Programmation par contraintes sur intervalles

Gilles.Trombettoni@lirmm.fr

Université de Montpellier
http://www.lirmm.fr/˜trombetton/cours/intervalles1415m1.pdf

Montpellier, 3 mars 2015

Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Plan

1 Introduction
- Définitions, notations
- Extension d'une fonction aux intervalles
- Schémas de résolution des systèmes de contraintes et d'optimisation globale
- Estimation de paramètres ensembliste
- Domaines d'application, bibliographie, outils

2 Extensions de fonctions aux intervalles

3 Algorithmes de bissection et contraction
- Heuristiques de branchement
- Algorithme HC4 de propagation de contraintes
- Algorithmes de consistance forte
- Autres algorithmes de contraction

Qu'est-ce qu'un intervalle ?

L' **arithmétique d'intervalles** permet des calculs **rigoureux** sur les ordinateurs en gérant des **flottants** et des **arrondis** conservatifs.

Définitions de base, notations

intervalle $[x_i] = [\underline{x_i}, \overline{x_i}]$	$\{x_i \in \mathbb{R}, \underline{x_i} \leq x_i \leq \overline{x_i}\}$
$\underline{x_i}$ et $\overline{x_i}$	bornes flottantes
$\mathbb{IR}$	ens. de tous les intervalles
$\emptyset$	intervalle vide
$\mathrm{m}([x_i])$	**point milieu** de $[x_i]$
$w([x_i]) := \overline{x_i} - \underline{x_i}$	**largeur** ou taille de $[x_i]$

Qu'est-ce qu'une boîte ?

- Une **boîte**, parallèle aux axes, $[x]$ est un produit cartésien d'intervalles : $[x_1] \times ... \times [x_i] \times ... \times [x_n]$.
- $w([x]) := Max_n(w([x_i]))$
- Certaines opérations ensemblistes applicables sur des boîtes (ex : inclusion, intersection).
- L'opérateur **Hull** d'*enveloppe* permet d'approximer un ensemble S de boîtes par la plus petite boîte (dite **extérieure**) contenant S.

En pratique, une boîte est un vecteur d'intervalles qui définit un **espace de recherche** dans lequel se trouvent les valeurs des inconnues d'un système de contraintes.

Plan

1. Introduction
    - Définitions, notations
    - Extension d'une fonction aux intervalles
    - Schémas de résolution des systèmes de contraintes et d'optimisation globale
    - Estimation de paramètres ensembliste
    - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
    - Heuristiques de branchement
    - Algorithme HC4 de propagation de contraintes
    - Algorithmes de consistance forte
    - Autres algorithmes de contraction

L'extension aux intervalles d'une fonction sur les réels permet un calcul conservatif de l'image de la fonction en chaque point.

Extension d'une fonction sur $\mathbb{IR}$ (fonction d'inclusion)

Considérons une fonction $f : \mathbb{R}^n \to \mathbb{R}$.
$[f] : \mathbb{IR}^n \to \mathbb{IR}$ est une **extension** de f aux intervalles ssi :

$$\forall [x] \in \mathbb{IR}^n \quad [f]([x]) \supseteq \Im f([x]) \equiv \{f(x),\ x \in [x]\}$$
$$\forall x \quad \in \mathbb{R}^n \qquad f(x) = [f](x)$$

Pour les fonctions élémentaires, l' **arithmétique des intervalles** calcule rapidement l'image (aux arrondis près). Exemples :

$$\begin{aligned}
\alpha[x] &= [\alpha\underline{x}, \alpha\overline{x}] \ \ si \ \ \alpha \geq 0, \quad \alpha \in \mathbb{R} \\
&= [\alpha\overline{x}, \alpha\underline{x}] \ \ si \ \ \alpha < 0 \\
[\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}] \\
[\underline{x}, \overline{x}] - [\underline{y}, \overline{y}] &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}]
\end{aligned}$$

Arithmétique d'intervalles

$$[x]\,.\,[y] = [min\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}, max\{\underline{x}\underline{y}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{x}\overline{y}\}]$$

$$sqr([-4,2]) = [0,16]$$

$$\begin{aligned} 1/[y] &= \emptyset && si\ \ [y]=[0,0] \\ &= [1/\overline{y}, 1/\underline{y}] && si\ \ 0 \notin [y] \\ &= [1/\overline{y}, +\infty] && si\ \ \underline{y}=0\ \ et\ \ 0<\overline{y} \\ &= [-\infty, 1/\underline{y}] && si\ \ \underline{y}<0\ \ et\ \ 0=\overline{y} \\ &= [-\infty, +\infty] && sinon\ \ (0 \in [y]) \end{aligned}$$

$$exp([x]) = [exp(\underline{x}), exp(\overline{x})]$$

Sinus, cosinus ??

Extension naturelle $[f]_N$

Evaluation naturelle

Extension **naturelle** $[f]_N$ d'une fonction f **composée** : l'arithmétique des intervalles est utilisée en chaque opérateur.

Exemple

- Considérons $f(x) = x(x+1)$.
- Quelle est l'évaluation naturelle de f dans $[x] = [-1, +1]$?

Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Schéma combinatoire (naïf) de résolution : bissection + évaluation

Soit $f : \mathbb{R}^2 \to \mathbb{R}^2 \quad x \mapsto (x_1^2 + x_2^2 - 4^2,\ (x_1 - 4)^2 + x_2^2 - 6^2)^T$
Cherchons les solutions de $f(x) = 0$, soit les solutions du système
$\{f_1(x_1, x_2) = x_1^2 + x_2^2 - 4^2 = 0\ ,\ f_2(x_1, x_2) = (x_1 - 4)^2 + x_2^2 - 6^2 = 0\}$

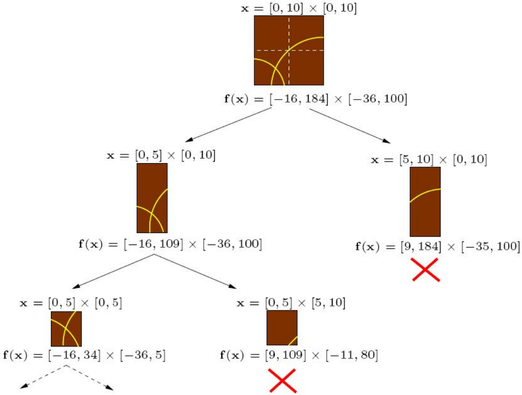


Schéma de résolution : bissection + contraction

Les méthodes à intervalles permettent de trouver/approximer **toutes** les solutions d'un système de contraintes sur les réels.

Schéma général : **Branch & Contract**

1. $L \leftarrow \{[x_0]\}$ /* On part d'une boîte initiale $[x_0]$ */
2. Tant que L est non vide faire :
    1. Choisir une boîte $[x]$ de L (et l'enlever de L).
    2. **Contracter** $[x]$ et essayer de **garantir** l'existence et/ou de l'unicité d'une solution dans $[x]$.
    3. $[x] = \emptyset \quad \Rightarrow \quad$ pas de solution dans $[x]$
    4. $[x] \neq \emptyset$ et $w([x]) \leq \epsilon \quad \Rightarrow$ $[x]$ est "solution" (garantie ou non)
    5. $[x] \neq \emptyset$ et $w([x]) > \epsilon \quad \Rightarrow$ **Bissection**: $[x]$ est découpée sur une dimension (variable) en deux sous-boîtes $[x_g]$ et $[x_d]$
    6. $L \leftarrow L \cup \{[x_g]\} \cup \{[x_d]\}$

Schéma général de résolution (suite)

Remarques

- Définition de **contraction** : réduction de la boîte courante sur les bornes sans perte de solution.
- Toutes les stratégies de résolution basées sur les intervalles trouvent un **sur-ensemble** des solutions (en utilisant des algorithmes de contraction rigoureux).

Trois types de contracteurs... et de communautés

- Relaxation linéaire ou convexe (programmation mathématique).
- Extension aux intervalles des algorithmes d'analyse numérique (analyse par intervalles).
- Propagation de contraintes et consistance forte (programmation par contraintes).

Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

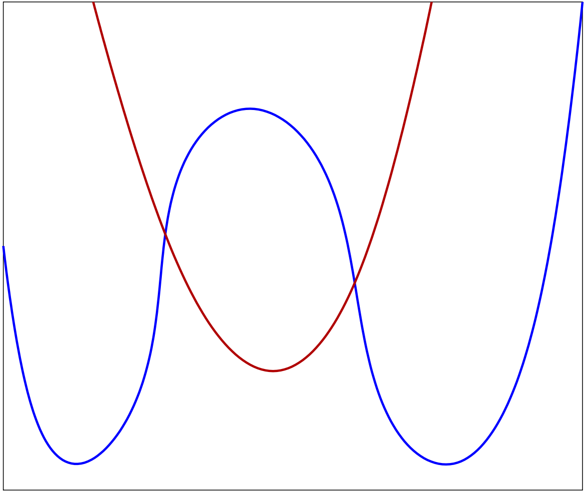


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

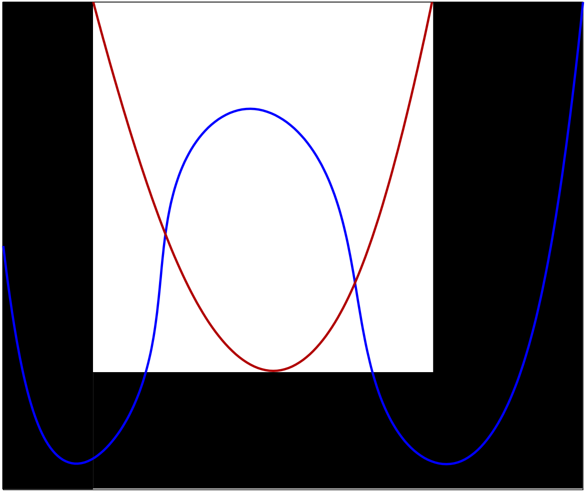


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

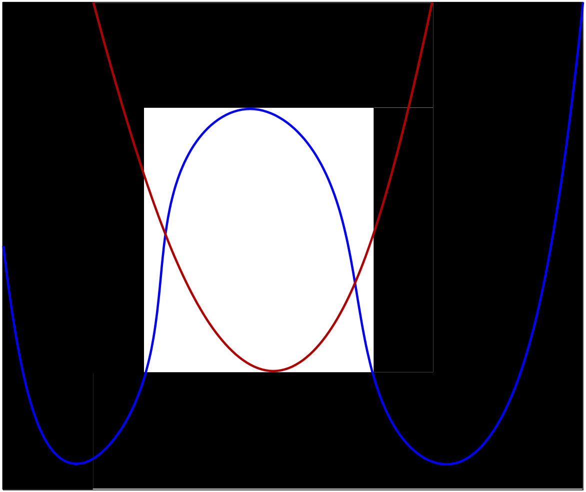


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

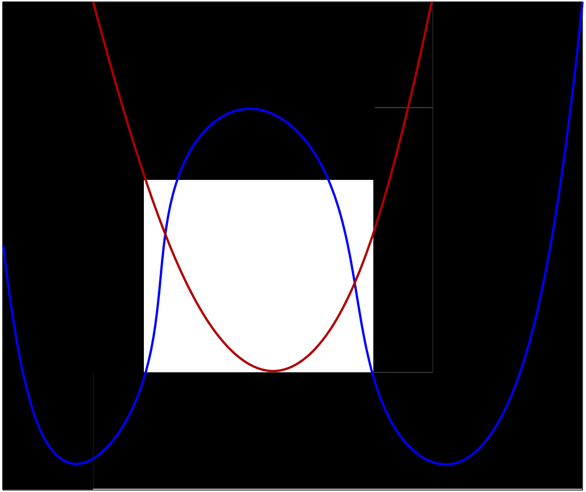


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

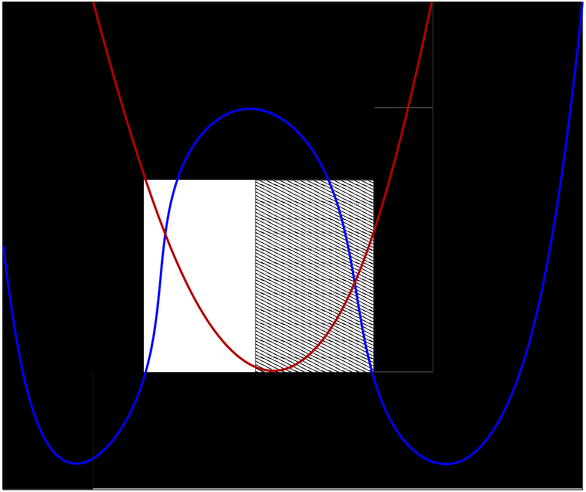


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

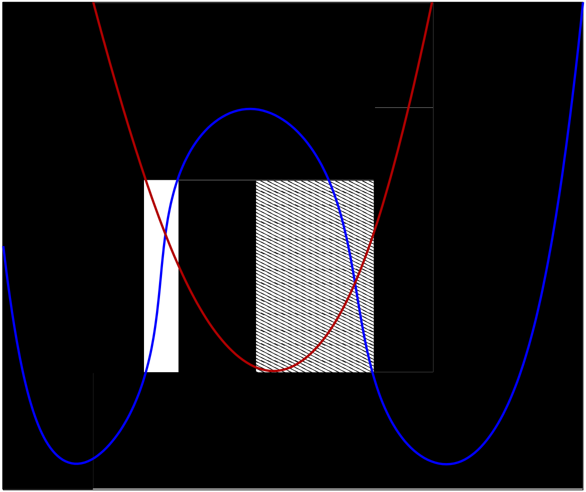


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

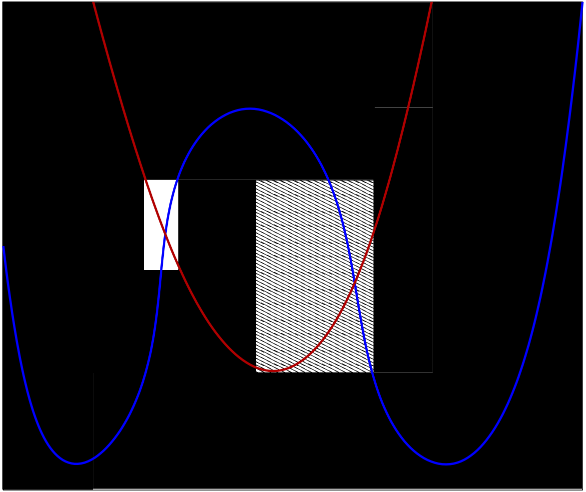


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

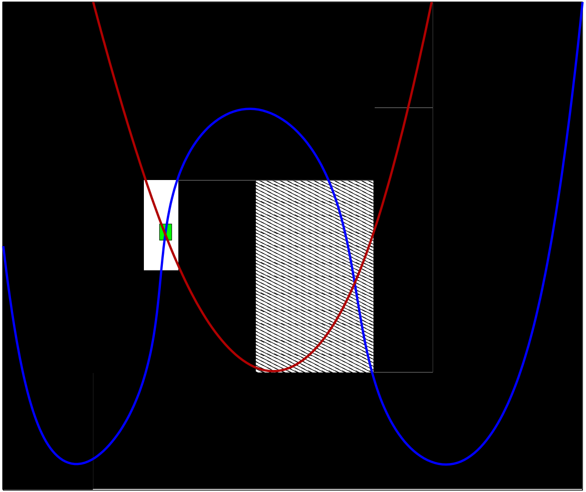


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$

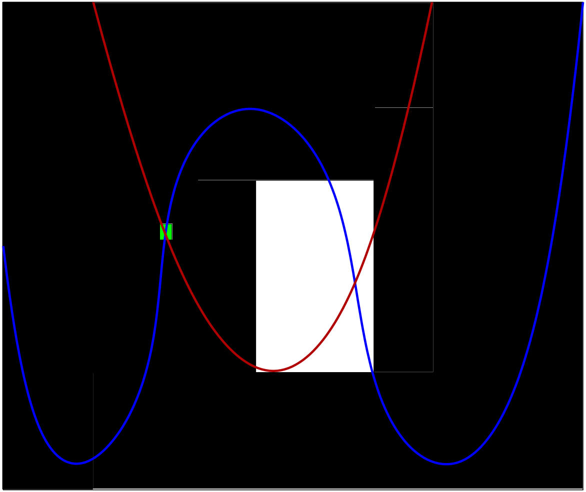
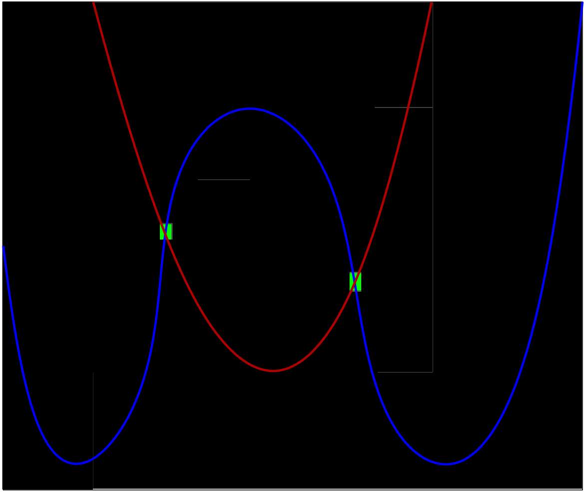


Schéma général de résolution de $\{f_1(x_1, x_2) = 0,\ f_2(x_1, x_2) = 0\}$



Optimisation globale sous contraintes rigoureuse

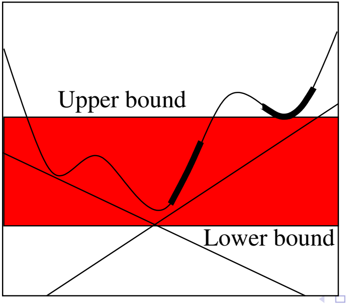
Optimisation globale continue sous contraintes, **rigoureuse**:

- Optimisation (continue) sous contraintes:
  $argmin_{x \in [x] \subset \mathbb{R}^n} f(x) \;\; s.c. \;\; g(x) \leq 0 \wedge h(x) = 0$
- La meilleure solution est garantie avec une erreur bornée sur le coût (ϵ-minimisation). Deux niveaux de rigueur :
  - Sortie : une petite **boîte** contenant un vecteur réel x ϵ-minimisant: $f(x) \;\; s.c. \;\; g(x) \leq 0 \wedge h(x) = 0$.
  - Sortie : un vecteur **flottant** x ϵ-minimisant:
    $f(x) \;\; s.c. \;\; g(x) \leq 0 \wedge -\epsilon_{eq} \leq h(x) \leq \epsilon_{eq}$.

Optimisation globale sous contraintes : ingrédients

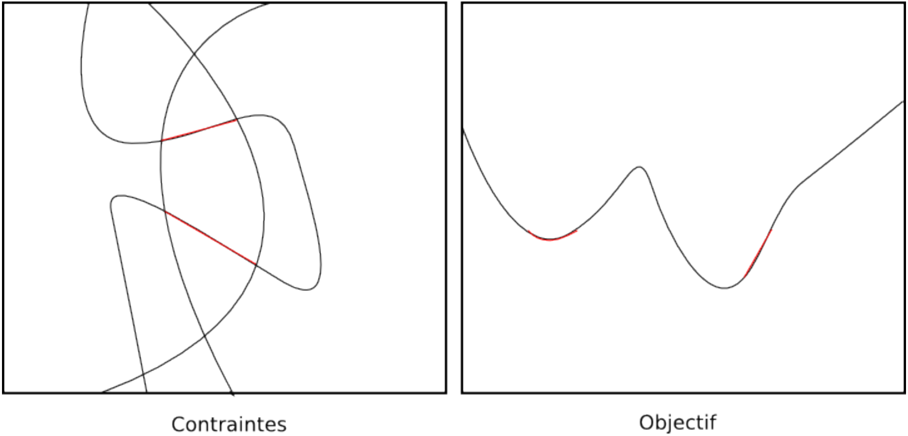
Algorithme de *Branch & Bound* sur intervalles contenant:

- exploration combinatoire (bissection)
- contraction
- recherche d'un minorant du coût (**lower bounding**)... convexification
- recherche d'un point réalisable (faisable) majorant le coût (**upper bounding**)... optimisation locale

