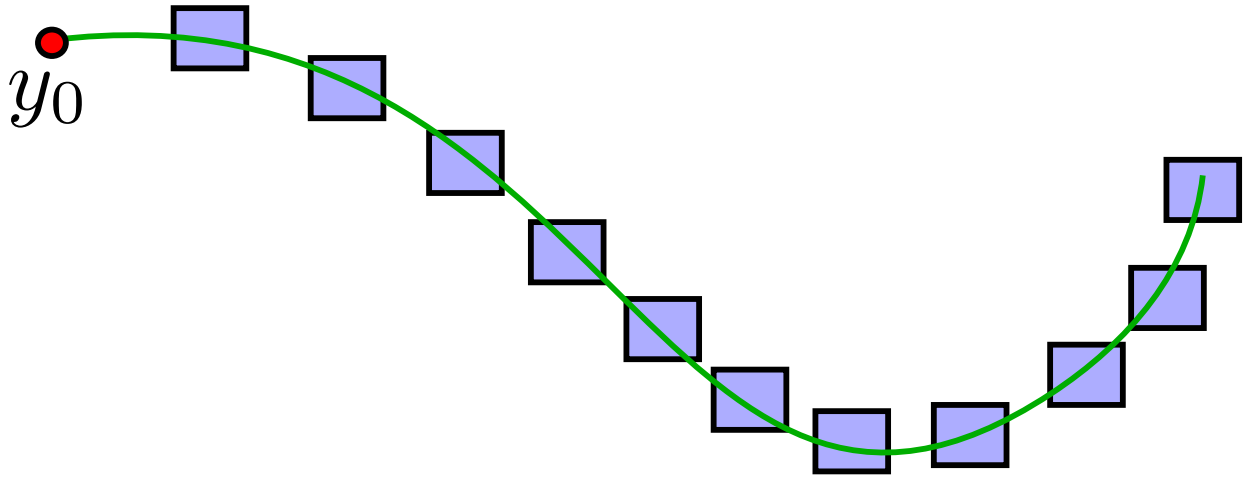


Exemples de problèmes

Equation différentielle

Soit $f : \mathbb{R}^n \to \mathbb{R}^n, \quad t_1, \ldots, t_n \in \mathbb{R}^+$.

$$\mathcal{S} \supseteq \{(y(t_1), \ldots, y(t_n)),\ y' = f(y) \wedge y(t_0) = y_0\}.$$



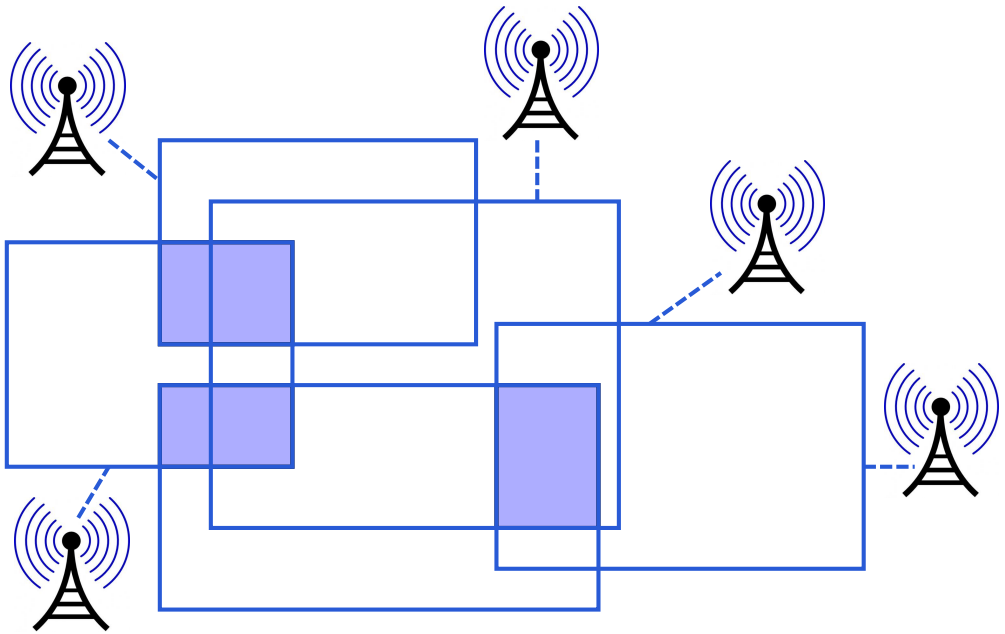
Exemples de problèmes

Contraintes combinatoires

Soit m mesures incertaines $[x]_i$ $(1 \leq i \leq m)$ d'une position de $\mathbb{R}^n$ dont q seulement sont correctes.

$$\mathcal{S} = \bigcup_{i_1,\ldots,i_q} \bigcap_{1 \leq j \leq q} [x]_i$$

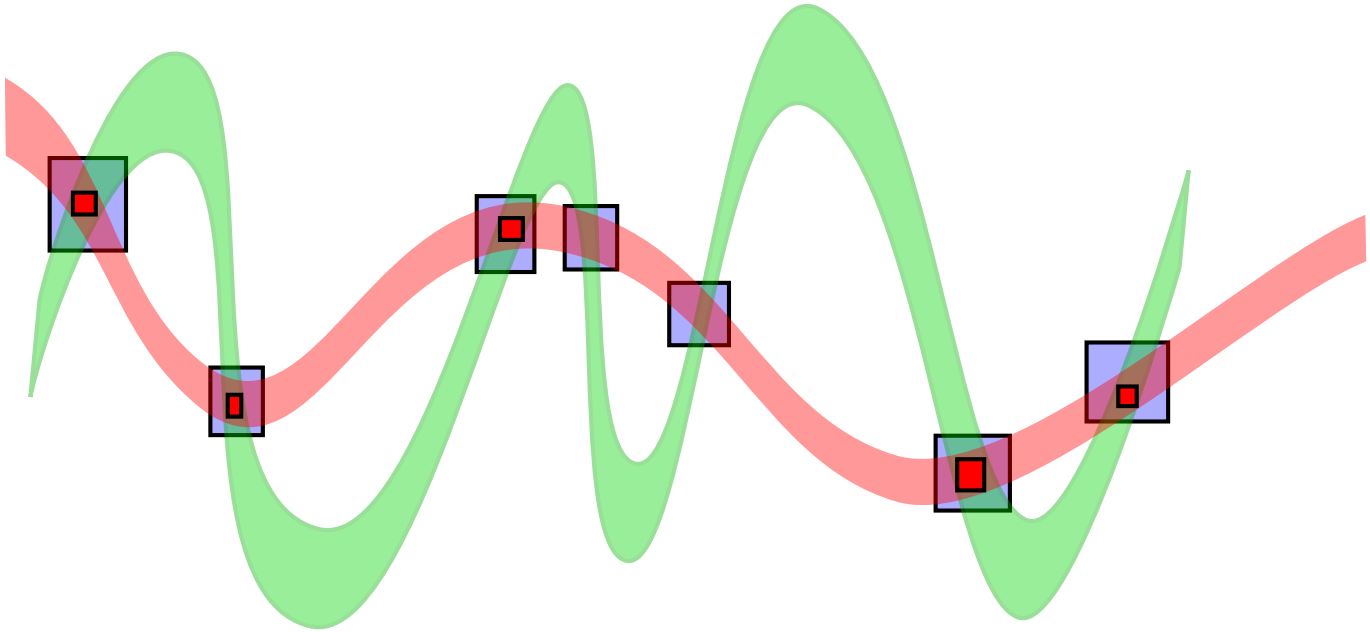
où $i_1, \ldots, i_q$ décrit des ensembles d'indices tous différents entre 1 et N.



Prise en compte d'incertitudes

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

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



Exemples de problèmes

Hétérogénéïté

Les contraintes peuvent être de différentes natures.

Exemples de problèmes

Des versions “boîtes noires” efficaces de certains algorithmes standards (résolution et d’optimisation) sont fournies avec IBEX, notamment :

- `defaultsolver`
- `defaultoptimizer`

→ démo

Plan de l'exposé

1 Introduction

2 Exemples de problèmes

3 TP n°1

4 TP n°2

5 TP n°3

Voyons maintenant comment de tels algorithmes peuvent être conçus, en combinant les trois types d'opérations dont nous avons parlé.

L'arithmétique d'intervalle tout d'abord permet de calculer un sur-ensemble de l'image d'une boîte :

Arithmétique d'intervalle

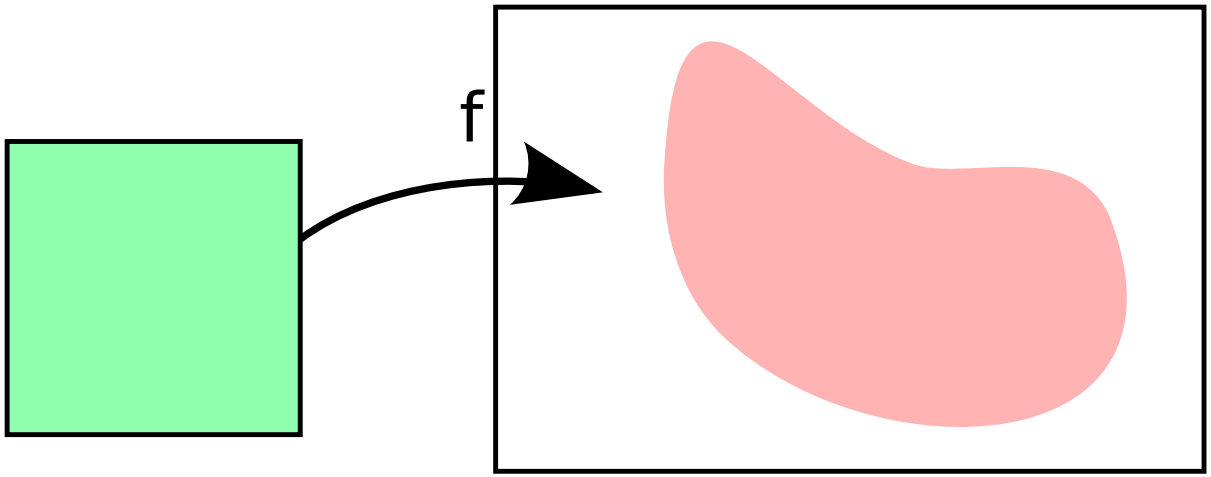
Question : si $x \in [1,2]$ comment varie $f(x) = x^2 - x$?

$$\begin{array}{rl} x & \in [1,2] \\ x^2 & \in [1,4] \\ x^2 - x & \in [-1,3] \end{array}$$

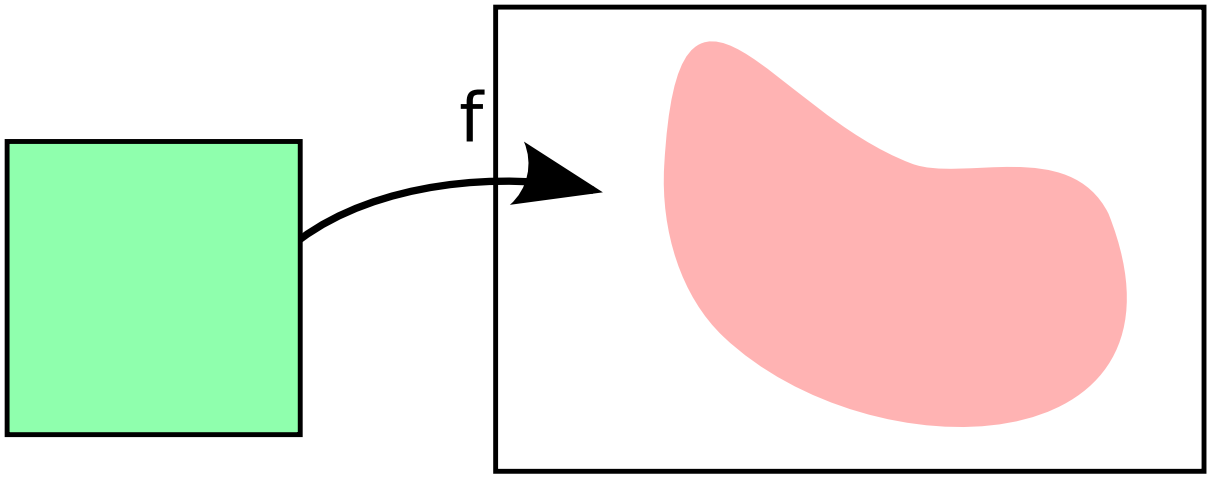
$$range(f,[1,2]) = [0,2] \subseteq [-1,3]$$

Généralisation :

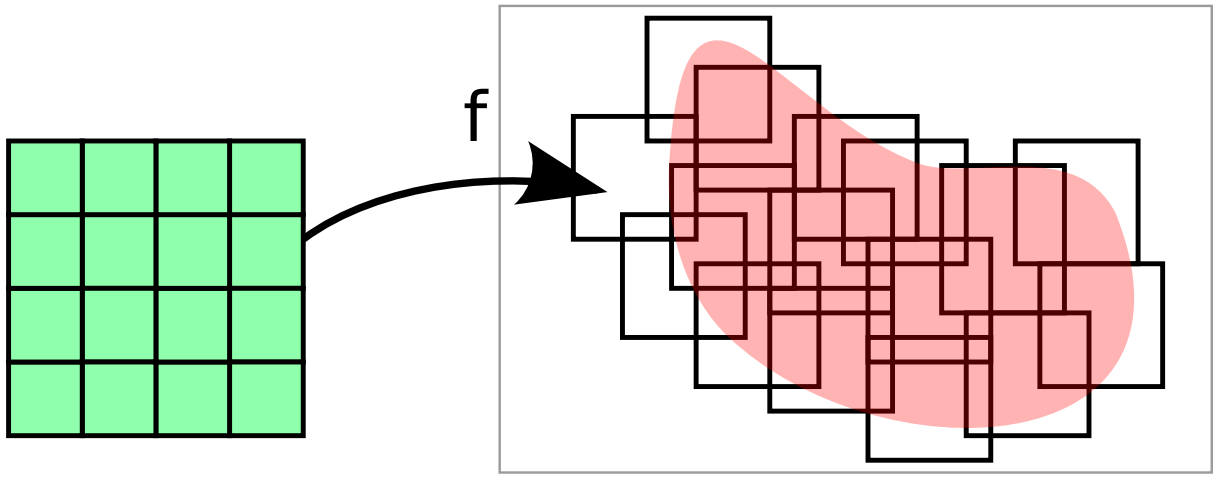
Généralisation :



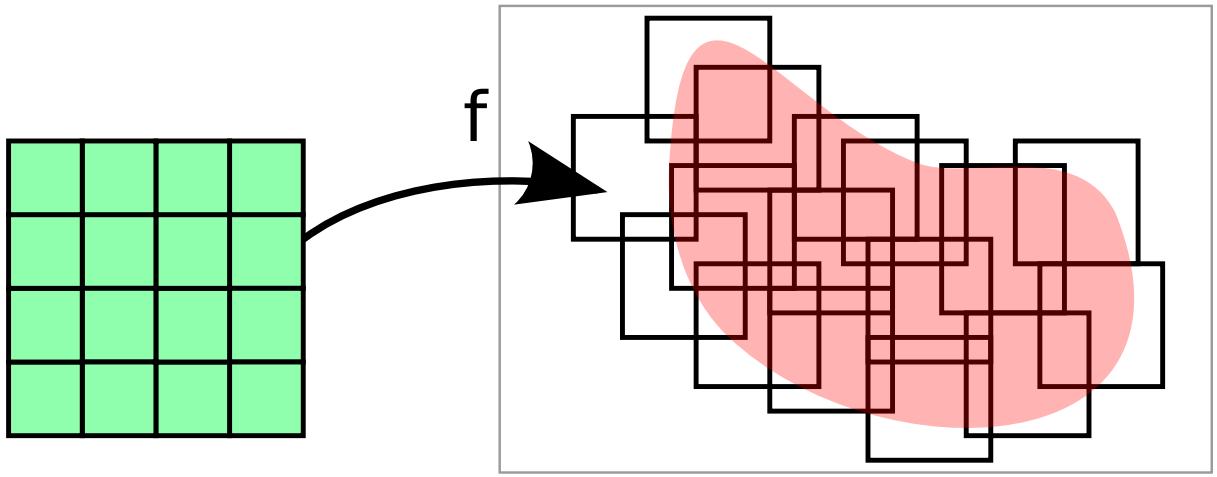
En combinant simplement cette arithmétique avec la bissection, nous pouvons déjà calculer une approximation non-triviale de l'image d'une fonction.



En combinant simplement cette arithmétique avec la bissection, nous pouvons déjà calculer une approximation non-triviale de l'image d'une fonction.



En combinant simplement cette arithmétique avec la bissection, nous pouvons déjà calculer une approximation non-triviale de l'image d'une fonction.



➡ c'est l'objet du premier TP.

