Recherche de zéros et systèmes non-linéaires

Vincent Nozick

Recherche de zéros

Problématique :

Étant donnée une fonction non-linéaire, on veut trouver le ou les points où cette fonction s'annule (s'ils existent).

Recherche de zéros

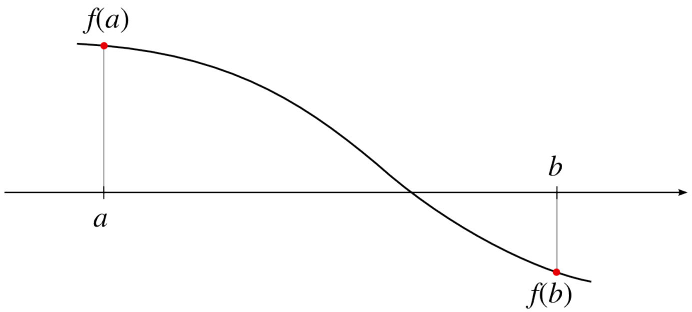
Fonctions étudiées :

- fonctions analytiques
  (fonctions dont on connait 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

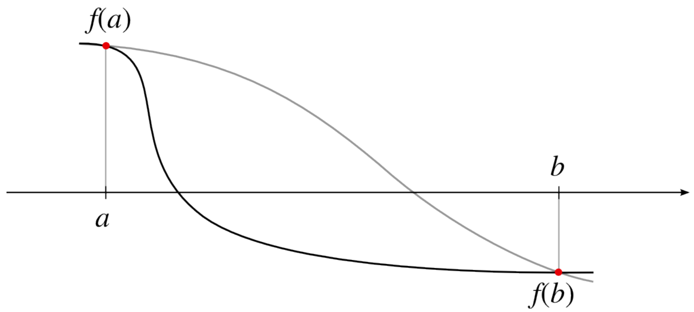
On peut évaluer cette fonction sur quelques points,
mais on ne sait pas à quoi ressemble la courbe.

Recherche de zéros



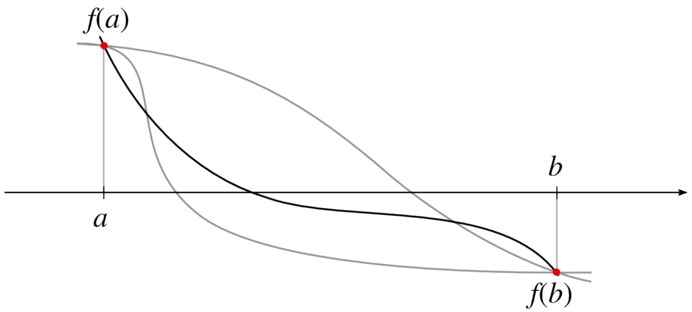
On peut évaluer cette fonction sur quelques points,
mais on ne sait pas à quoi ressemble la courbe.

Recherche de zéros



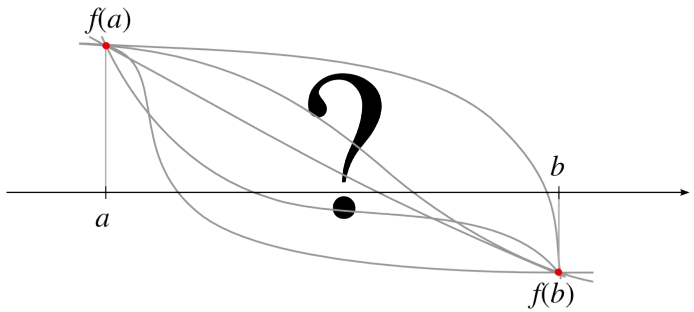
On peut évaluer cette fonction sur quelques points,
mais on ne sait pas à quoi ressemble la courbe.

Recherche de zéros



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

Recherche de zéros



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

Méthode naïve

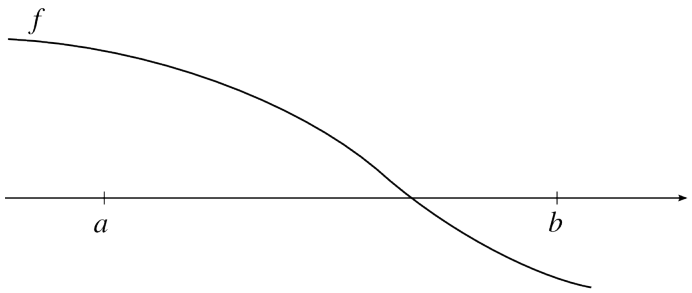
Soit f une fonction :

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

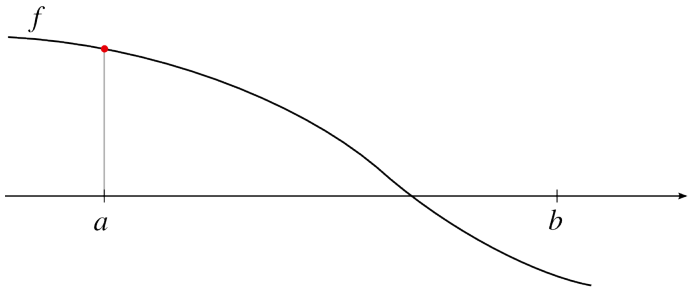
Méthode :