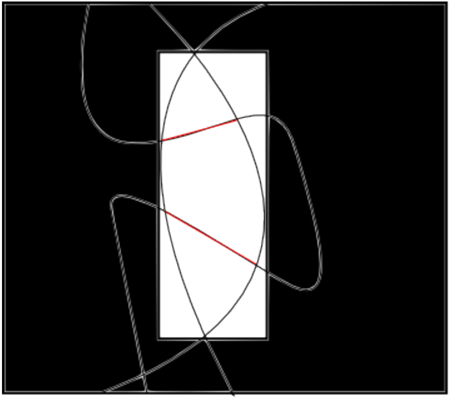


Objective function

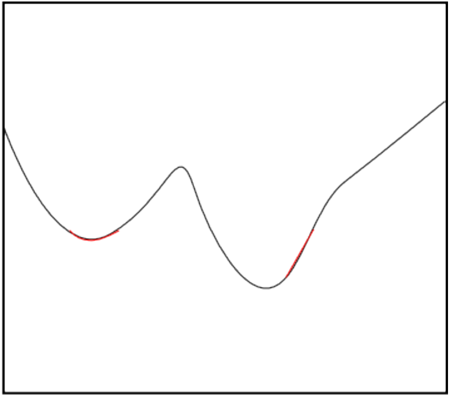
Illustration



Illustration

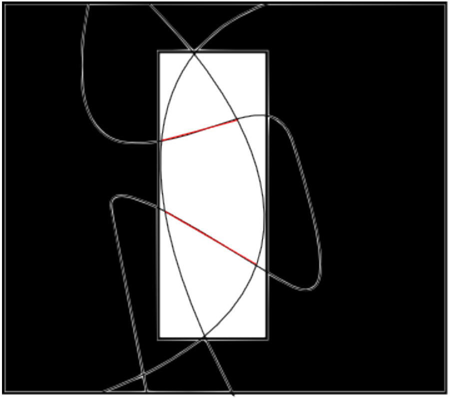


Contraintes

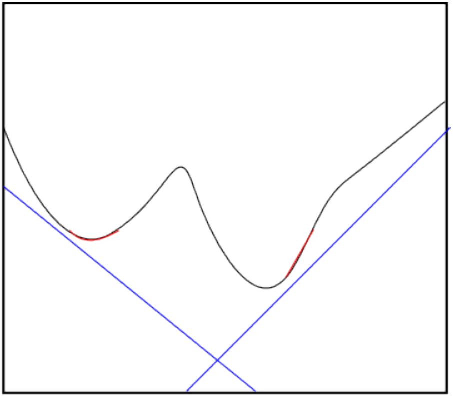


Objectif

Illustration

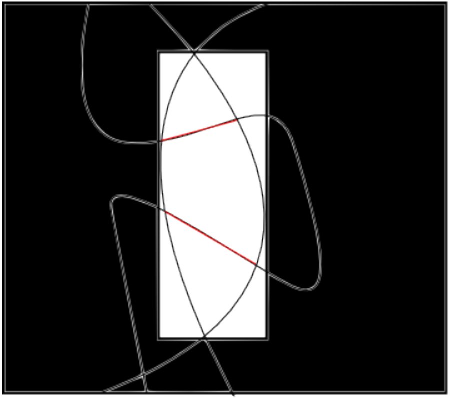


Contraintes

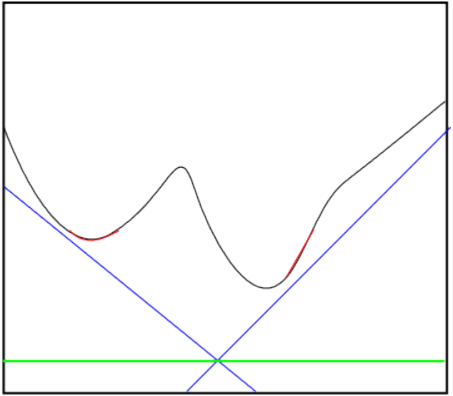


Objectif

Illustration

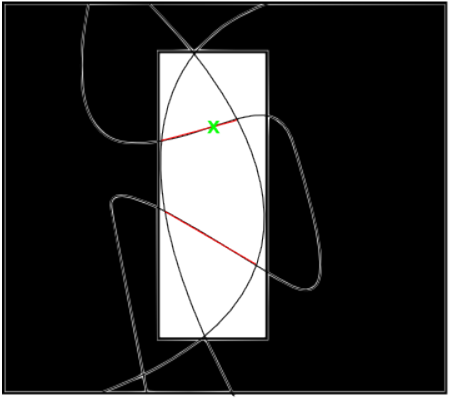


Contraintes

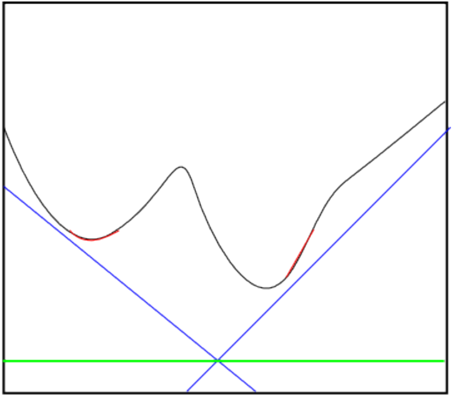


Objectif

Illustration

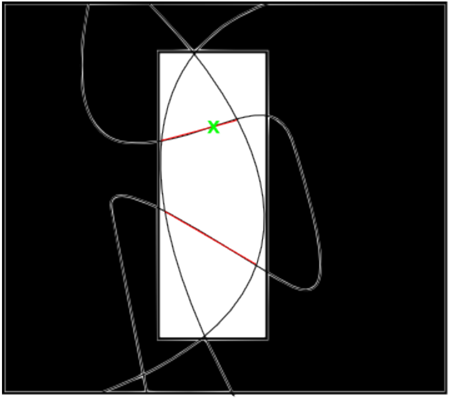


Contraintes

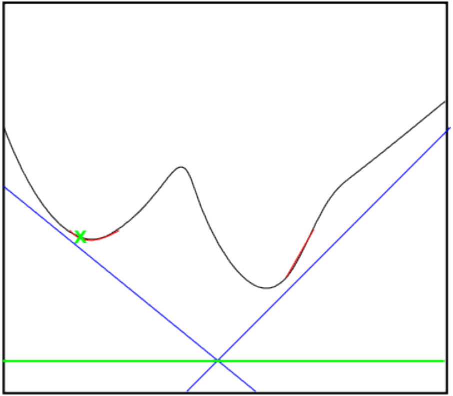


Objectif

Illustration

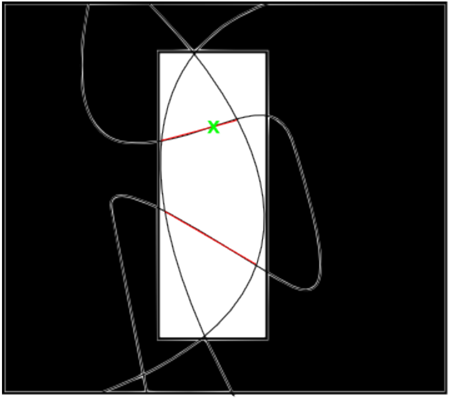


Contraintes

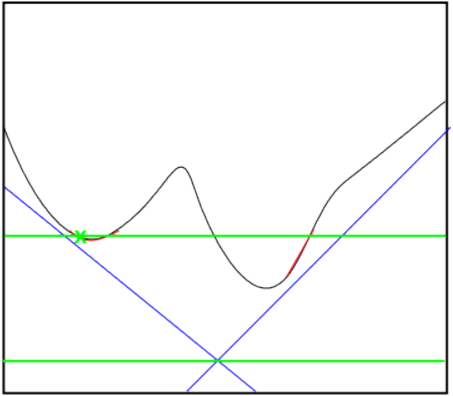


Objectif

Illustration

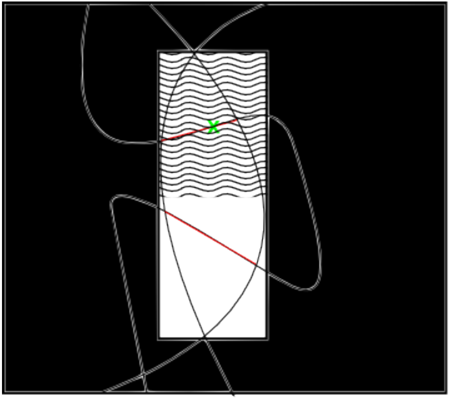


Contraintes

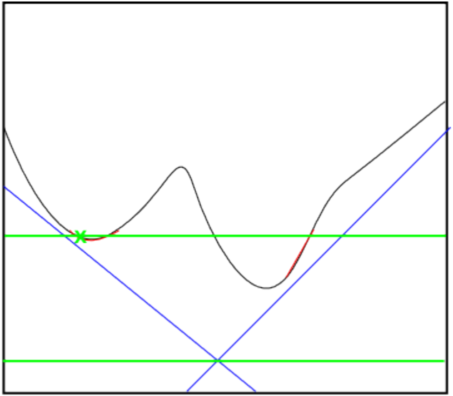


Objectif

Illustration

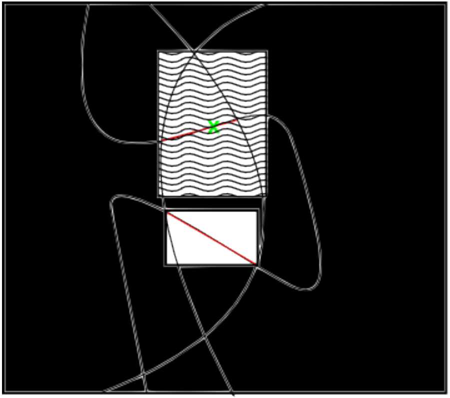


Contraintes

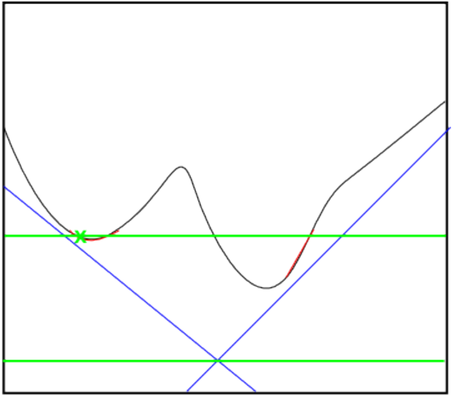


Objectif

Illustration

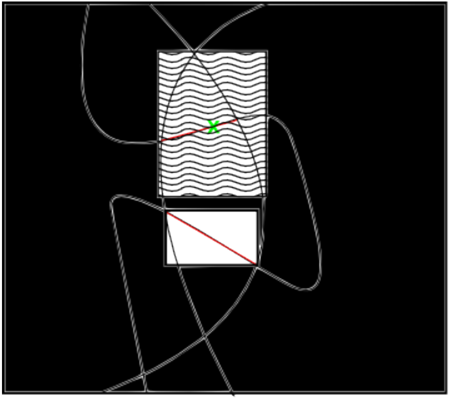


Contraintes

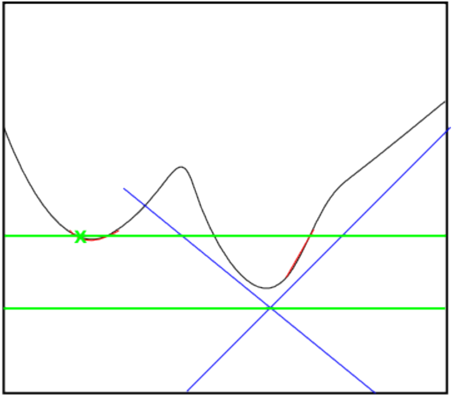


Objectif

Illustration

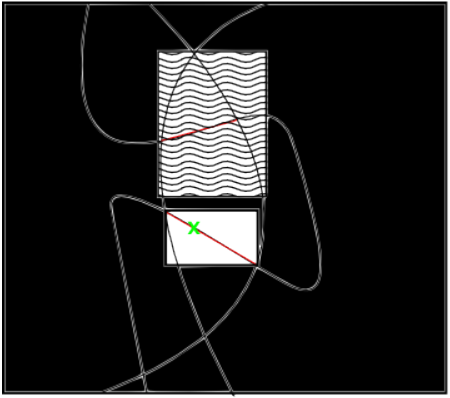


Contraintes

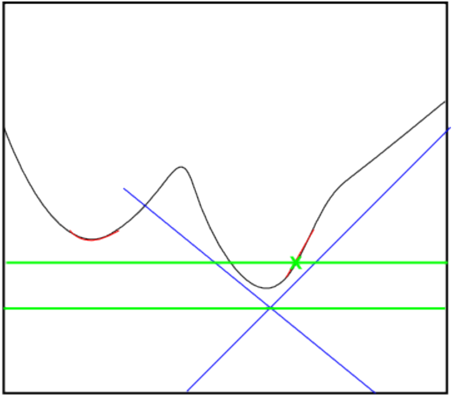


Objectif

Illustration

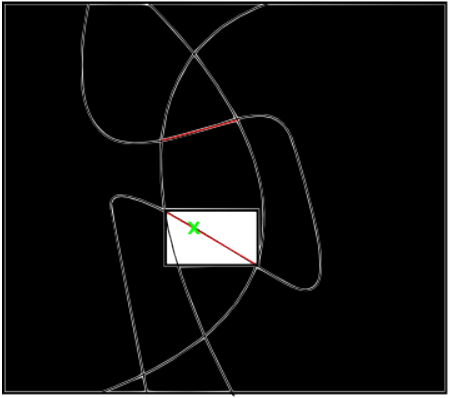


Contraintes

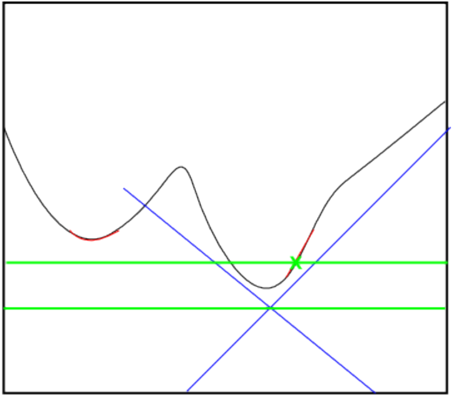


Objectif

Illustration

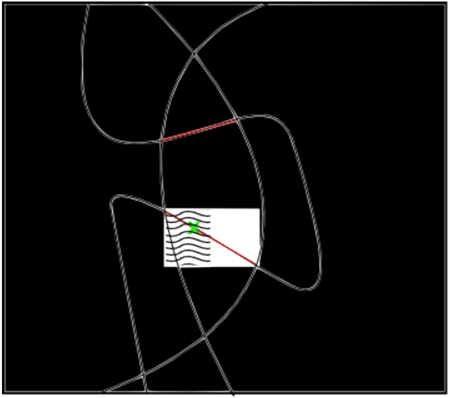


Contraintes

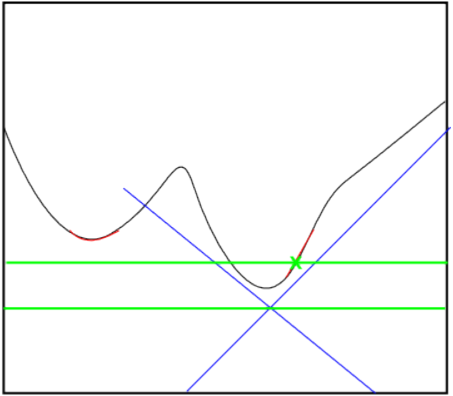


Objectif

Illustration

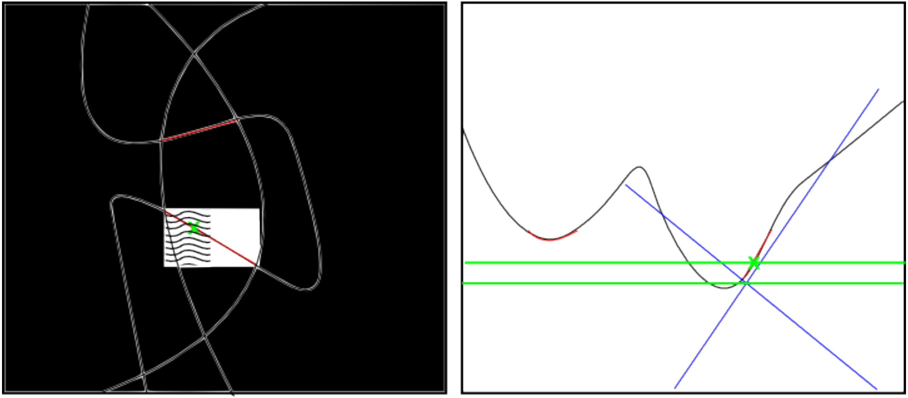


Contraintes



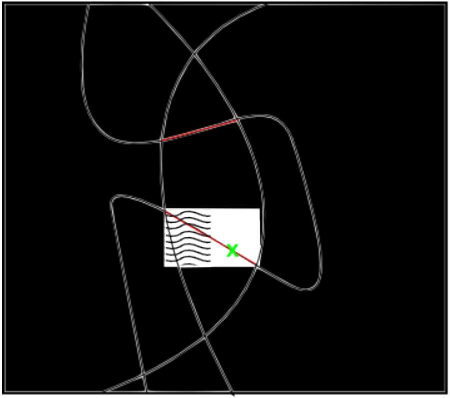
Objectif

Illustration

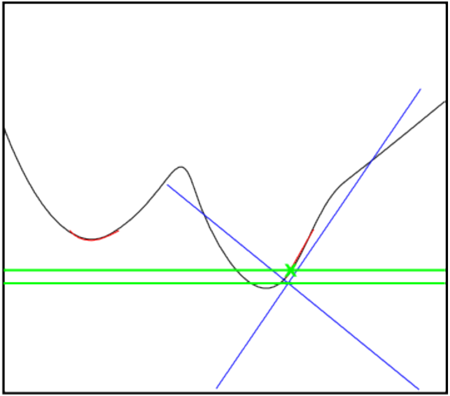


Contraintes   Objectif

Illustration

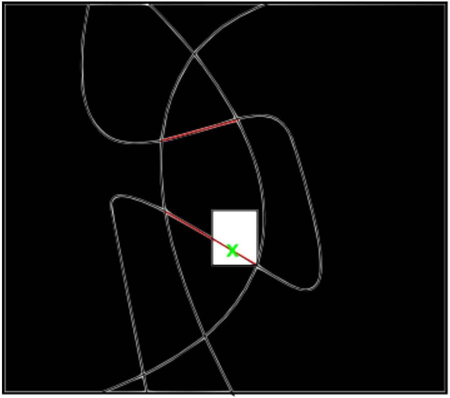


Contraintes

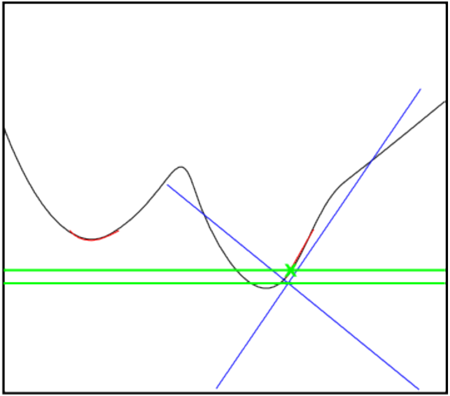


Objectif

Illustration

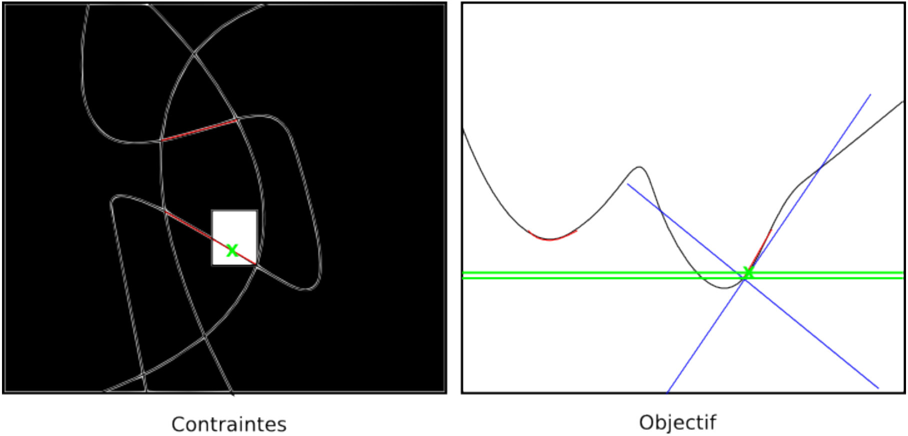


Contraintes

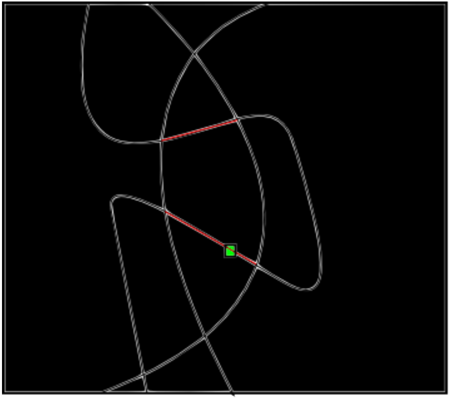


Objectif

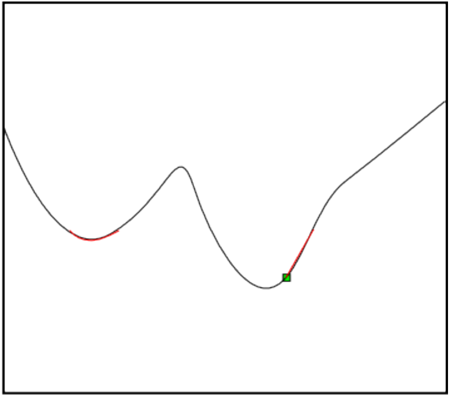
Illustration



Illustration



Contraintes



Objectif

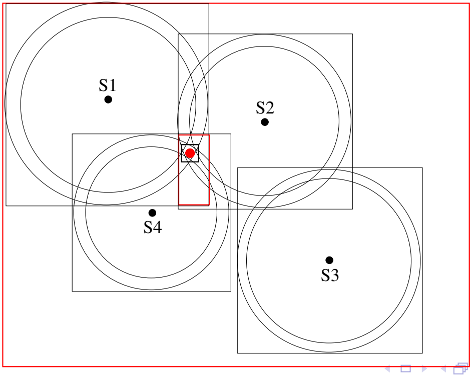
Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Exemple du GPS – –

De nombreux problèmes d'estimation non linéaire de paramètres peuvent être résolus par une approche CP (contracteurs) due à Jaulin :

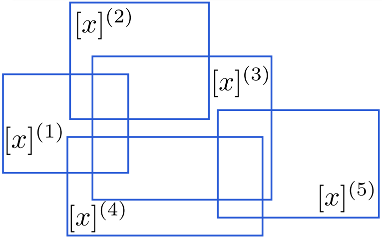
$FixedPoint(QInter(box, 60\%, MeasureContractor_1, ..., MeasureContractor_p))$



