

Recherche de zéros
ooooo

Méthode naïve
oooo

Dichotomie
oooo

Méthode de la fausse position
oooooo

Sécante
oooooooooooo

Newton
oooooooooooo

n-dimensions
oooooooooooo

Lever
oooo

Recherche de zéros et systèmes non-linéaires

Pascal Romon



Recherche de zéros

Problématique :

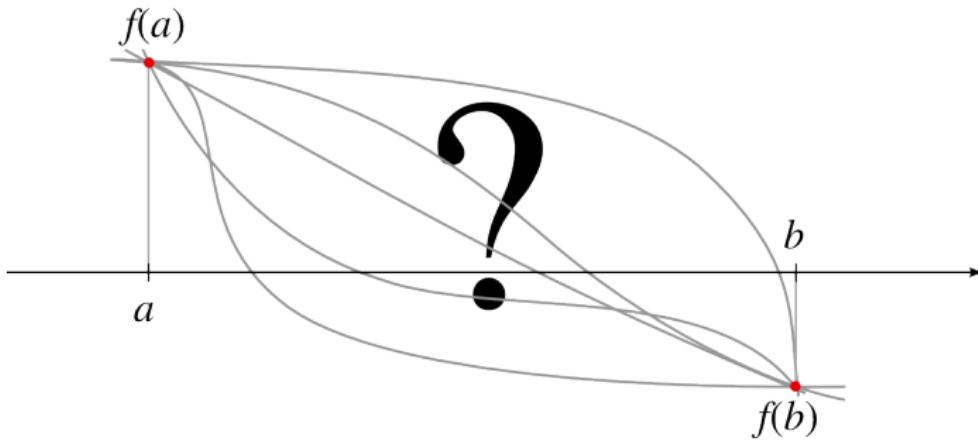
Étant donnée une fonction non-linéaire, on veut trouver les points où cette fonction s'annule (s'ils existent): ses *zéros* (aussi appelés *racines*)

Recherche de zéros

Fonctions étudiées :

- fonctions analytiques
(fonctions dont on connaît la formule)
 $\mathbb{R} \rightarrow \mathbb{R}$, continues et dérivables
 - fonctions non analytiques, mais évaluables en n'importe quel point.
 - un ensemble de points représentant une fonction supposée continue.

Recherche de zéros



On peut évaluer cette fonction sur quelques points, mais on ne sait pas à quoi ressemble la courbe.

Recherche de zéros

Fonction non-linéaire :

Un phénomène est dit non-linéaire lorsque des grandeurs caractéristiques du phénomène reliées entre elles ne

Recherche de zéros

Applications :

Résolution d'équations non linéaires
(permet de résoudre des problèmes mathématiques)

Méthode naïve

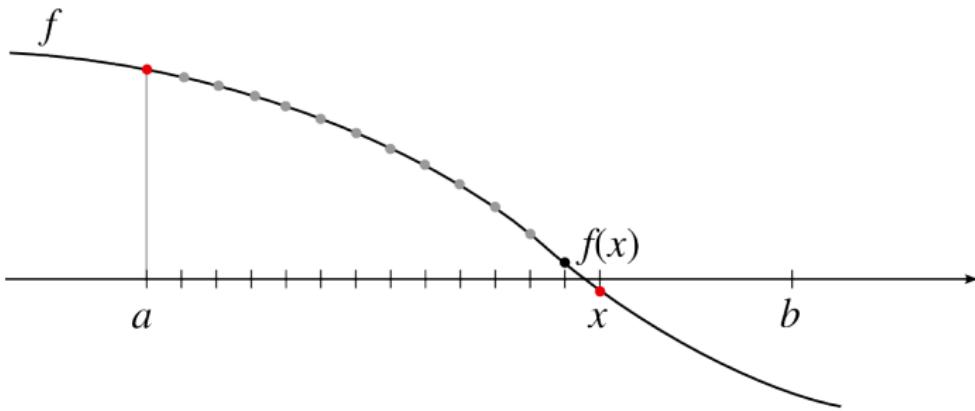
Soit f une fonction :

- continue sur $[a, b]$
 - monotone sur $[a, b]$
 - telle que $f(a)f(b) < 0$

Méthode :

- on choisit un pas dx
 - on teste $f(a + k \cdot dx)$ jusqu'à ce que f change de signe
(k entier positif)

Méthode naïve



Méthode naïve

Algorithm 1: méthode naïve

input: une fonction f et un intervalle $[a, b]$

$x = a + dx$

while $f(x)f(a) > 0$ **do**

| $x = x + dx$

end

return x

Remarque :

Le résultat est l'intervalle $[x - dx, x]$. On peut relancer le même algo sur cet intervalle avec un pas dx plus petit.