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

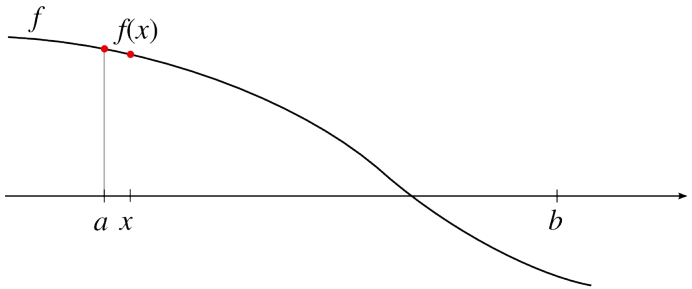
Méthode naïve



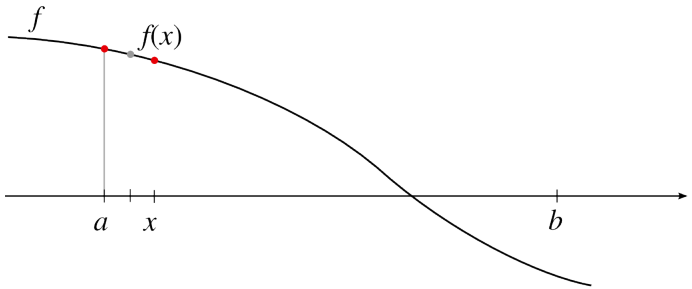
Méthode naïve



Méthode naïve

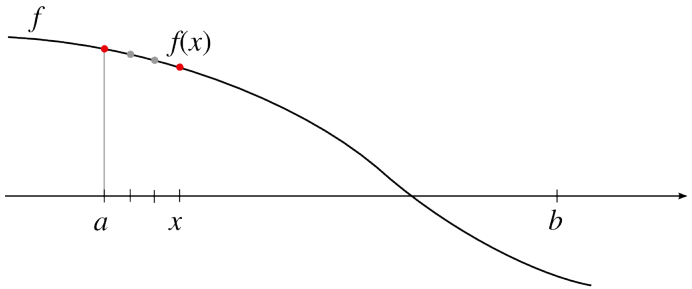


Méthode naïve



Méthode naïve Dichotomie Positions fausses Newton n-dimensions Levenberg-Marquardt Applications

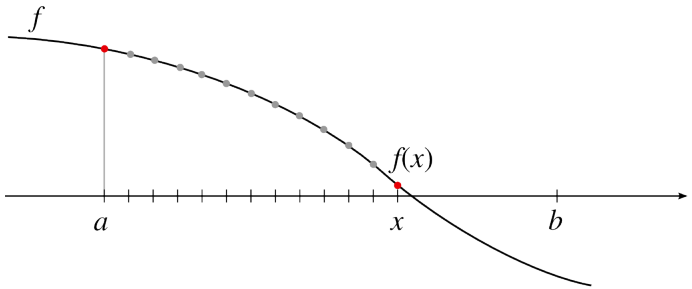
Méthode naïve



Vincent Nozick Recherche de zéros et systèmes non-linéaires 6 / 48

Méthode naïve Dichotomie Positions fausses Newton n-dimensions Levenberg-Marquardt Applications

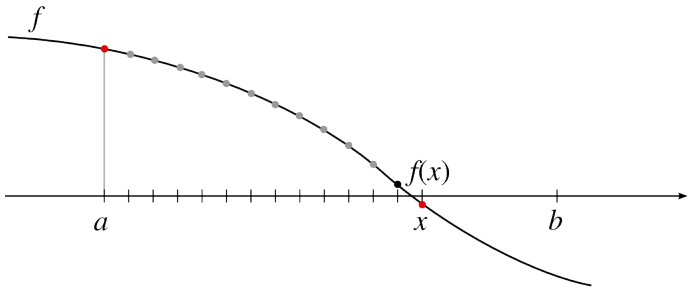
Méthode naïve



Vincent Nozick Recherche de zéros et systèmes non-linéaires 6 / 48

Méthode naïve Dichotomie Positions fausses Newton n-dimensions Levenberg-Marquardt Applications

Méthode naïve



Vincent Nozick Recherche de zéros et systèmes non-linéaires 6 / 48

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
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 cherche un intervalle petit contenant le 0

