Computer-Assisted Proofs
Introduction to Interval Arithmetic

Cours de recherche master informatique

27 November 2025

Agenda

Taylor expansion of order 1

Mean value theorem:

$$\text{for } f \quad : \quad \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R}^n \\ x & \mapsto & f(x) \end{array}$$

for x and y, there exists $\xi_y \in (x, y)$ (or (y, x)) such that

$$f(y) = f(x) + \nabla f(\xi_y) \cdot (y - x).$$

We deduced that for $x \in \boldsymbol{x}$ and $y \in \boldsymbol{x}$,

$$f(y) \in f(x) + \nabla f(\boldsymbol{x}) \cdot (y - x)$$

and then

$$f(\boldsymbol{x}) \subset f(x) + \nabla f(\boldsymbol{x}) \cdot (\boldsymbol{x} - x).$$

Similarly, in several dimensions: $f : \mathbb{R}^m \to \mathbb{R}^n$

For x and y, let us consider $g : t \mapsto (1-t)x + ty$.
We have $h(t) = f(g(t))$ is a function from $\mathbb{R}$ to $\mathbb{R}^n$ such that $h(0) = f(x)$ and $h(1) = f(y)$.
For $t \in [0,1]$, there exists $\xi_t \in (0,t)$ such that

$$h(t) = h(0) + \nabla h(\xi_t)(t-0), \text{ which can be written as}$$

$$f(g(t)) = f(x) + \text{Jac}(f(g(\xi_t))) \cdot \nabla g(\xi_t).t = f(x) + t\text{Jac}(f(g(\xi_t))) \cdot (y-x)$$

thus, with $t = 1$,

$$f(y) = f(x) + \text{Jac}(f(g(\xi_1))) \cdot \nabla g(\xi_1) = f(x) + \text{Jac}(f(g(\xi_1))) \cdot (y-x)$$

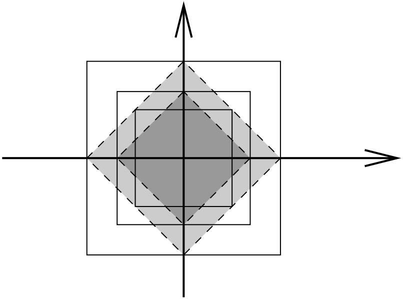
that is, for $x \in \boldsymbol{x}$ and $y \in \boldsymbol{x}$,

$$f(y) \in f(x) + \text{Jac}f(\boldsymbol{x}) \cdot (y-x)$$

and then, for $\tilde{x} \in \boldsymbol{x}$,

$$f(\boldsymbol{x}) \subset f(\tilde{x}) + \text{Jac}f(\boldsymbol{x}) \cdot (\boldsymbol{x} - \tilde{x}).$$

Matrix-vector product: wrapping effect!



2 successive rotations of $\pi/4$
of the little central square

Ubiquity of the Wrapping Effect (after Lohner, 2001)

Where does the Wrapping Effect appear?

- matrix-vector iterations: $\boldsymbol{x}_{n+1} = A_n \boldsymbol{x}_n + b_n$, $\boldsymbol{x}_0 \in I\!R^n$;
- discrete dynamical systems: $\boldsymbol{x}_{n+1} = f(\boldsymbol{x}_n)$, $\boldsymbol{x}_0$ given and f sufficiently smooth;
- continuous dynamical systems (ODEs): $x'(t) = g(t, x(t))$, $x(0) = x_0$, which is studied through a numerical one step method (or more) of the kind $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + h\Phi(\boldsymbol{x}_n, t_n) + \boldsymbol{z}_{n+1}$;
- difference equations: $a_0 \boldsymbol{z}_n + a_1 \boldsymbol{z}_{n+1} + \ldots + a_m \boldsymbol{z}_{n+m} = b_n$ with $\boldsymbol{z}_0, \ldots \boldsymbol{z}_m$ given;
- automatic differentiation.

The matrix-vector iteration is archetypal of the wrapping effect in all of these cases.

Classical approach: coordinate transformations

(after Lohner, 2001)

$$\boldsymbol{x}_{n+1} = A_n \boldsymbol{x}_n + \boldsymbol{b}_n$$

Let's assume that A and b are constant ($A_n = A$, $b_n = b$ for all n).
Well-known problem with the power method: $\boldsymbol{x}_n$ becomes aligned with the eigenvector corresponding to the largest eigenvalue (in module).

Principle: replace

$$\boldsymbol{x}_{n+1} = A\boldsymbol{x}_n + \boldsymbol{b}$$

by

$$\begin{aligned} \boldsymbol{x}_{n+1} &= B\boldsymbol{y}_{n+1} \\ \boldsymbol{y}_{n+1} &= B^{-1}AB\boldsymbol{y}_n + B^{-1}\boldsymbol{b} \end{aligned}$$

Choice of B?

better choose an orthogonal transformation.

Classical approach: *QR*-preconditioning

(after Lohner, and Nedialkov&Jackson, 2001)

Principle: Factor A as $A = QR$ with Q orthogonal : $Q^{-1} = Q^T$, and R upper triangular.