TP n°1

Trois classes servent à représenter des intervalles sous IBEX :

- `Interval`
- `IntervalVector`
- `IntervalMatrix`

Deux classes servent à construire des fonctions :

- `Variable`
- `Function`

➡ consultez le tutorial en ligne et l'API sur
`http://www.emn.fr/z-info/ibex/`.
pour apprendre à manipuler ces objets.

Pour évaluer l'image d'une boîte par une fonction, on utilise la fonction C++ `eval`.

Exemple

```
Variable x,y;
Function f(x,y,x+y);
IntervalVector box(2);
box[0]=Interval(0,1);
box[1]=Interval(1,2);
Interval z=f.eval(box);
```

Règle d'utilisation

L'évaluation considère les arguments comme formant un unique vecteur de $\mathbb{R}^n$, (quel que soit le nombre d'arguments de la fonction f et leur type). Ici :

$$f : x \mapsto x_1 + x_2.$$

Conséquence : la fonction C++ `eval` n'accepte en argument qu'un vecteur d'intervalles, c.a.d., une boîte.

Remarque : il y a en fait une façon simple et systématique de construire et manipuler cette boîte (cf. TP n˚3).

En revanche elle retourne :

un intervalle	si elle est à valeur dans $\mathbb{R}$
une boîte	si elle est à valeur dans $\mathbb{R}^m$
une matrice d'intervalles	si elle est à valeur dans $\mathbb{R}^{m\times p}$

Plan de l'exposé

TP n˚2

Nous allons combiner maintenant

- l'évaluation (puis le backward)
- l'intersection (& logique)
- la bissection

pour obtenir un algorithme de résolution garantie d'équations.

Garantie ?

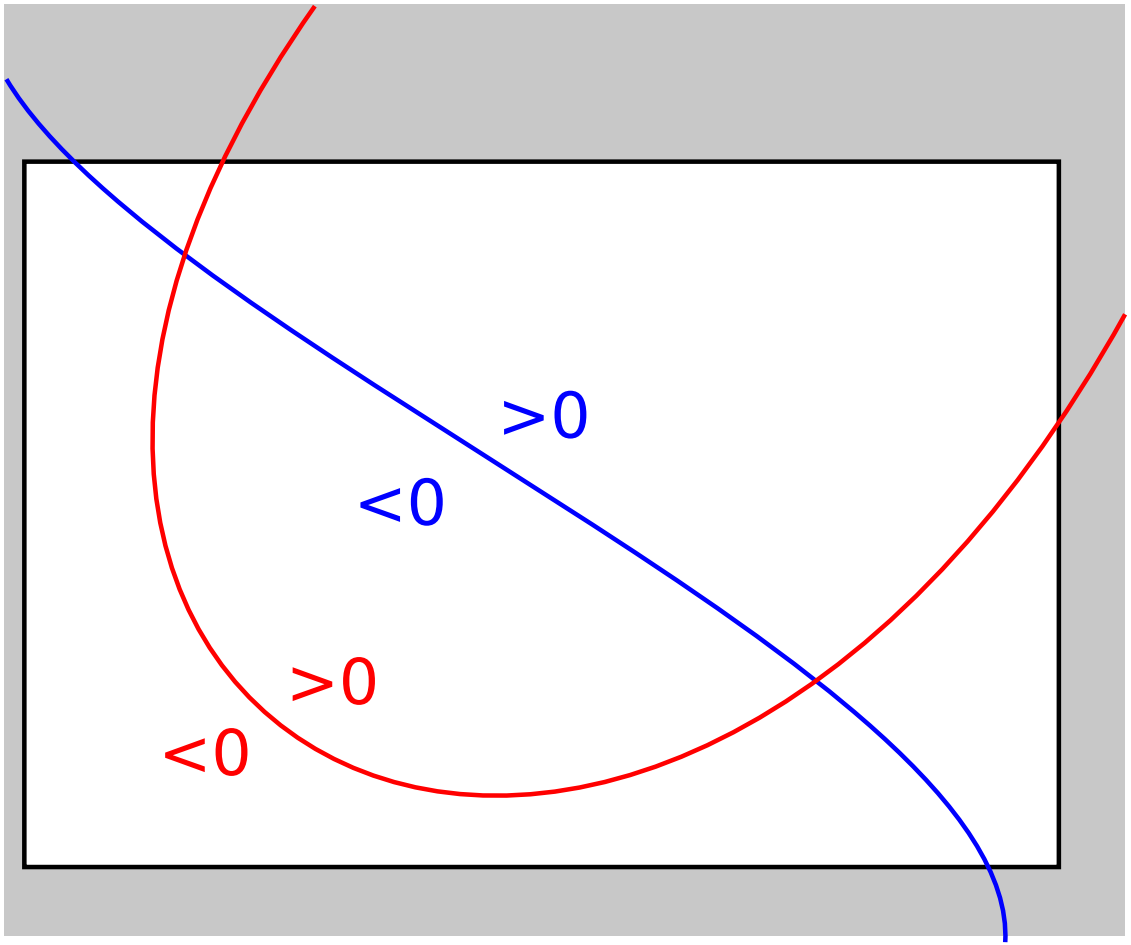
- aucune perte de solution
- chaque solution est encadrée par une boîte de précision arbitraire
- prise en compte des incertitudes sur les paramètres du modèle

Dans le TP n˚2, vous devez écrire un algorithme similaire mais pour un problème légèrement différent (SIVIA).

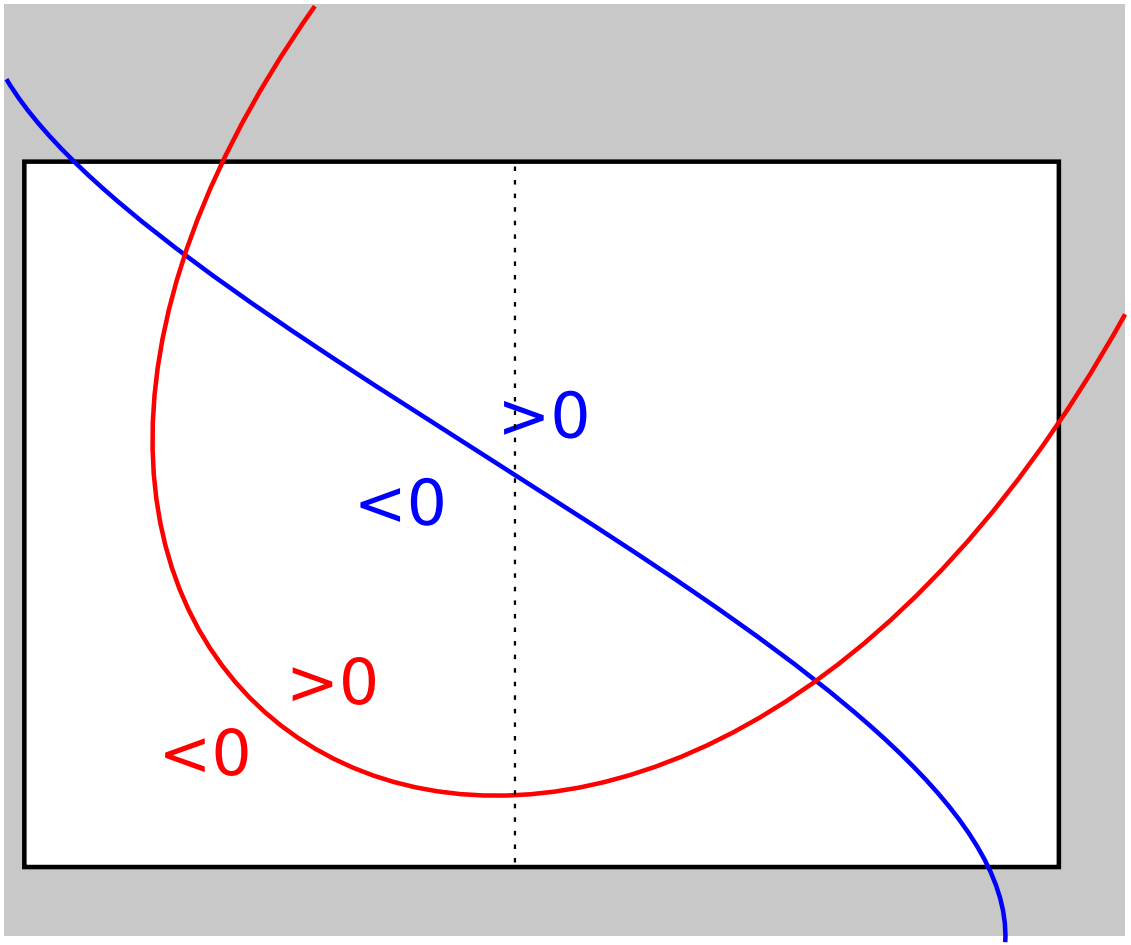
Le TP est constitué de 2 phases

1. **phase 1** : utilisation de l'évaluation uniquement
2. **phase 2** : utilisation du backward

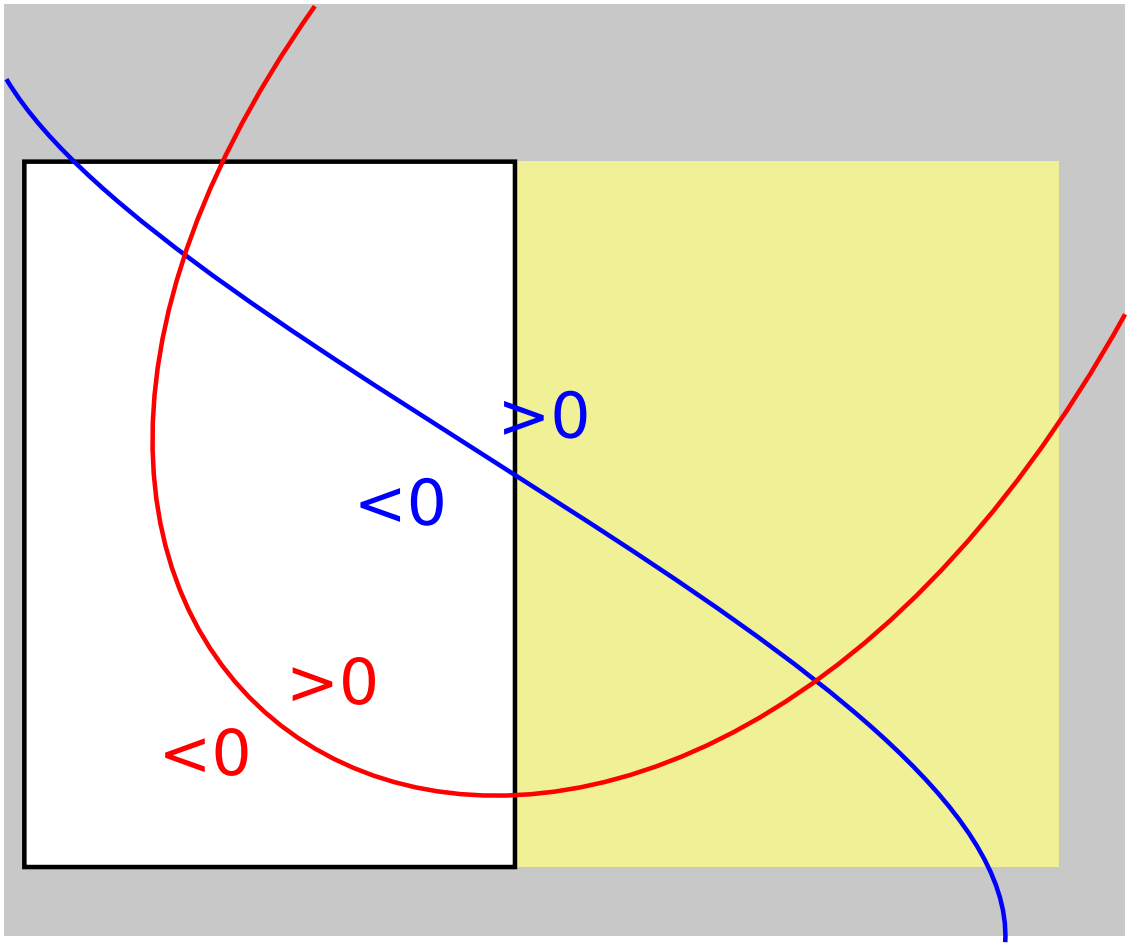
Principe (avec évaluation uniquement).



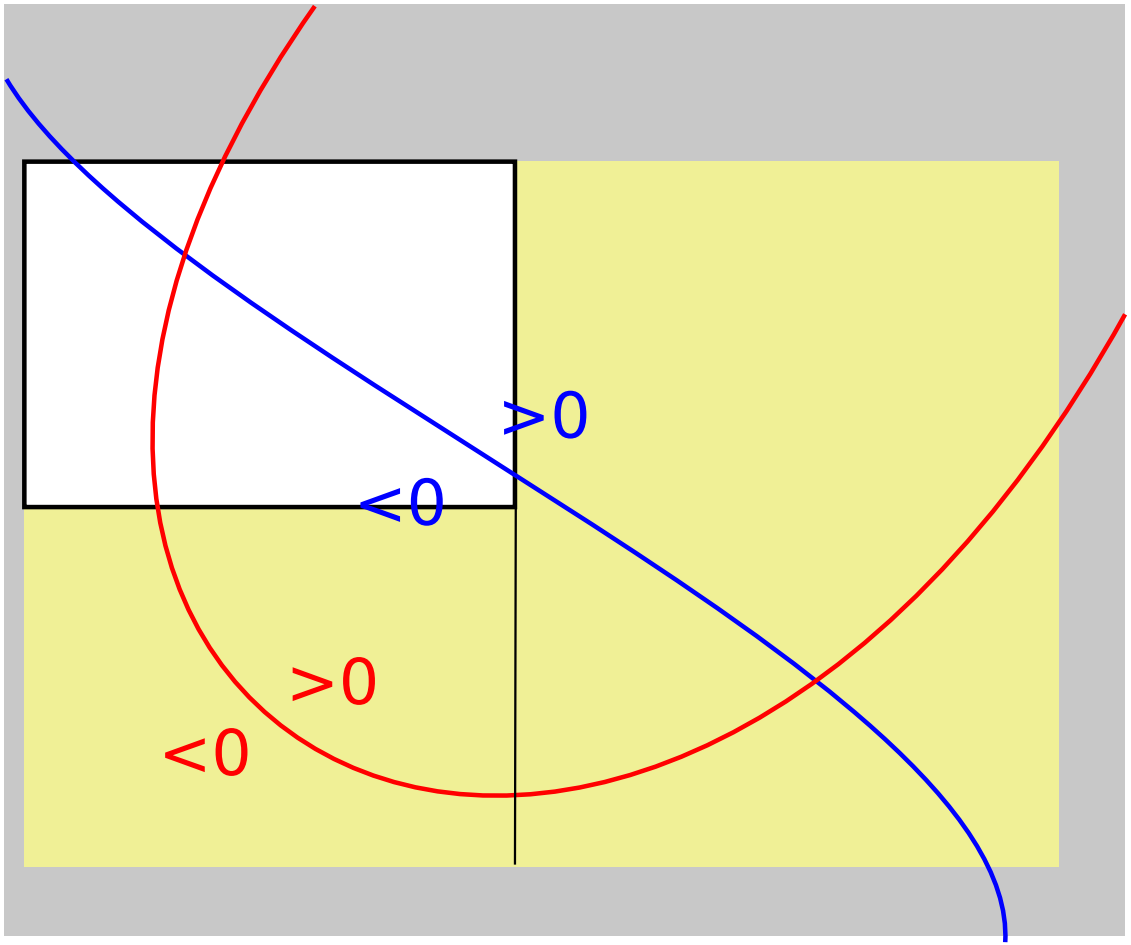
Principe (avec évaluation uniquement).