Dichotomie

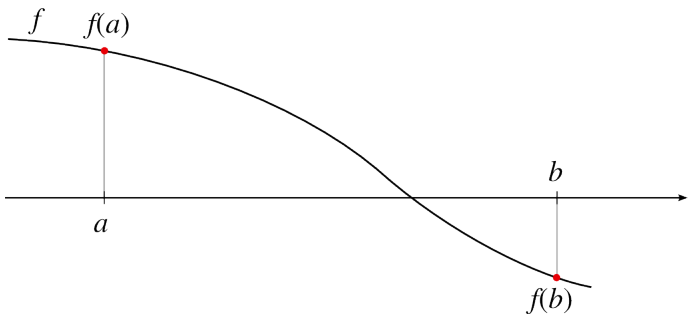
Soit f une fonction :

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

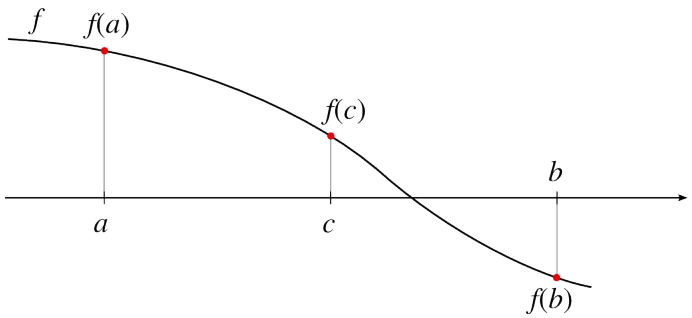
Méthode :
On cherche un intervalle petit contenant le 0