In

$$x_{n+1} = Ax_n + b$$

replace x_n by

$$\begin{cases} x_n &= Qy_n \\ \boldsymbol{y}_{n+1} &= RQ\boldsymbol{y}_n + Q^T\boldsymbol{b} \end{cases} \Leftrightarrow y_n = Q^T x_n \text{ and thus } \boldsymbol{y}_n = Q^T \boldsymbol{x}_n$$

Classical approach: QR-preconditioning

(after Lohner, and Nedialkov&Jackson, 2001)

Iteration:

$$\begin{cases} \mathbf{y}_n & = & Q^T \mathbf{x}_n \\ \mathbf{y}_{n+1} & = & RQ\mathbf{y}_n + Q^T \mathbf{b} \end{cases}$$

Theoretical results:

$$w(\mathbf{y}_n]) \leq \mathrm{cond}(Q^T P)\rho(A)^n w(\mathbf{y}_0) + \frac{\mathrm{cond}(Q^T P)\rho(A)^{n-1} - 1}{\mathrm{cond}(Q^T P)\rho(A) - 1} w(\mathbf{b}) + |Q^T| w(\mathbf{b})$$

where A diagonalizable: $A = P\Lambda P^{-1}$.

Agenda

Affine arithmetic

Comba, Stolfi and Figueiredo (1993, 2004)

Definition: each input or computed quantity x is represented by
$x = x_0 + \alpha_1\varepsilon_1 + \alpha_2\varepsilon_2 + \cdots + \alpha_n\varepsilon_n$
where x_0, α_1, $\ldots\alpha_n$ are known real / floating-point numbers,
and ε_1, ε_2 $\ldots\varepsilon_n$ are symbolic variables $\in [-1, +1]$.
Example: $x \in [3, 7]$ is represented by $x = 5 + 2\varepsilon$.

Operations:
$(x_0 + \sum_k \alpha_k\varepsilon_k) + (y_0 + \sum_k \beta_k\varepsilon_k) = (x_0 + y_0) + \sum_k(\alpha_k + \beta_k)\varepsilon_k$.
$(x_0 + \sum_k \alpha_k\varepsilon_k) \times (y_0 + \sum_k \beta_k\varepsilon_k) =$
$(x_0 \times y_0) + \sum_k(x_0\beta_k + y_0\alpha_k)\varepsilon_k + \gamma_l\varepsilon_l$ with ε_l a new variable.

Affine arithmetic: example

Comba, Stolfi and Figueiredo (1993, 2004)

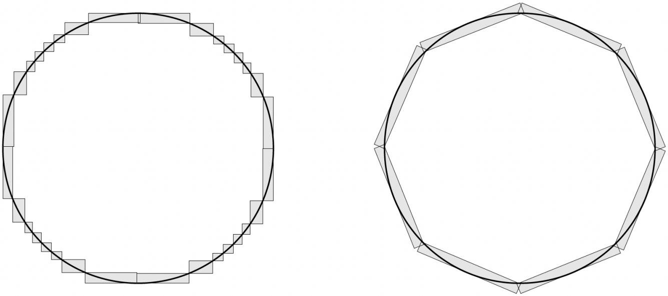


Figure 3. Rectangle approximation of a circle computed with interval arithmetic (left) and with affine arithmetic (right), using the same tolerance.

Affine arithmetic: example in 2D

for a given function f of two variables x and y,
if $x = x_0 + \sum_{k=1}^{n} \alpha_k \varepsilon_k$ and $y = y_0 + \sum_{k=1}^{n} \beta_k \varepsilon_k$,
we are looking for an approximation of

$$f(x,y) = f(x_0 + \sum_{k=1}^{n} \alpha_k \varepsilon_k, y_0 + \sum_{k=1}^{n} \beta_k \varepsilon_k)$$

as

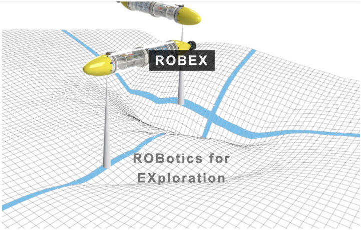
$$z_0 + \sum_{k=1}^{n} \gamma_k \varepsilon_k + \gamma_{n+1} \varepsilon_{n+1}$$

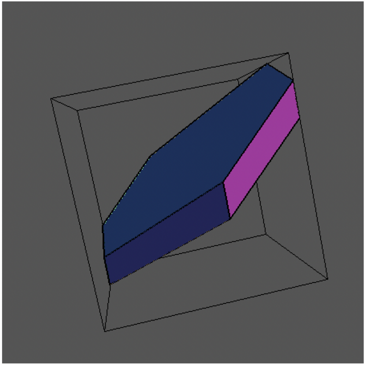
where z_0 and the γ_k are easy to compute and γ_{n+1}, which is the error, is small.

Roundoff errors: compute δ_l an upper bound of all roundoff errors and add it to γ_l.