Méthode naïve

Réflexion :

C'est nul comme méthode.

Dichotomie

Soit f une fonction :

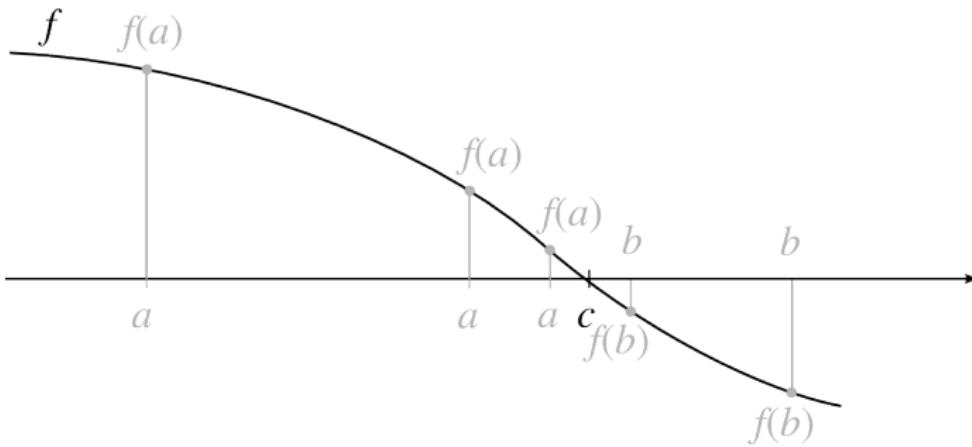
- continue sur $[a, b]$
- monotone sur $[a, b]$
- telle que $f(a)f(b) < 0$

Méthode :

On recherche un petit intervalle contenant le zéro.

- on coupe l'intervalle de départ en 2
- on garde l'intervalle qui contient le 0
- on recommence ...

Dichotomie



Dichotomie

Algorithm 2: Dichotomie

input: fonction f , intervalle $[a, b]$ et nombre d'itérations n

```
for  $i = 1$  to  $n$  do
     $c = (a + b)/2$ 
    if  $f(a)f(c) < 0$  then
        |  $b = c$ 
    else
        |  $a = c$ 
    end
end
return  $(a + b)/2$ 
```

Remarque :

Pour obtenir une précision 2 fois supérieure, il suffit de faire une itération de plus.

Dichotomie

Réflexions :

- La dichotomie ne fait appel qu'au signe, pas aux valeurs de la fonction. On n'utilise pas d'intuition supplémentaire sur son comportement.
- La dichotomie est une amélioration de la méthode naïve récursive, où le pas vaut la moitié de l'intervalle étudié.
- Ça marche aussi si la fonction n'est pas monotone, mais on perd l'unicité s'il y a plusieurs zéros.

Méthode de la fausse position

Soit f une fonction :

- continue sur $[a, b]$
- monotone sur $[a, b]$
- telle que $f(a)f(b) < 0$

Méthode :

- on fait une approximation linéaire de f sur $[a, b]$
 - on approxime la courbe de f par sa *corde*
- on résout l'équation linéaire
 - on calcule l'équation de la droite passant par $(a, f(a))$, $(b, f(b))$
 - on calcule son intersection avec l'axe des abscisses
- on obtient un nouvel intervalle
- on recommence