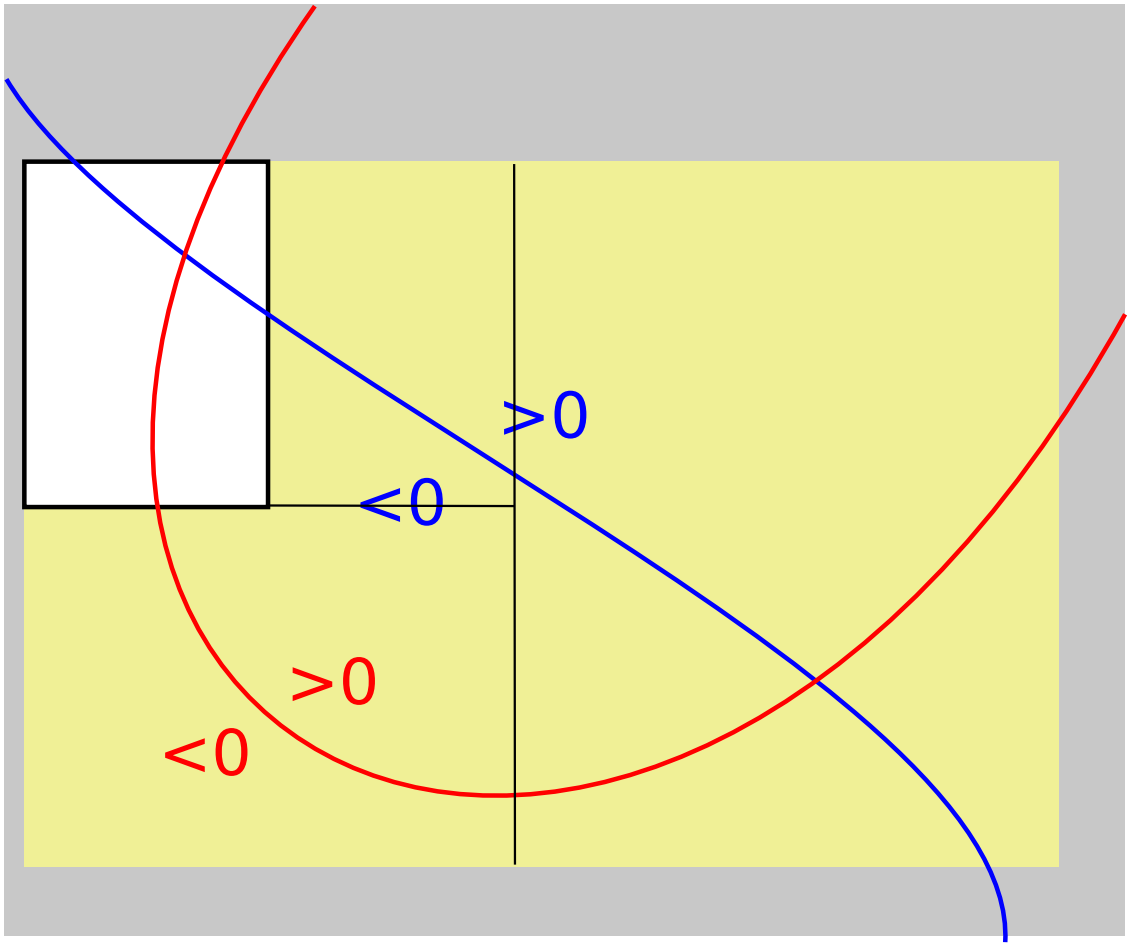
Principe (avec évaluation uniquement).



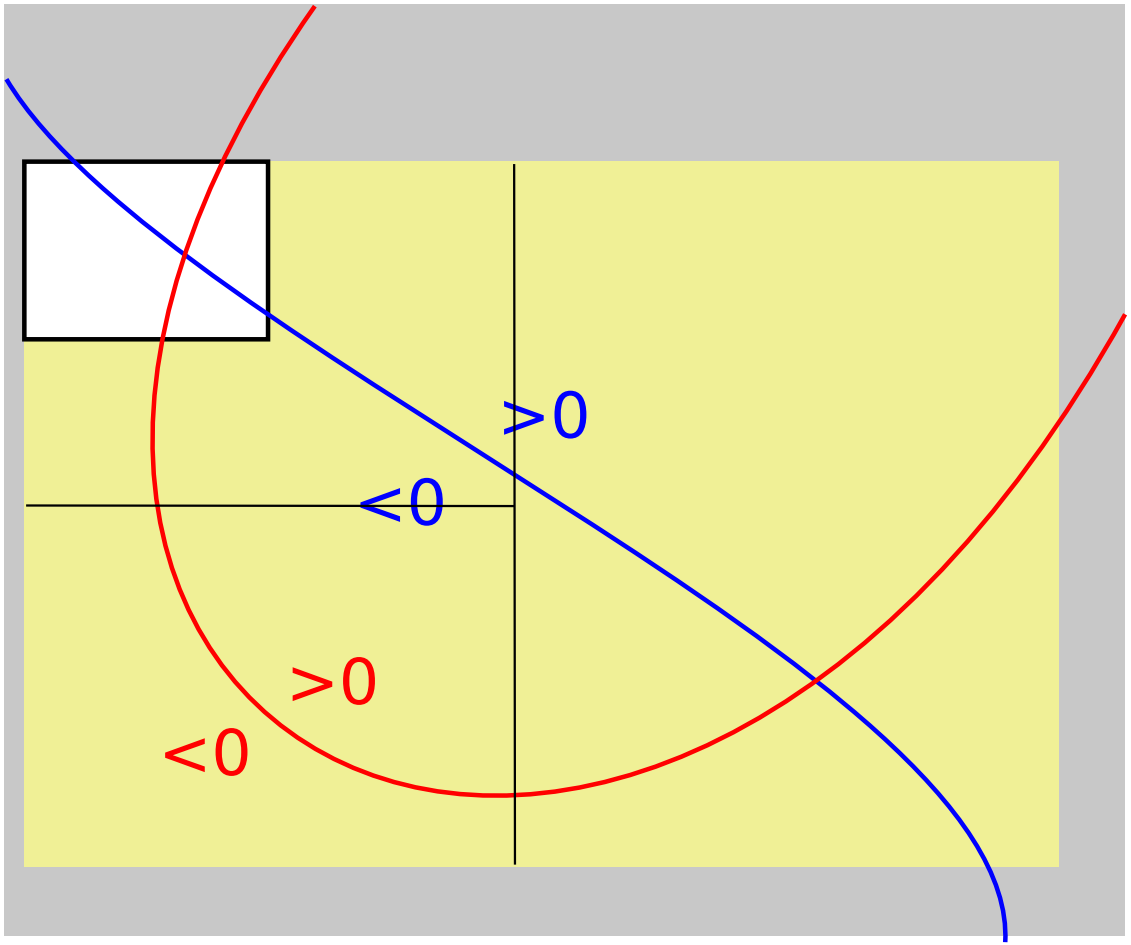
Principe (avec évaluation uniquement).



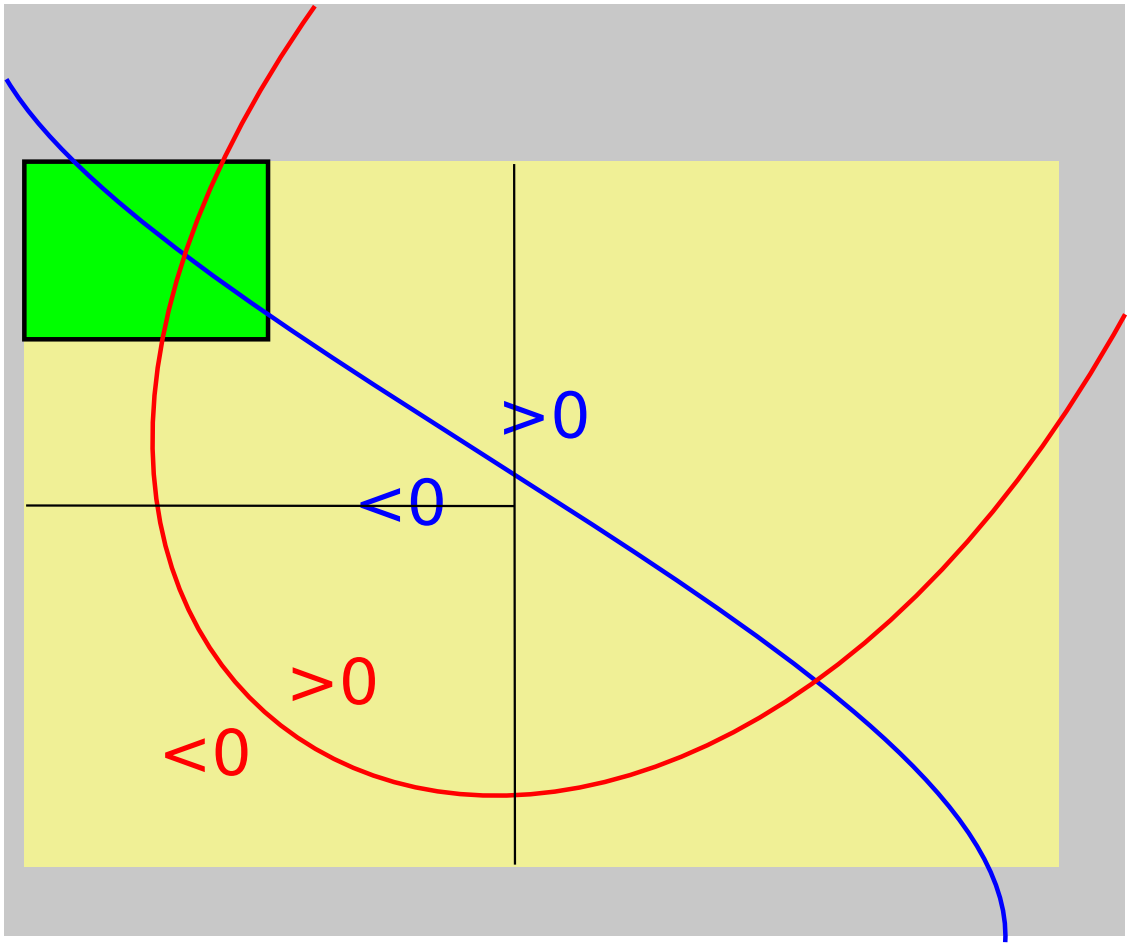
Principe (avec évaluation uniquement).



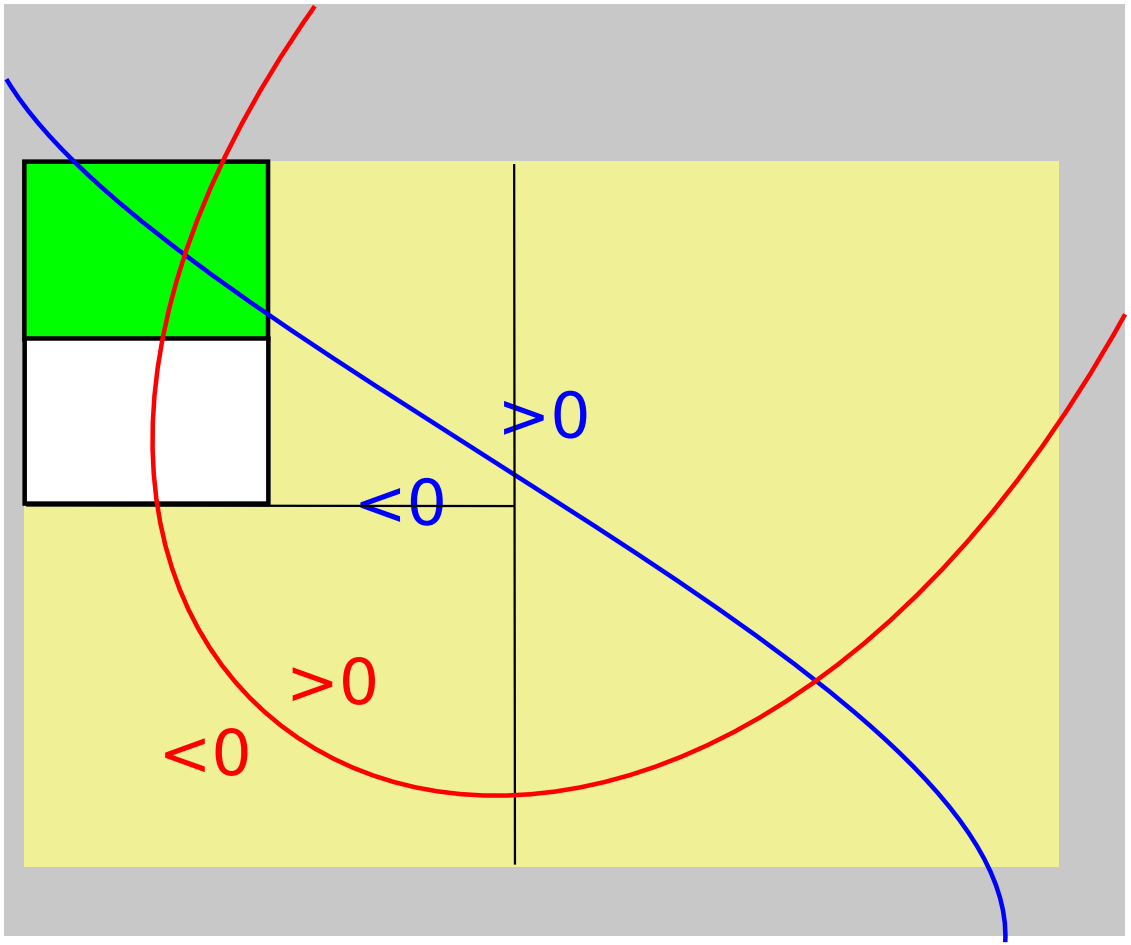
Principe (avec évaluation uniquement).



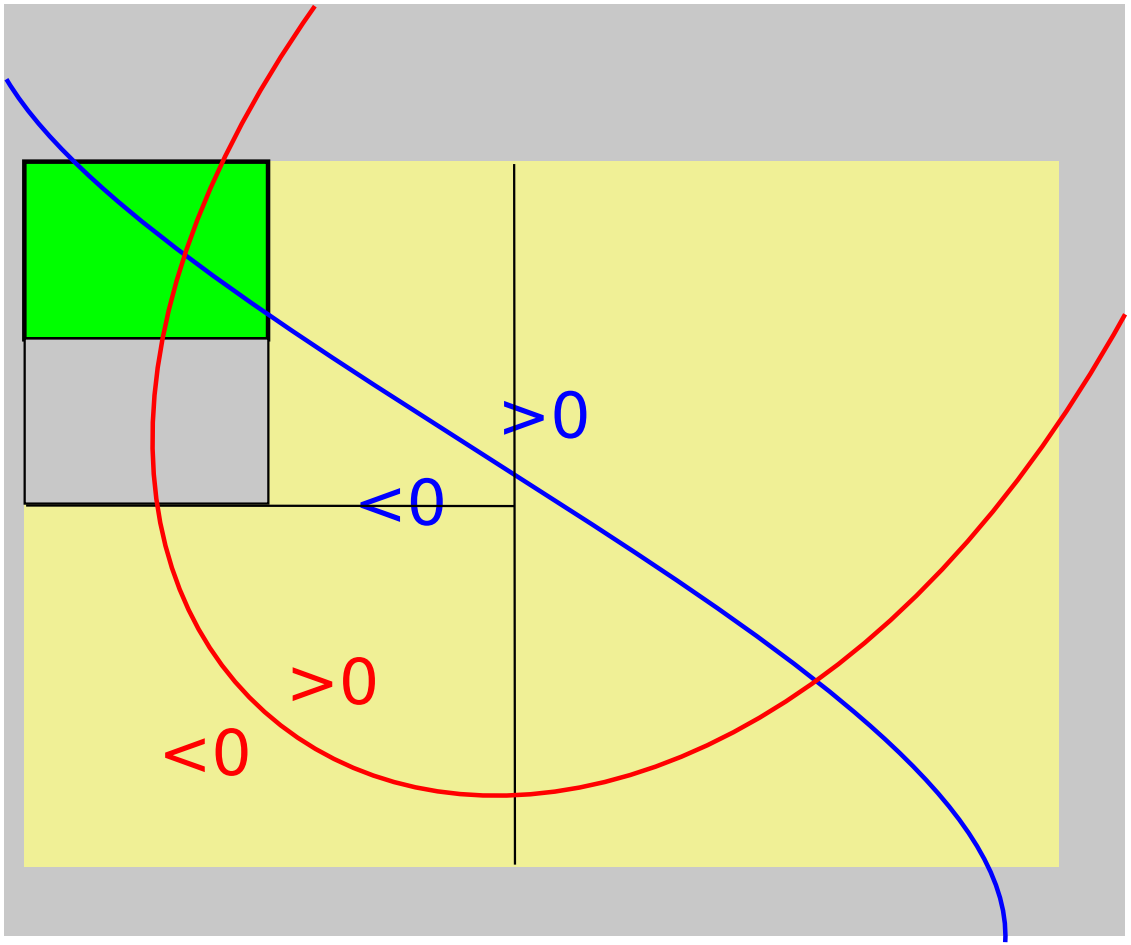
Principe (avec évaluation uniquement).