Affine arithmetic: applications

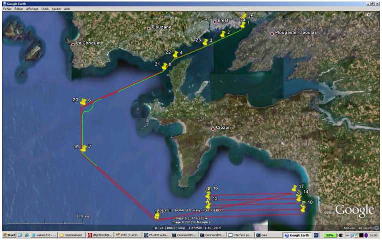
Robex team in Brest, Cosynus team in LIX





Affine arithmetic: applications

Robex team in Brest



Agenda

Taylor models

Berz, Hoefkens and Makino 1998, Nedialkov, Neher

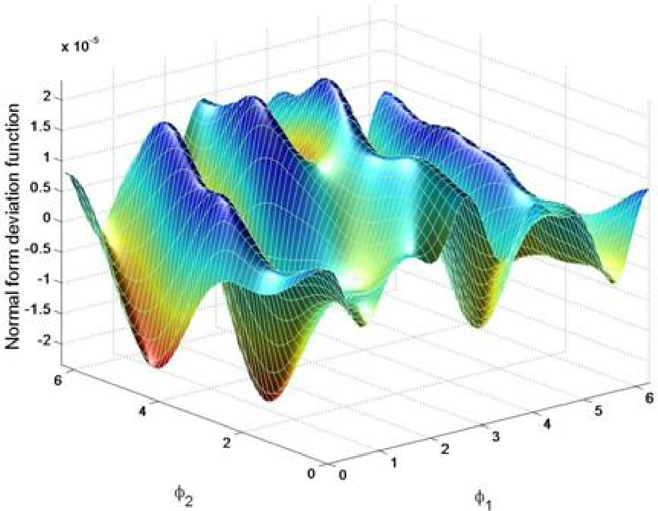
Principle: represent a function $f(x)$ for $x \in [-1, 1]$ by a polynomial part $p(x)$ of degree d and a remainder part (a big bin) $\boldsymbol{I}$ such that $\forall x \in [-1, 1],\ f(x) \in p(x) + \boldsymbol{I}$.

Operations:

- affine operations: straigthforward;
- non-affine operations: enclose the nonlinear terms of degree higher than d and add this enclosure to the remainder.

Roundoff errors: determine an upper bound b on the roundoff errors and add $[-b, b]$ to the remainder.

Taylor models: applications (HEP, beam physics)



Agenda

Modal arithmetic

Let $\boldsymbol{x} = [\underline{x}, \bar{x}]$, $\boldsymbol{y} = [\underline{y}, \bar{y}]$, $\boldsymbol{z} = [\underline{z}, \bar{z}]$,

$\boldsymbol{x} + \boldsymbol{y} = \boldsymbol{z} \Leftrightarrow$
$[\underline{x}, \bar{x}] + [\underline{y}, \bar{y}] = [\underline{x} + \underline{y}, \bar{x} + \bar{y}] = [\underline{z}, \bar{z}]$
but $\boldsymbol{z} - \boldsymbol{y} \neq \boldsymbol{x}$
$\boldsymbol{z} - \boldsymbol{y} = [\underline{z} - \bar{y}, \bar{z} - \underline{y}] \neq \boldsymbol{x}$

$\forall x \in \boldsymbol{x}, \forall y \in \boldsymbol{y}, x + y \in \boldsymbol{x} + \boldsymbol{y}$
$\forall z \in \boldsymbol{z}, \forall y \in \boldsymbol{y}, z - y \in \boldsymbol{z} - \boldsymbol{y} \supset \boldsymbol{x}$

what if $\boldsymbol{y'} = [\bar{y}, \underline{y}]$?
$\boldsymbol{z} - \boldsymbol{y'} = \boldsymbol{x}$?
$\boldsymbol{z} - \boldsymbol{y'} = [\underline{z} - \underline{y}, \bar{z} - \bar{y}] = \boldsymbol{x}$!

Meaning of $\boldsymbol{y'}$?

$\forall z \in \boldsymbol{z}, \exists y' \in \boldsymbol{y}, z - y \in \boldsymbol{z} - \boldsymbol{y'} = \boldsymbol{x}$
true as $\boldsymbol{z} = \boldsymbol{x} + \boldsymbol{y}$

In modal arithmetic, a **proper** interval is interpreted as quantified universally,
an **improper** interval is interpreted as quantified existentially.
$\forall$ **must precede** $\exists$.

Modal arithmetic

Applications: compute inner approximations.

Pros and cons:

Pro recover algebraic properties (subtraction as the inverse of addition. . .)

Con how do you interpret an improper interval as the final result of a computation?

In practice: use of some "reverse" operations when needed.

Agenda

Algorithm: solving a nonlinear system: Newton Why a specific iteration for interval computations?

Usual formula:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Direct interval transposition:

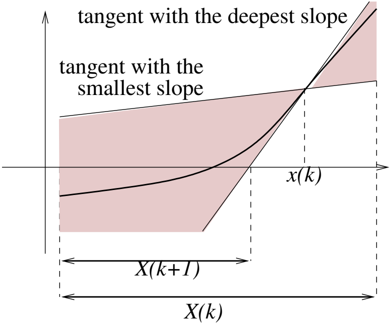
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \frac{f(\boldsymbol{x}_k)}{f'(\boldsymbol{x}_k)}$$

$$w(\boldsymbol{x}_{k+1}) = w(\boldsymbol{x}_k) + w\left(\frac{f(\boldsymbol{x}_k)}{f'(\boldsymbol{x}_k)}\right) > w(\boldsymbol{x}_k)$$

divergence!