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

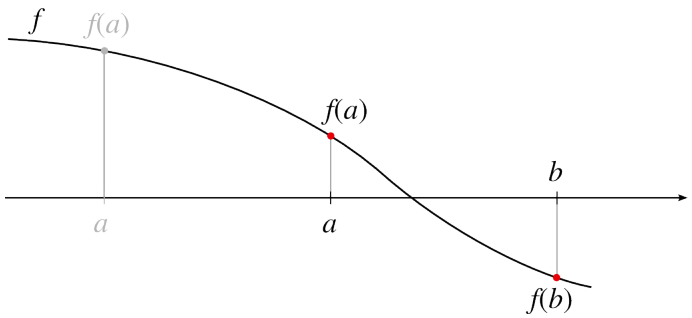
Dichotomie



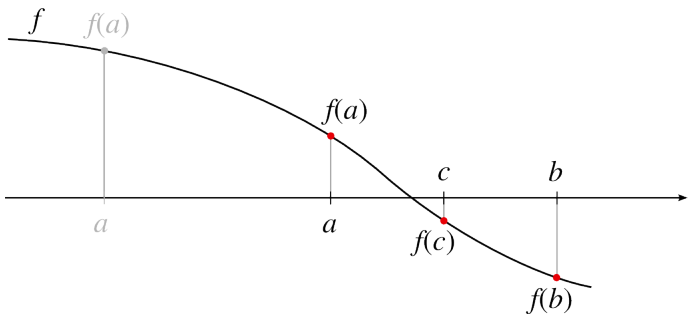
Dichotomie



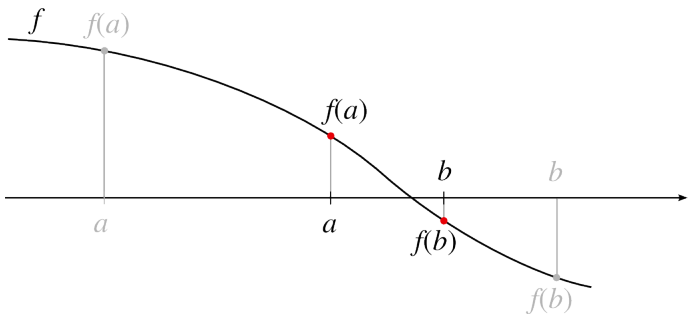
Dichotomie



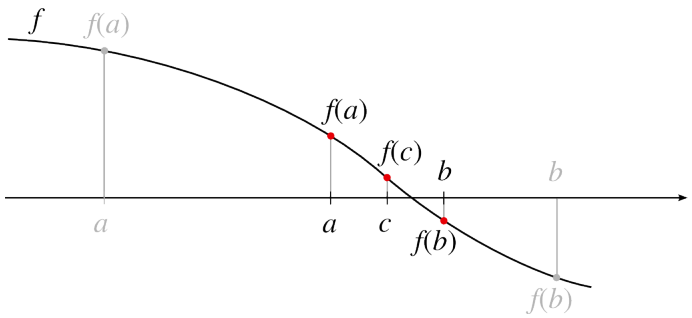
Dichotomie



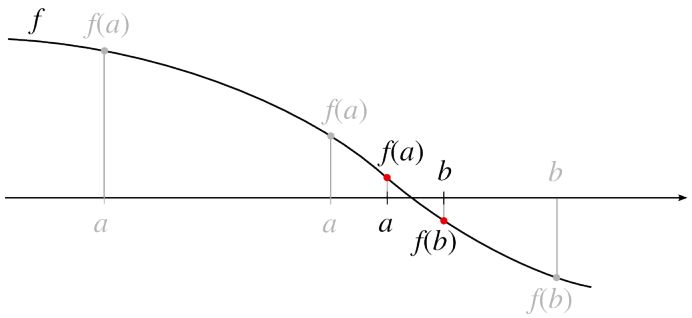
Dichotomie



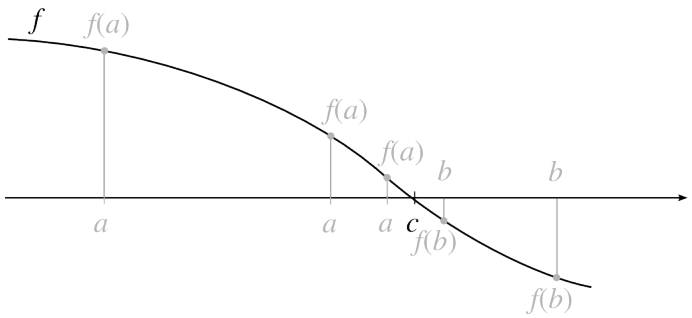
Dichotomie



Dichotomie



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
return (a + b)/2
```

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

Dichotomie

Réflexions :

- A travers la méthode de dichotomie, on considère que la probabilité que f s'annule d'un coté ou de l'autre du milieu de l'intervalle $[a, b]$ est la même.
- Finalement, la dichotomie est une extension de la méthode naïve récursive, où le pas vaut la moitié de l'intervalle étudié.

Positions fausses

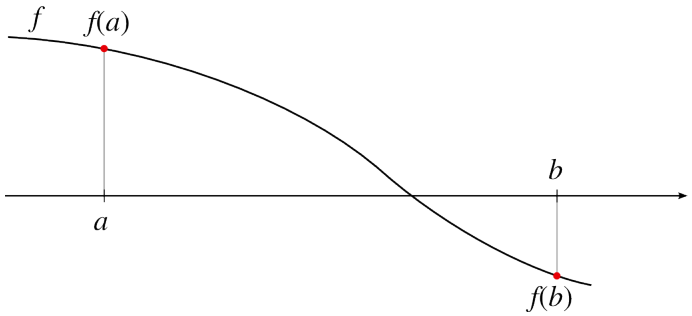
Soit f une fonction :

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

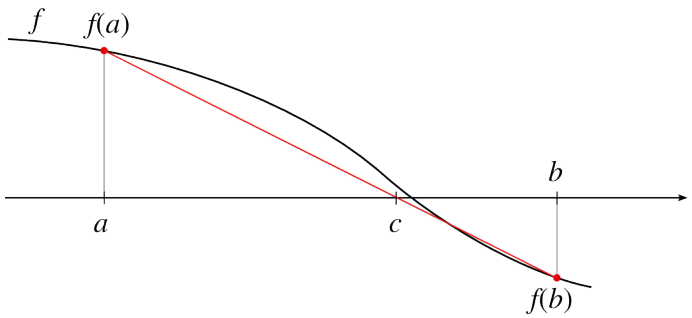
Méthode :

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

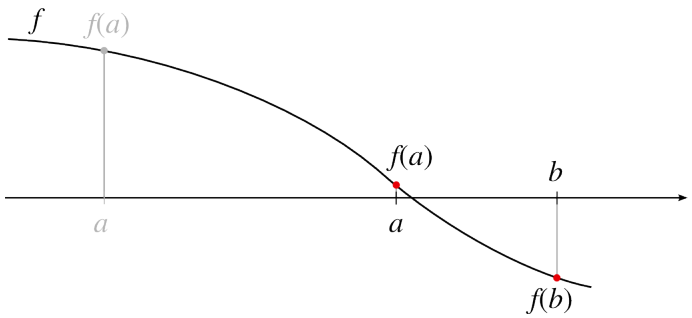
Positions fausses



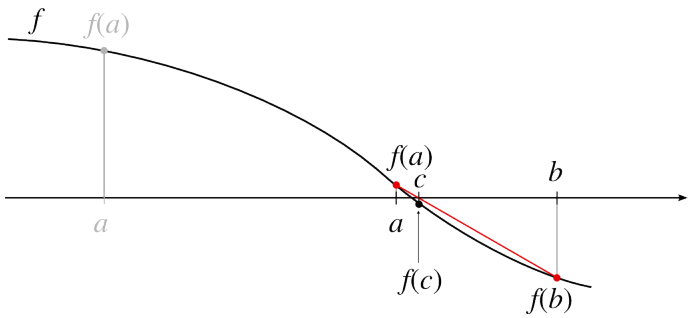
Positions fausses



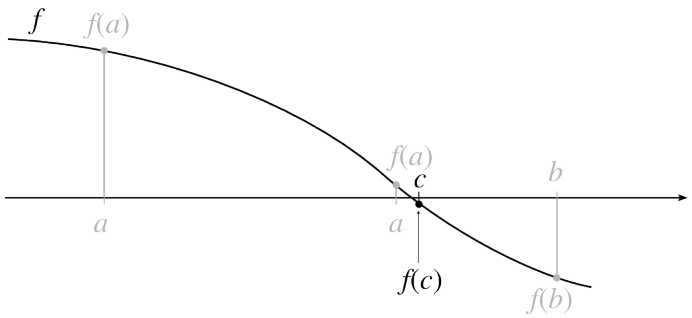
Positions fausses



Positions fausses



Positions fausses



Positions fausses

Algorithm 3: Positions fausses

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

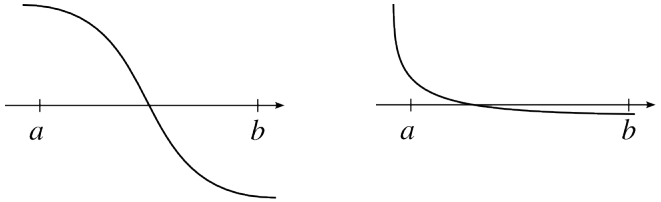
```
repeat
    c = a + f(a)(b − a) / (f(a) − f(b))
    if f(a)f(c) < 0 then
        b = c
    else
        a = c
until |f(c)| < seuil
return c
```

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

Positions fausses