Recherche de zéros
ooooo

Méthode naïve
oooo

Dichotomie
oooo

Méthode de la fausse position
○●ooo

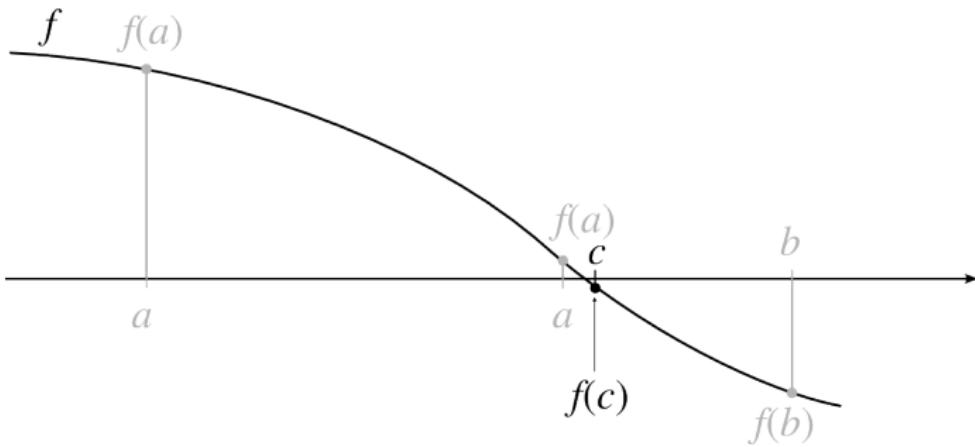
Sécante
oooooooooooo

Newton
oooooooooooo

n-dimensions
oooooooooooo

Lever
oooooo

Méthode de la fausse position



Méthode de la fausse position

Algorithm 3: Fausse position

input: fonction f , intervalle $[a, b]$ et un *seuil*

repeat

$$c = a + f(a)(b - a) / (f(a) - f(b))$$

if $f(a)f(c) < 0$ **then**

$$\quad | \quad b = c$$

else

$$\quad | \quad a = c$$

end

until $|f(c)| < \text{seuil}$

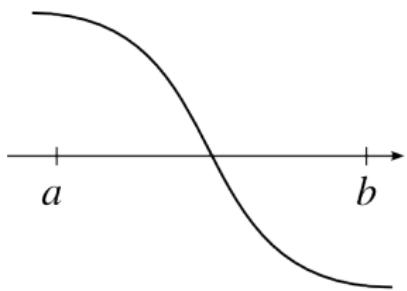
return c

on s'arrête quand $|f(c)|$ est suffisamment petit

Méthode de la fausse position

Réflexion :

Cette méthode peut être inadaptée pour certaines fonctions.



OK



PAS OK

Méthode de la fausse position

Réflexion :

La méthode de la fausse position est adaptée aux fonctions dont le comportement est à peu près linéaire sur $[a, b]$, ce qui permet de restreindre $[a, b]$ plus rapidement qu'avec la méthode de dichotomie.

Elle fait appel aux valeurs de f (contrairement à la méthode par dichotomie) mais pas à celles de la dérivée.

Recherche de zéros

Méthode naïve
oooo

Dichotomie

Méthode de la fausse position Sécante

Newton
●ooooo

n-dimensions

Lever
o ooooo

voir numerical recipes p. 450

Newton

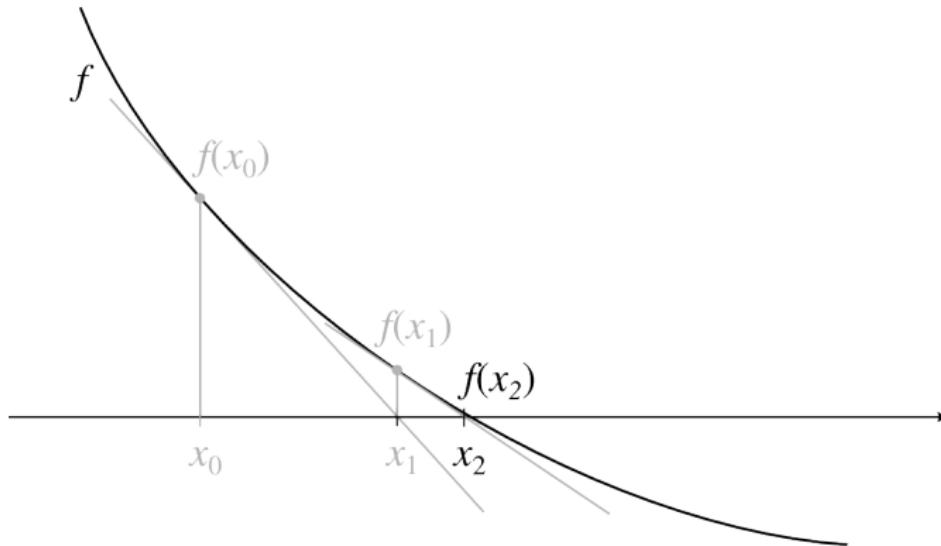
Principe :

