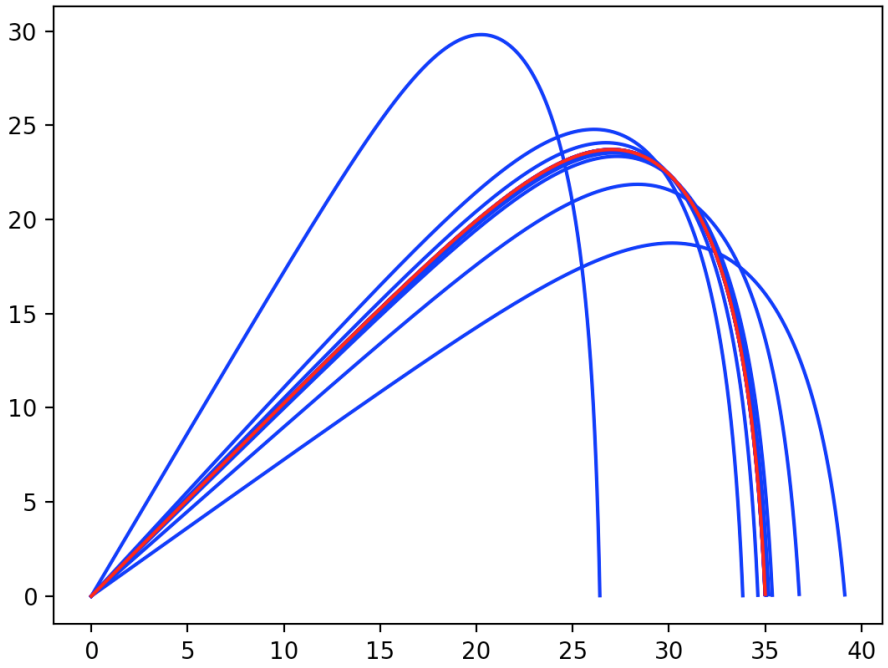
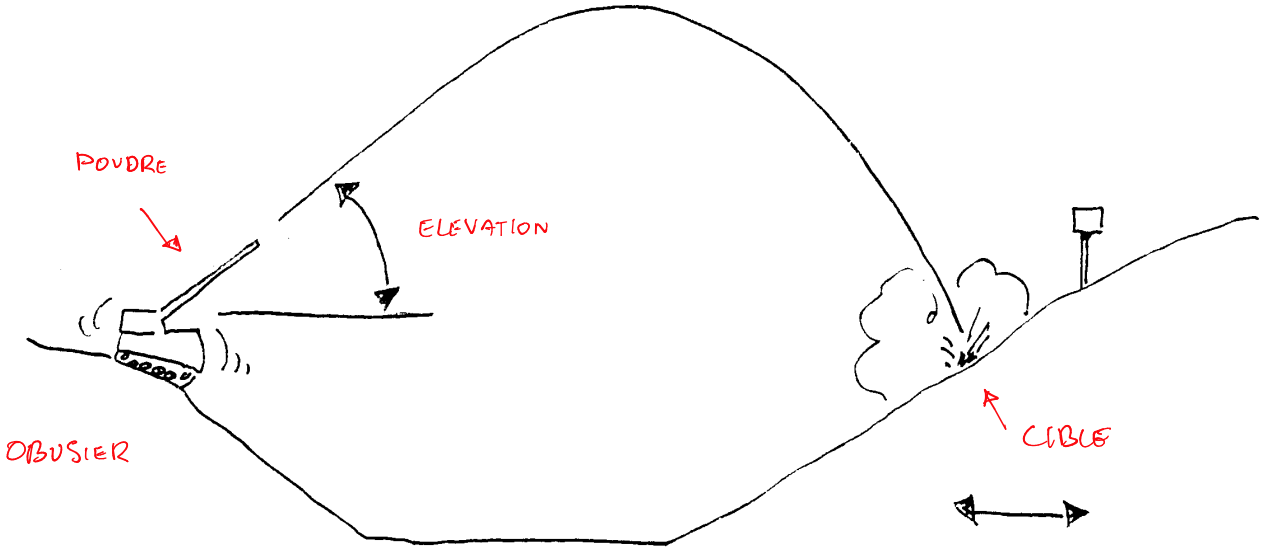
A whole civilisation
will die tonight…



Trump had promised
to end the war in Ukraine in a day

Obusier autopropulsé M109 155 mm

A quoi servent les méthodes numériques ?