Réflexion :

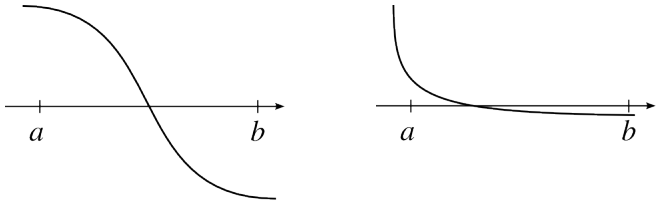
Cette méthode peut être inadaptée pour certaines fonctions.



Positions fausses

Réflexion :

Cette méthode peut être inadaptée pour certaines fonctions.



OK **PAS OK**

Positions fausses

Réflexion :

A travers la méthode des positions fausses, on espère que f est à peu près linéaire sur $[a, b]$, ce qui permet de restreindre $[a, b]$ plus rapidement qu'avec la méthode de dichotomie.

Newton

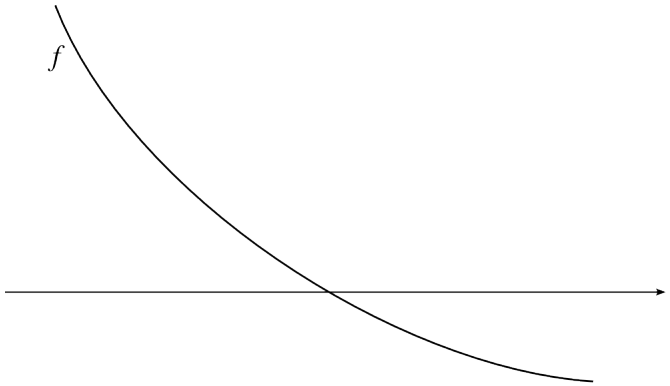
Principe :

- hypothèse : x_0 proche du résultat
- méthode itérative

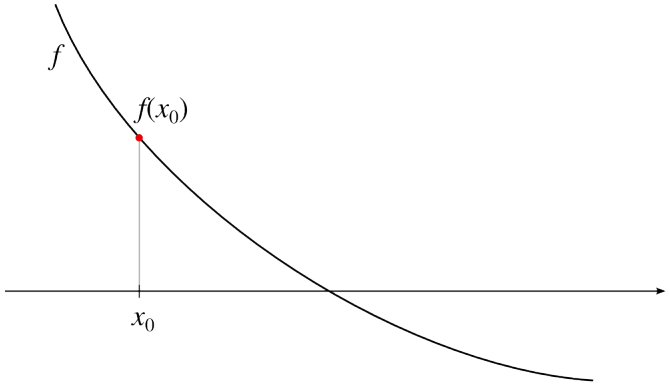
Méthode :

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

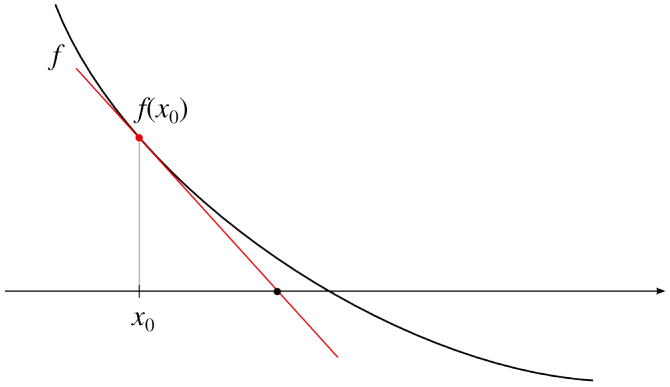
Newton



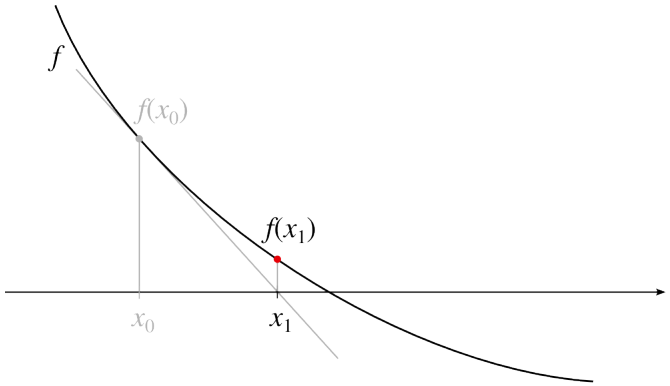
Newton



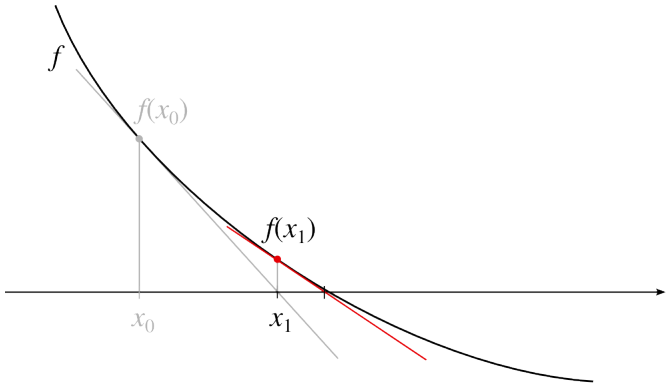
Newton



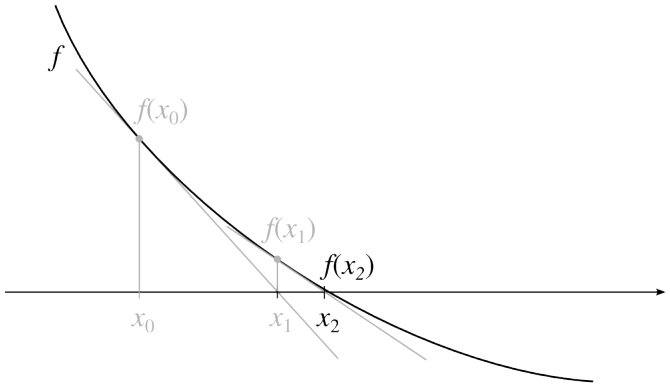
Newton



Newton



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 $y_0 = f(x_0)$)

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

Newton

Algorithm 4: Newton

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

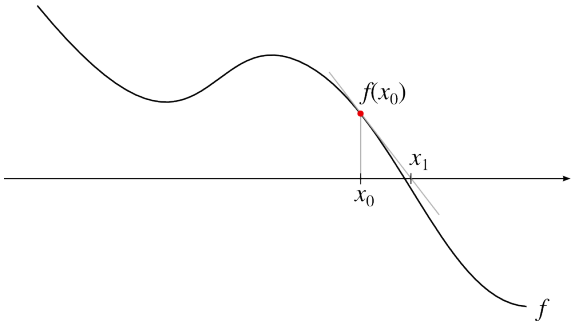
1 $x = x_0$
2 **repeat**
3 $\quad x = x - f(x)/f'(x)$
4 **until** $|f(x)| < seuil$
5 **return** x

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

Newton

Remarque :

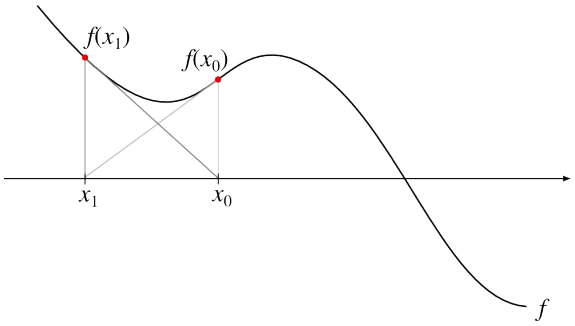
Selon les conditions initiales ...



Newton

Remarque :

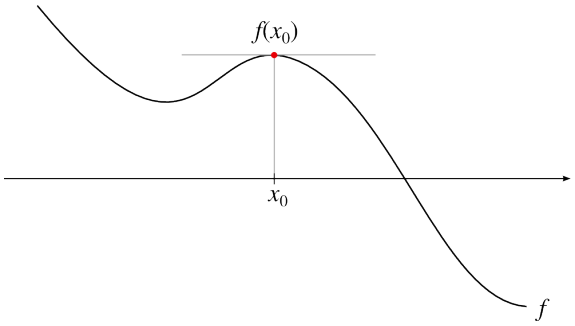
Selon les conditions initiales ...



Newton

Remarque :

Selon les conditions initiales ...



Newton

Réflexion :

- le choix de la position initial est très important
- la convergence n'est pas garantie