- hypothèse : x_0 proche du résultat
- méthode itérative

Méthode :

- on calcule la tangente au point $(x_i, f(x_i))$
- on calcule l'intersection de la tangente avec l'axe des abscisses
- on obtient x_{i+1}
- on recommence

Newton



Newton

Tangente ...

- tangente : $y = mx + p$
- dérivée de f en $x_0 \rightarrow$ coefficient directeur de la tangente
 \rightarrow tangente : $y = f'(x_0)x + p$
- avec $p = y_0 - f'(x_0)x_0$
 \rightarrow tangente : $y = f'(x_0)x + y_0 - f'(x_0)x_0$

Newton

Intersection avec les abscisses ...

- tangente : $y = f'(x_0)x + y_0 - f'(x_0)x_0$
 - si $y_1 = 0$ (et $y_0 = f(x_0)$)

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Newton

Algorithm 4: Newton

input: fonction f , solution initiale x_0 et un *seuil*

$x = x_0$

repeat

| $x = x - f(x)/f'(x)$

until $|f(x)| < \text{seuil}$

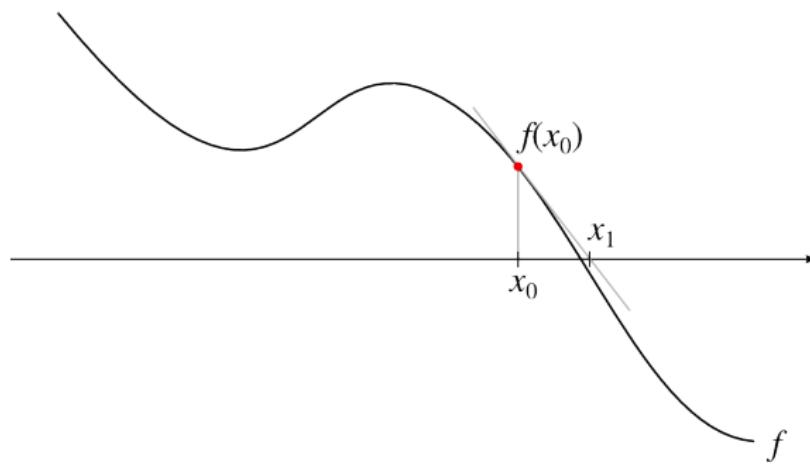
return x

On s'arrête quand $|f(x)|$ est suffisamment petit

Newton

Remarque :

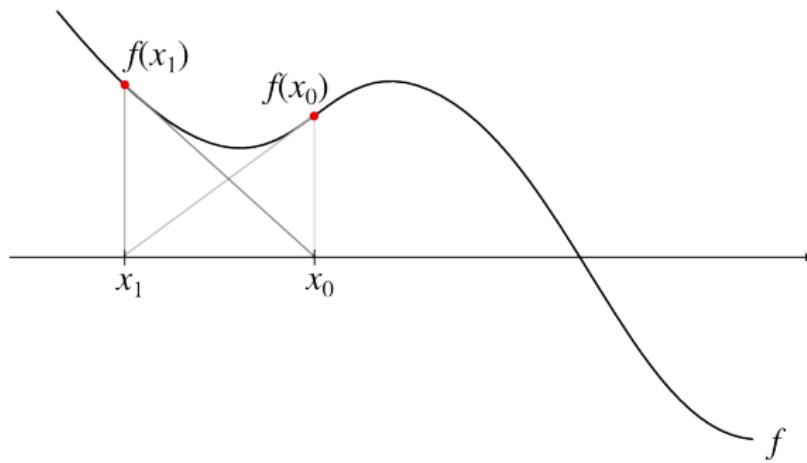
Selon les conditions initiales ...