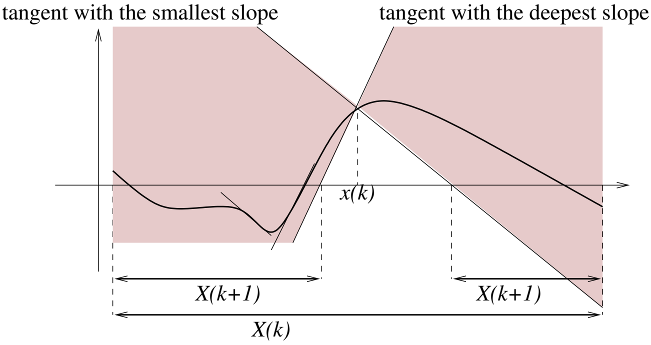
Algorithm: interval Newton principle of an iteration

(Hansen-Greenberg 83, Baker Kearfott 95-97, Mayer 95, van Hentenryck et al. 97)



$$\mathbf{x}_{k+1} := \left(x_k - \frac{\mathbf{f}(\{x_k\})}{\mathbf{f}'(\mathbf{x}_k)} \right) \bigcap \mathbf{x}_k$$

Algorithm: interval Newton principle of an iteration



$$(\boldsymbol{x}_{k+1,1}, \boldsymbol{x}_{k+1,2}) := \left(x_k - \frac{\boldsymbol{f}(\{x_k\})}{\boldsymbol{f}'(\boldsymbol{x}_k)} \right) \bigcap \boldsymbol{x}_k$$

Algorithm: interval Newton

Input: $\boldsymbol{f}$, $\boldsymbol{f'}$, $\boldsymbol{x}_0$ // $\boldsymbol{x}_0$ initial search interval
Initialization: $\mathcal{L} = \{\boldsymbol{x}_0\}$, $\alpha = 0.75$ //any value in $]0.5, 1[$ is suitable
Loop: while $\mathcal{L} \neq \emptyset$
Suppress $(\boldsymbol{x}, \mathcal{L})$
$x := \text{mid}(\boldsymbol{x})$
$(\boldsymbol{x}_1, \boldsymbol{x}_2) := \left(x - \frac{\boldsymbol{f}(\{x\})}{\boldsymbol{f'}(\boldsymbol{x})}\right) \bigcap \boldsymbol{x}$ // $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ can be empty
if $w(\boldsymbol{x}_1) > \alpha w(\boldsymbol{x})$ or $w(\boldsymbol{x}_2) > \alpha w(\boldsymbol{x})$ then $(\boldsymbol{x}_1, \boldsymbol{x}_2) := \text{bisect}(\boldsymbol{x})$
if $\boldsymbol{x}_1 \neq \emptyset$ and $\boldsymbol{f}(\boldsymbol{x}_1) \ni 0$ then
if $w(\boldsymbol{x}_1)/|\text{mid}(\boldsymbol{x}_1)| \leq \varepsilon_X$ or $w(\boldsymbol{f}(\boldsymbol{x}_1)) \leq \varepsilon_Y$ then Insert $\boldsymbol{x}_1$ in *Res*
else Insert $\boldsymbol{x}_1$ in $\mathcal{L}$
same handling of $\boldsymbol{x}_2$
Output: *Res*, a list of intervals that may contain the roots.

Algorithm: interval Newton

properties

Existence and uniqueness of a root are proven:
if there is no hole and if the new iterate (before $\bigcap$) is contained in the interior of the previous one.

Existence of a root is proven:

- using the mean value theorem:
  OK if $f(\inf(\boldsymbol{x}))$ and $f(\sup(\boldsymbol{x}))$ have opposite signs.
  (Miranda theorem in higher dimensions).
- using Brouwer theorem: if the new iterate (before $\bigcap$) in contained in the previous one.

Algorithm: interval Newton ε-inflation

Existence and uniqueness of a root are proven:

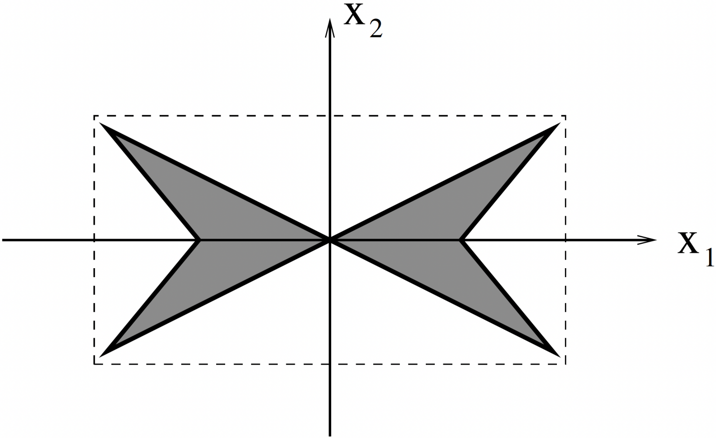
once one has a satisfying iterate $\mathbf{x}$,