- la convergence peut être très rapide

Méthode de la sécante

Méthode :

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

Méthode de la sécante

Méthode :

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

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

Méthode de la sécante

Dérivée :

$$f'(x_0) = \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0}$$

Estimation numérique de la dérivée (dérivée arrière) :

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

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 - éstimation de la dérivée

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

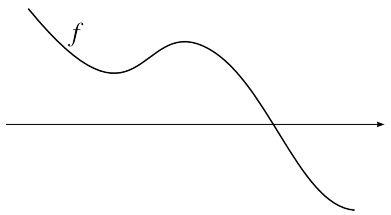
1 $x = x_0$
2 **repeat**
3 $\quad x = \frac{f(x)(x-\Delta x) - x.f(x-\Delta x)}{f(x) - f(x-\Delta x)}$
4 **until** $|f(x)| <$ *seuil*
5 **return** x

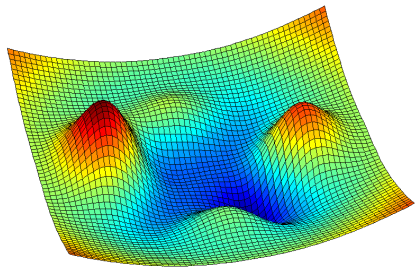
On s'arrête quand $|f(x)|$ est suffisament petit

Systèmes non-linéaires

Introduction :

Et pour les fonctions à plusieurs variables?



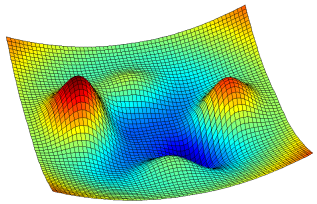


$z = f(x)$
1D

$z = f(x, y)$
2D

Systèmes non-linéaires

D'ailleurs, certaines fonctions renvoient plusieurs variables :



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

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

Par exemple, les hauteurs respectives d'un groupe de soldats méchants groupés autour du chef, se déplaçant dans la montagne.

Systèmes non-linéaires

Méthode :

soit la fonction $\mathbf{z} = f(\mathbf{x})$ et une valeur $\widehat{\mathbf{z}}$ connue
on cherche $\widehat{\mathbf{x}}$ tel que $f(\widehat{\mathbf{x}}) = \widehat{\mathbf{z}}$

$\rightarrow$ on veut minimiser $\epsilon = |f(\mathbf{x}) - \widehat{\mathbf{z}}|$