Newton

Remarque :

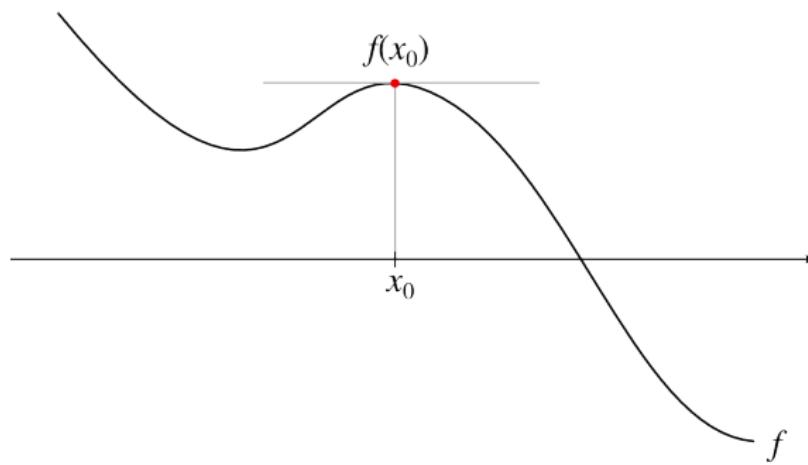
Selon les conditions initiales ...



Newton

Remarque :

Selon les conditions initiales ...



Newton

Réflexion :

- le choix de la position initial est très important
- la convergence n'est pas garantie (contrairement aux autres méthodes)

mais ...

- la convergence peut être très rapide

Méthode de la sécante

Méthode :

On veut utiliser la méthode de Newton, mais on ne connaît pas la dérivée f' de f .

→ on fait une estimation de la dérivée de f en x_0 qu'on réinjecte dans la méthode de Newton.

Méthode de la sécante

$$\text{Dérivée : } f'(x_0) = \lim_{x \rightarrow x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Estimation numérique de la dérivée (dérivée arrière) :

$$f'(x) \simeq \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

Itération

$$x_{n+1} = x_n - \Delta x \frac{f(x_n)}{f(x_n) - f(x_n - \Delta x)}$$

$$= \frac{(x_n - \Delta x)f(x_n) - x_n f(x_n - \Delta x)}{f(x_n) - f(x_n - \Delta x)}$$

Méthode de la sécante

Méthode :

- on évalue la dérivée de $f(x_i)$
- on calcule la tangente au point $(x_i, f(x_i))$
- on calcule l'intersection de la tangente avec l'axe des abscisses
- on obtient x_{i+1}
- on recommence

Méthode de la sécante

Algorithm 5: Newton - estimation de la dérivée

input: fonction f , solution initiale x_0 , un pas Δx et un *seuil*

$x = x_0$

repeat

$$\left| \quad x = \frac{f(x)(x-\Delta x) - x.f(x-\Delta x)}{f(x)-f(x-\Delta x)}$$