1. inflate $\mathbf{x}$: $\mathbf{y} = [1-\varepsilon, 1+\varepsilon]\mathbf{x}$
2. apply one iteration step to $\mathbf{y}$, obtain $\mathbf{y'}$
3. if $\mathbf{y'} \subset \mathring{\mathbf{y}}$ then $\mathbf{y}$ contains a unique root

Agenda

Linear systems solving: example

$$\boldsymbol{A} = \begin{pmatrix} 2 & [-1, 1] \\ [-1, 1] & 2 \end{pmatrix} \text{ and } \boldsymbol{b} = \begin{pmatrix} [-2, 2] \\ 0 \end{pmatrix}$$



Algorithm: linear systems solving forget about LU factorization

Problem: solve $\boldsymbol{Ax} = \boldsymbol{b}$, i.e. determine
Hull $(\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})) =$ Hull $(\{x\ :\ \exists A \in \boldsymbol{A}, \exists b \in \boldsymbol{b}, Ax = b\})$.

Classical approach:

- compute the LU factorization of the matrix: $L \cdot U = A$ with L lower triangular with unit diagonal, U upper triangular
- solve $Ly = b$
- solve $Ux = y$

Interval approach:
forget about solving triangular linear systems,
forget about LU factorization:
amplification of the dependency problem.

Algorithm: linear systems solving forget about LU factorization

Problem: solve $\boldsymbol{Ax} = \boldsymbol{b}$, i.e. determine
Hull $(\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b})) = $ Hull $(\{x : \exists A \in \boldsymbol{A}, \exists b \in \boldsymbol{b}, Ax = b\})$.

Classical approach:

- compute the LU factorization of the matrix: $L \cdot U = A$ with L lower triangular with unit diagonal, U upper triangular
- solve $Ly = b$
- solve $Ux = y$

Interval approach:

forget about solving triangular linear systems,
forget about LU factorization:
amplification of the dependency problem.

Algorithm: linear systems solving

example (Rohn, 2005)

Solve using interval arithmetic, based on floating-point arithmetic:

$$\begin{pmatrix} 1-\varepsilon & & & \\ 1 & 1 & & \\ \vdots & & \ddots & \\ 1 & \dots & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

where $\varepsilon = 2^{-53}$.

Exact solution, exact inverse:

$$\text{solution } = \begin{pmatrix} 1/(1-\varepsilon) \\ -\varepsilon/(1-\varepsilon) \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \text{ inverse } = \begin{pmatrix} 1/(1-\varepsilon) & & & & \\ -1/(1-\varepsilon) & 1 & & & \\ & -1 & 1 & & \\ & & \ddots & \ddots & \\ & & & -1 & 1 \end{pmatrix}$$

Algorithm: linear systems solving

example (Rohn, 2005)

Solve using interval arithmetic, based on floating-point arithmetic:

$$\begin{pmatrix} 1-\varepsilon & & & \\ 1 & 1 & & \\ \vdots & & \ddots & \\ 1 & \dots & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

where $\varepsilon = 2^{-53}$: $1-\varepsilon$ and $1+2\varepsilon$ are FP but not $1+\varepsilon$.

Computed interval solution:

$$\left([1, 1+2\varepsilon], [-2\varepsilon, 0], [-2\varepsilon, 2\varepsilon], \dots, [-2^{i-2}\varepsilon, 2^{i-2}\varepsilon], \dots\right)'$$

Algorithm: linear systems solving (Hansen-Sengupta)

Problem: solve $\boldsymbol{Ax} = \boldsymbol{b}$, i.e. determine
Hull $(\Sigma_{\exists\exists}(\boldsymbol{A}, \boldsymbol{b}))$ = Hull $(\{x : \exists A \in \boldsymbol{A}, \exists b \in \boldsymbol{b}, Ax = b\})$.

Pre-processing:
multiply the system by an approximate $mid(\boldsymbol{A})^{-1}$.
New system $= mid(\boldsymbol{A})^{-1}\boldsymbol{Ax} = \boldsymbol{b}$.
Hope: contracting iteration.

Algorithm: apply Gauss-Seidel iteration

while convergence not reached loop
  for $i = 1$ to n do
$$\boldsymbol{x}_i := \left(\boldsymbol{b}_i - \sum_{j \neq i} \boldsymbol{A}_{i,j}\boldsymbol{x}_j\right) / \boldsymbol{A}_{i,i}$$

Agenda

Algorithm: certify

Let A be a matrix, b a vector, x^* such that $Ax^* = b$ and $\tilde{x}$ an approximate solution to $Ax = b$.

Problem: determine an enclosure of the error $x^* - \tilde{x}$.

Algorithm: certify

Rump (1980), Nguyen (2011)

Let A be a matrix, b a vector, x^* such that $Ax^* = b$ and $\tilde{x}$ an approximate solution to $Ax = b$.

Iterative refinement:
determine an enclosure of the error $x^* - \tilde{x}$
and refine it.

`while (not convergence)`

$\boldsymbol{r} = [b - A\tilde{x}]$

$\boldsymbol{e} \supset A^{-1}\boldsymbol{r}$

$\tilde{x} = \tilde{x} + \mathtt{mid}(\boldsymbol{e})$

$\boldsymbol{e} = \boldsymbol{e} - \mathtt{mid}(\boldsymbol{e})$

Algorithm: certify

in practice

Let A be a matrix, b a vector, x^* such that $Ax^* = b$ and $\tilde{x}$ an approximate solution to $Ax = b$.