Problèmes non linéaires

Fonction non linéaire

Trouver x tel que

$$f(x) = 0$$

Suite de candidats…

$$x_1, x_2, x_3, \ldots$$

… qui devrait converger vers une racine

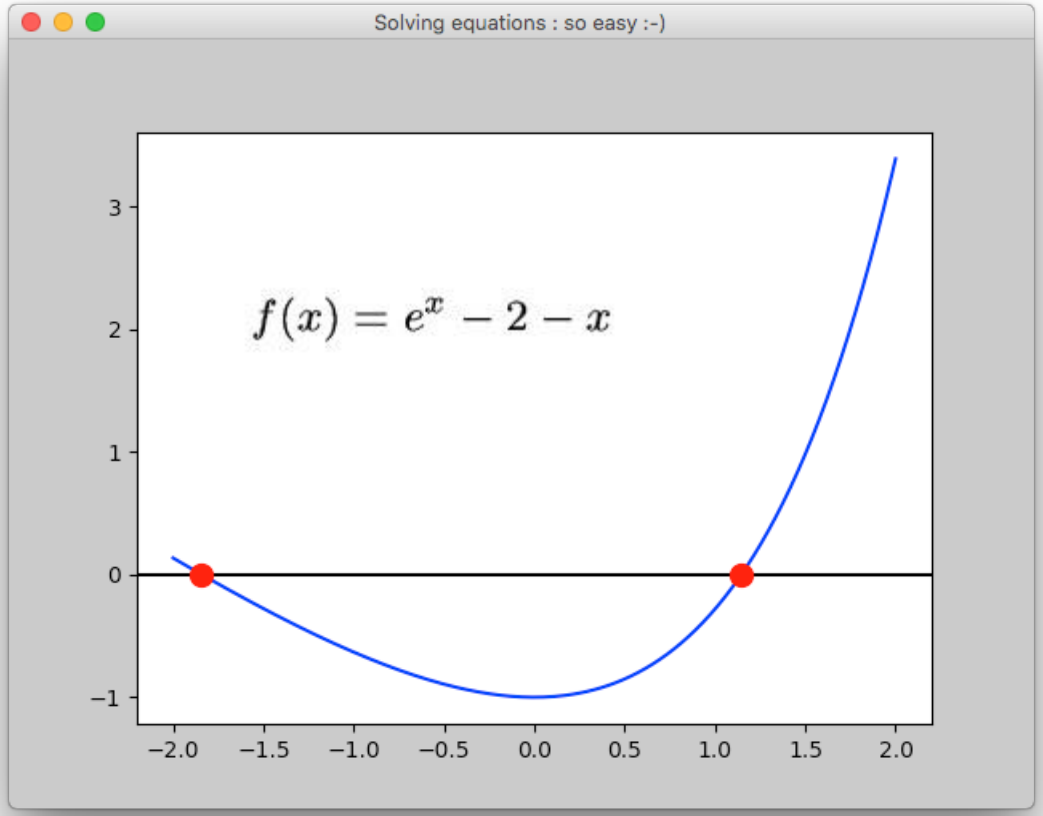
Questions

- Convergence vers une racine
- Candidat initial
- Encadrement

Méthodes numériques itératives

- Méthode de bissection
- Méthode du point fixe
- Méthode de Newton-Raphson

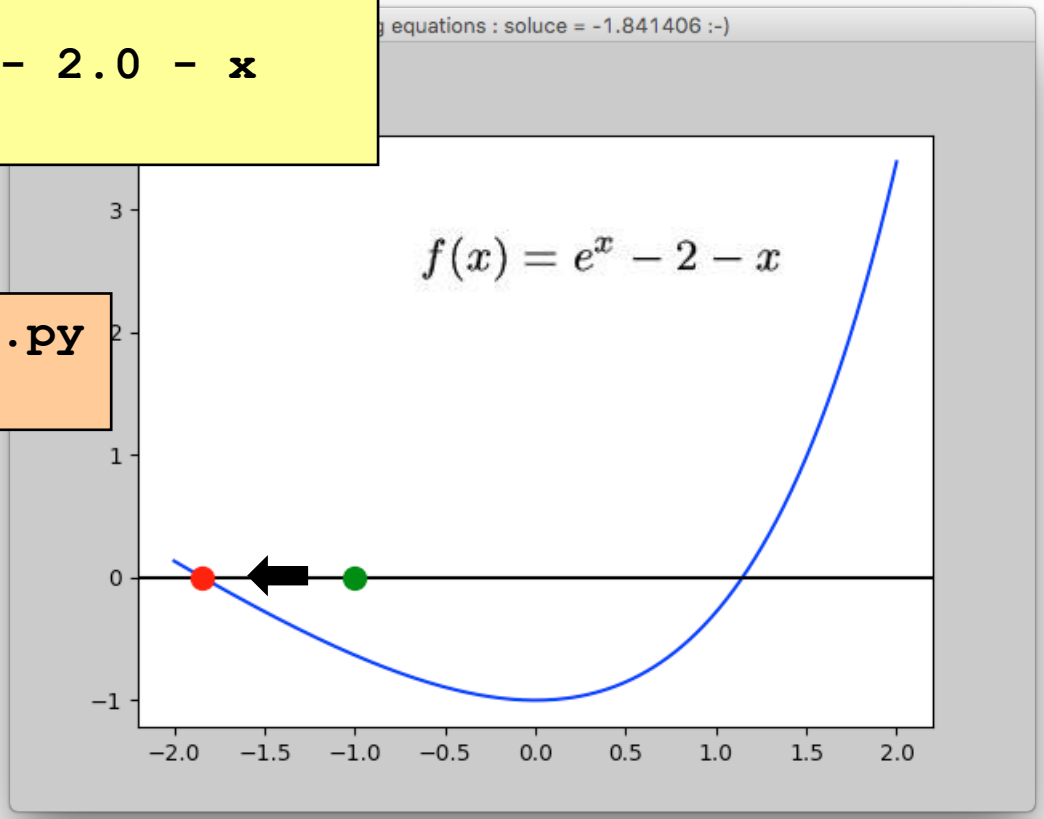
Exemple simple



Avec scipy, c'est très facile…

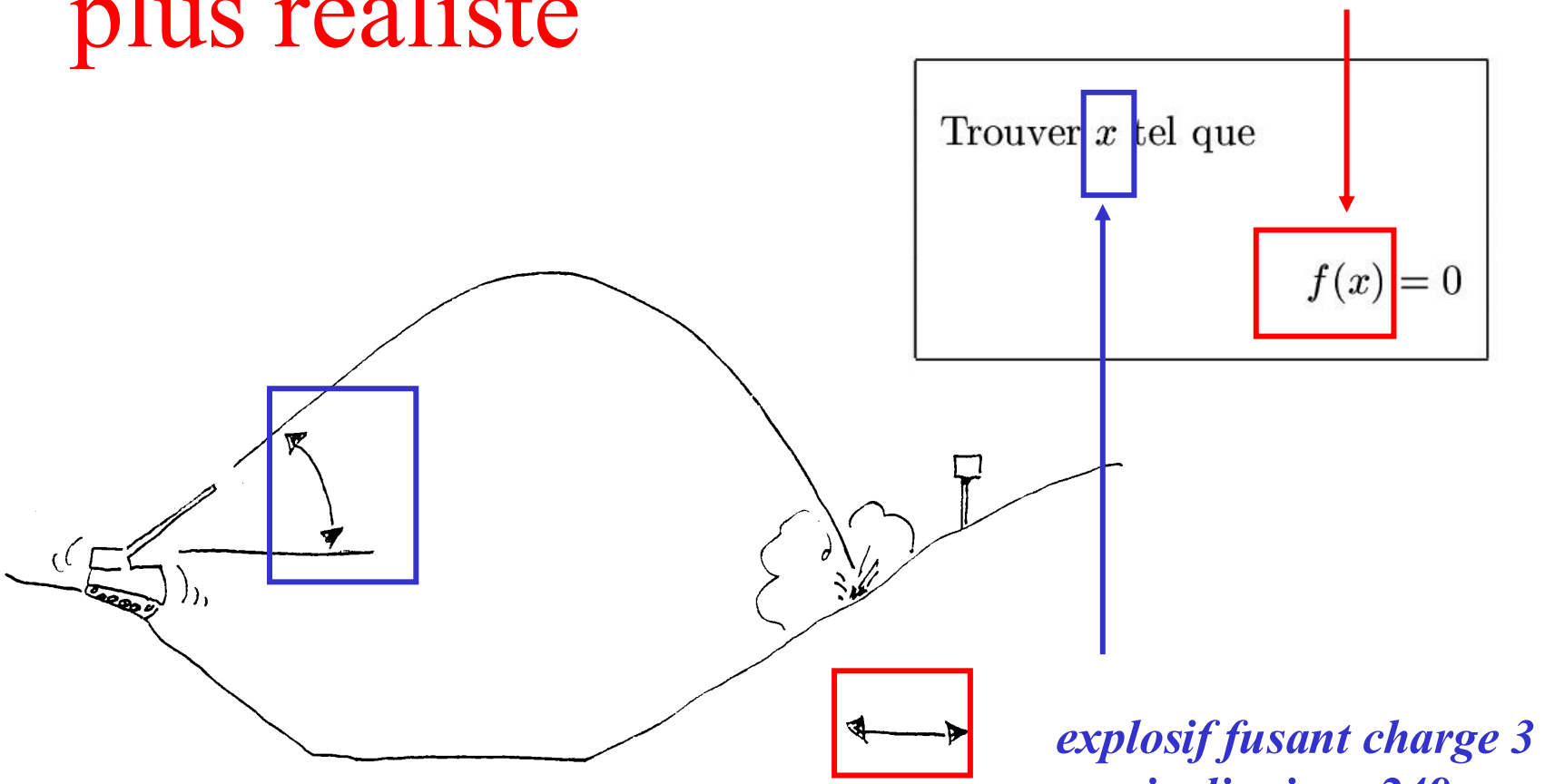
```
from math import exp
from scipy.optimize import fsolve

f = lambda x : exp(x) - 2.0 - x
print(fsolve(f,-1))
```

```
bash-3.2$ python solve.py
[-1.84140566]
```



Exemple plus réaliste

Distance entre l'explosion et la cible



explosif fusant charge 3
inclinaison 249
déplacement 4559

Comment prédire ou prévoir f ?

Trouver x tel que

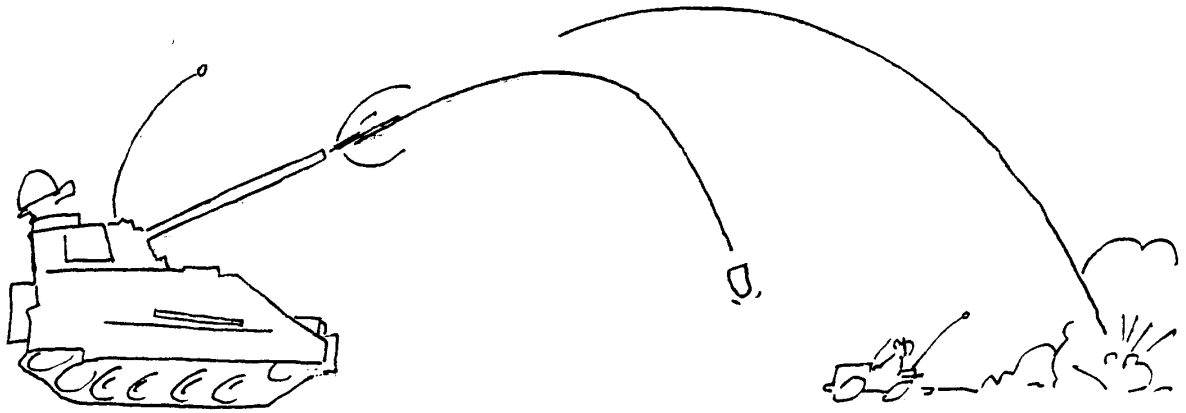
$$f(x) = 0$$

Trajectoire de l'obus = solution d'une équation différentielle ordinaire
Paramètres à tenir en compte :

- ***Charge de poudre, type d'obus,***
- ***Usure du tube (change après chaque coup !)***
- ***Calibrage de l'obusier (différent pour chaque pièce !)***
- ***Position calculée de la pièce d'artillerie (travail de topographie)***
- ***Position estimée de la cible (observateur avancé)***
- ***Vents dominants, humidité de l'air***
- ***Rotation de la terre***