autrement dit, on veut faire tendre $f(\mathbf{x})$ vers $\widehat{\mathbf{z}}$ autant que possible.

$\rightarrow$ on part d'une estimation $\mathbf{x}_0$

Systèmes non-linéaires

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 n-dimensions :

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

Matrice jacobienne

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

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

Définition:
La matrice jacobienne $\mathbf{J}$ est la matrice des dérivées partielles du premier ordre d'une fonction vectorielle f.

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

Systèmes non-linéaires

Dérivée 1D :

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

Dérivée n-dimensions :

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

Systèmes non-linéaires

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

Principe :
on cherche $\Delta_\mathbf{x}$ tel que $f(\mathbf{x} + \Delta_\mathbf{x}) - \widehat{\mathbf{z}} = \mathbf{0}$
$\rightarrow \underbrace{f(\mathbf{x}) - \widehat{\mathbf{z}}}_{\epsilon} + \mathbf{J}\Delta_\mathbf{x} = \mathbf{0}$, soit $\epsilon + \mathbf{J}\Delta_\mathbf{x} = \mathbf{0}$

Méthode :

- $\rightarrow$ on veut minimiser : $\mathbf{J}\Delta_\mathbf{x} = -\epsilon$
- $\rightarrow$ la solution : $\Delta_\mathbf{x} = -\mathbf{J}^+\epsilon$ $\qquad \mathbf{J}^+ = (\mathbf{J}^\top\mathbf{J})^{-1}\mathbf{J}^\top$
- $\rightarrow$ on avance : $\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta_\mathbf{x}$
- $\rightarrow$ on recommence jusqu'à convergence

Systèmes non-linéaires

Algorithm 6: Newton n-dimensions

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