Iterative refinement: determine an enclosure of the error $x^* - \tilde{x}$.

- use R an approximate inverse of A (or use LU factorization of A and solve triangular systems)
- use an initial enclosure $\boldsymbol{e_0}$ of the error (start with some guess, inflate it if needed until it works)
- replace the procedure to solve an interval linear system by a (hopefully contractant) iteration: apply $[I - RA]$.

Why? if R is close enough to A^{-1}, then RA is close to I and thus $I - RA$ is small / has a spectral radius < 1, thus repeatedly multiplying by it contracts $\boldsymbol{e}$.

Agenda

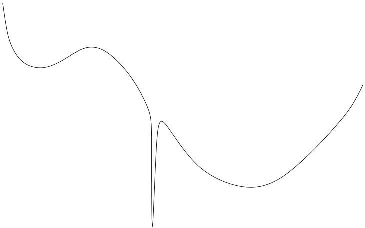
Algorithm: optimize a continuous function

Problem: $f : \mathbb{R}^n \to \mathbb{R}$, determine x^* and f^* that verify

$$f^* = f(x^*) = \min_x f(x)$$

Assumptions:

- search within a box $\boldsymbol{x}_0$
- $x^* \in$ in the interior of $(\boldsymbol{x}_0)$, not at the boundary
- f continuous enough: $\mathcal{C}^2$



Algorithm: optimize a continuous function

(Ratschek and Rokne 1988, Hansen 1992, Kearfott 1996. . .)

Goal: find the minimum of f, continuous function on a box $\boldsymbol{x}_0$.

$\boldsymbol{x}_0$ current box

$\bar{f}$ current upper bound of f^*

while there is a box in the waiting list

    if $f(\boldsymbol{x}) > \bar{f}$ then

        reject $\boldsymbol{x}$

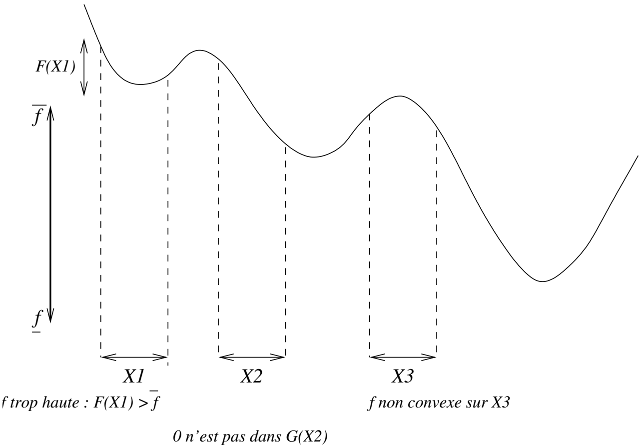
    otherwise

        update $\bar{f}$: if $f(mid(\boldsymbol{x})) < \bar{f}$ then $\bar{f} = f(mid(\boldsymbol{x}))$

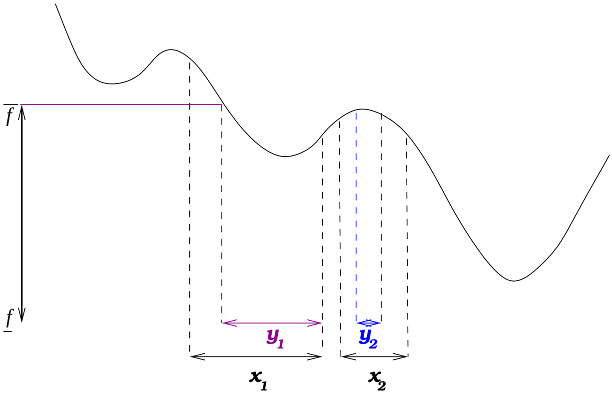
        bisect $\boldsymbol{x}$ into $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$

        examine $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$

Algorithm: optimize a continuous function the rejection procedure



Algorithm: optimize a continuous function the reduction procedure

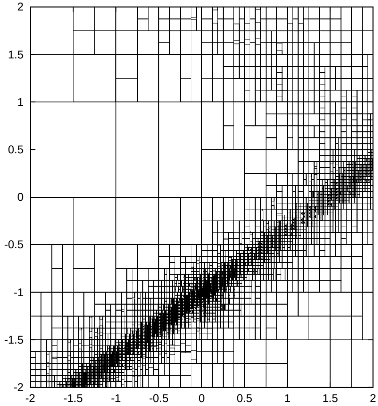


Algorithm: optimize a continuous function

Hansen algorithm Hansen 1992

$\mathcal{L}$ = list of not yet examined boxes := $\{\boldsymbol{x}_0\}$
while $\mathcal{L} \neq \emptyset$ **loop**
  remove $\boldsymbol{x}$ from $\mathcal{L}$
  reject $\boldsymbol{x}$**?**
    yes if $f(\boldsymbol{x}) > \bar{f}$
    yes if Grad$f(\boldsymbol{x}) \not\ni 0$
    yes if $Hf(\boldsymbol{x})$ has its diagonal non > 0
  reduce $\boldsymbol{x}$
    Newton applied to the gradient
    solve $\boldsymbol{y} \subset \boldsymbol{x}$ such that $f(\boldsymbol{y}) \leq \bar{f}$
  bisect $\boldsymbol{y}$: insert the resulting $\boldsymbol{y}_1$ and $\boldsymbol{y}_2$ in $\mathcal{L}$