On obtient donc f(x) en utilisant ode45…

Evidemment entre modèle et réalité…



Méthodes itératives

C constante positive et inférieure à l'unité

$$\lim_{i \to \infty} \frac{|e_i|}{|e_{i-1}|^r} = C$$

taux de convergence

$$e_i = x - x_i$$

La séquence converge si on peut trouver C et r

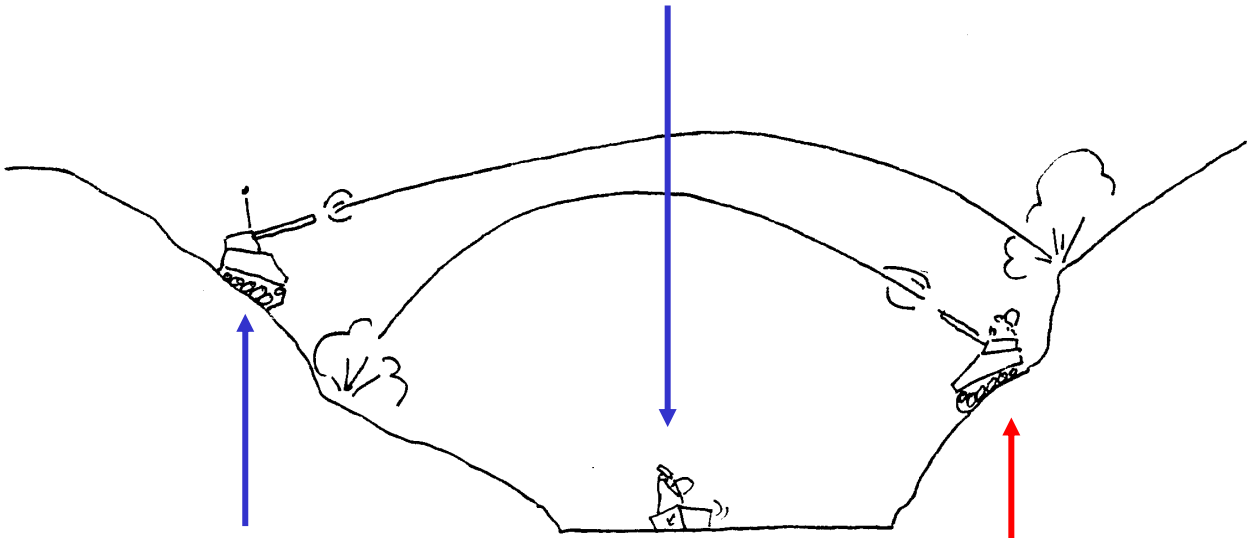
r = taux de convergence

r=1 taux de convergence linéaire

r=2 taux de convergence quadratique

Est-il important de converger rapidement ?

Observateur avancé ajustant le tir (espérance de vie très limitée en général…)



Obusier tirant sur l'obusier adverse…

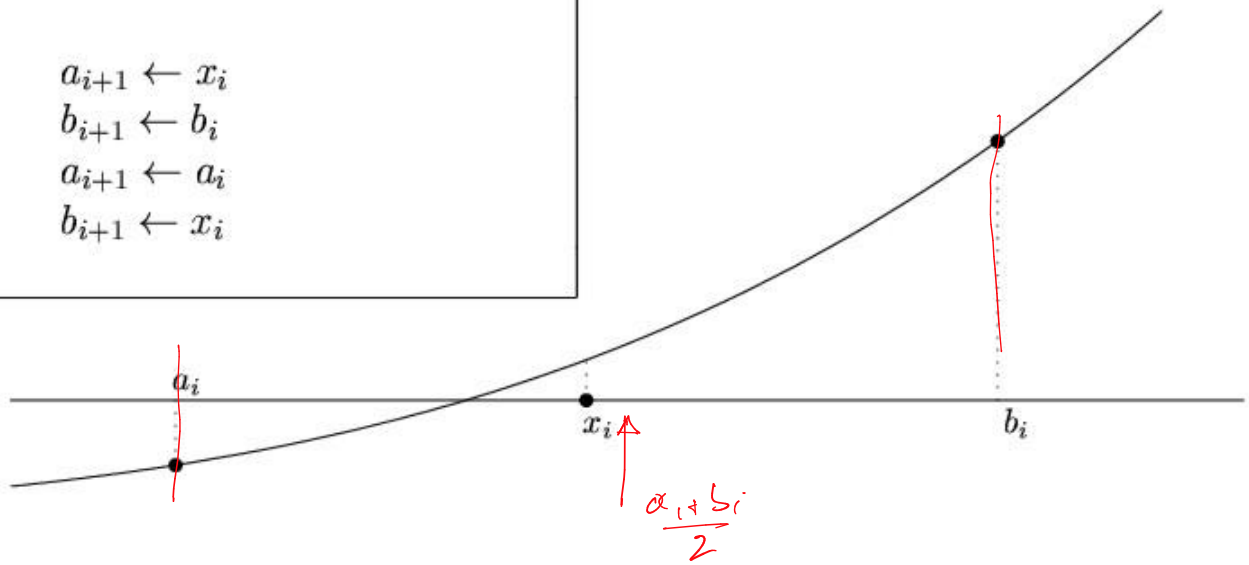
Obusier adverse tentant de vous détruire au même moment…

La technique de bissection est une méthode d'encadrement

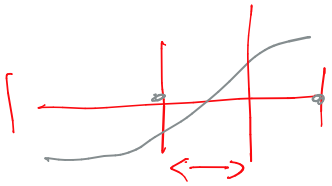
On fournit un intervalle $[a_0, b_0]$ avec $f(a_0)f(b_0) < 0$ et on calcule trois suites a_i, b_i, x_i par

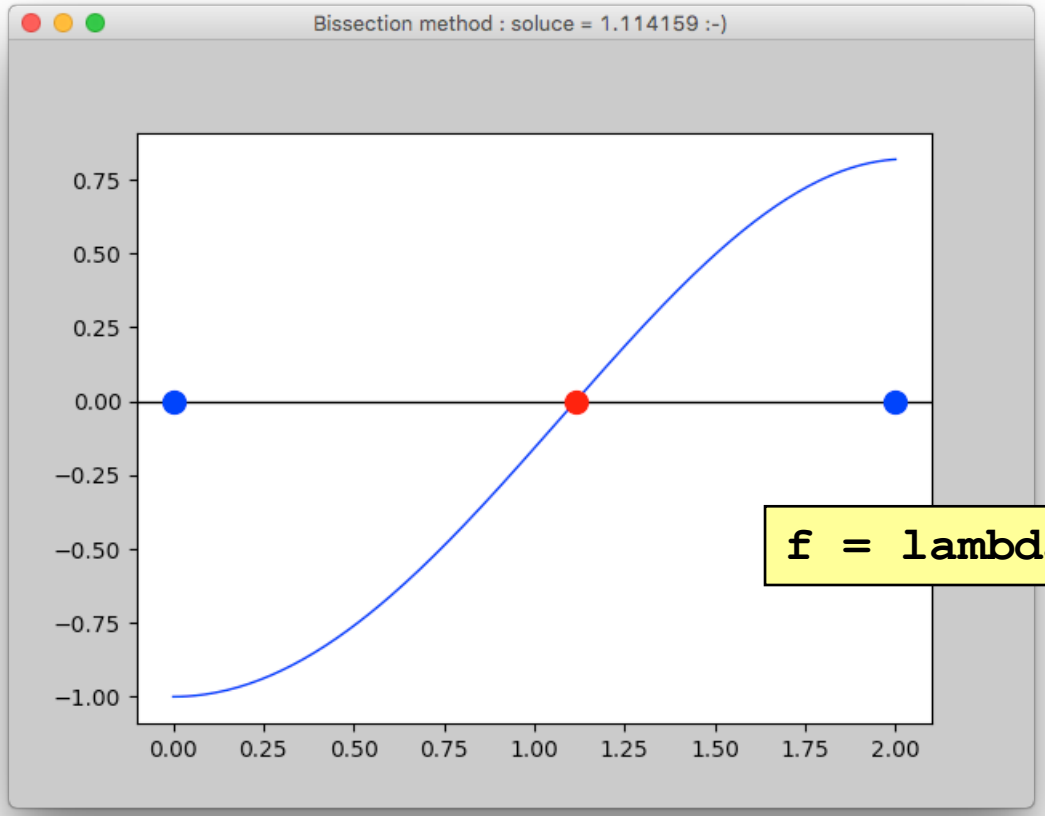
$$x_i = \frac{(a_i + b_i)}{2}$$

Si $f(x_i) = 0$	On a une solution !
Si $f(x_i)f(a_i) > 0$	$a_{i+1} \leftarrow x_i$ $b_{i+1} \leftarrow b_i$
Sinon	$a_{i+1} \leftarrow a_i$ $b_{i+1} \leftarrow x_i$