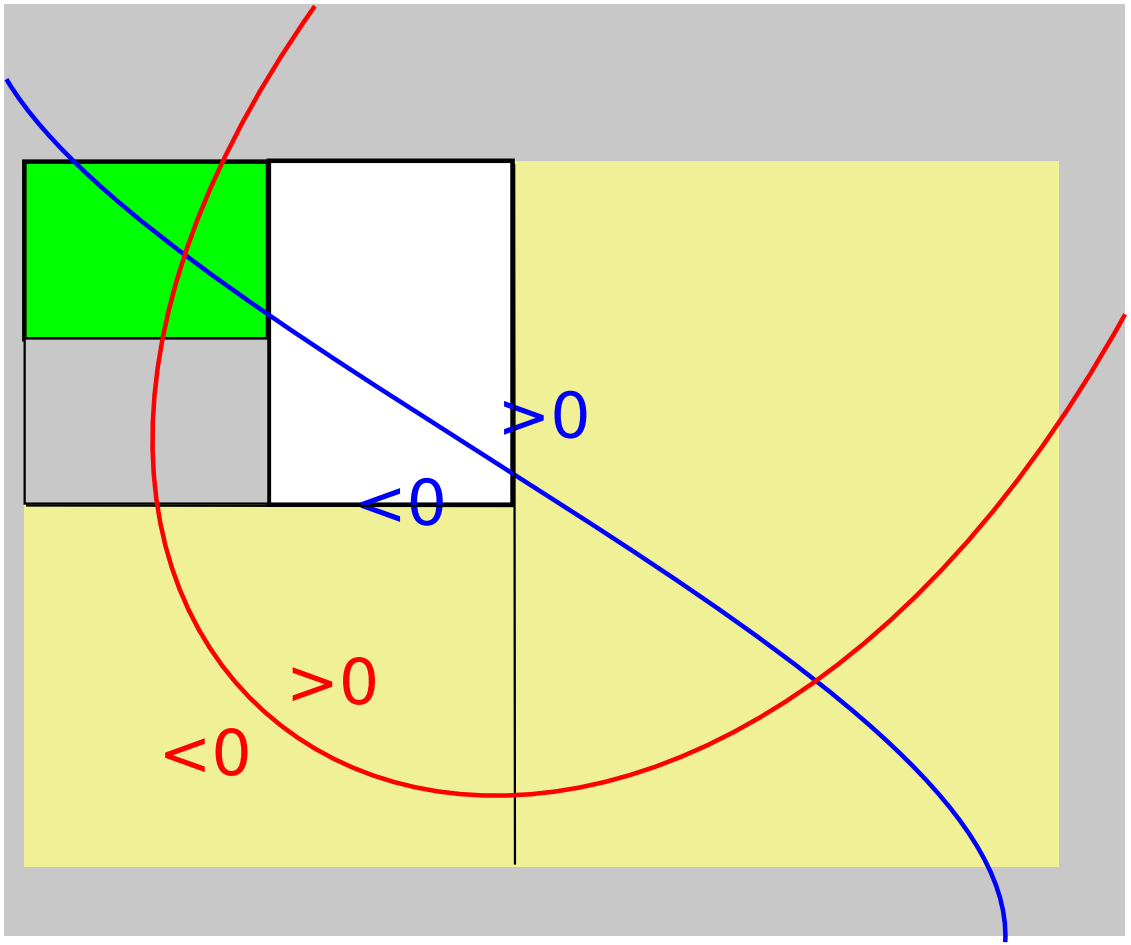
Principe (avec évaluation uniquement).



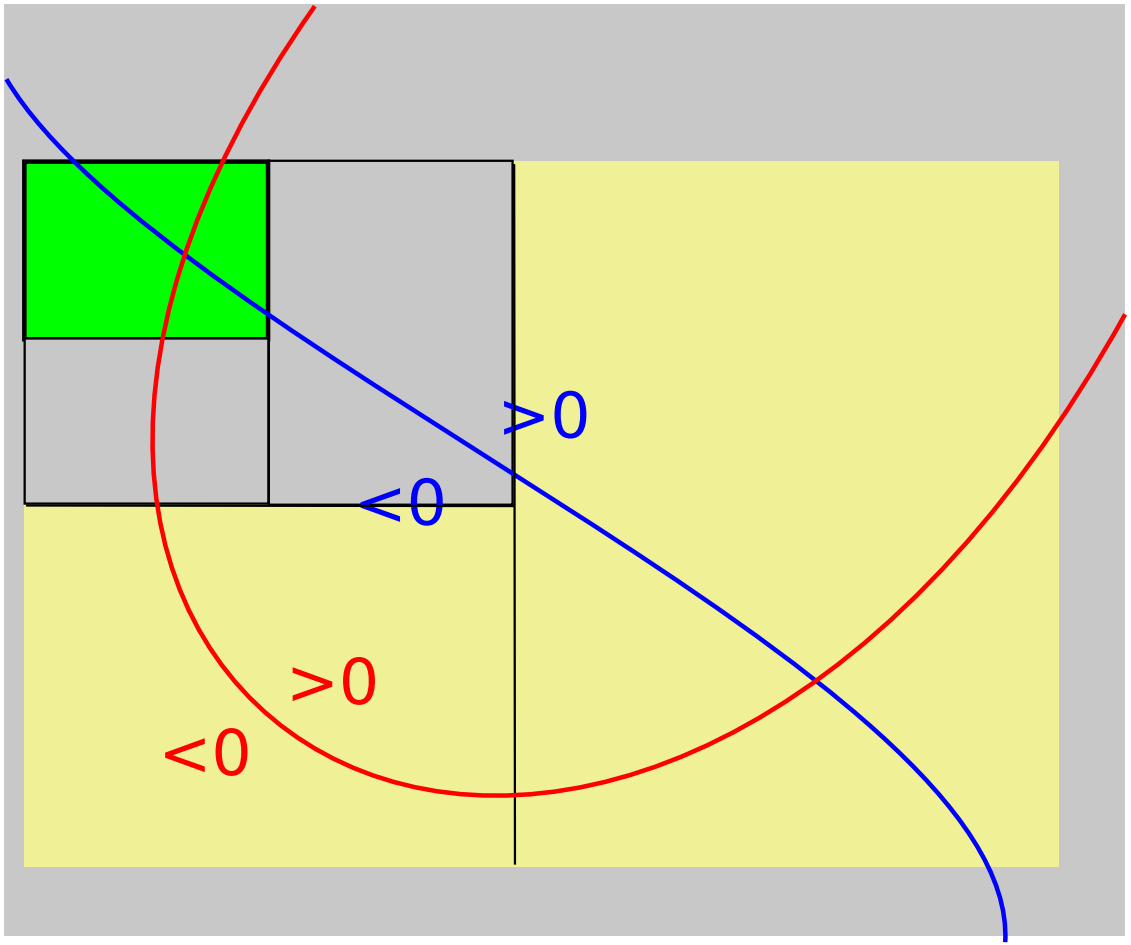
Principe (avec évaluation uniquement).



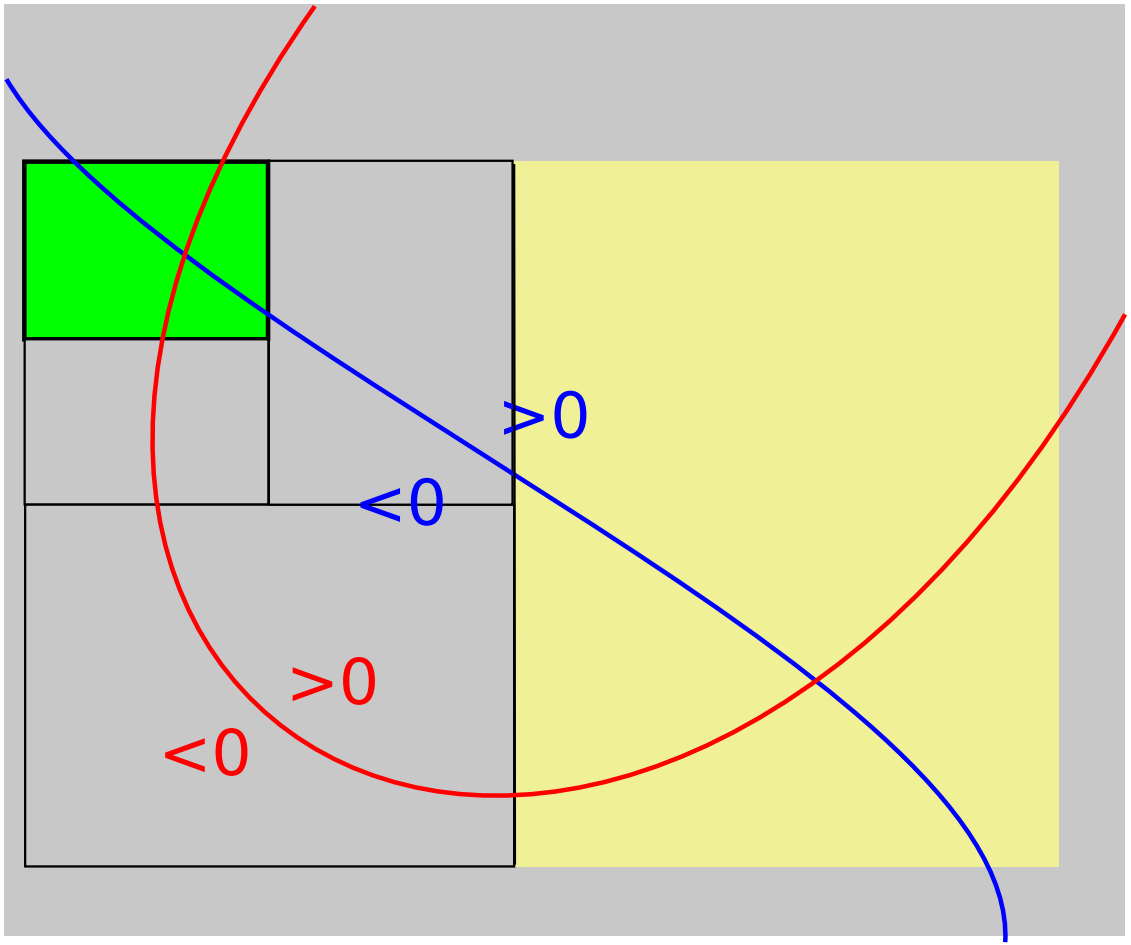
Principe (avec évaluation uniquement).



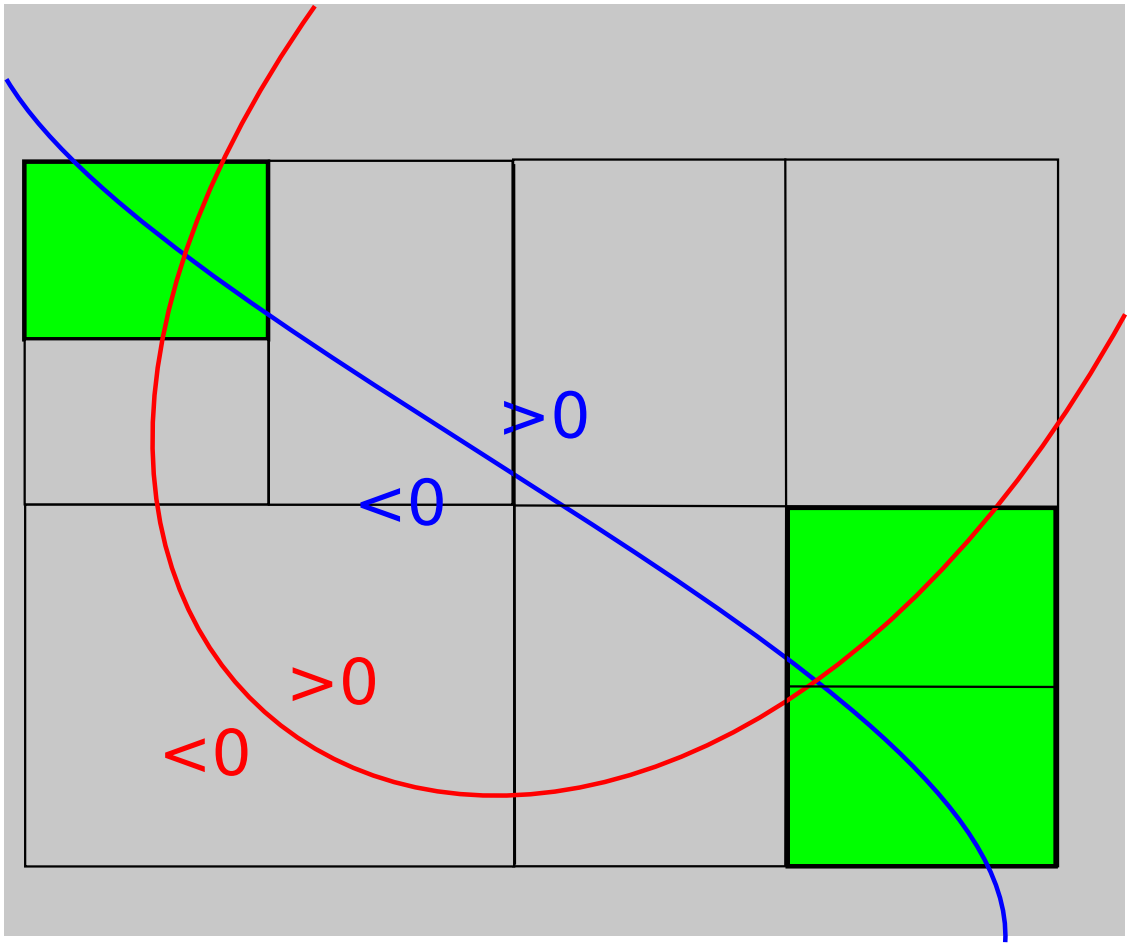
Principe (avec évaluation uniquement).



Principe (avec évaluation uniquement).



Principe (avec évaluation uniquement).



input : une fonction f, une boîte $[x]^{(0)}$, $\varepsilon > 0$
output : un vecteur de ε-boîtes encadrant $\{f = 0\}$

stck $\leftarrow \{[x]^{(0)}\}$
sols$\leftarrow \{\}$
while (stck$\neq \{\}$)
    $[x] \leftarrow$ stck.pop()
    if ($0 \in [f]([x])$)
        if (size($[x]$)$< \varepsilon$)
            sols $\leftarrow$ sols $\cup \{[x]\}$
        else
            $([x]^{(1)}, [x]^{(2)}) \leftarrow$ bisect($[x]$)
            stck $\leftarrow$ stck $\cup \{[x]^{(1)}, [x]^{(2)}\}$
        end
    end
return sols

Dans la seconde phase, vous devez écrire une version plus efficace de SIVIA, basée sur la notion de **contraction** et que nous introduisons ici pour notre problème de résolution via l'algorithme de **forward-backward**.

Definition (Contracteur)

On appelle contracteur tout opérateur de $\mathbb{IR}^n$ dans $\mathbb{IR}^n$ tel que

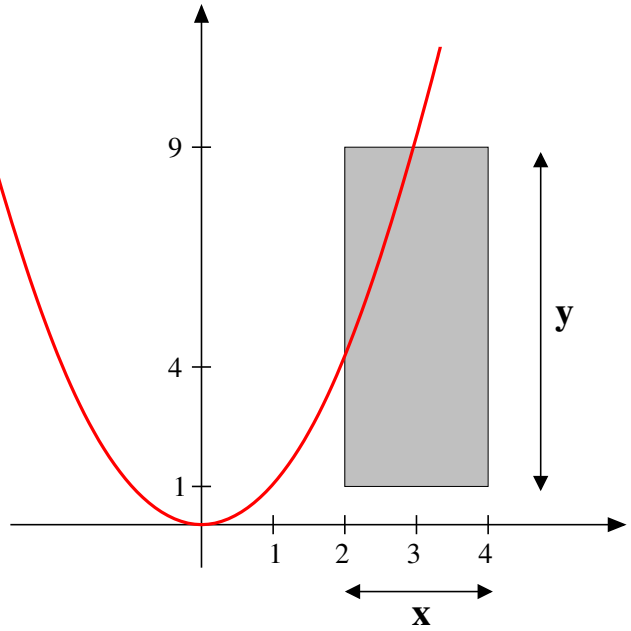
$$\forall [x] \in \mathbb{IR}^n, \quad C([x]) \subseteq [x].$$

Note : $\mathbb{IR}^n$ = ensemble des boîtes de dimension n.

Forward-backward.

Utilisation de la forme symbolique

Forward-backward.



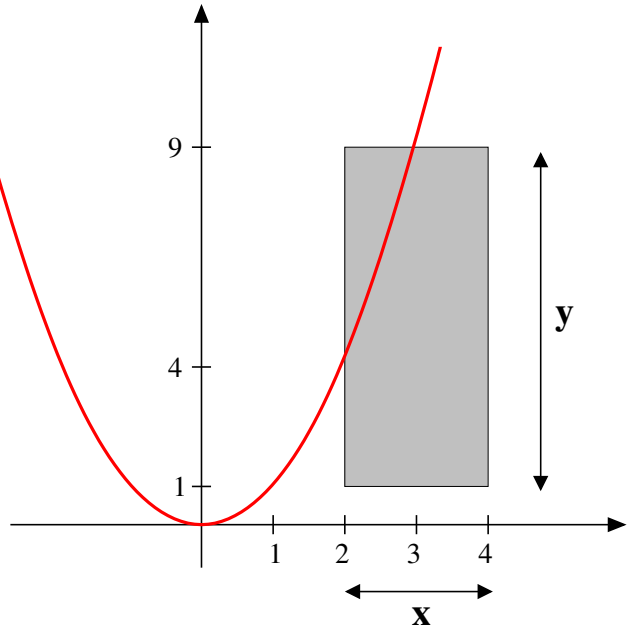
$x \in [2,4] \quad y \in [1,9]$

Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

Forward-backward.