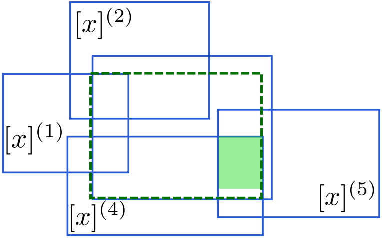
Q-intersection

Définition

La **q-intersection** d'un ensemble de boîtes S est la plus petite boîte incluant tous les points qui appartiennent à au moins q boîtes de S.



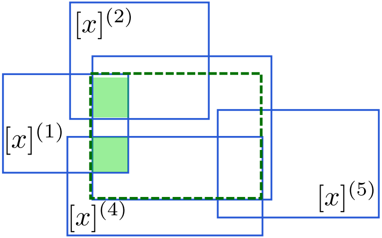
Boîtes (S)



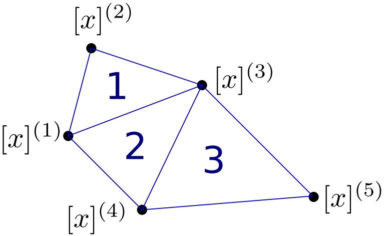
3-intersection

Q-intersection

La q-intersection peut se calculer à partir du **graphe d'intersection**.



Boîtes



Graphe d'intersection

Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Domaines d'application

- Chimie : exemple : calculer les états stables et instables des processus de distillation $\Rightarrow$ calculer toutes les solutions d'un gros système de contraintes non-linéaires.
- Robotique mobile : SLAM, meutes de drones, robot voilier.
- Estimation de paramètres : étalonnage, localisation (système GPS du futur proche, robot mobile), reconstruction 3D
- Preuve de propriété : exemple en conception de robot : prouver qu'un robot ne casse pas quand son organe terminal atteint **n'importe quel point** de son espace de travail.

  $\Rightarrow$ prouver que $det(J(x)) = 0$ n'a pas de solution sur tout l'espace.

Le point fort des méthodes à intervalles réside dans leur capacité à prendre en compte des incertitudes bornées dans les modèles (ex: erreurs de mesure), les erreurs de calcul sur les flottants et à manipuler des **ensembles infinis** de points (calcul ensembliste).

Exemple de NCSP (AOL-legentil.bch from COPRIN benchmark)

```
Variables

x in [1e-10, pi/2-1e-10];
y in [0, pi/2-1e-10];
z in [-1e8, 1e8];

Constraints

10/3*cos(x)/sin(x)^2+4*(1+tan(x)^2)/cos(y)
  +z*(-50/3*sin(y)*cos(x)/(sin(x)^2*(3.5-5*sin(y)))
  -10/3*cos(x)/sin(x)^2-4*(1+tan(x)^2)/cos(y))=0;

4*tan(x)*sin(y)/cos(y)^2+z*(50/3*cos(y)/(sin(x)
  *(3.5-5*sin(y)))+250/3*sin(y)*cos(y)/(sin(x)
  *(3.5-5*sin(y))^2)-4*tan(x)*sin(y)/cos(y)^2)=0;

50/3*sin(y)/(sin(x)*(3.5-5*sin(y)))+20+10/3/sin(x)
  -4*tan(x)/cos(y)=0;
```

Exemple de problème d'optimisation sous contraintes (Coconut: ex_6_2_9)

```
Variables
  x2, x3, x4, x5 in [1e-7,0.5];

Minimize      (31.4830434782609*x2 + 6*x4)*log(4.8274*x2 + 0.92*x4) -
              1.36551138119385*x2 + 2.8555953099828*x4 + 11.5030434782609*x2*
              log(x2/(4.8274*x2 + 0.92*x4)) + 20.98*x2*log(x2/(4.196*x2 + 1.4*
              x4)) + 7*x4*log(x4/(4.196*x2 + 1.4*x4)) + (4.196*x2 + 1.4*x4)*