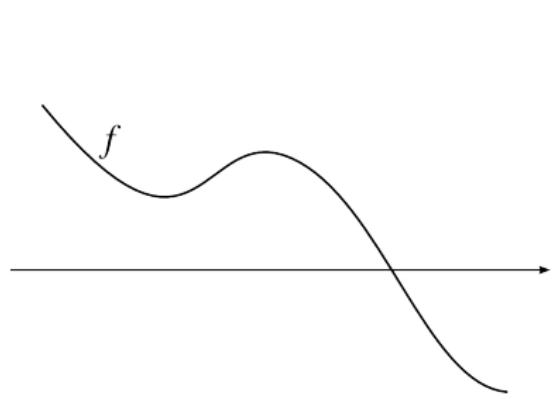
until $|f(x)| < \text{seuil}$

return x

On s'arrête quand $|f(x)|$ est suffisamment petit

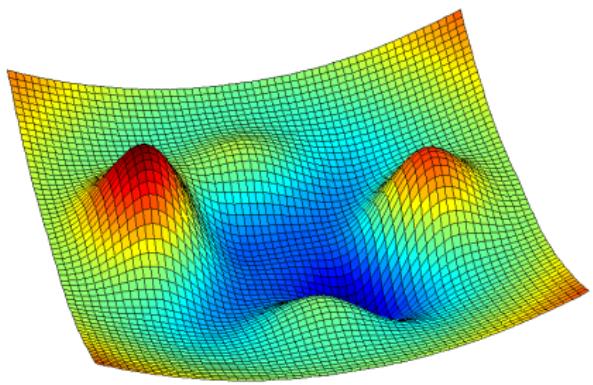
Systèmes non-linéaires

Et pour les fonctions de plusieurs variables?



$$z = f(x)$$

1D

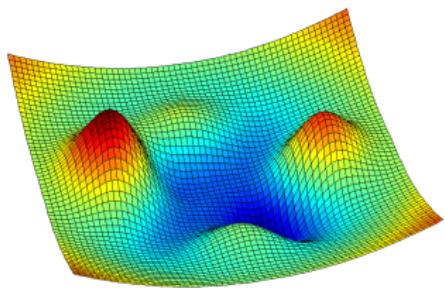


$$z = f(x, y)$$

2D

Systèmes non-linéaires

Certaines fonctions ont des images vectorielles :



$$\mathbf{z} = f(\mathbf{x})$$

avec $\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}$ et $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$

Zéros d'une fonction de plusieurs variables

Problème associé, la minimisation :

Deux cas de figure pour une fonction $z = f(x)$:

- ① on suppose que f possède un minimum (connu) $\hat{\mathbf{z}}$
on cherche $\hat{\mathbf{x}}$ réalisant le minimum ; il faut donc résoudre
 $f(\hat{\mathbf{x}}) = \hat{\mathbf{z}}$
 - ② on veut résoudre $f(\hat{\mathbf{x}}) = \hat{\mathbf{z}}$
alors on cherche le minimum de $\phi(\mathbf{x}) = \|f(\mathbf{x}) - \hat{\mathbf{z}}\|$

Analogie avec le cas unidimensionnel

Dérivée 1D :

$$f'(x) \simeq \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Soit

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x$$

Dérivée en n dimensions :

$$f(\mathbf{x} + \Delta_{\mathbf{x}}) \simeq f(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta_{\mathbf{x}}$$

$J(\mathbf{x})$ est la matrice jacobienne en \mathbf{x}

Optimisation non-linéaire

Soit une fonction f à plusieurs variables de $\mathbb{R}^n \mapsto \mathbb{R}^m$:

$$f : \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \longmapsto \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{pmatrix}$$

Définition:

La dérivée partielle de f est la dérivée par rapport à l'une de ses variables, les autres étant gardées constantes. Exemple:

$$\frac{\partial f(\mathbf{x})}{\partial x_i}$$

est la **dérivée partielle** de f par rapport à la variable x_i du vecteur \mathbf{x} . Lors du calcul de cette dérivée, les autres variables x_j $j \neq i$ sont considérées comme constantes. On peut dériver le vecteur ou chacune

Matrice jacobienne

Soit une fonction f à plusieurs variables de $\mathbb{R}^n \mapsto \mathbb{R}^m$:

$$f : \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \longmapsto \begin{pmatrix} f_1(x_1, x_2, \dots, x_n) \\ f_2(x_1, x_2, \dots, x_n) \\ \vdots \\ f_m(x_1, x_2, \dots, x_n) \end{pmatrix}$$

Définition:

La matrice jacobienne J est la matrice des dérivées partielles du premier ordre d'une fonction vectorielle f lorsqu'elles existent.

$$J = \frac{\partial(f_1, \dots, f_m)}{\partial(x_1, \dots, x_n)} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Optimisation non-linéaire

Dérivée 1D :

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x$$

Dérivée n -dimensionnelle :

$$f(\mathbf{x} + \Delta_{\mathbf{x}}) \simeq f(\mathbf{x}) + J\Delta_{\mathbf{x}}$$

Méthode itérative

$$f(\mathbf{x} + \Delta_{\mathbf{x}}) \simeq f(\mathbf{x}) + J\Delta_{\mathbf{x}}$$