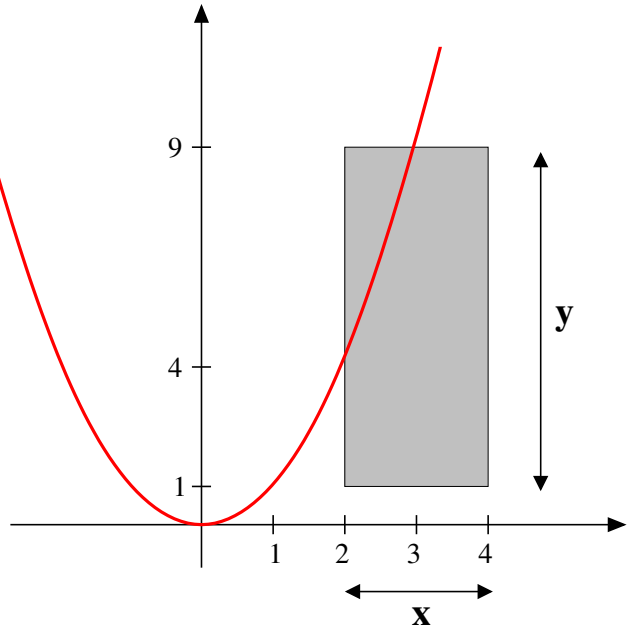
$x \in [2,4] \quad y \in [1,9]$
$[x]^2 = [4,16]$

Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

Forward-backward.



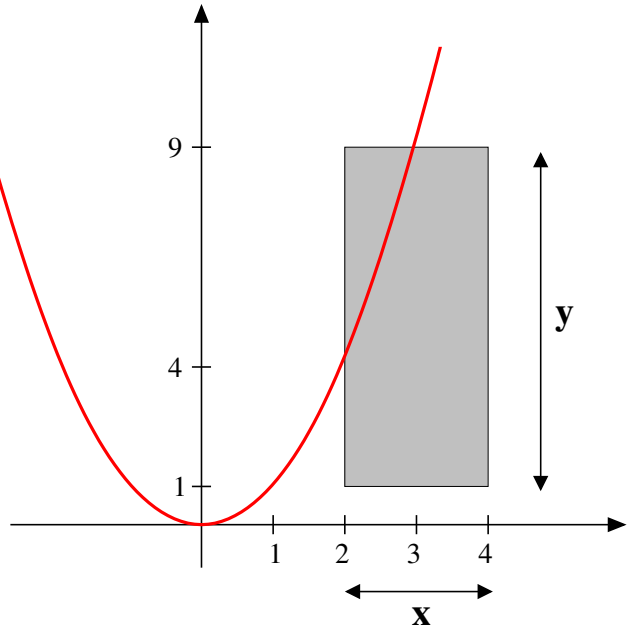
$x \in [2,4] \quad y \in [1,9]$
$[x]^2 = [4,16]$
$-[x]^2 = [-16,-4]$

Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

Forward-backward.



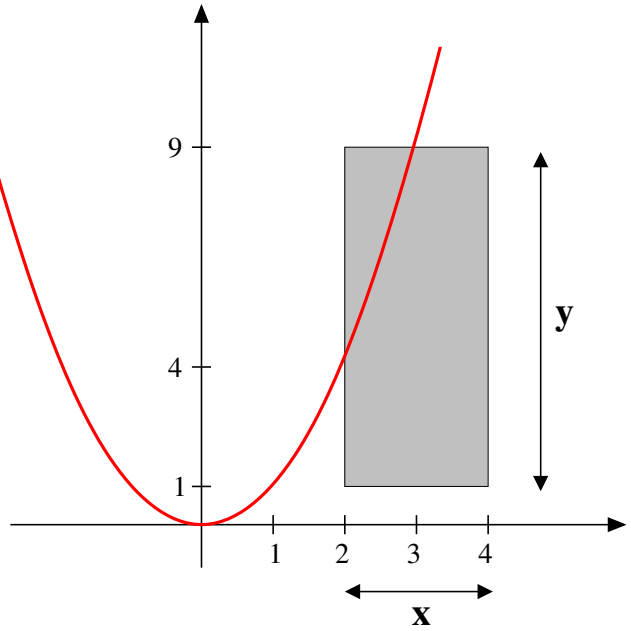
$x \in [2,4] \quad y \in [1,9]$
$[x]^2 = [4,16]$
$-[x]^2 = [-16,-4]$
$[y] - [x]^2 = [-15,5]$

Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$

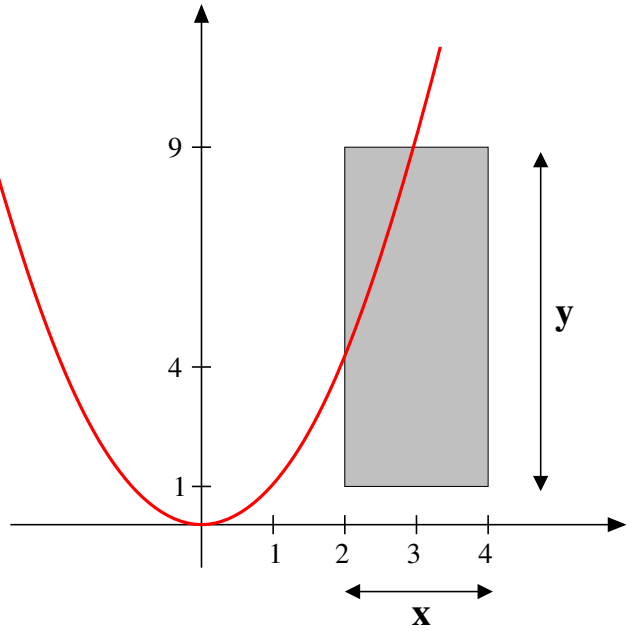
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$y = x^2 \qquad \rightsquigarrow \qquad y \in [x]^2 \cap [y]$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$
$[x]^2 = [4,16]$

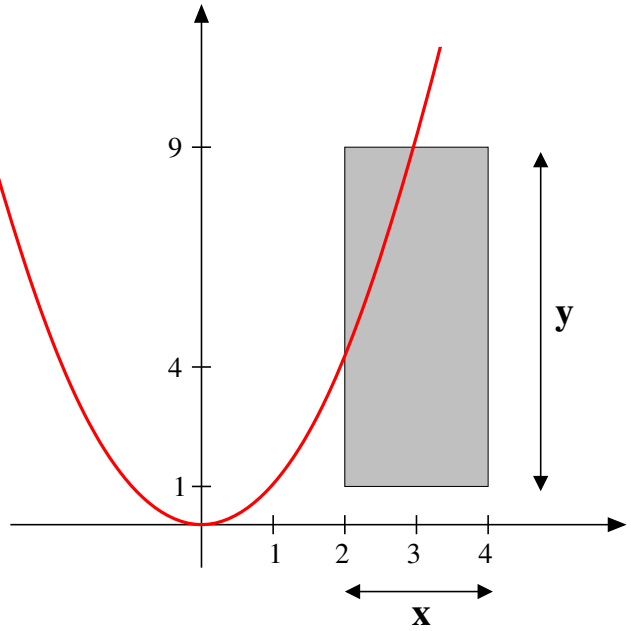
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$y = x^2 \qquad \rightsquigarrow \qquad y \in [x]^2 \cap [y]$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$
$[x]^2 = [4,16]$
$[x]^2 \cap [y] = [4,9]$

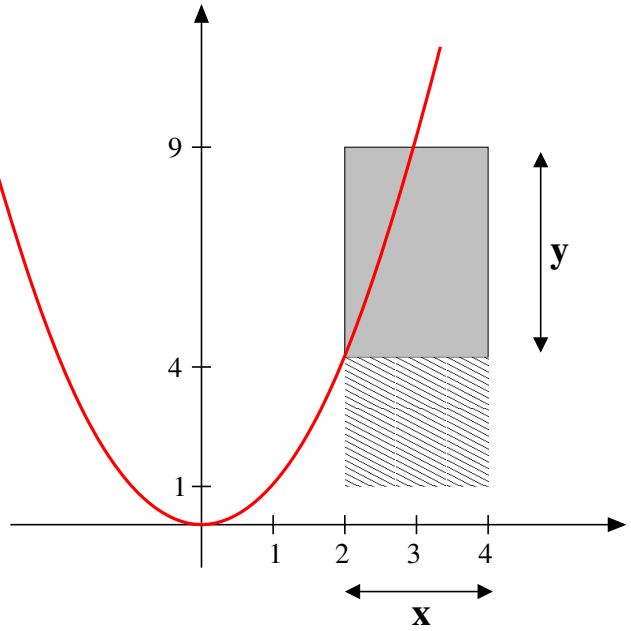
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$y = x^2 \qquad \rightsquigarrow \qquad y \in [x]^2 \cap [y]$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$
$[x]^2 = [4,16]$
$[x]^2 \cap [y] = [4,9]$

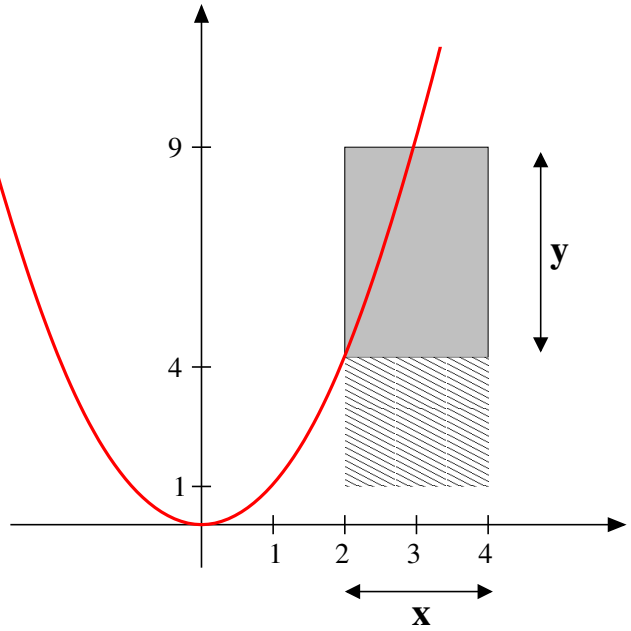
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$y = x^2 \qquad \rightsquigarrow \qquad y \in [x]^2 \cap [y]$$

Forward-backward.



$$x \in [2,4] \quad y \in [1,9]$$

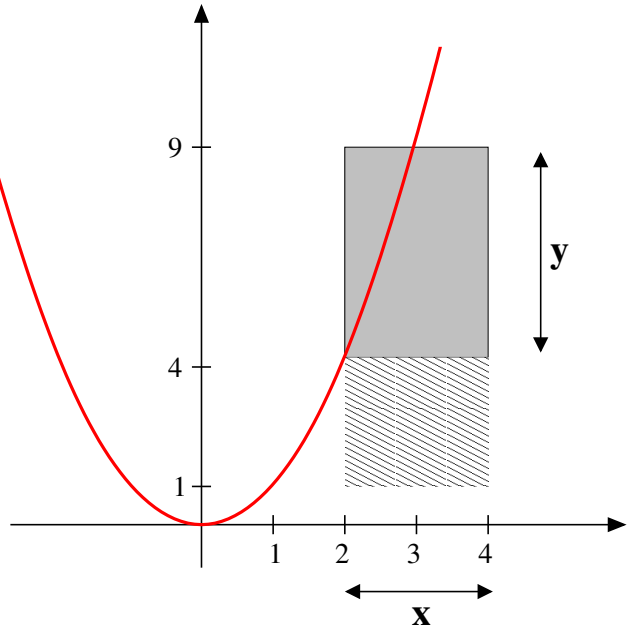
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$x = \sqrt{y} \qquad \rightsquigarrow \qquad x \in \sqrt{[y]} \cap [x]$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$
$\sqrt{[y]} = [1,3]$

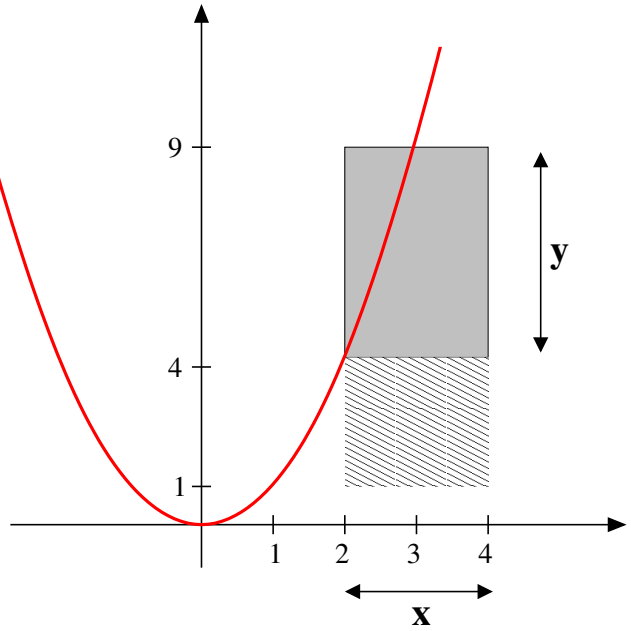
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$x = \sqrt{y} \qquad \rightsquigarrow \qquad x \in \sqrt{[y]} \cap [x]$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$
$\sqrt{[y]} = [1,3]$
$\sqrt{[y]} \cap [x] = [2,3]$

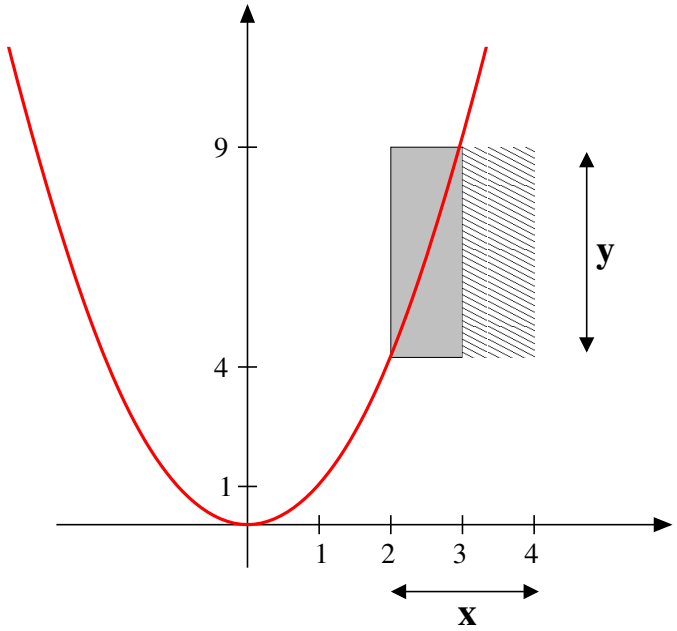
Utilisation de la forme symbolique

Exemple

$$y - x^2 = 0$$

$$x = \sqrt{y} \qquad \rightsquigarrow \qquad x \in \sqrt{[y]} \cap [x]$$

Forward-backward.