Exemple de découpage d'un pavé $[-2, 2]^2$



Algorithm: constrained optimization

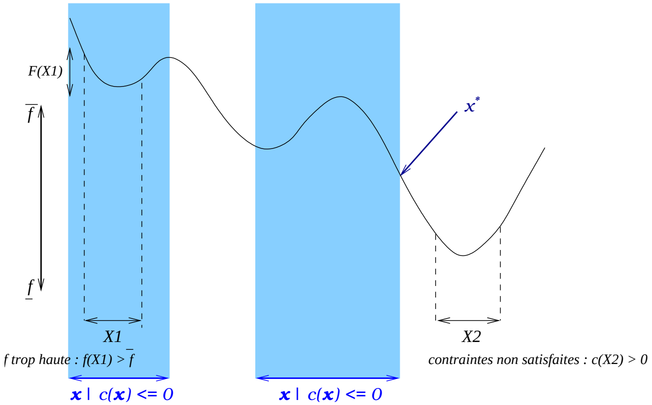
Problem: $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$,
determine x^* and f^* that verify

$$f^* = f(x^*) = \min_{\{x | c(x) \leq 0\}} f(x)$$

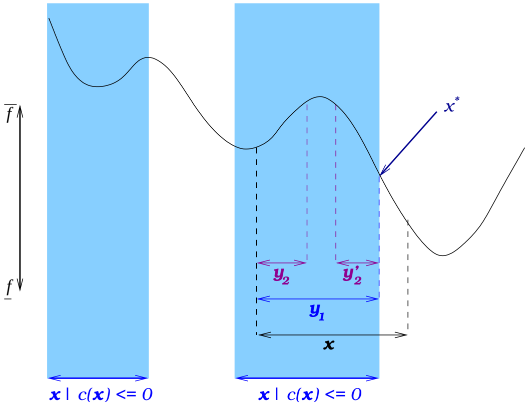
Assumptions:

- search within a box $\boldsymbol{x}_0$
- f continuous enough: $\mathcal{C}^2$
- c continuous enough: $\mathcal{C}^1$

Algorithm: constrained optimization $c(x) \leq 0$ the rejection procedure



Algorithm: constrained optimization $c(x) \leq 0$ the reduction procedure



Algorithm: constrained optimization $c(x) \leq 0$

$\mathcal{L} := \{x_0\}$
while $\mathcal{L} \neq \emptyset$ **loop**
  remove x from $\mathcal{L}$
  reject x**?**
    yes if $f(x) > \bar{f}$
    yes if Grad$f(x) \not\ni 0$
    yes if f not convex on x
  reduce x
    solve $y \subset x \mid f(y) \leq \bar{f}$
    Newton applied to the gradient
  bisect y into y_1 and y_2
    insert y_1 and y_2 in $\mathcal{L}$

$\mathcal{L} := \{x_0\}$
while $\mathcal{L} \neq \emptyset$ **loop**
  remove x from $\mathcal{L}$
  reject x**?**
    yes if $f(x) > \bar{f}$
    yes if $c(x) > 0$
  reduce x
    solve $y \subset x$ such that $c(y) \leq 0$
    Newton applied to the Lagrangian
  bisect y into y_1 and y_2
    insert y_1 and y_2 in $\mathcal{L}$

Agenda

Algorithm: constraints programming

Cleary 1987, Benhamou et al. 1999, Jaulin et al. 2001

Problem:

$$\left\{\begin{array}{lcl} c_1(x_1,\dots,x_n) & = & 0 \\ \vdots & & \\ c_p(x_1,\dots,x_n) & = & 0 \end{array}\right.$$

expressed as:

$$\begin{array}{llcll} & y_i & = & x_i & \text{for } 1 \le i \le n \\ & y_k & = & y_i \diamond y_j & \text{for } n+1 \le k \le m \text{ and } i,j < k \\ & & & & y_k \text{ auxiliary variable} \\ \text{where} & y_k & = & \varphi(y_i) & \text{for } n+1 \le k \le m \text{ and } i < k \end{array}$$

Algorithm: constraints programming

Initializations: $\boldsymbol{y}_1 := \boldsymbol{x}_1, \ldots, \boldsymbol{y}_n := \boldsymbol{x}_n$

Propagation: forward mode

for $k = n+1$ to m loop

$$\boldsymbol{y}_k := \boldsymbol{y}_i \diamond \boldsymbol{y}_j \text{ or } \boldsymbol{y}_k := \varphi(\boldsymbol{y}_i)$$

Propagation: backward mode

for $k = m$ to n loop

if $\boldsymbol{y}_k$ is defined as $\boldsymbol{y}_i \diamond \boldsymbol{y}_j$ then

$$\boldsymbol{y}_i := \left(\boldsymbol{y}_k \diamond^{-r} \boldsymbol{y}_j\right) \cap \boldsymbol{y}_i$$
$$\boldsymbol{y}_j := \left(\boldsymbol{y}_i \diamond^{-l} \boldsymbol{y}_k\right) \cap \boldsymbol{y}_j$$

else if $\boldsymbol{y}_k$ is defined as $\varphi(\boldsymbol{y}_i)$ then

$$\boldsymbol{y}_i := \varphi^{-1}(\boldsymbol{y}_k) \cap \boldsymbol{y}_i$$

using **reverse** ("modal-like") operations.