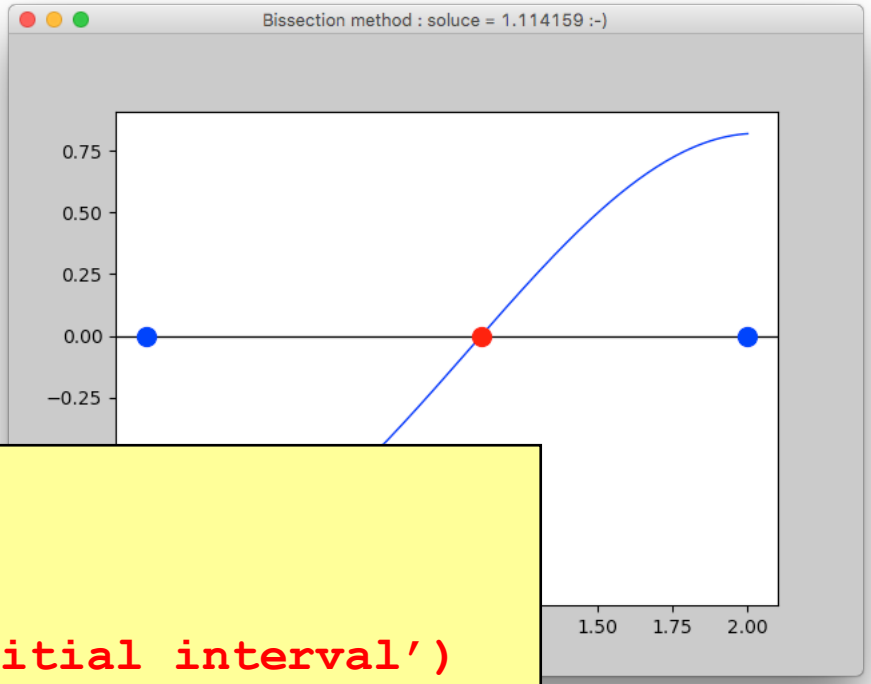
Exemple pour la méthode de bissection…






```
f = lambda x : x * sin(x) - 1
```

Programme python




```
def bissect(a,b,f,tol,nmax):
  n = 0; delta = (b-a)/2
  if (f(a)*f(b) > 0) :
    raise RuntimeError('Bad initial interval')
  while (abs(delta) >= tol and n <= nmax) :
    delta = (b-a)/2; n = n + 1;
    x = a + delta
    if (f(x)*f(a) > 0) :
      a = x
    else :
      b = x
  if (n > nmax) :
    raise RuntimeError('Too much iterations')
  return x
```

Une fioriture numérique…

```
x = a + (b-a)/2
```

```
>>> import numpy as np
>>> b = np.finfo(float).max
>>> a = b - 10**300
>>> (a+b)/2
inf
>>> a + (b-a)/2
1.7976931298623156e+308
>>> (b-a)/2
4.999999900185599e+299
```

…ou…

```
x = (a+b)/2
```

Et le résultat…

```
x =  1.0000000e+00 (Estimated error 1.0000000e+00 at iteration 1)
x =  1.5000000e+00 (Estimated error 5.0000000e-01 at iteration 2)
x =  1.2500000e+00 (Estimated error 2.5000000e-01 at iteration 3)
x =  1.1250000e+00 (Estimated error 1.2500000e-01 at iteration 4)
x =  1.0625000e+00 (Estimated error 6.2500000e-02 at iteration 5)
x =  1.0937500e+00 (Estimated error 3.1250000e-02 at iteration 6)
x =  1.1093750e+00 (Estimated error 1.5625000e-02 at iteration 7)
x =  1.1171875e+00 (Estimated error 7.8125000e-03 at iteration 8)
x =  1.1132812e+00 (Estimated error 3.9062500e-03 at iteration 9)
x =  1.1152344e+00 (Estimated error 1.9531250e-03 at iteration 10)
x =  1.1142578e+00 (Estimated error 9.7656250e-04 at iteration 11)
x =  1.1137695e+00 (Estimated error 4.8828125e-04 at iteration 12)
x =  1.1140137e+00 (Estimated error 2.4414062e-04 at iteration 13)
x =  1.1141357e+00 (Estimated error 1.2207031e-04 at iteration 14)
x =  1.1141968e+00 (Estimated error 6.1035156e-05 at iteration 15)
x =  1.1141663e+00 (Estimated error 3.0517578e-05 at iteration 16)
x =  1.1141510e+00 (Estimated error 1.5258789e-05 at iteration 17)
x =  1.1141586e+00 (Estimated error 7.6293945e-06 at iteration 18)
```

On observe un taux de convergence linéaire…

Taux de convergence linéaire

$$|e_0| \leq \frac{b_0 - a_0}{2}$$

En effectuant une nouvelle itération

$$|e_1| \leq \frac{b_1 - a_1}{2} = \frac{b_0 - a_0}{2^2}$$

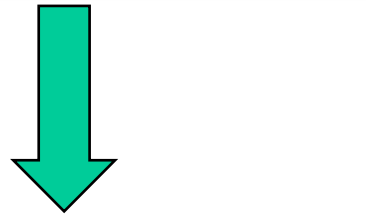
$$|e_n| \leq \frac{b_0 - a_0}{2^{n+1}}$$

$$|e_i| \leq \frac{1}{2}|e_{i-1}|$$

Problème aux conditions aux limites

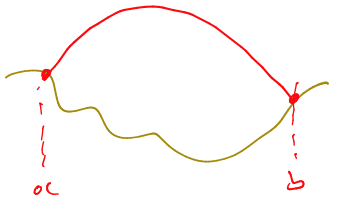
Trouver $u(x)$ tel que

$$\begin{cases} u''(x) = f(x, u(x)), & x \in [a, b] \\ u(a) = u^a \\ u(b) = u^b \end{cases}$$



Trouver $(u(x), v(x))$ tels que

$$\begin{cases} u'(x) = v(x) \\ v'(x) = f(x, u(x)), & x \in [a, b] \\ u(a) = u^a \\ u(b) = u^b \end{cases}$$