$x \in [2,4] \quad y \in [1,9]$
$\sqrt{[y]} = [1,3]$
$\sqrt{[y]} \cap [x] = [2,3]$

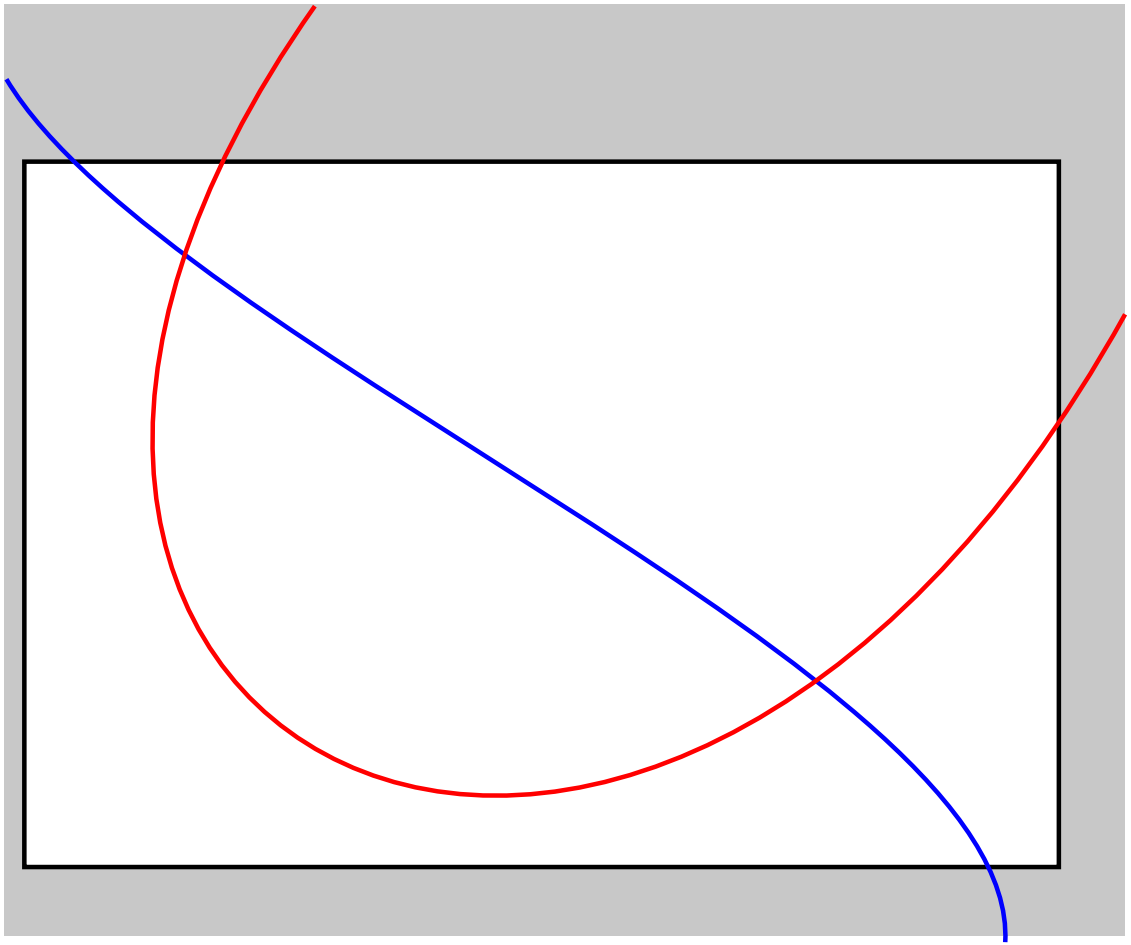
Utilisation de la forme symbolique

Exemple

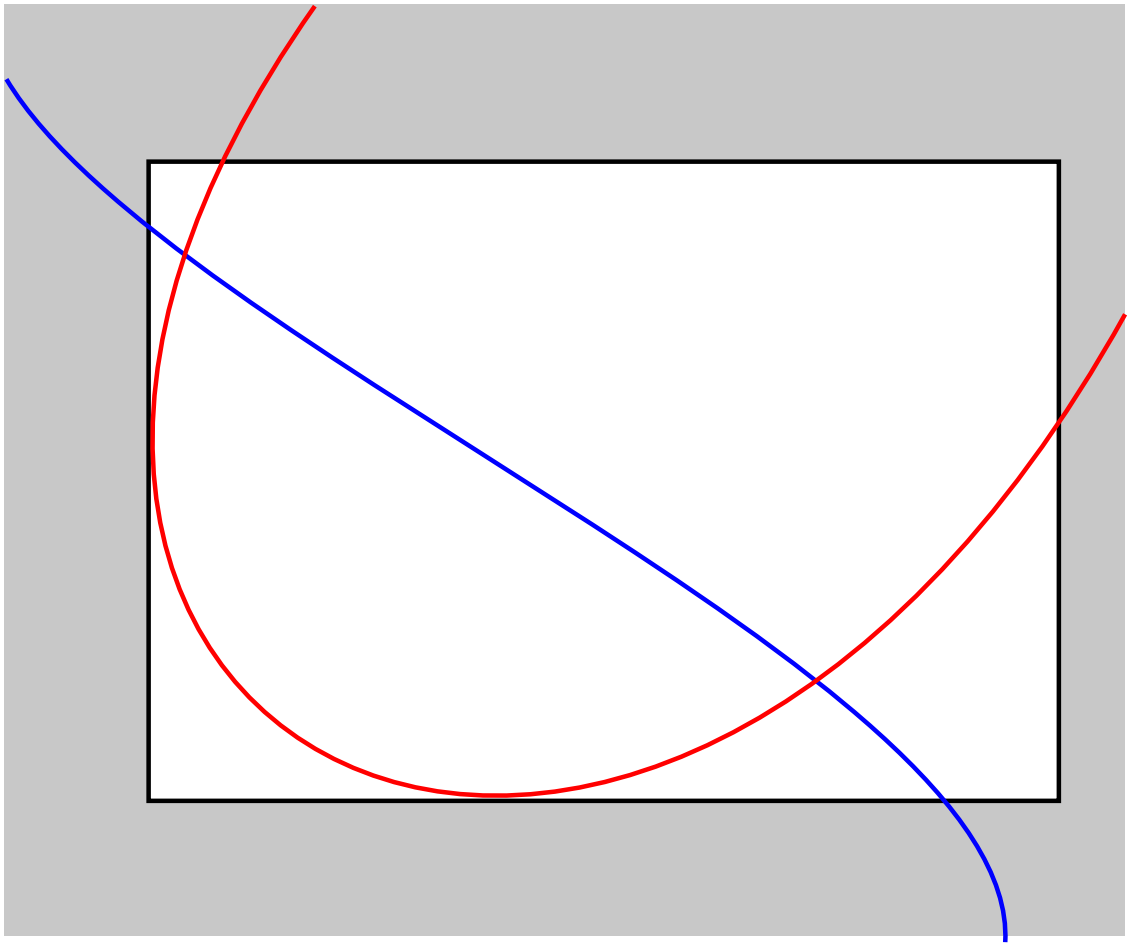
$$y - x^2 = 0$$

$$x = \sqrt{y} \qquad \rightsquigarrow \qquad x \in \sqrt{[y]} \cap [x]$$

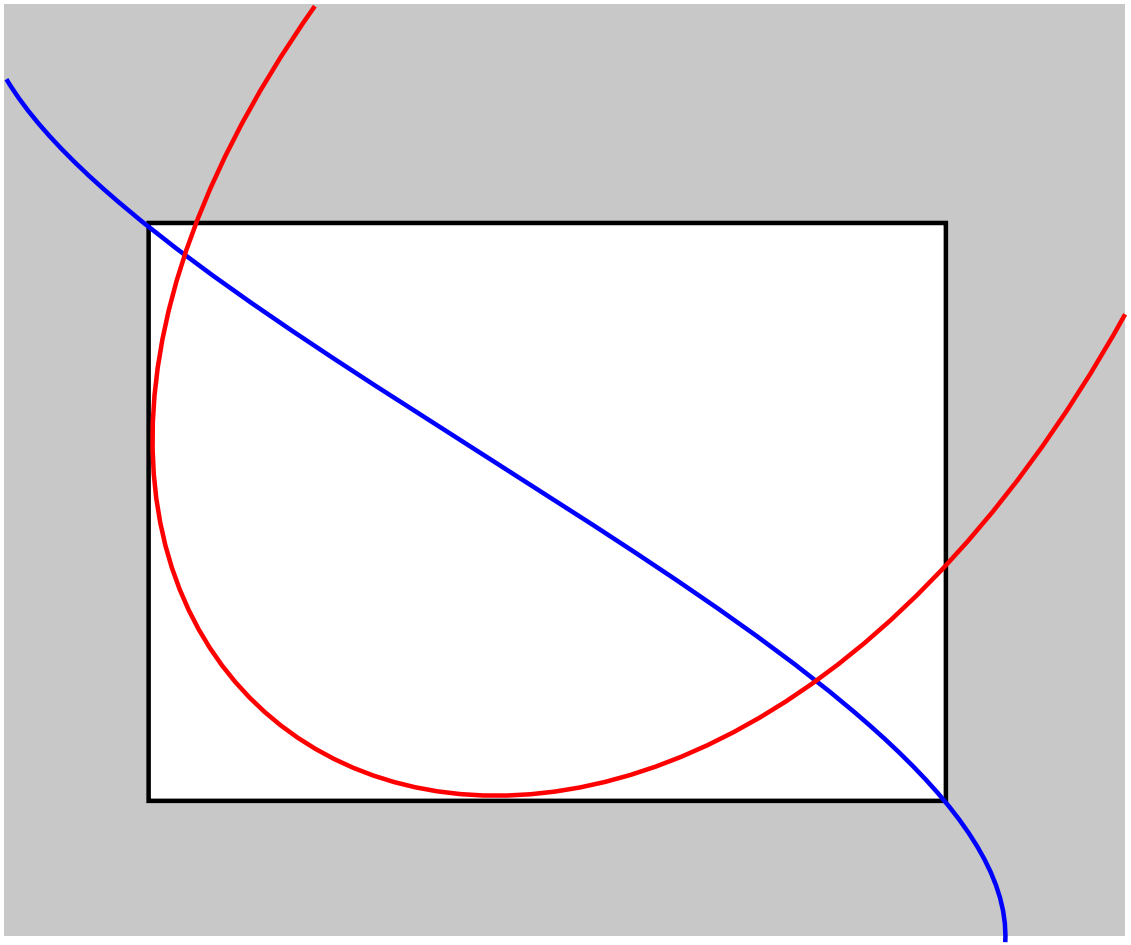
Principe (avec forward-backward).



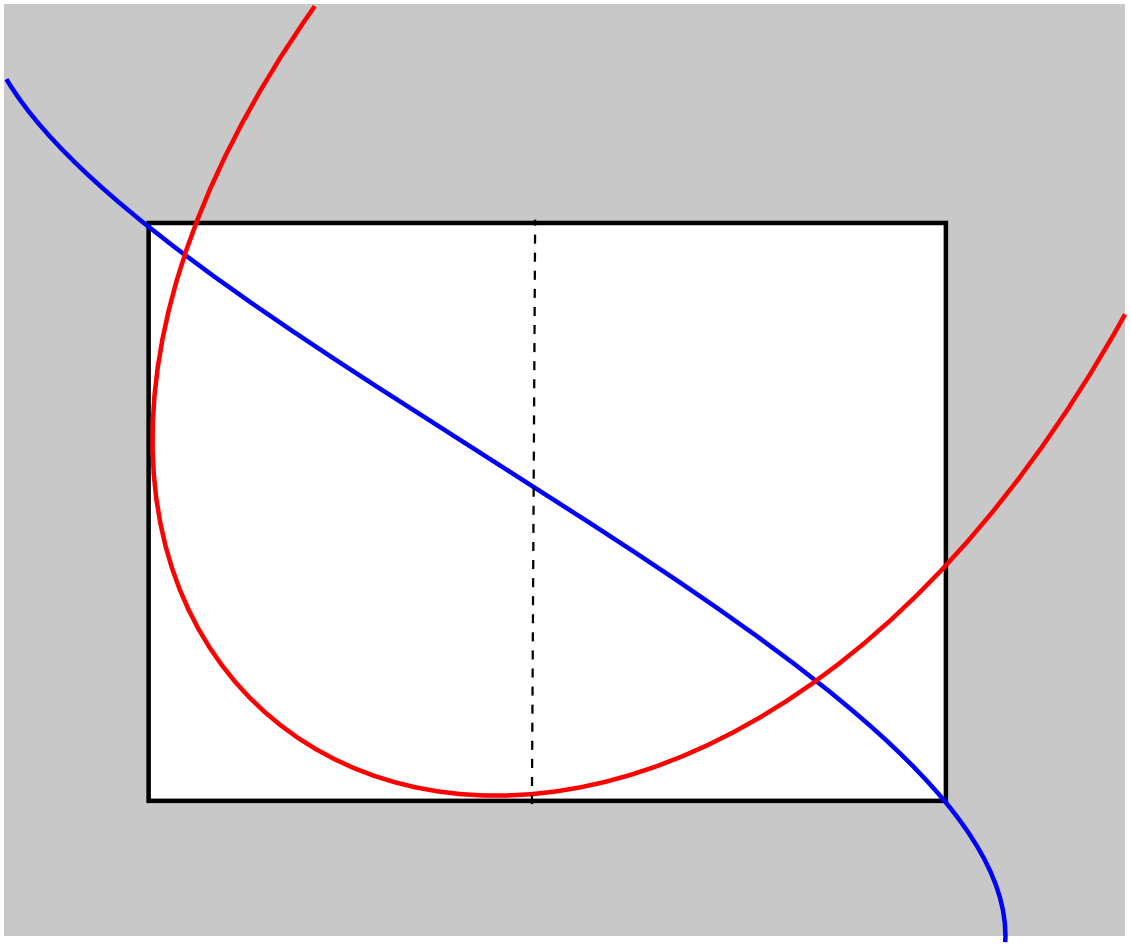
Principe (avec forward-backward).



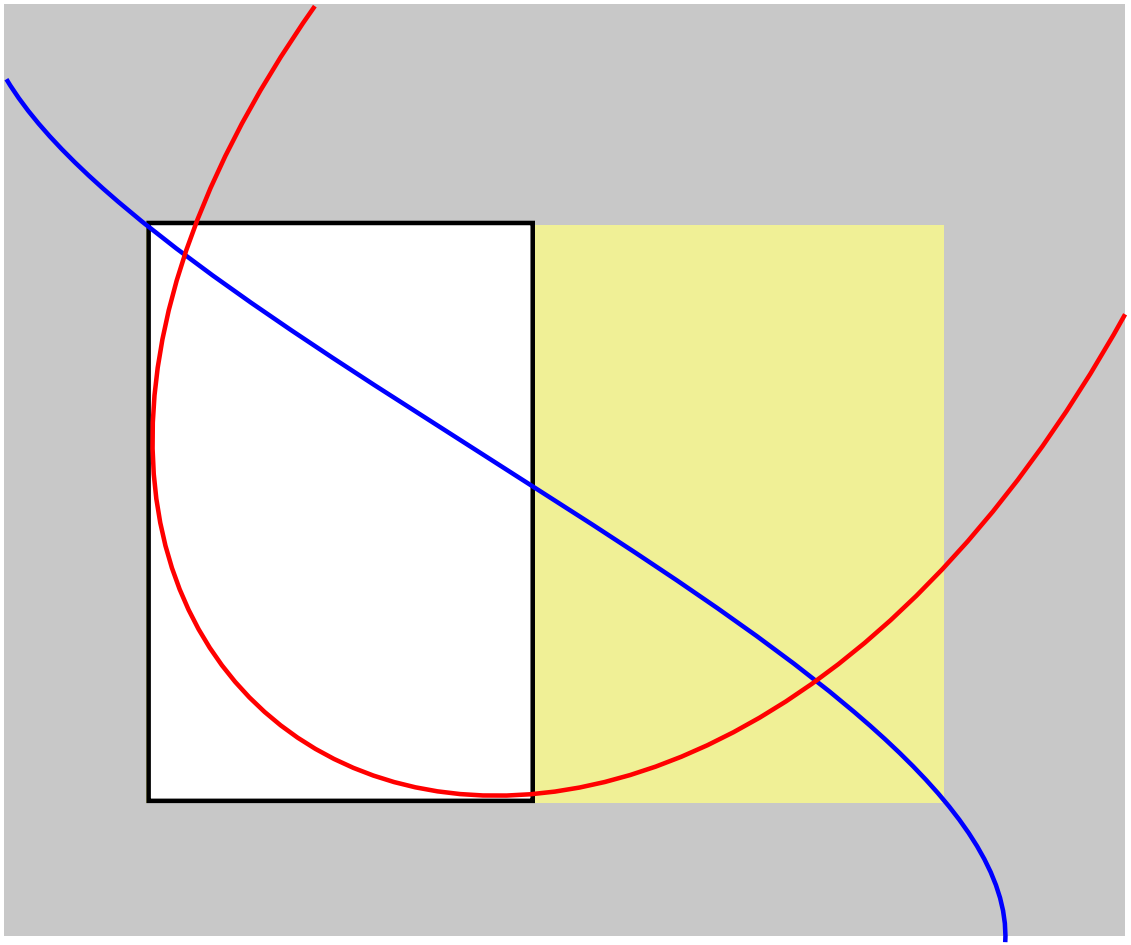
Principe (avec forward-backward).