              log(4.196*x2 + 1.4*x4) + 1.62*x2*log(x2/(7.52678200680961*x2 +
              0.443737968424621*x4)) + 0.848*x2*log(x2/(7.52678200680961*x2 +
              0.443737968424621*x4)) + 1.728*x2*log(x2/(1.82245052351472*x2 +
              1.4300083598626*x4)) + 1.4*x4*log(x4/(0.504772348000588*x2 + 1.4*
              x4)) + (31.4830434782609*x3 + 6*x5)*log(4.8274*x3 + 0.92*x5) -
              1.36551138119385*x3 + 2.8555953099828*x5 + 11.5030434782609*x3*
              log(x3/(4.8274*x3 + 0.92*x5)) + 20.98*x3*log(x3/(4.196*x3 + 1.4*
              x5)) + 7*x5*log(x5/(4.196*x3 + 1.4*x5)) + (4.196*x3 + 1.4*x5)*
              log(4.196*x3 + 1.4*x5) + 1.62*x3*log(x3/(7.52678200680961*x3 +
              0.443737968424621*x5)) + 0.848*x3*log(x3/(7.52678200680961*x3 +
              0.443737968424621*x5)) + 1.728*x3*log(x3/(1.82245052351472*x3 +
              1.4300083598626*x5)) + 1.4*x5*log(x5/(0.504772348000588*x3 + 1.4*
              x5)) - 35.6790434782609*x2*log(x2) - 7.4*x4*log(x4) -
              35.6790434782609*x3*log(x3) - 7.4*x5*log(x5);

Subject to    x2 + x3 = 0.5;    x4 + x5 = 0.5;
```

Exemple de problème d'optimisation sous contraintes (Coconut: ex_7_2_3)

```
Variables
  x1                  in [100,10000];
  x2, x3              in [1000,10000];
  x4, x5, x6, x7, x8 in [10,1000];

Minimize x1 + x2 + x3;

Subject to

 833.33252*x4/x1/x6 + 100/x6 - 83333.333/(x1*x6) <= 1;
 1250*x5/x2/x7 + x4/x7 - 1250*x4/x2/x7 <= 1;
 1250000/(x3*x8) + x5/x8 - 2500*x5/x3/x8 <= 1;

 0.0025*x4 + 0.0025*x6 <= 1;
 -0.0025*x4 + 0.0025*x5 + 0.0025*x7 <= 1;
 -0.01*x5 + 0.01*x8 <= 1;
```

Différentes communautés travaillant sur les intervalles

Plusieurs centaines de chercheurs s'intéressent aux intervalles, mais une minorité se préoccupe des systèmes non convexes. De plus, ils sont éparpillés dans différentes communautés scientifiques.

- opérateurs sur les flottants et arithmétique d'intervalles ;
- analyse par intervalles (analyse numérique) ;
- optimisation globale robuste (programmation mathématique, recherche opérationnelle) ;
- programmation par contraintes (informatique, RO) ;
- “néo applicatifs” : chercheurs et ingénieurs appliquant les méthodes à intervalles dans divers domaines comme la robotique, l'automatique (robuste), le traitement du signal, chimie, etc ;

  Communauté émergente !