$m x''(t) = -C x'(t)$

$m y''(t) = -C y'(t) - mg$

Première idée :
Méthode du tir

$$r(v^a) = 0$$

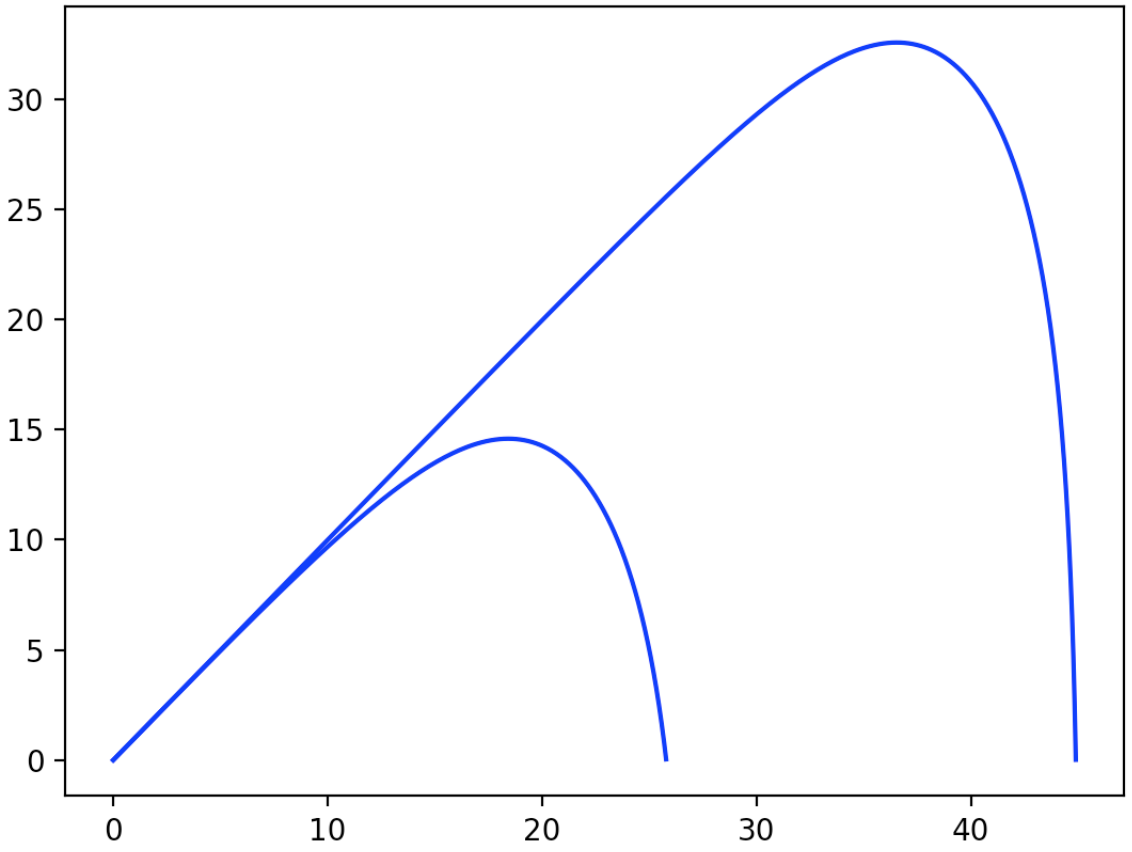
C'est la technique de l'artilleur

Technique d'ajustement
par essais et erreurs…
et une méthode d'encadrement

$$r \; : \; (v^a) \; \rightarrow \; u^b - u^h(b, v^a)$$

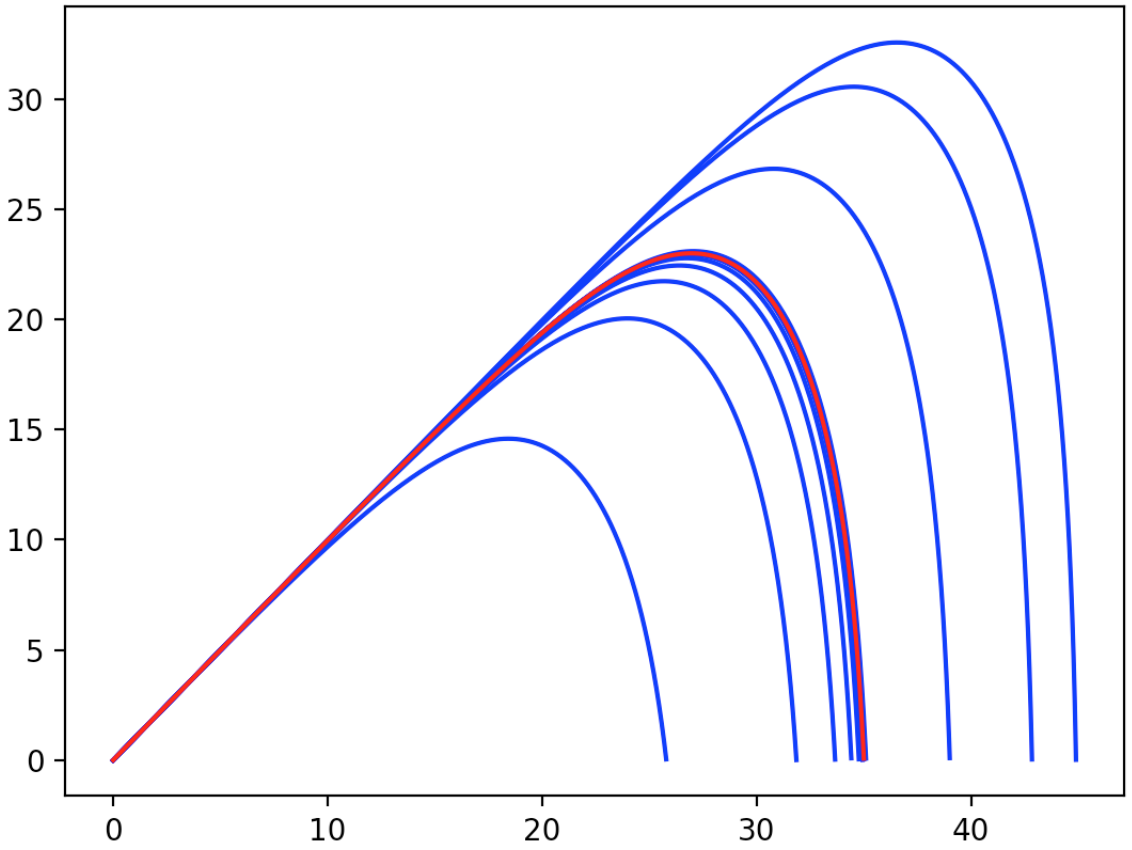
Pour la trajectoire d'un vrai obus…

```
def f(u):

  friction = 0.1 * sqrt(u[1]*u[1]+u[3]*u[3])
  mass = 1

  dxdt = u[1]
  dudt = - (friction * u[1]) / mass
  dydt = u[3]
  dvdt = - (friction * u[3]) / mass - 9.81

  return array([dxdt,dudt,dydt,dvdt])
```



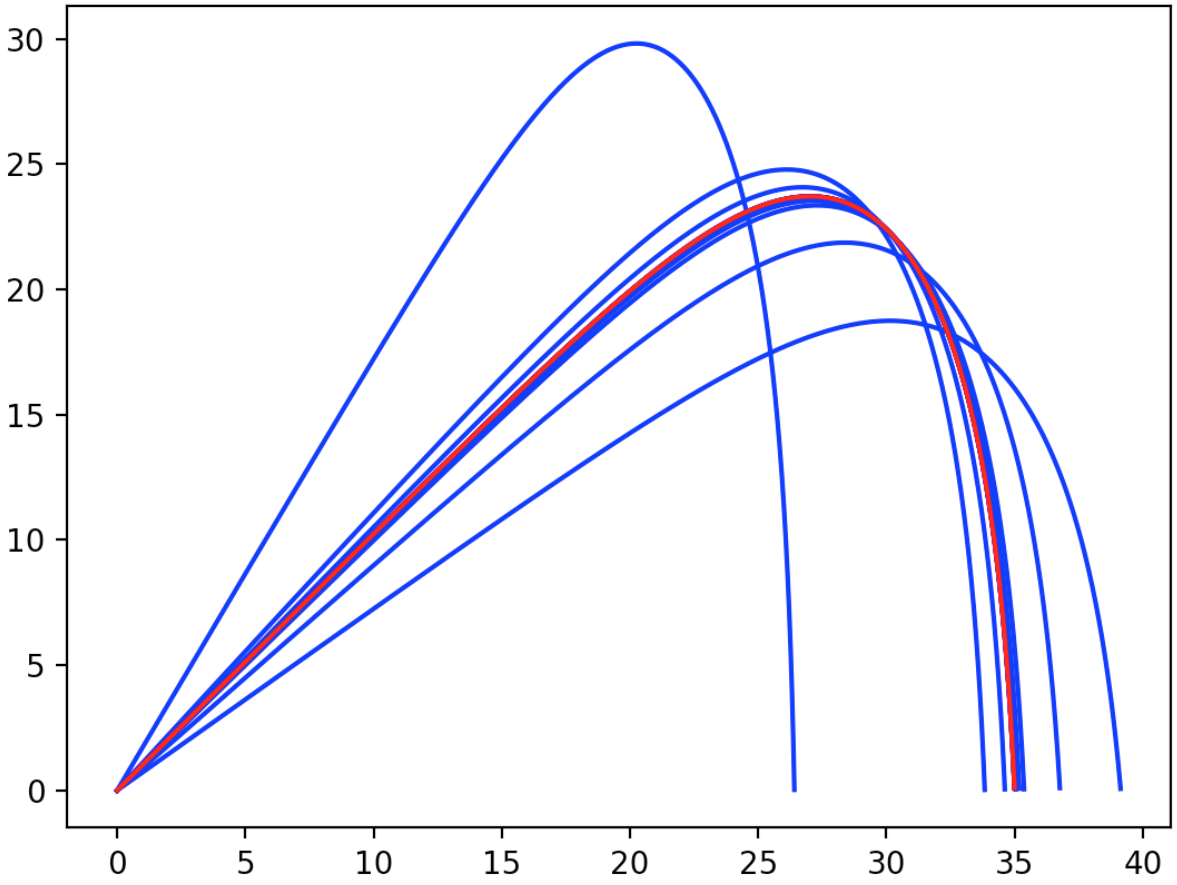
Ajustons la puissance du tir…

Utilisons la technique bissection pour ajuster la vitesse initiale de l'obus.