Principe :

on cherche Δ_x tel que $f(x + \Delta_x) - \hat{z} = 0$

$$\rightarrow \underbrace{f(\mathbf{x}) - \hat{\mathbf{z}}}_{\varepsilon} + J\Delta_{\mathbf{x}} = \mathbf{0}, \text{ soit } \varepsilon + J\Delta_{\mathbf{x}} = \mathbf{0}$$

Méthode :

→ on veut résoudre : $J\Delta_x = -\varepsilon$ (si possible)

→ la solution : $\Delta_x = -J^+ \varepsilon$

$$J^+ = (J^\top J)^{-1} J^\top$$

→ on avance : $x_{i+1} = x_i + \Delta_x$

→ on recommence jusqu'à convergence

Méthode itérative

Algorithm 6: Newton n -dimensions

input: fonction f , solution initiale \mathbf{x}_0 , $\hat{\mathbf{z}}$ et un seuil

$$x = x_0$$

repeat

$$J = [\partial f_i / \partial x_j]$$

$$\varepsilon = f(x) - \hat{z}$$

$$\Delta_x = -J^+ \varepsilon$$

$$x = x + \Delta_x$$

until $|f(\mathbf{x}) - \hat{\mathbf{z}}| < \text{seuil}$

return x

Retour à la dimension 1

Remarque :

Il s'agit en fait de la méthode de Newton :

$$\begin{aligned}\mathbf{x}_{i+1} &= \mathbf{x}_i + \Delta_{\mathbf{x}} \\ &= \mathbf{x}_i - J^+ \varepsilon \\ &= \mathbf{x}_i - J^+ (f(\mathbf{x}_i) - \hat{\mathbf{z}})\end{aligned}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad \longleftrightarrow \quad x_{i+1} = x_i - J^+(f(x_i) - \hat{z})$$

En dimension 1: $J = f'(x)$ est une matrice 1×1 , $J^+ = 1/f'(x)$.

Levenberg-Marquardt

Introduction :

Il s'agit d'une amélioration de la méthode de Newton où le pas Δ_x a une importance variable selon la situation.

Levenberg-Marquardt

Newton :

$$\begin{aligned} \mathbf{J}\Delta_{\mathbf{x}} &= -\varepsilon &\rightarrow \Delta_{\mathbf{x}} &= -\mathbf{J}^+ \varepsilon \\ &&\rightarrow \Delta_{\mathbf{x}} &= -(\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \varepsilon \end{aligned}$$

$\Delta_{\mathbf{x}}$ est un pas acceptable si $|f(\mathbf{x} + \Delta_{\mathbf{x}})| < |f(\mathbf{x})|$, ce qui n'est pas toujours le cas.

$\Delta_{\mathbf{x}}$ trop petit :

- résolution trop longue
- risque de tomber dans un minimum local

$\Delta_{\mathbf{x}}$ trop grand :

- risque de dépasser le minimum global et de diverger

Levenberg-Marquardt

Newton ;

$$\begin{aligned} \mathbf{J}\Delta_{\mathbf{x}} = -\varepsilon &\quad \rightarrow \quad \Delta_{\mathbf{x}} = -\mathbf{J}^+ \varepsilon \\ &\quad \rightarrow \quad \Delta_{\mathbf{x}} = -(\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \varepsilon \end{aligned}$$

Levenberg-Marquardt :

$$\begin{aligned} \mathbf{J}\Delta_{\mathbf{x}} = -\varepsilon &\quad \rightarrow \quad \mathbf{J}^\top \mathbf{J} \Delta_{\mathbf{x}} = -\mathbf{J}^\top \varepsilon \\ &\quad \rightarrow \quad (\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{Id}) \Delta_{\mathbf{x}} = -\mathbf{J}^\top \varepsilon \\ &\quad \rightarrow \quad \Delta_{\mathbf{x}} = -(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{Id})^{-1} \mathbf{J}^\top \varepsilon \end{aligned}$$

- Δ_x trop petit \rightarrow on diminue λ
 - Δ_x trop grand \rightarrow on augmente λ

Levenberg-Marquardt

En pratique :