Bibliographie

- Ramon E. Moore, R. Baker Kearfott, Michael J. Cloud, *"Introduction to Interval Analysis"*, SIAM, 2009
- Eldon Hansen, *"Global Optimization using Interval Analysis"*,
- Arnold Neumaier, *"Interval Methods for Systems of Equations"*, Cambridge University Press, 1990
- Pascal Van Hentenryck, Laurent Michel, Yves Deville, *"Numerica: A Modeling Language for Global Optimization"*, MIT Press, 1997.
- Frédéric Benhamou, Laurent Granvilliers, *"Continuous and Interval Constraints"*, chapitre 16 du livre *"Handbook of Constraint Programming"*, Elsevier, 2006.
- **Luc Jaulin, Michel Kieffer, Olivier Didrit, Eric Walter, "Applied Interval Analysis", Springer, 2001.**

Outils logiciels rigoureux (optim globale et/ou résolution)

Nom	Création	Communauté	Langage	Problème
GlobSol	années 1990	analyse par int.	Fortran	Optim
Numerica	années 1990	PPC	Numerica	Rés, Optim
RealPaver	années 2000	PPC	RealPaver, C++	Rés. (optim)
Icos	années 2000	PPC	C++	Optim
IBBA	années 2000	Optim globale	Fortran	Optim
Alias	années 2000	analyse par int.	C++, Maple	Rés. (optim)
GloptLab	années 2010	analyse par int.	plateforme	Rés. (optim)
Ibex	2007	PPC	C++	Rés., optim

Plus d'outils commerciaux ; peu de Matlab ; pas de large distribution

Ibex (Interval-Based EXplorer) : bibliothèque en C++ libre :
`www.emn.fr/z-info/ibex/`

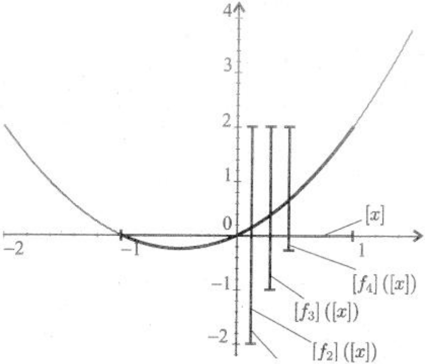
Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Non optimalité du calcul de l'image

Rappel: Extension **naturelle** $[f]_N$ d'une fonction f **composée** : l'arithmétique des intervalles est utilisée en chaque opérateur.

Exemple : $f_1(x) = x(x+1)$, $f_2(x) = x\,.\,x + x$, $f_3(x) = x^2 + x$, $f_4(x) = (x+\frac{1}{2})^2 - \frac{1}{4}$ évaluées naturellement en $[x] = [-1, +1]$.



Causes de non optimalité du calcul de l'image

Toutes les extensions aux intervalles des fonctions produisent en général une **surestimation** de l'image. Calculer l'image optimale d'un système polynomial (sur les rationnels) est un problème NP-difficile. Trois causes de non optimalité :

- les **arrondis** liés aux calculs sur les flottants ;
- le manque de **continuité** des fonctions. Exemple :
  $f(x) = (\frac{1}{x})^2$, avec $[x] = [-1, 1]$
  $\frac{1}{[-1,1]} = Hull([-\infty, -1] \cup [1, +\infty]) = [-\infty, +\infty]$
  D'où : $(\frac{1}{[-1,1]})^2 = [0, +\infty]$
  Or, l'image optimale $[1, +\infty]$ peut s'obtenir en faisant un point de choix sur $[-\infty, -1]$ et $[1, +\infty]$.
- les variables qui apparaissent **plusieurs fois** dans l'expression.

Problème des occurrences multiples de variables

Exemple : $[x] - [x] = [\underline{x} - \overline{x}, \overline{x} - \underline{x}] \supset [0,0] \neq [0,0]$

$(\mathbb{IR}, +)$ et $(\mathbb{IR} \setminus \{[0,0]\}, \ . \ )$ ne sont pas des groupes !

Sous-distributivité : $[x] . ([y] + [z]) \subset [x] . [y] + [x] . [z]$

Remarque : Si f est monotone par rapport à x_i dans une boîte donnée, alors le problème de dépendance lié à cette variable disparaît... comme nous allons le voir illico !

Autres extensions aux intervalles d'une fonction

Evaluations sur intervalles basées sur des calculs de **dérivées**.

Dérivées pour approximer les fonctions (Taylor)

- Extension de Taylor (ordre 1)
- Extension de Hansen
- ...

Dérivées pour détecter les monotonies

- Extension basée sur la monotonie :
  Chaque dérivée partielle $\frac{\partial f}{\partial x_i}$ permet de savoir si f est monotone par rapport à x_i dans la boîte courante...
- Extension par regroupement d'occurrences

Extension de Taylor [Moore 1966]

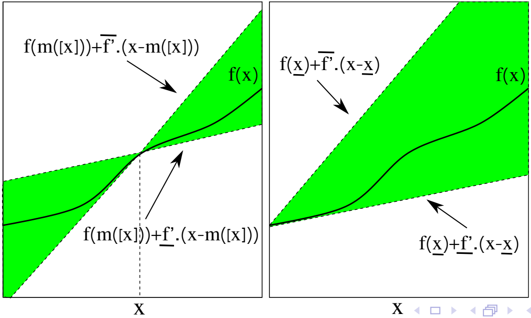
Extension de Taylor/centrée à l'ordre 1 $[f]_T$ de f

$$[f]_T([x]) = f(m([x])) + \sum_{i=1}^{n} \left(\left[\frac{\partial f}{\partial x_i} \right] ([x]) \,.\, ([x_i] - m([x_i])) \right)$$

- $f : \mathbb{R}^n \to \mathbb{R}; x = (x_1, ..., x_n)$
- L'extension de Taylor limite la surestimation quand $w([x])$ est petit (< 1) ou quand les largeurs des dérivées partielles sont proches de 0.
  Elle est sinon incomparable avec l'extension naturelle.

Illustration de l'extension de Taylor

- Considérons en dimension 1, la fonction $[f_T] : \mathbb{R} \to \mathbb{IR}$: $[f_T](x) = f(x_0) + [f']([x]) \, . \, (x - x_0)$, avec $x_0 = m([x])$.
- $[f_T]$ est affine en x et a une pente appartenant à $[f']_N([x])$.
- Le graphe de $[f_T](x)$ est un cône, un papillon de centre $(x_0, f(x_0))$. Cf. figure de gauche.



Petit exercice

Considérons $f(x_1, x_2) = 3x_1^2 + x_2^2 + x_1 x_2$ dans la boîte $[x] = [-1, 3] \times [-1, 5]$.

- Quelle est l'évaluation (naturelle) de $[f]_N$?
- Quelle est l'évaluation (de Taylor) de $[f]_T$?

Extension de monotonie : exemple

1. $f(x) = x^3 - 3x^2 + x$ à évaluer dans $[x] = [3, 4]$.
2. $f'(x) = 3x^2 - 6x + 1$ d'où $[f']_N([3, 4]) = [4, 31]$.
3. Comme $0 \notin [4, 31]$, f est monotone croissante par rapport x sur $[x] = [3, 4]$.
4. L'image calculée par monotonie $[f]_M([3, 4]) = [f(3), f(4)] = [3, 20]$.

Extension de monotonie

Définition

- $f : \mathbb{R}^{n+p} \to \mathbb{R}$; $v = (x_1, \ldots, x_n, w_1, \ldots, w_p)$
- x est l'ensemble des *variables monotones* dans $[v]$.
- Si x_i est croissante, alors $x_i^- = \underline{x_i}$ et $x_i^+ = \overline{x_i}$.
  Si x_i est décroissante, alors $x_i^- = \overline{x_i}$ et $x_i^+ = \underline{x_i}$.
- $f_{min}(w) = f(x_1^-, \ldots, x_n^-, w)$
  $f_{max}(w) = f(x_1^+, \ldots, x_n^+, w)$
- $[f]_M([v]) \equiv \left[\underline{[f_{min}]_N([w])}, \overline{[f_{max}]_N([w])}\right]$

Propriétés

- $[f]_{OPT}([v]) \subseteq [f]_M([v]) \subseteq [f]_N([v])$
- Si $w = \emptyset$, alors $[f]_M([v]) = [f]_{OPT}([v])$

Conclusion sur les extensions peu coûteuses

Deux grandes catégories :

- $[f]_{OG}([x]) \subseteq [f]_M([x]) \subseteq [f]_N([x])$
- $[f]_H([x]) \subseteq [f]_T([x])$

Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Principales heuristiques de bissection

Dans l'arbre de recherche, le choix de la prochaine variable à bissecter est très important. Trois heuristiques sont très employées :

- à **tour de rôle** ! (ce qui souligne l'importance de n'oublier aucune variable) ;
- le **plus large intervalle** d'abord (ce qui souligne encore l'importance de n'oublier personne) ;
- la fonction **smear** [Kearfott 1990] :
  - pour chaque couple (f_j, x_i), dans la boîte $[x]$ courante : calculer $smear(f_j, x_i) = |[\frac{\partial f_j}{\partial x_i}]_N([x])| \,.\, w([x])$;
  - pour une var x_i impliquée dans différentes contraintes f_j : $smear(x_i) = \sum_j (smear(f_j, x_i))$ (ou $Max_j(smear(f_j, x_i)))$;
  - bissecter la variable x_i de plus grand “impact” : $smear(x_i)$.

Deux grands types de contracteurs

Propagation de contraintes

La boîte est contractée vis à vis de chaque contrainte **individuellement** (procédure *revise*) :

- `HC4-Revise` ;
- `Box-Revise` (`BoxNarrow`) ;
- `Mohc-Revise`

L'intersection entre boîtes individuelles est mise en œuvre incrémentalement par une procédure de **propagation de contraintes** de type AC3.