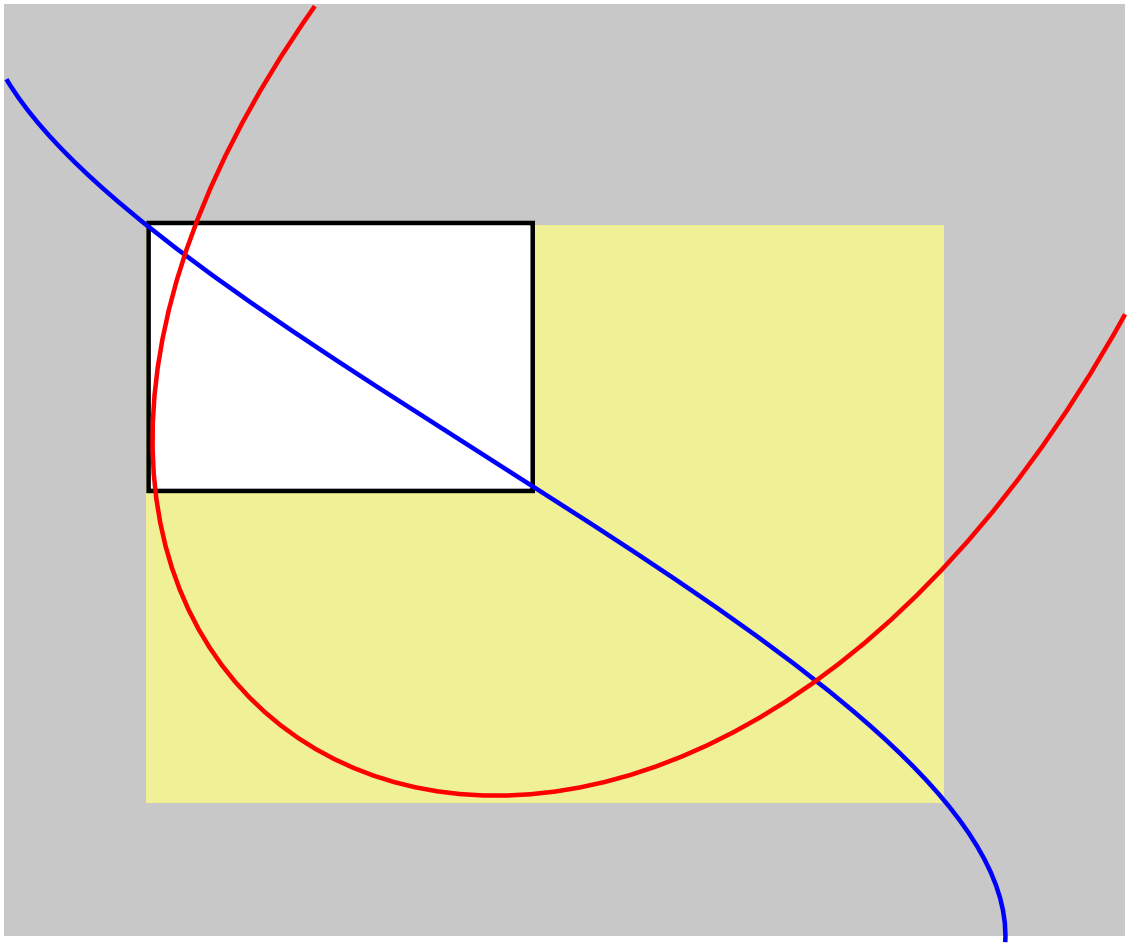
Principe (avec forward-backward).



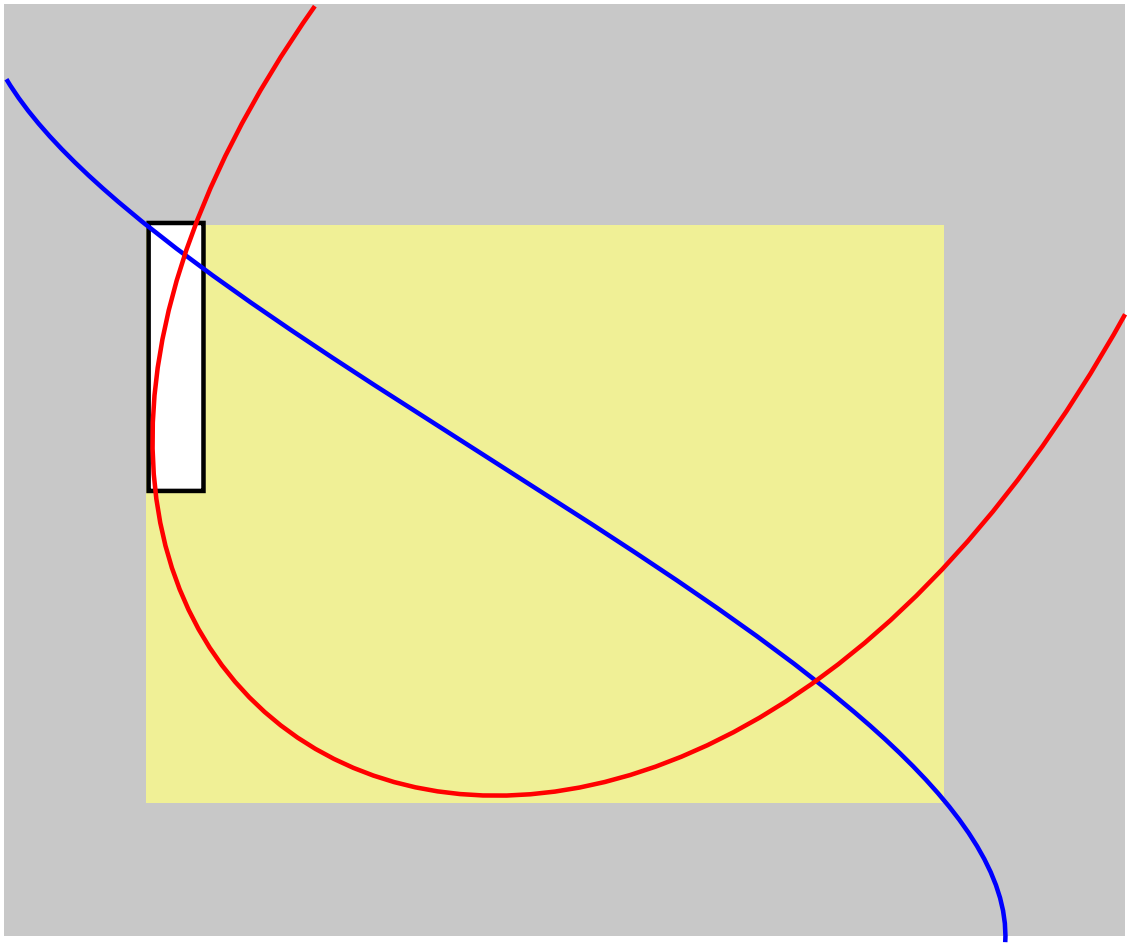
Principe (avec forward-backward).



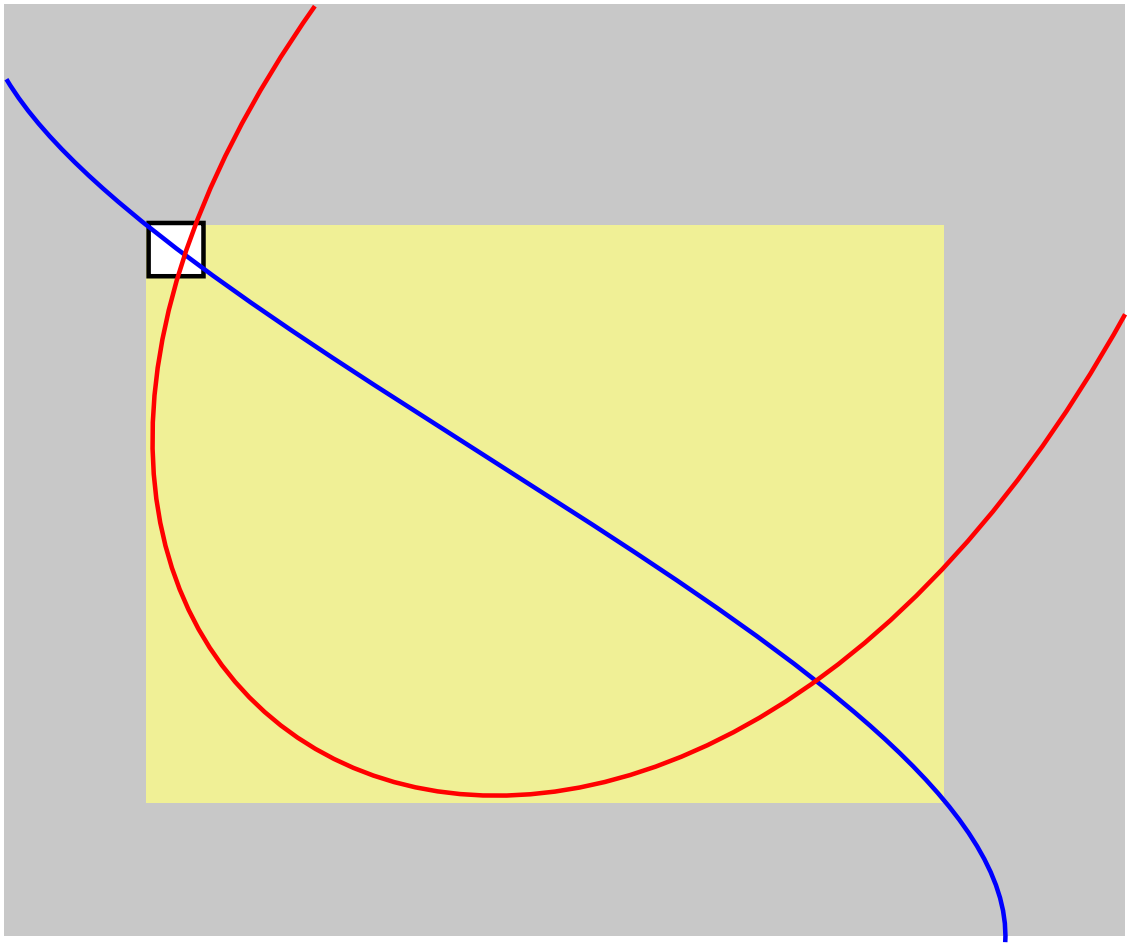
Principe (avec forward-backward).



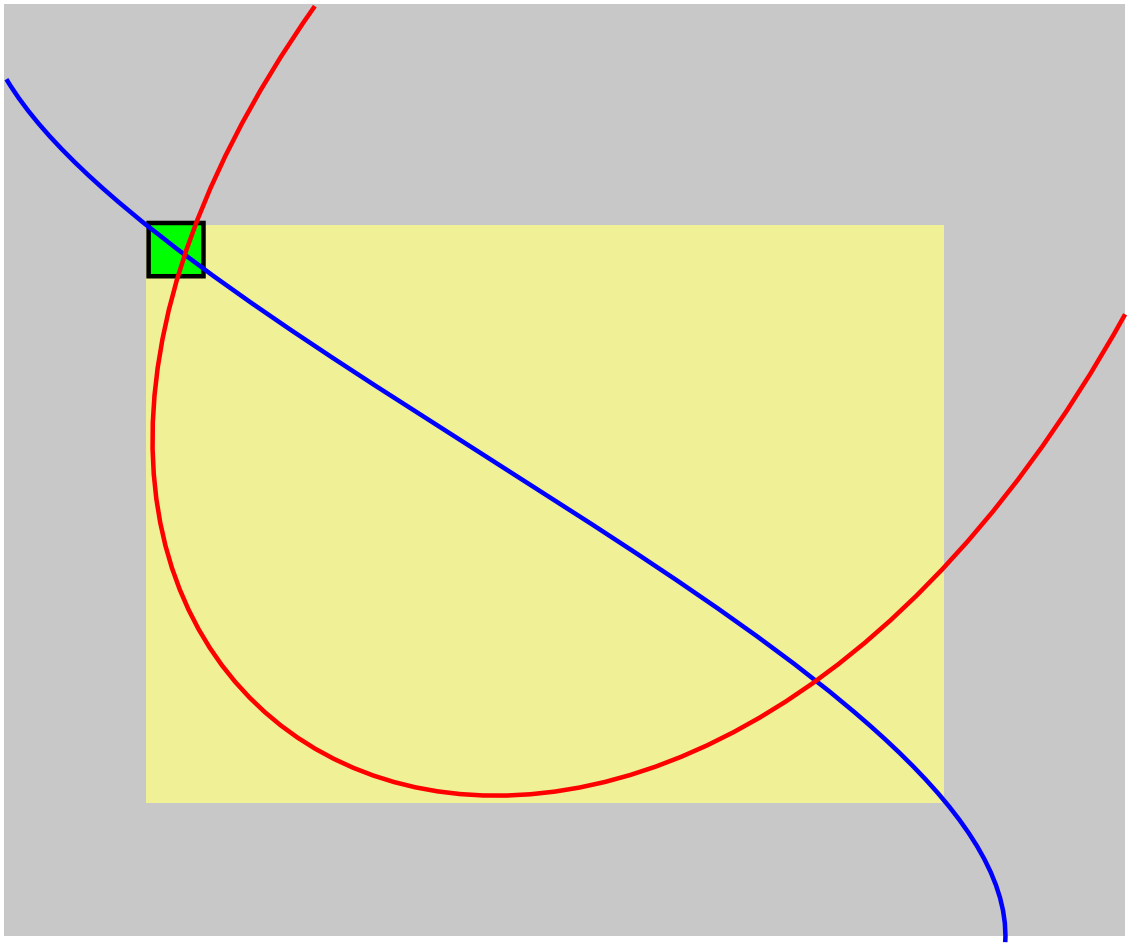
Principe (avec forward-backward).



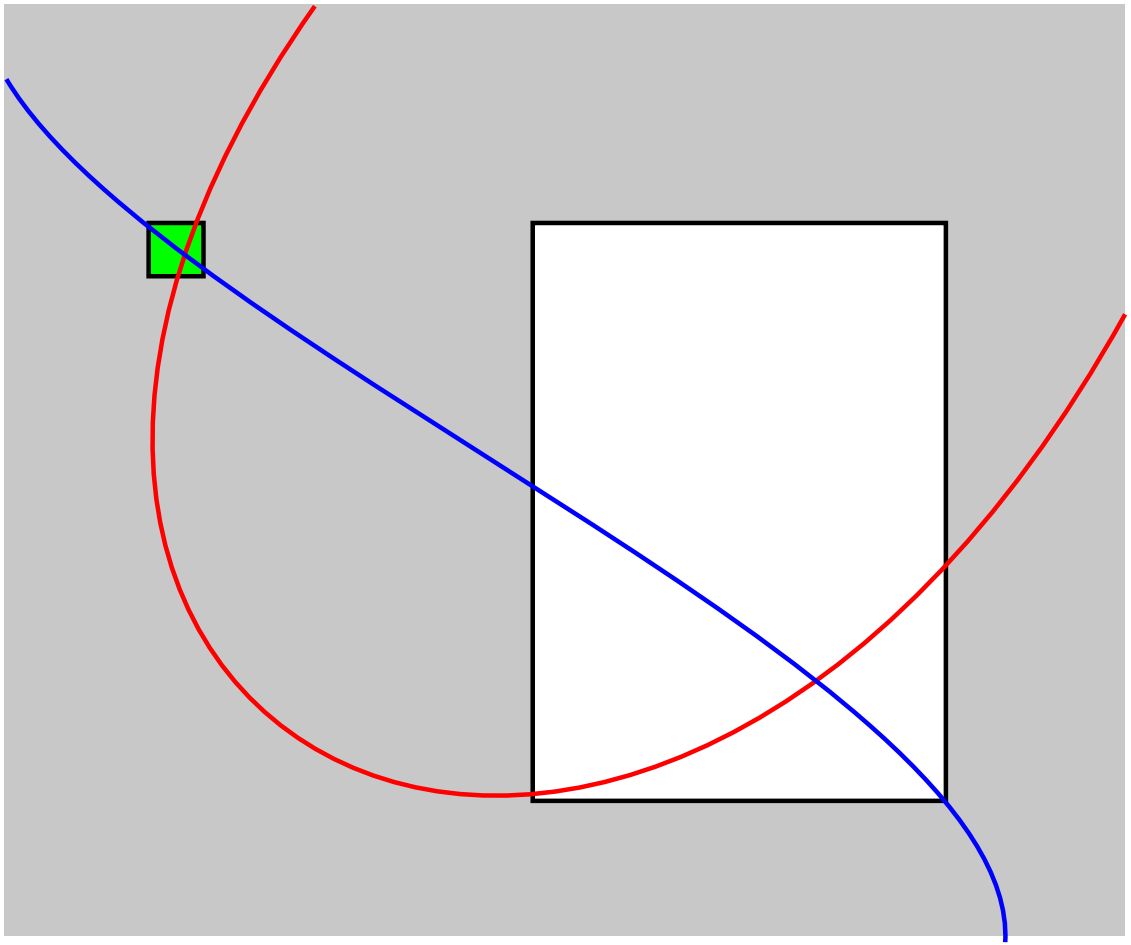
Principe (avec forward-backward).