On choisit d'abord une valeur initiale de λ_0 :

$$\lambda_0 = 10^{-3} \times \frac{\sum_{i=1}^n (\mathbf{J}\mathbf{J}^\top)_{ii}}{n}$$

- si Δ_x fait converger f : $\lambda = \lambda \div 10$
pour accélérer la convergence
- si Δ_x ne fait pas converger f : $\lambda = 10 \times \lambda$
 Δ_x était trop grand

Algorithm 7: Levenberg-Marquardt

input: fonction f , solution initiale \mathbf{x}_0 , $\hat{\mathbf{z}}$ et un *seuil*

$\mathbf{x} = \mathbf{x}_0$

$\lambda = \frac{1}{n} \sum_{i=1}^n (\mathbf{J}\mathbf{J}^\top)_{ii} \times 10^{-3}$

repeat

$\mathbf{J} = [\partial f_i / \partial x_j]$

compteur = 0

accepté = **false**

repeat

$\Delta_{\mathbf{x}} = -(\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{Id})^+ \mathbf{J}^\top (f(\mathbf{x}) - \hat{\mathbf{z}})$

if $|f(\mathbf{x} + \Delta_{\mathbf{x}})| < |f(\mathbf{x})|$ **then**

$\mathbf{x} = \mathbf{x} + \Delta_{\mathbf{x}}$

$\lambda = \lambda / 10$

accepté = **true**

end

else $\lambda = \lambda \times 10$

compteur = *compteur* + 1

if *compteur* > 100 **then return** \mathbf{x}

until *accepté* = **true**

until $|f(\mathbf{x}) - \hat{\mathbf{z}}| < \text{seuil}$

return \mathbf{x}

Applications

Rectification d'images :



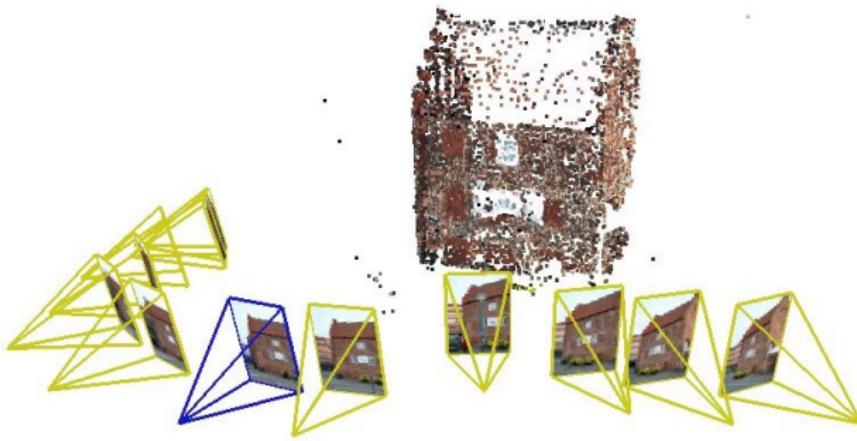
Applications

Correction de la distorsion radiale : (certains modèles sont linéaires)



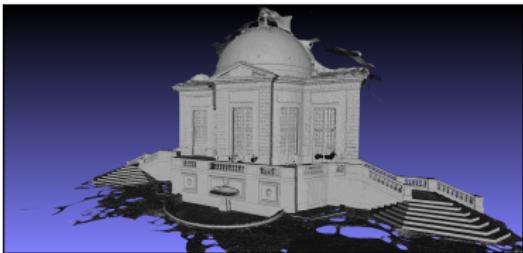
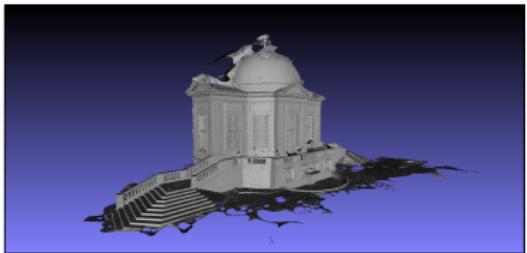
Applications

Bundle adjustment :



Applications

Bundle adjustment :



Applications

Trajectoire optimale : (courbe minimale vs. chemin le plus court)

SP
MCP