Ajustons la puissance du tir…

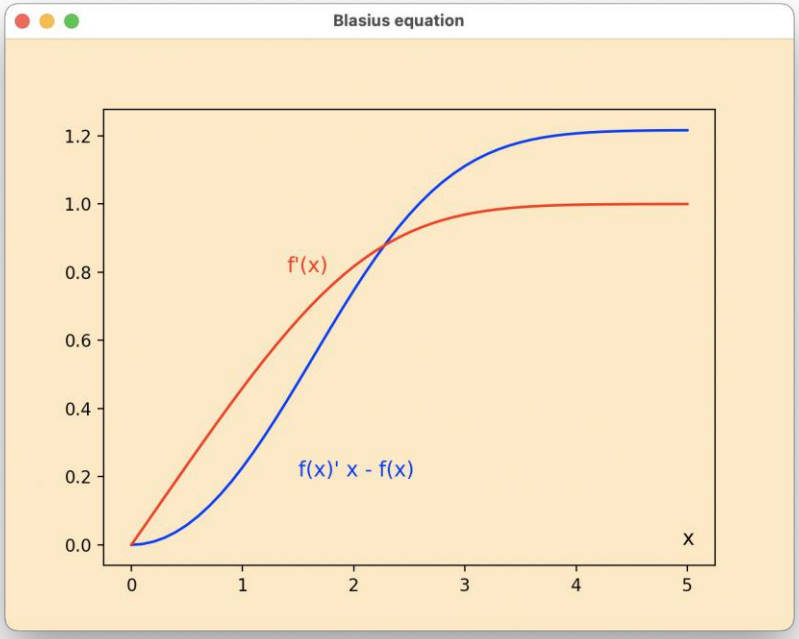
Utilisons la technique bissection pour ajuster la vitesse initiale de l'obus.



Ajustons l'angle du tir…

Utilisons la technique bissection pour ajuster l'inclinaison du tube.

$$\begin{cases} f'''(x) + f(x)f''(x) &= 0 \\ \\ f(0) &= 0 \\ f'(0) &= 0 \\ \lim_{x\to\infty} f'(x) &= 1 \end{cases}$$



Equation de Blasius

Seconde idée : Différences finies

$$u(X_i) \approx u^h(X_i) = U_i$$

C'est la vision satellitaire

Synthèse globale des conditions limites par la résolution d'un système d'équations

$$0 = u''(X_i) - f(X_i, U_i) \approx (u^h)''(X_i) - f(X_i, U_i)$$

En utilisant une différence finie centrée d'ordre deux,

$$\approx \frac{(U_{i-1} - 2U_i + U_{i+1})}{h^2} - f(X_i, U_i)$$

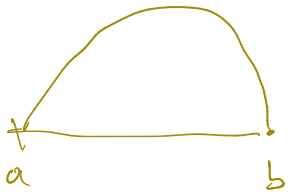
Il est alors nécessaire de résoudre un système…

$$\frac{1}{h^2}\begin{bmatrix} h^2 & 0 & & & & & & & \\ 1 & -2 & 1 & & & & & & \\ & 1 & -2 & 1 & & & & & \\ & & 1 & -2 & 1 & & & & \\ & & & 1 & -2 & 1 & & & \\ & & & & \cdot & \cdot & \cdot & & \\ & & & & & 1 & -2 & 1 & \\ & & & & & & 1 & -2 & 1 \\ & & & & & & & 0 & h^2 \end{bmatrix} \begin{bmatrix} U_0 \\ U_1 \\ U_2 \\ U_3 \\ U_4 \\ \vdots \\ U_{m-2} \\ U_{m-1} \\ U_m \end{bmatrix} = \begin{bmatrix} u^a \\ f(X_1, U_1) \\ f(X_2, U_2) \\ f(X_3, U_3) \\ f(X_4, U_4) \\ \vdots \\ \vdots \\ f(X_{m-1}, U_{m-1}) \\ u^b \end{bmatrix}$$

Système linéaire si f linéaire en u
Système non-linéaire si f non-linéaire en u

Exemple (linéaire)

$$u''(x) = x - u(x)$$



Deux constantes à choisir pour satisfaire les deux conditions aux limites !