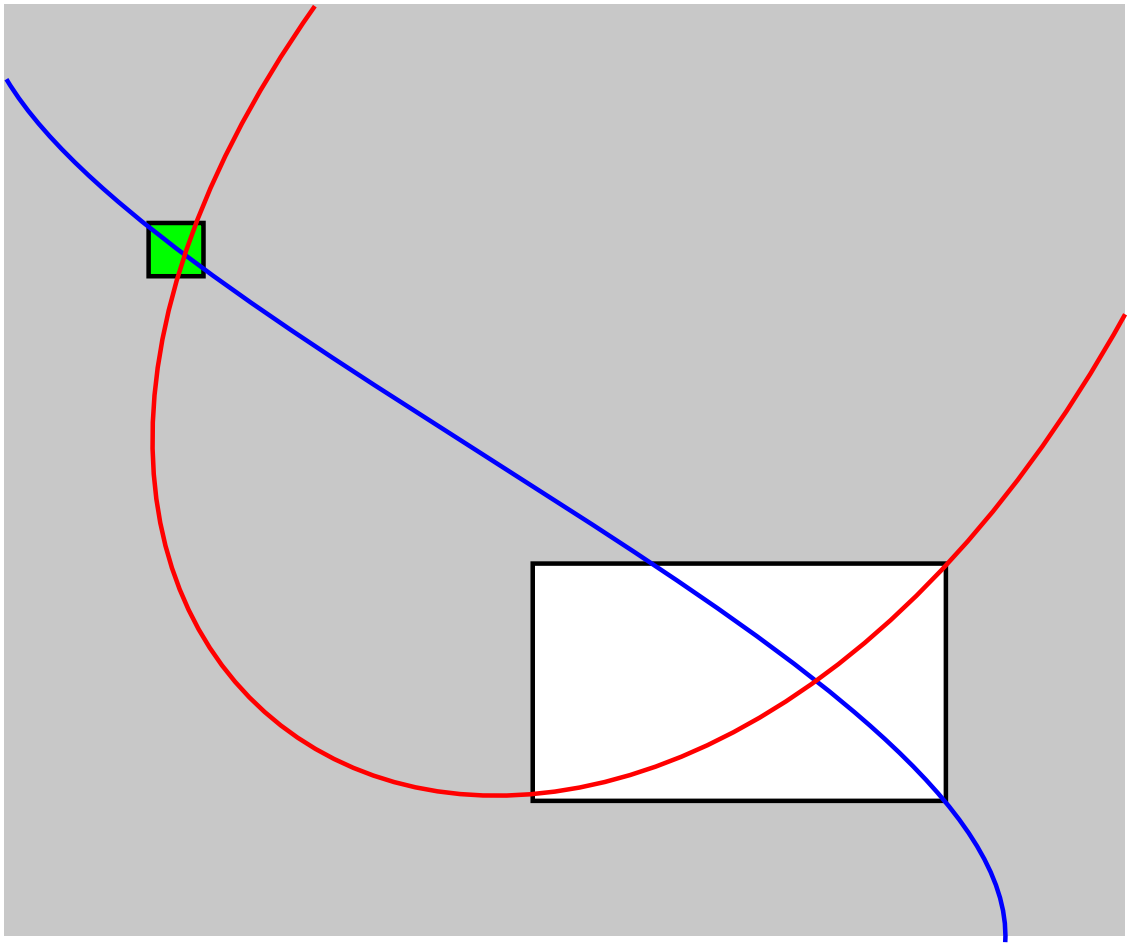
Principe (avec forward-backward).



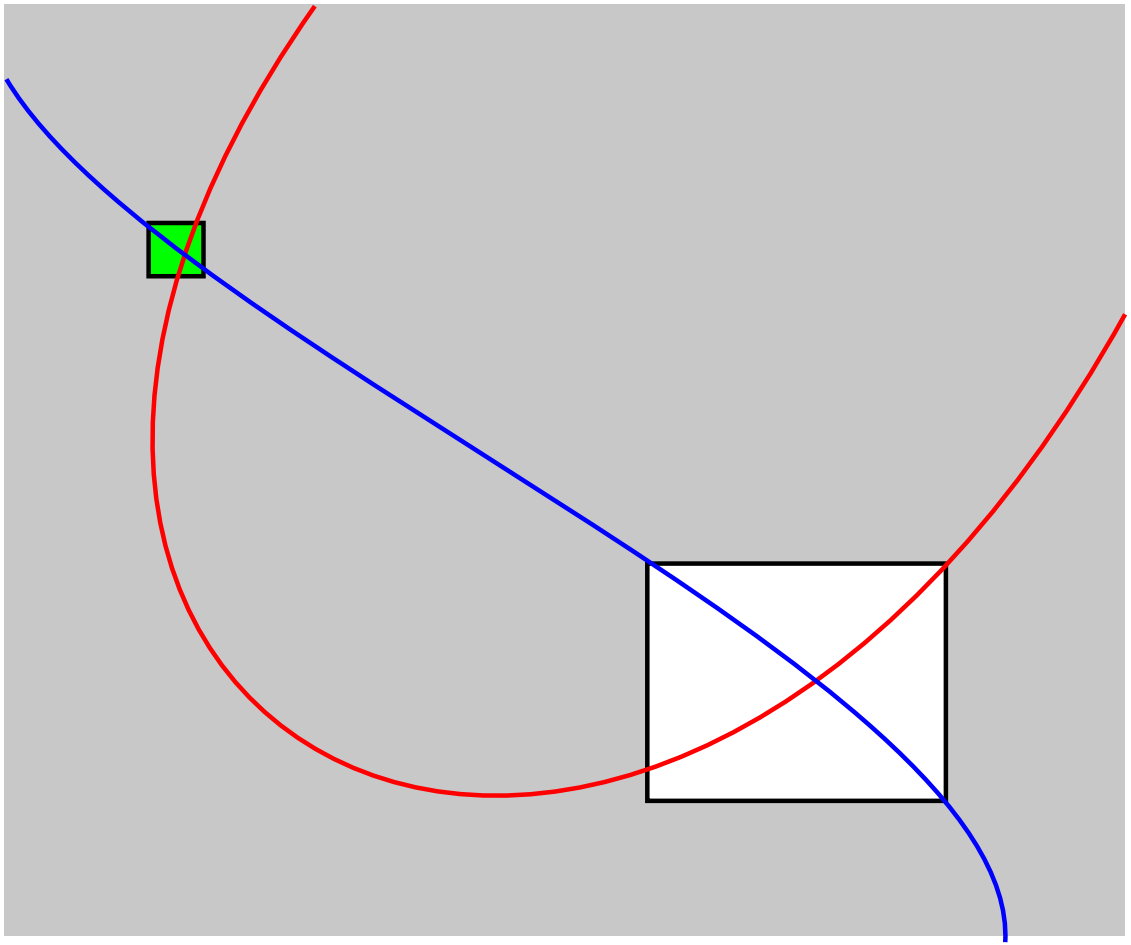
Principe (avec forward-backward).



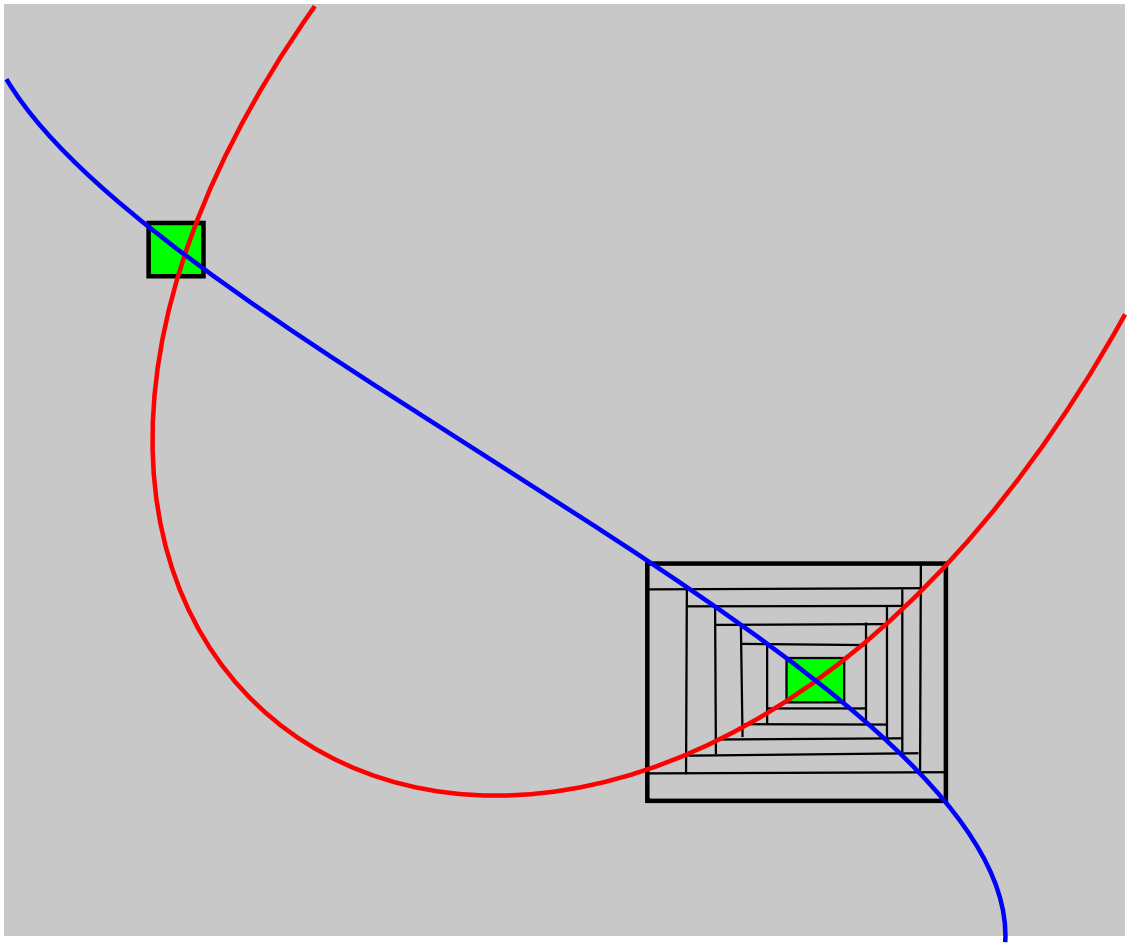
Principe (avec forward-backward).



Principe (avec forward-backward).



Principe (avec forward-backward).



Nous avons vu que le **forward-backward** pouvait contracter *extérieurement* une contrainte :

Definition (Contracteur extérieur)

Soit c une contrainte et C un contracteur. C est un **contracteur extérieure** pour c si

$$\forall [x] \subseteq \mathbb{R}^n, \quad \forall x \in [x], \quad c(x) \Longrightarrow x \in C([x]) \subseteq [x].$$

Nous définissons de façon similaire un contracteur *intérieur* pour une contrainte c comme un contracteur extérieur pour sa négation.

Ainsi, un contracteur *intérieur* pour c ne **supprime que des points solutions**.

TP n˚2

Pour le TP n˚2 vous devrez paver à la fois l'extérieur et l'intérieur d'une contrainte de type

$$f(x) \leq 0.$$

L'idée-clef consiste alors à prendre la négation de cette contrainte (c.a.d., $f(x) > 0$) : un contracteur extérieur pour la seconde contrainte permettra d'obtenir le pavage intérieur souhaité.

TP n˚2

Nouvelles classes IBEX à utiliser :

- `LargestFirst` : objet chargé de couper des boîte suivant la dimension de largeur maximale.
- `CtcFwdBwd` : contracteur basé sur le forward-backward

Nouvelles fonctions C++ à utiliser :

- Pour `LargestFirst` : la fonction `bisect`. Elle prend une boîte en argument et en retourne deux, dans une paire. On accède à la première boîte via `first`, à la seconde via `second`.
- Pour `CtcFwdBwd` : la fonction `contract` prenant une boîte et la contractant (en place).

Plan de l'exposé

1 Introduction

2 Exemples de problèmes

3 TP n˚1

4 TP n˚2

5 TP n˚3

TP n°3

L'idée-clef de la programmation par contracteurs est d'abstraire l'algorithme de contraction de sa contrainte sous-jacente et de le voir en tant que tel comme une fonction C

$$C : \mathbb{IR}^n \rightarrow \mathbb{IR}^n$$

telle que

$$C([x]) \subseteq [x]$$

Des opérateurs entre contracteurs (retournant des contracteurs) peuvent alors être définis.

TP n°3

L'idée-clef de la programmation par contracteurs est d'abstraire l'algorithme de contraction de sa contrainte sous-jacente et de le voir en tant que tel comme une fonction C

$$C : \mathbb{IR}^n \to \mathbb{IR}^n$$

telle que

$$C([x]) \subseteq [x]$$

Des opérateurs entre contracteurs (retournant des contracteurs) peuvent alors être définis.
Les opérateurs de base sont :

$$\begin{aligned}
(C_1 \cap C_2)([x]) &:= C_1([x]) \cap C_2([x]) && \text{(intersection)} \\
(C_1 \cup C_2)([x]) &:= \square\Big(C_1([x]) \cup C_2([x])\Big) && \text{(union)} \\
(C_1 \circ C_2)([x]) &:= C_1(C_2([x])) && \text{(composition)}
\end{aligned}$$

Ils peuvent être composés d'une manière arbitrairement complexe, ex :

$$((C_1 \cap C_2) \cup (C_3 \circ C_4)) \circ C_5$$

Des opérateurs plus complexes que $\cap$ ou $\circ$ existent. Ainsi la **q-intersection** est-elle généralisée à une opération entre contracteurs. Notons

$$Q = \mathtt{q-inter}(C_1, \ldots, C_n, q).$$

Alors

$$Q([x]) := \bigcup_{i_1,\ldots,i_q} \bigcap_{1 \leq j \leq q} C_i([x]).$$

L'intérêt fondamental de généraliser les opérations aux contracteurs est qu'on peut ainsi, opérationnellement, passer de simples boîtes à des ensembles quelconques.

La q-intersection peut, par exemple, être appliquée non plus entre de simple boîtes mais entre des ensembles définis eux-mêmes par des contraintes.

➞ c'est l'objet du TP n˚3.

TP n°3

La difficulté technique principale dans ce TP est de savoir passer du modèle mathématique aux contracteurs.

Les transparents suivants indiquent comment procéder.

En fait, dans IBEX, deux “mondes” cohabitent

Le monde mathématique

Il correspond au modèle (de notre problème en entrée).
Dans ce “monde”, on considère :

- trois types différents : le type scalaire, vectoriel et matriciel (+tableau de matrices)
- des variables ayant chacune un type particulier
- des fonctions (mathématique) pouvant avoir potentiellement plusieurs arguments (de différents types) et retournant une valeur d’un type particulier

Le monde numérique

L’un des intérêts d’IBEX est qu’il est orienté contracteurs : il offre un langage de haut niveau de manipulation de contracteurs qui permet de rapidement obtenir des stratégies sophistiquées, par simple composition.

Le prix à payer pour avoir ce langage est que les contracteurs soient des objets très simples, notamment :

- ils ne prennent en entrée qu'une **unique** boîte (un seul argument)
- cette boîte est un objet numérique pur, totalement "anonyme" :
  ➡ *il est impossible d'attacher un contracteur à des variables du problème* !

Conséquence : lorsqu'un contracteur C effectue le backward d'une fonction f prenant n arguments, et qu'on souhaite passer à C le domaine de n variables, il faut donc "agglomérer" ces intervalles dans une seule boîte. On parlera d'**écriture** du domaine dans une boîte.

Inversement, il faut pouvoir **lire** le domaine d'une variable à partir de la boîte contractée par C.

TP n˚3

Problème : pour pouvoir lire/écrire le domaine d'une variable depuis/dans une boîte, il semble, a priori, qu'il faille

- d'une part, connaître la façon particulière dont les domaines sont "encodés" dans une boîte
- d'autre part, effectuer des calculs d'indices alambiqués

En réalité, ni l'un ni l'autre n'est nécessaire !

Principe-clé : c'est la notion de **fonction de projection** qui va permettre de passer d'un monde à l'autre, sans concept supplémentaire et, du point de vue pratique, sans risque de bug.

TP n˚3

Quelle que soit sa signature, une fonction prend en entrée **une boîte**, et donne en sortie un objet d'un type précis :

- intervalle
- un vecteur d'intervalles (... *accessoirement également représenté par une boîte !*)
- ou une matrice d'intervalles.

Ainsi, la fonction suivante

$$proj : (x \in \mathbb{R}^7, y \in \mathbb{R}^2, z \in \mathbb{R}^{2\times 2}) \mapsto y$$

prend en entrée une boîte de dimension

$$13 = 7 + 2 + (4 \times 4)$$

et retourne un vecteur d'intervalles à 2 composantes.

Il est possible d'effectuer un `backward` sur un objet de type `Function` qui fonctionne suivant un principe similaire :

Le backward prend deux arguments : en premier le domaine de y (son type précis : un intervalle, une boîte ou une matrice intervalle) et en second le domaine de x (une boîte).

Exemple

```
Function f(x,sin(x));
IntervalVector x(1,Interval::ALL_REALS);
f.backward(Interval::ALL_REALS, x);
//--> x vaut [-1,1]
```

C'est cette asymétrie (entre input et output) qui va nous permettre de passer d'un monde à l'autre car :

- appliqué en mode **forward**, la fonction *proj* permet de lire le domaine de la variable *y* depuis une boîte de dimension 13
- appliqué en mode **bacwkard**, la fonction *proj* permet d'écrire le domaine de la variable *y* dans une boîte de dimension 13

⟹ on peut donc complètement ignorer la façon dont les domaines sont agglomérés dans une boîte.