Agenda

Exercise 1

Test the interval Newton method for these two functions f and g,

$$f \ : \ x \mapsto x^2 - 2 \text{ or } x \mapsto \prod_{i=1}^{5}(x-i)$$

starting from different intervals $\boldsymbol{x}_0$, and

$$g \ : \ x \mapsto x^2 - 2x + 1$$

starting from different intervals $\boldsymbol{x}_0$.

For both cases, observe the speed of convergence, that is, the evolution of the widths of the iterates.

Exercise 1: useful commands

`extended_div`: division by an interval possibly containing 0, returns 2 results
`return x1, x2`: to return 2 results
Always return 2 results even if empty, it is easier to handle
`isempty_interval(I)`: true if I is empty
`interval(4.5)`: transforms 4.5 into an interval, for subsequent evaluation
`intersect_interval`, `isinterior`: may be useful

Exercise 2

Create random matrices A, badly scaled (pre-multiply by D, a diagonal matrix with largely varying coefficients), and/or badly conditioned ($A = Q \cdot D \cdot Q'$ where D as above and Q orthogonal).
Create random vectors x and right-hand side $b = Ax$.
Pretend you do not know x.
Solve (i.e. determine x) such that $Ax = b$ and certify the solution.

`using LinearAlgebra`: to have access to useful functionalities
`A=rand(10,10)`: creates a random matrix of dims 10×10
`A[5,5]`: $A_{5,5}$
`D=diagm([1,2,3,4,5])`: creates a diagonal matrix having 1, 2, 3, 4 and 5 as diagonal elements
`(Q,R)=qr(A)`: to obtain the QR factorization of A, Q is orthogonal
`(L,U)=lu(A)`: to obtain the LU factorization of A
`inv(A)`: to obtain an approximate inverse of A
`A*B`: to multiply two matrices A and B

Homework

Choose one of the exercises.

Send your Julia files including your results,
and your comments (in any form: text file, legibly handwritten document)
to Nathalie.Revol@ens-lyon.fr
at latest Thursday, December 4 at 10h15.

Agenda

Conclusions

Interval algorithms

- can solve problems that other techniques are not able to solve
- is a simple version of set computing
- give effective versions of theorems which did not seem to be effective (Brouwer)
- can determine all zeros or all extrema of a continuous function
- overestimate the result
- is less efficient than floating-point arithmetic (theoretical factor: 4, practical factor: 2 to 15 or worse)
  $\Rightarrow$ solve "small" problems.

Philosophical conclusion

Morale

- forget one's biases:
  - do not use without thinking algorithms which are supposed to be good ones (Newton)
  - do not reject without thinking algorithm which are supposed to be bad ones (Gauss-Seidel)
- prefer contracting iterations whenever possible

Teaser

In relation with formal proof:

- compute your result using interval arithmetic
- clean and simplify the computing path: avoid bisection when not useful, put first the most promising branch...
- give this "neat" computing path to your favorite proof assistant.

Magical conclusion

Magical conclusion

22	6	24	31
28	27	11	17
12	20	25	26
21	30	23	9

References on interval arithmetic

- R. Moore: *Interval Analysis*, Prentice Hall, Englewood Cliffs, 1966.
- A. Neumaier: *Interval methods for systems of equations*, CUP, 1990.
- R. Moore, R.B. Kearfott, M.J. Cloud: *Introduction to interval analysis*, SIAM, 2009.
- W. Tucker: *Validated numerics*, Princeton U. Press, 2011.
- S.M. Rump: *Computer-assisted proofs and Self-Validating Methods*. In B. Einarsson ed., Handbook on Accuracy and Reliability in Scientific Computation, pp. 195-240. SIAM, 2005.
- S.M. Rump: *Verification methods: Rigorous results using floating-point arithmetic*, Acta Numerica, vol. 19, pp. 287-449, 2010.

References on interval arithmetic

- J. Rohn: *A Handbook of Results on Interval Linear Problems*, `http://www.cs.cas.cz/rohn/handbook` 2006.
- E. Hansen and W. Walster: *Global optimization using interval analysis*, MIT Press, 2004.
- R.B. Kearfott: *Rigorous global search: continuous problems*, Kluwer, 1996.
- V. Kreinovich, A. Lakeyev, J. Rohn, P. Kahl: *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Dordrecht, 1997.
- L.H. Figueiredo, J. Stolfi: *Affine arithmetic* `http://www.ic.unicamp.br/~stolfi/EXPORT/projects/affine-arith/`.
- *Taylor models arith.:* M. Berz and K. Makino, N. Nedialkov, M. Neher.