Contracteurs plus puissants

Algorithme de 3B-consistance (consistance forte)

Plan

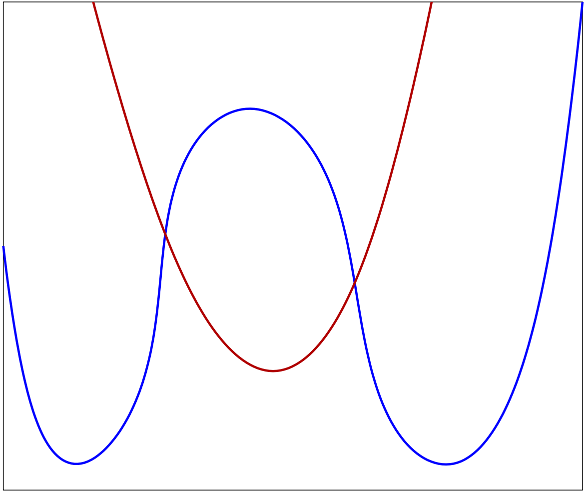
1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Propagation de contraintes

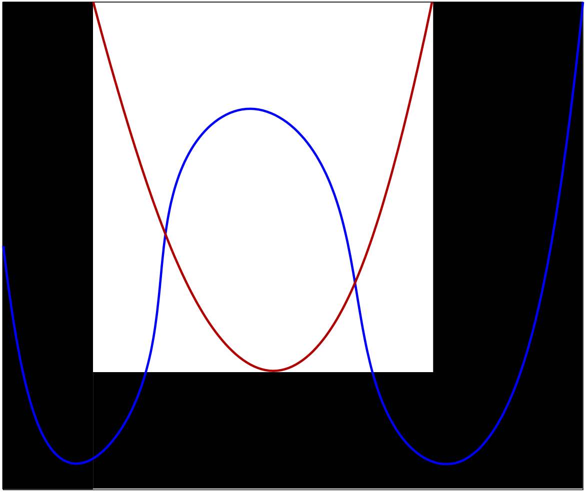
Un système $(c, x, [x])$ est **Hull-cohérent** si la boîte $[x]$ est optimale pour chaque contrainte de $c_j \in c$ prise individuellement.
But des algorithmes de **propagation de contraintes** : contracter la boîte courante vis à vis d'une seule contrainte, à tour de rôle.
Principe (AC3) : Au départ, on empile toutes les contraintes dans une *queue de propagation Q*.
Jusqu'à ce que Q devienne vide :

1. sélection dans Q d'une contrainte c_j (et élimination) ;
2. application sur c_j d'une procédure de contraction/révision qui réduit (optimalement ?) les intervalles des variables x_i de c ;
3. propagation : $\forall x_i \in x$, si $[x_i]$ est **suffisamment** réduit, on ajoute dans Q chaque contrainte c_j' impliquant x_i $(c_j' \neq c_j)$ (si c_j' ne se trouve pas déjà dans Q).

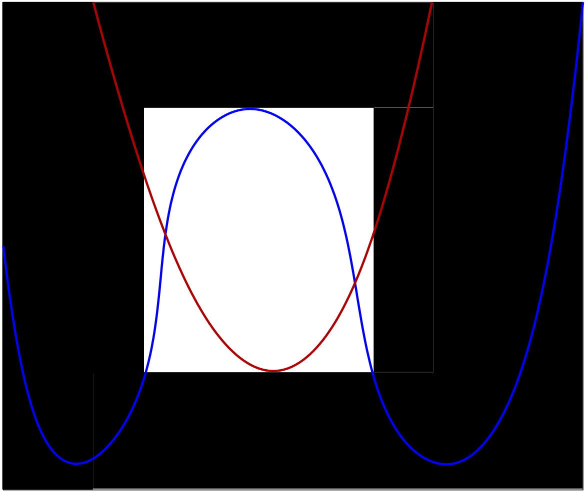
Exemple de propagation de contraintes



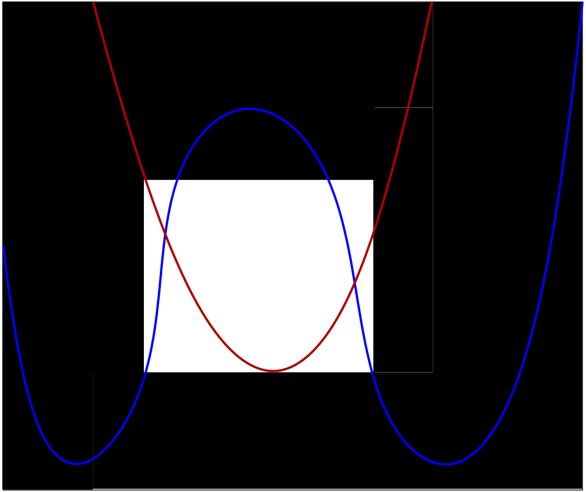
Exemple de propagation de contraintes



Exemple de propagation de contraintes



Exemple de propagation de contraintes



Algorithme `HC4-Revise`

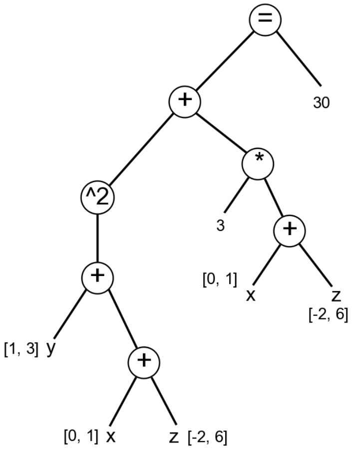
L'algorithme `HC4-Revise` contracte la boîte courante vis à vis d'une contrainte c_j.

Principe de `HC4-Revise` : contracter chaque **occurrence** de variable en l'isolant dans c_j :
évaluer les **fonctions de projection** ainsi obtenus et intersecter avec la boîte courante.

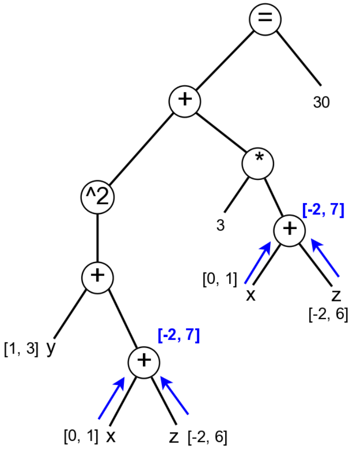
Exemple : application à $(x+y+z)^2 + 3(x+z) = 30$

- $[y] \leftarrow [y] \cap (Hull(-\sqrt{30-3([x]+[z])}\,,\ +\sqrt{30-3([x]+[z])}\,) - [x] - [z])$
- $[z] \leftarrow [z] \cap (Hull(-\sqrt{30-3([x]+[z])}\,,\ +\sqrt{30-3([x]+[z])}\,) - [x] - [y])$
- $[z] \leftarrow [z] \cap (\frac{1}{3}(30 - ([x]+[y]+[z])^2)) - [x])$
- ...

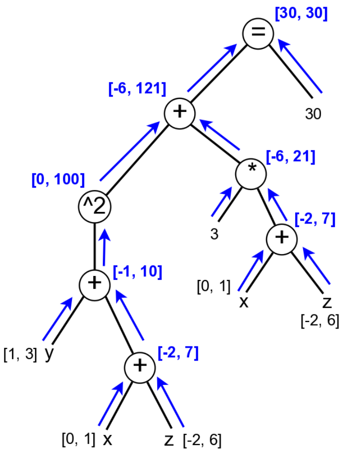
`HC4-Revise` appliqué à $(x+y+z)^2+3(x+z)=30$



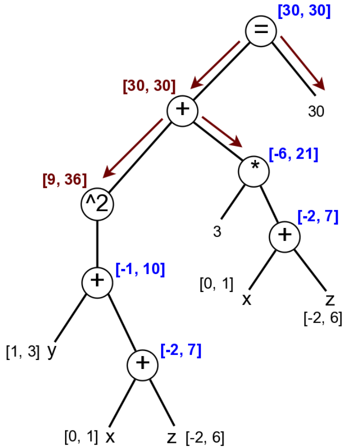
HC4-Revise appliqué à $(x + y + z)^2 + 3(x + z) = 30$



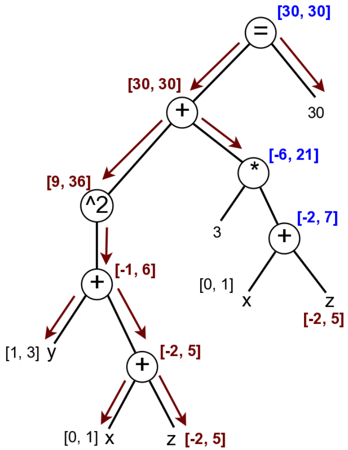
`HC4-Revise` appliqué à $(x+y+z)^2+3(x+z)=30$



`HC4-Revise` appliqué à $(x + y + z)^2 + 3(x + z) = 30$



`HC4-Revise` appliqué à $(x + y + z)^2 + 3(x + z) = 30$

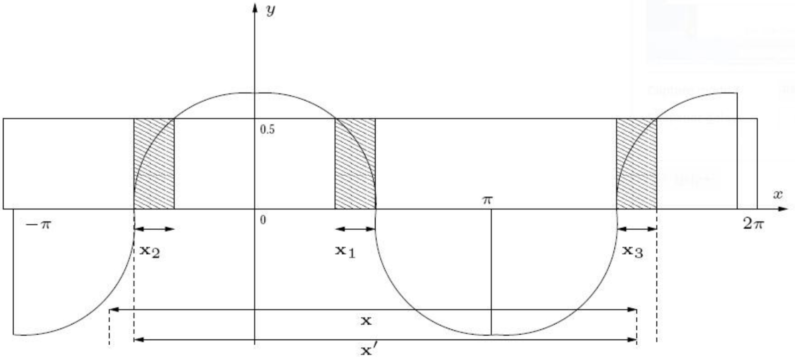


Fonctions de projection élémentaires

- Toute contrainte s'obtient par composition de fonctions **élémentaires**.
- Pour les contraintes élémentaires, on sait calculer la boîte optimale en utilisant les **fonctions de projection élémentaires** (appelées aussi **fonctions inverses**).
- Contraintes élémentaires :

$y = x$	$y = \exp(x)$	$y = \cos(x)$	$y = \arccos(x)$
$y = x + z$	$y = \ln(x)$	$y = \sin(x)$	$y = \arcsin(x)$
$y = x - z$	$y = \log(x)$	$y = \tan(x)$	$y = \arctan(x)$
$y = x \times z$	$y = \mathrm{sqr}(x)$	$y = \cosh(x)$	$y = \mathrm{arccosh}\,(x)$
$y = x/z$	$y = \sqrt{x}$	$y = \sinh(x)$	$y = \mathrm{arcsinh}\,(x)$
$y = x^a$	$y = 1/x$	$y = \tanh(x)$	$y = \mathrm{arctanh}\,(x)$

Fonctions de projection élémentaires (ex : $y = cos(x)$)



Les fonctions de projection primitives sont toutes strictement monotones par morceaux sur leur domaine de définition, et sont inversibles par morceaux.
⇒ On peut implémenter les procédures correspondantes en utilisant les bornes des opérandes, en étudiant les changements de sens de variation et en gérant les bornes flottantes.

Bilan sur la propagation de contraintes

Souci : Pour **une** contrainte polynomiale $c_j := f_j(x) = 0$, calculer la boîte optimale (Hull-cohérente) est NP-difficile.

Causes de non optimalité : arrondis sur les flottants, non continuité des fonctions et occurrences multiples de variables.