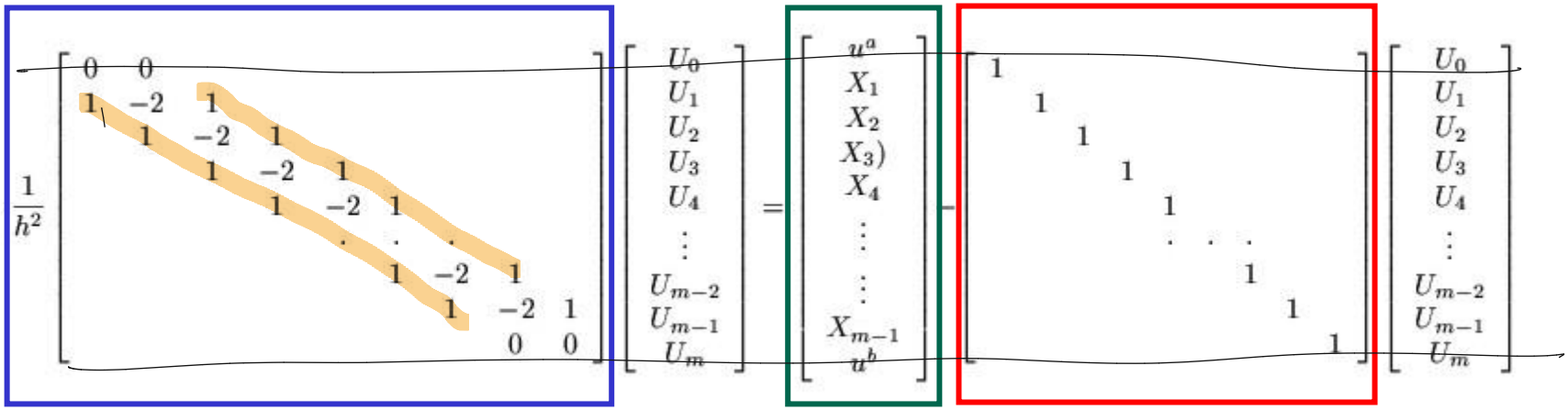
$$u(x) = x + Acos(x) + Bsin(x)$$

Solution analytique

$$\frac{U_{i-1} - 2U_i + U_{i+1}}{h^2} = X_i + U_i$$

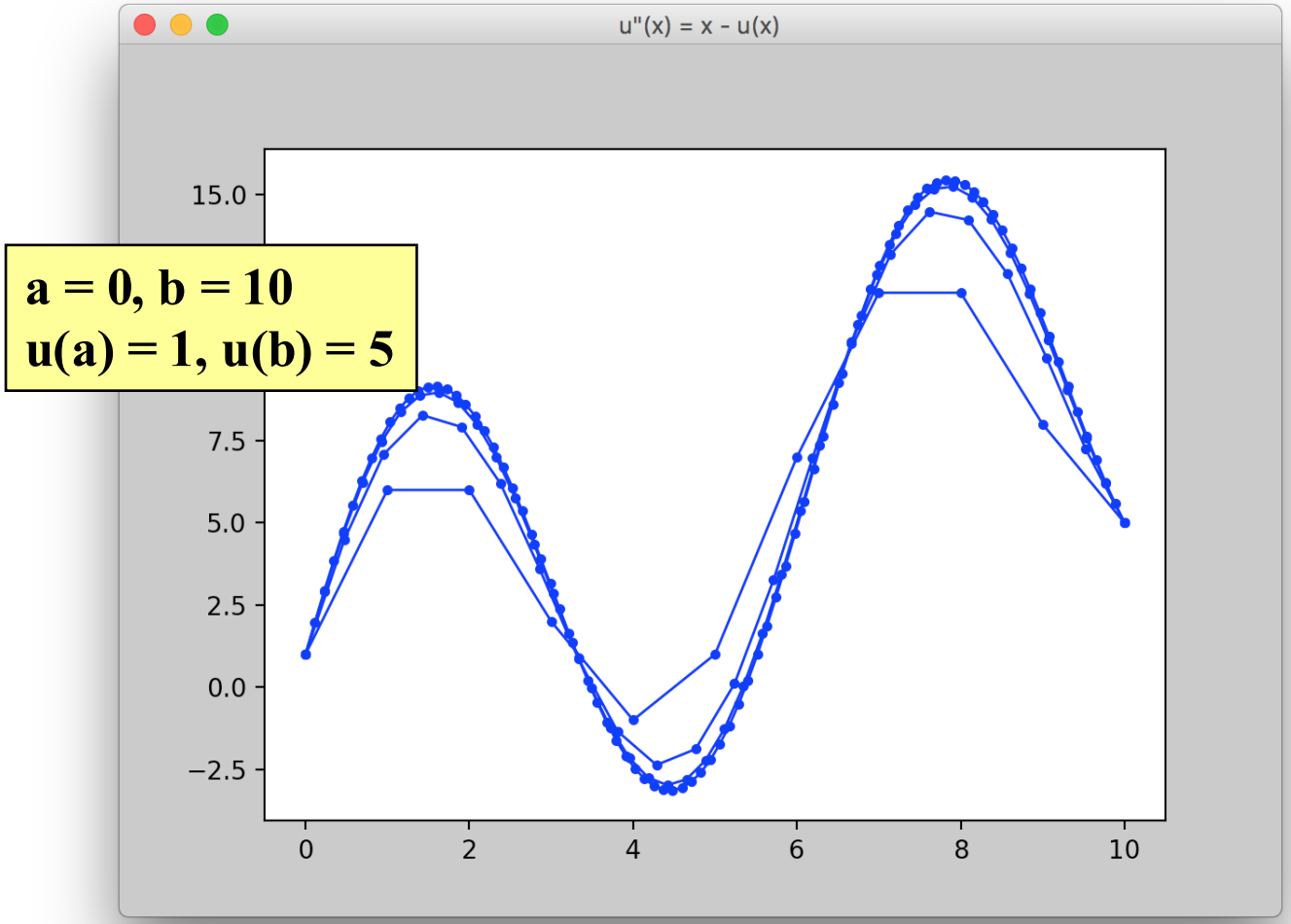
$$\underline{\underline{A}}\, x = b$$

Problème numérique

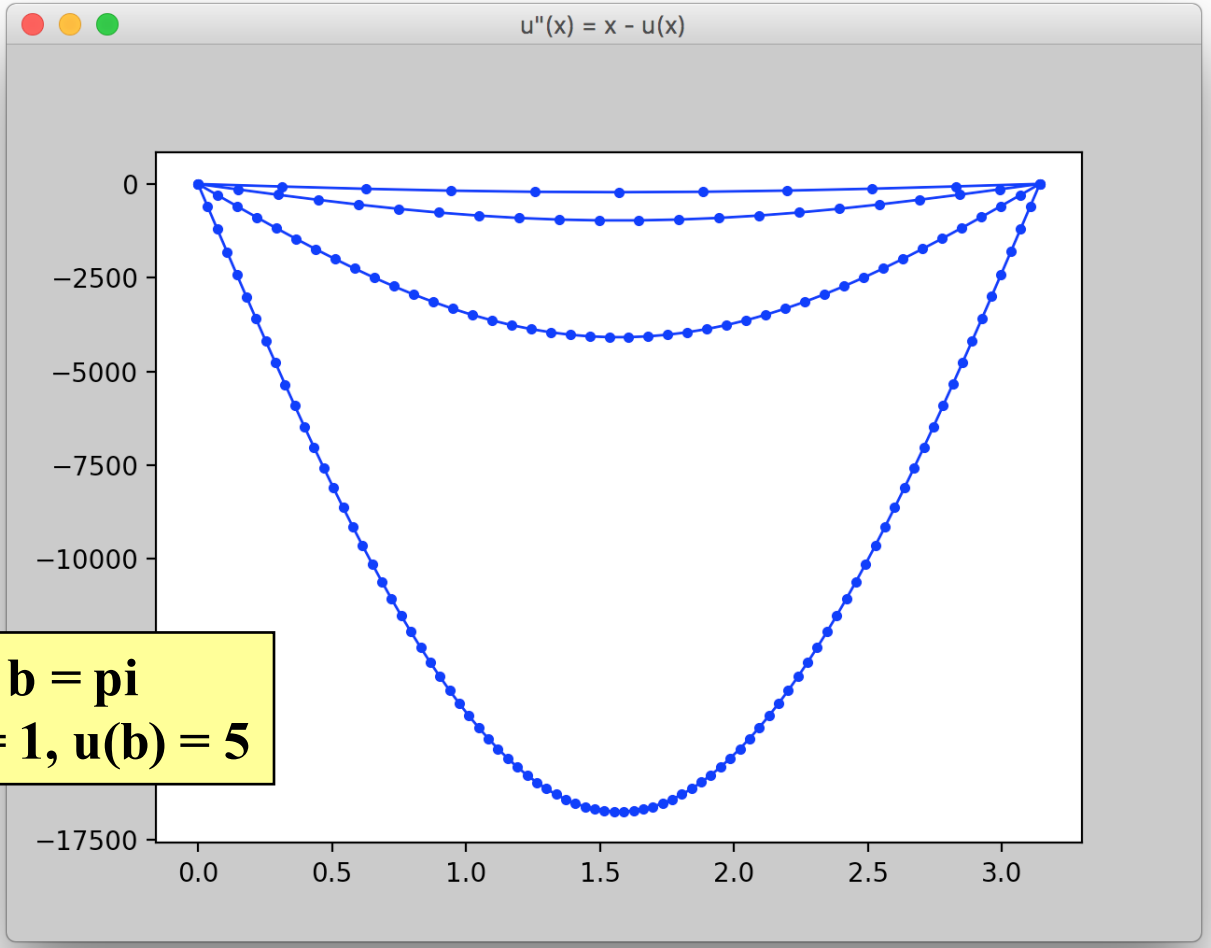


```
X = linspace(a,b,n)
h = (b-a)/(n-1)

b = array([0,*ones(n-2),0])
A = spdiags([b,-2*b,b],[-1,0,1],n,n)/(h*h)
B = copy(X); B[0] = Ua; B[-1] = Ub

U = spsolve(A.T + eye(n,n),B)
```

Et hop, un joli résultat :-)