```
1 x = x_0
2 repeat
3 |   J = [∂f_i/∂x_j]
4 |   ε = f(x) − ẑ
5 |   Δ_x = −J^+ ε
6 |   x = x + Δ_x
7 until |f(x) − ẑ| < seuil
8 return x
```


Systèmes non-linéaires

Remarque :
Il s'agit en fait de la méthode de newton :

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

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \qquad \longleftrightarrow \qquad \mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{J}^+(f(\mathbf{x}_i) - \widehat{\mathbf{z}})$$

Levenberg-Marquardt

Introduction :

Il s'agit d'une optimisation de la méthode de Newton où le pas $\Delta_{\mathbf{x}}$ a une importance variable selon la situation.

Levenberg-Marquardt

Newton :

$$\begin{array}{rcl} \mathbf{J}\Delta_{\mathbf{x}} = -\epsilon & \longrightarrow & \Delta_{\mathbf{x}} = -\mathbf{J}^{+}\epsilon \\ & \longrightarrow & \Delta_{\mathbf{x}} = -(\mathbf{J}^{\top}\mathbf{J})^{-1}\mathbf{J}^{\top}\epsilon \end{array}$$

$\Delta_{\mathbf{x}}$ 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
- risques de tomber dans un minimum local

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

- risque de dépasser le minimum global et de diverger

Levenberg-Marquardt

Newton :

$$\begin{array}{rcl} \mathbf{J}\Delta_{\mathbf{x}} = -\epsilon & \longrightarrow & \Delta_{\mathbf{x}} = -\mathbf{J}^{+}\epsilon \\ & \longrightarrow & \Delta_{\mathbf{x}} = -(\mathbf{J}^{\top}\mathbf{J})^{-1}\mathbf{J}^{\top}\epsilon \end{array}$$

Levenberg-Marquardt :

$$\begin{array}{rcl} \mathbf{J}\Delta_{\mathbf{x}} = -\epsilon & \longrightarrow & \mathbf{J}^{\top}\mathbf{J}\Delta_{\mathbf{x}} = -\mathbf{J}^{\top}\epsilon \\ & \longrightarrow & (\mathbf{J}^{\top}\mathbf{J} + \lambda\mathbf{Id})\Delta_{\mathbf{x}} = -\mathbf{J}^{\top}\epsilon \\ & \longrightarrow & \Delta_{\mathbf{x}} = -(\mathbf{J}^{\top}\mathbf{J} + \lambda\mathbf{Id})^{-1}\mathbf{J}^{\top}\epsilon \end{array}$$

- $\Delta_{\mathbf{x}}$ trop petit $\rightarrow$ on diminue λ
- $\Delta_{\mathbf{x}}$ trop grand $\rightarrow$ on augmente λ

Levenberg-Marquardt

En pratique :

On choisi 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 $\Delta_{\mathbf{x}}$ fait converger f : $\lambda = \lambda \div 10$
  pour accélérer la convergence
- si $\Delta_{\mathbf{x}}$ ne fait pas converger f : $\lambda = 10 \times \lambda$
  $\Delta_{\mathbf{x}}$ était trop grand

Algorithm 7: Levenberg-Marquardt

input: fonction f, solution initiale $\mathbf{x}_0$, $\widehat{\mathbf{z}}$ 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
    $\Delta_{\mathbf{x}} = -(\mathbf{J}^\top\mathbf{J} + \lambda\mathbf{Id})^{+}\mathbf{J}^\top(f(\mathbf{x}) - \widehat{\mathbf{z}})$
    if $|f(\mathbf{x} + \Delta_{\mathbf{x}})| < |f(\mathbf{x})|$ **then**
      $\mathbf{x} = \mathbf{x} + \Delta_{\mathbf{x}}$
      $\lambda = \lambda / 10$
      accepté $=$ **true**
    else $\lambda = \lambda \times 10$
    compteur $=$ *compteur* $+ 1$
    if *compteur* > 100 **then return** $\mathbf{x}$
  until *accepté* $=$ **true**
until $|f(\mathbf{x}) - \widehat{\mathbf{z}}| <$ *seuil*
return $\mathbf{x}$

Applications

Rectification d'images :


Applications

Rectification d'images :

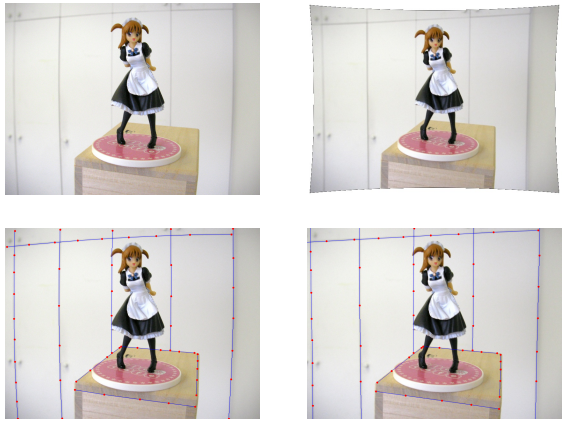



Applications

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

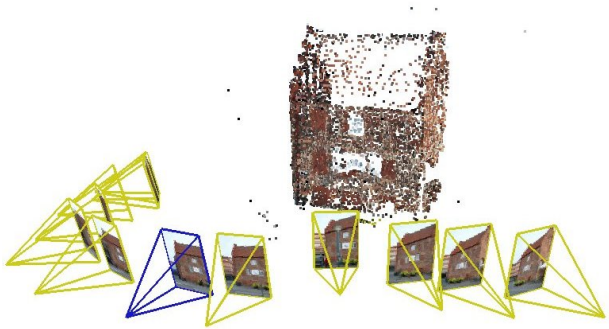

Applications

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



Applications

Bundle adjustment :



Applications

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