Propriété de HC4

`HC4-Revise` calcule la boîte optimale (aux arrondis près) quand f_j est continue et ne contient **aucune** occurrence multiple de variables.

Plan

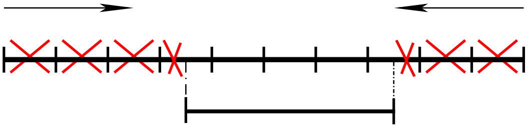
1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Algorithme `3B`

Consistance : L'intervalle $[x_i]$ est 3B-consistant (c'est-à-dire, ne peut être plus réduit) si on ne peut pas éliminer $\underline{x_i}$ ni $\overline{x_i}$ par un algorithme de 2B-consistance (`SubContractor` : `HC4`, `Mohc`).

`3B` (**in-out** $P(x, c, [x])$; **in** `SubContractor`, ϵ_{outer}, ϵ_{inner})

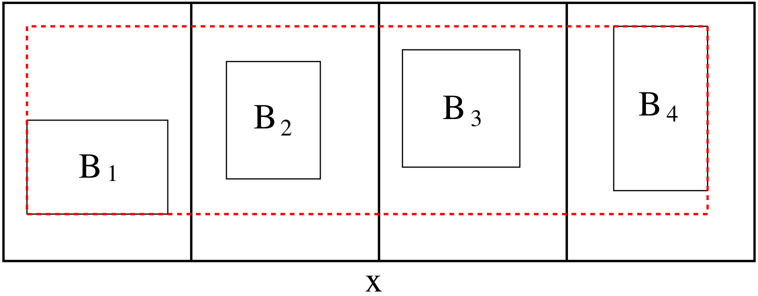
repeat
  for all variable $x_i \in x$ **do**
    `Var3B` (x_i, P, `SubContractor`, ϵ_{inner})
  end for
until aucun intervalle n'a été réduit de plus de ϵ_{outer}



Algorithme `CID`

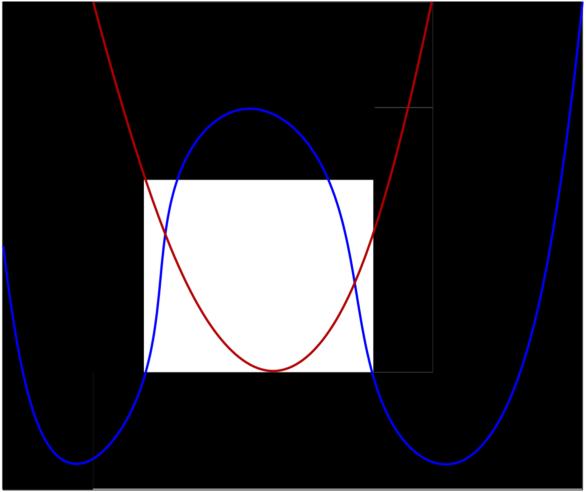
Plusieurs améliorations à cet algorithme, dont :

- la **disjonction constructive sur intervalles** (CID, 3BCID) :

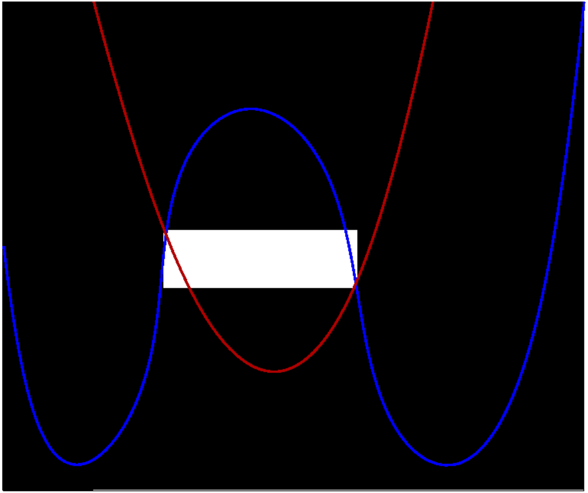


- ne pas travailler sur toutes les variables : **Adaptive CID**
  $\Rightarrow$ efficace en satisfaction comme en optimisation.

Exemple de 3B-consistance

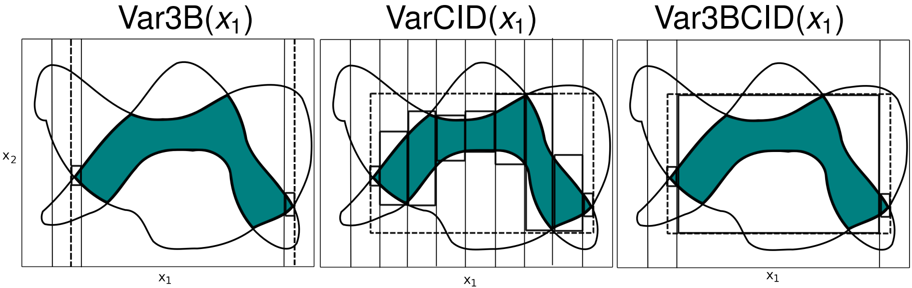


Exemple de 3B-consistance



Exemple avec 3B, CID, 3BCID

Exemple avec deux contraintes d'inégalité, deux variables (x_1 et x_2).
Découpage de $[x_1]$ en (au plus) 10 sous-intervalles par la procédure `Var3B`.



Plan

1. Introduction
   - Définitions, notations
   - Extension d'une fonction aux intervalles
   - Schémas de résolution des systèmes de contraintes et d'optimisation globale
   - Estimation de paramètres ensembliste
   - Domaines d'application, bibliographie, outils
2. Extensions de fonctions aux intervalles
3. Algorithmes de bissection et contraction
   - Heuristiques de branchement
   - Algorithme HC4 de propagation de contraintes
   - Algorithmes de consistance forte
   - Autres algorithmes de contraction

Opérateurs de l'analyse par intervalles

Adaptation aux intervalles des algos d'analyse numérique (Newton).

Principe

- On part d'une boîte initiale $[x]$.
- On calcule jusqu'à point-fixe : $[x] \leftarrow [x] \cap N([x])$.
- Si $N([x]) \subset [x]$, alors garantie de solution unique dans $[x]$ et convergence quadratique vers cette solution.

En pratique

- Quand le système contient plusieurs solutions, la matrice "jacobienne" contient une singularité et Newton intervalles échoue.
- Les opérateurs d'analyse par intervalles sont donc généralement efficaces quand les boîtes sont petites.

Opérateurs de l'analyse par intervalles

A retenir

- Algo numérique itératif classique : part d'un point faux et converge vers un point (parfois) faux.
- Algo d'analyse par intervalles : part d'une boîte initiale et converge vers une boîte (parfois strictement) contractée contenant nécessairement la solution.

Opérateurs de relaxation linéaire/polyédrale

Principe

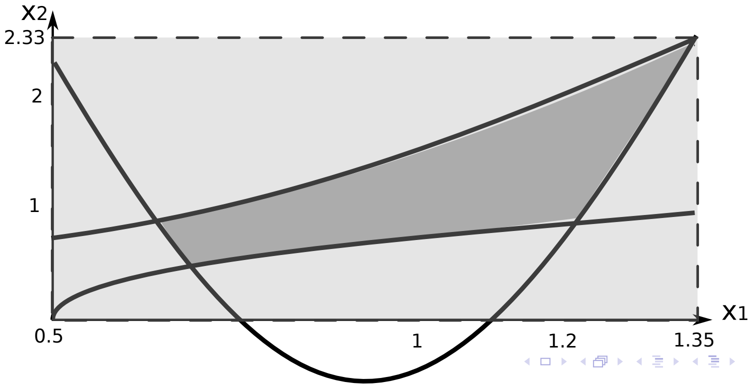
- Approximer les contraintes non-convexes par un ensemble d'inégalités linéaires englobantes traitées généralement par un algorithme de programmation linéaire (simplexe, point intérieur)
- Le système convexifié contient plus de solutions que le système non-linéaire initial.

Relaxation rigoureuse

- La linéarisation doit être conservative (arrondis sur des nombres flottants) pour ne pas perdre de solution.
- Les algorithmes du simplexe disponibles sont non rigoureux. Correction par un post-traitement [Neumaier, Shcherbina 2004].

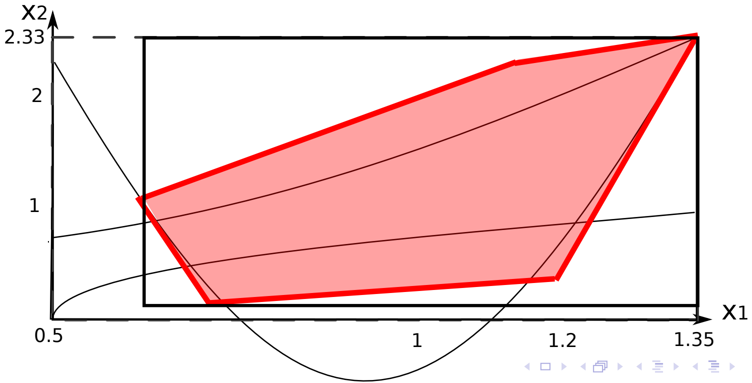
Exemple (2 hyperplans par inégalité)

$x_2 \geq x_1^2 + 0.5$
$x_2 \leq 2.5\, sin(4x_1 + 1) + 2$ Domaine $= [0.5, 1.35] \times [0, 2.33]$
$x_2 \geq sqrt(x_1 - 0.5)$



Exemple (2 hyperplans par inégalité)

$x_2 \geq x_1^2 + 0.5$
$x_2 \leq 2.5\, sin(4x_1 + 1) + 2$ Domaine $= [0.5, 1.35] \times [0, 2.33]$
$x_2 \geq sqrt(x_1 - 0.5)$



Conclusion : points forts des intervalles

Intervalles et non-convexité

- Technique de choix pour les problèmes non convexes.
- Des applications phares à venir, notamment liées à l'optimisation non convexe et à l'estimation de paramètres : étalonnage, géolocalisation (système GPS du futur proche, robot mobile), reconstruction 3D.

Intervalles et calcul ensembliste

Preuves de propriétés sur des ensembles infinis de points (calcul ensembliste). Ex: le robot ne casse pas...

Intervalles et erreurs

- Prise en compte des incertitudes bornées dans les modèles (ex: erreurs de mesure).
- Prise en compte des erreurs de calcul sur les flottants.
- Prise en compte des mesures aberrantes (*outliers*).