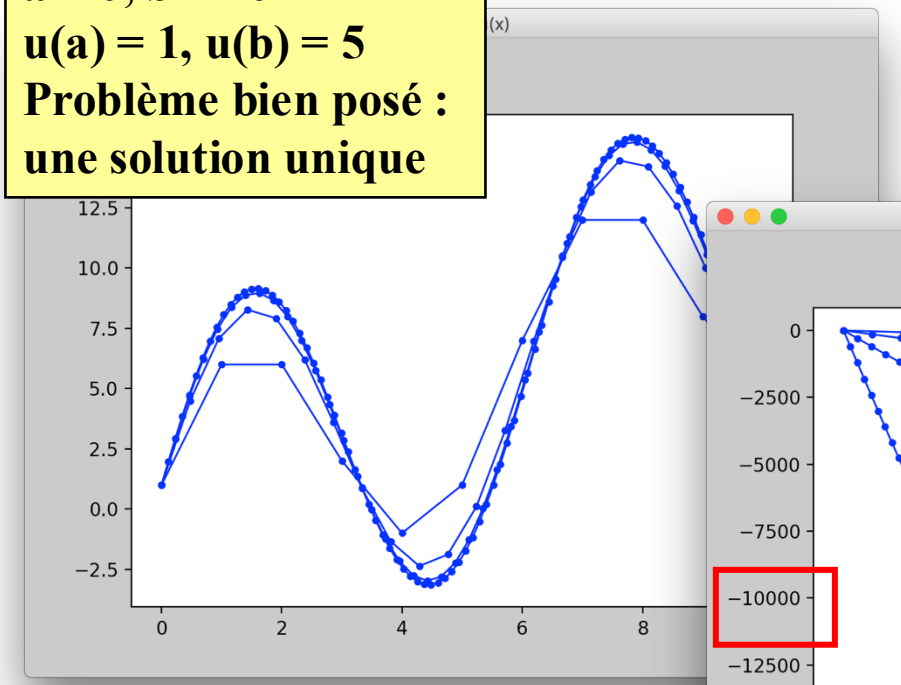


Un autre joli résultat :-(

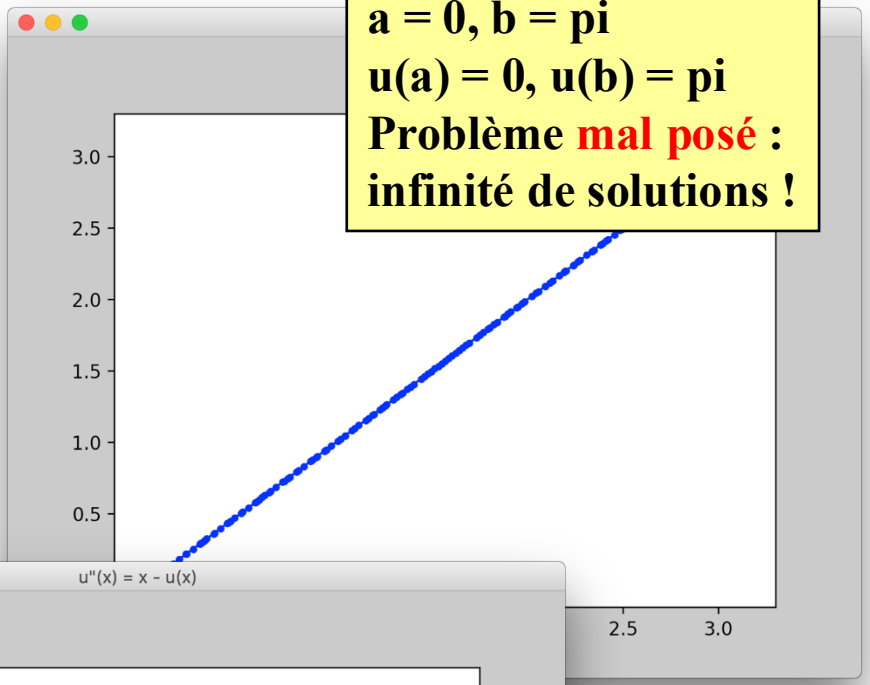


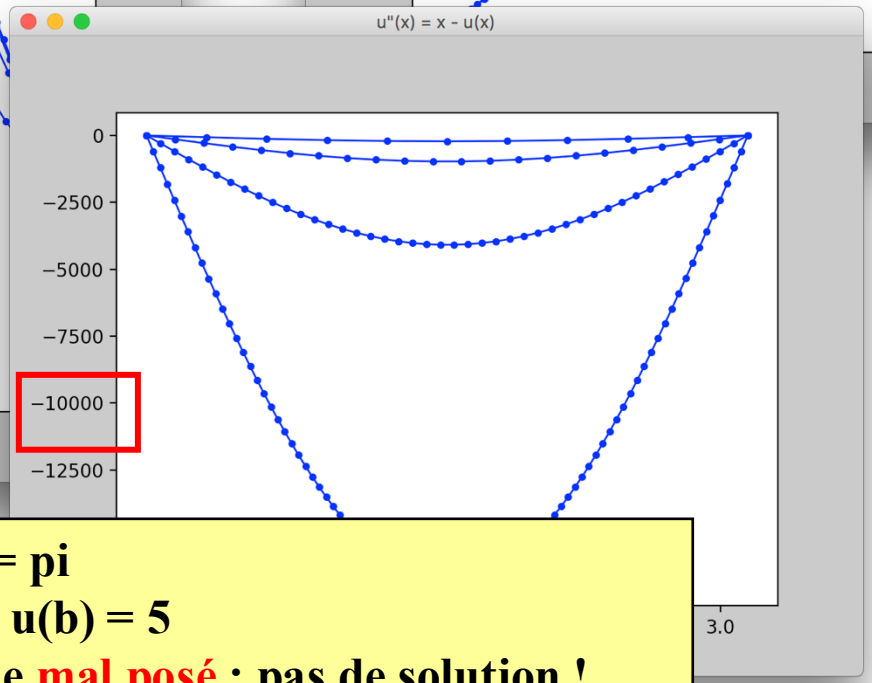
Est-ce que cela marche toujours ?

a = 0, b = 10
u(a) = 1, u(b) = 5
Problème bien posé :
une solution unique



a = 0, b = pi
u(a) = 0, u(b) = pi
Problème mal posé :
infinité de solutions !



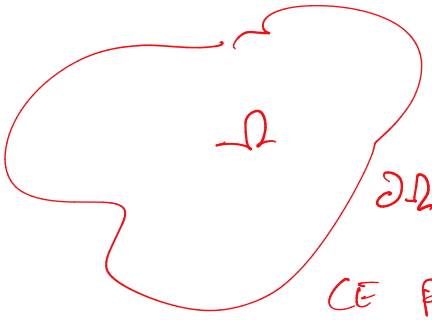


a = 0, b = pi
u(a) = 1, u(b) = 5
Problème mal posé : pas de solution !
Les solutions obtenues n'ont aucun sens !

EQUATION
DE POISSON

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = f(x,y) \quad \text{sur } \Omega$$

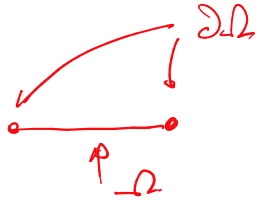
$$u(x,y) = 0 \quad \text{sur } \partial\Omega$$



CE PROBLEME
A TOUJOURS
UNE SOLUTION

$$u''(x) = f(x)$$

$$u(0) = 0$$
$$u(1) = 1$$



CE PROBLEME
N'EST PAS
TJRS BIEN
POSE

1D et …. 2D !

Pas le même monde !

Problèmes non linéaires

Fonction non linéaire

Trouver x tel que

$$f(x) = 0$$

Suite de candidats…

$$x_1, x_2, x_3, \ldots$$

… qui devrait converger vers une racine

Questions

- Convergence vers une racine
- Candidat initial
- Encadrement

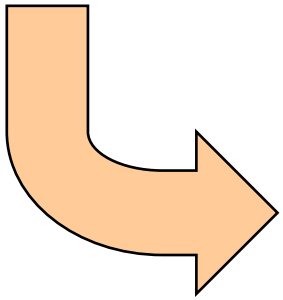
Méthodes numériques itératives

- Méthode de bissection
- Méthodes du point fixe
- Méthode de Newton-Raphson

Méthode du point fixe

Trouver x tel que

$$f(x) = 0$$



On reformule le problème…

Il existe plein de possibilités de choix pour g !

Trouver x tel que

$$x = g(x)$$

Méthode du point fixe

On fournit x_0

Tant que $|\Delta x| > \epsilon$, on calcule x_{i+1} à partir de x_i avec

$$x_{i+1} = g(x_i)$$

Si on converge, la solution x est le dernier x_{i+1} calculé

$$x_{i+1} = g(x_i)$$

Comment « bien » définir g à partir de f ?

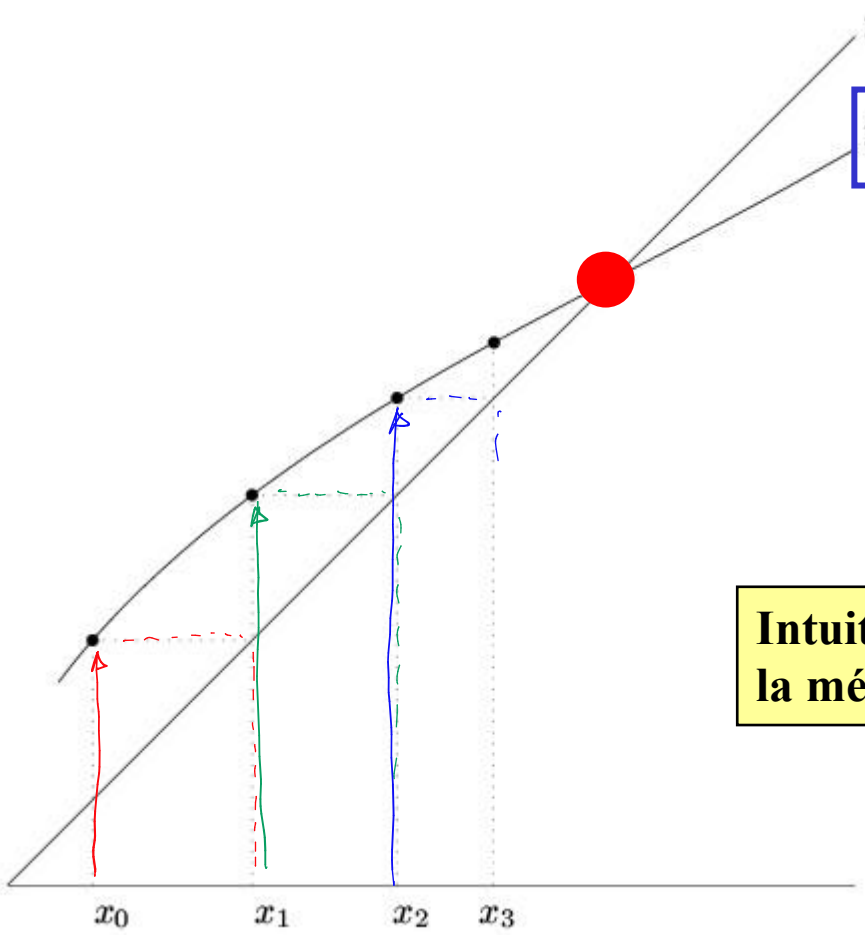
Réécrivons l'équation $f(x) = 0$ sous une forme[2]

[2]Il existe une infinité de façons d'écrire une équation $f(x) = 0$ sous une forme $x = g(x)$. A titre d'exemple considérons $f(x) = x^3 - 3x + 1$

On peut ainsi écrire $x = \dfrac{(x^3 + 1)}{3}$, ou $x = (3x - 1)^{1/3}$ ou encore $x = x - x^3 + 3x - 1$...

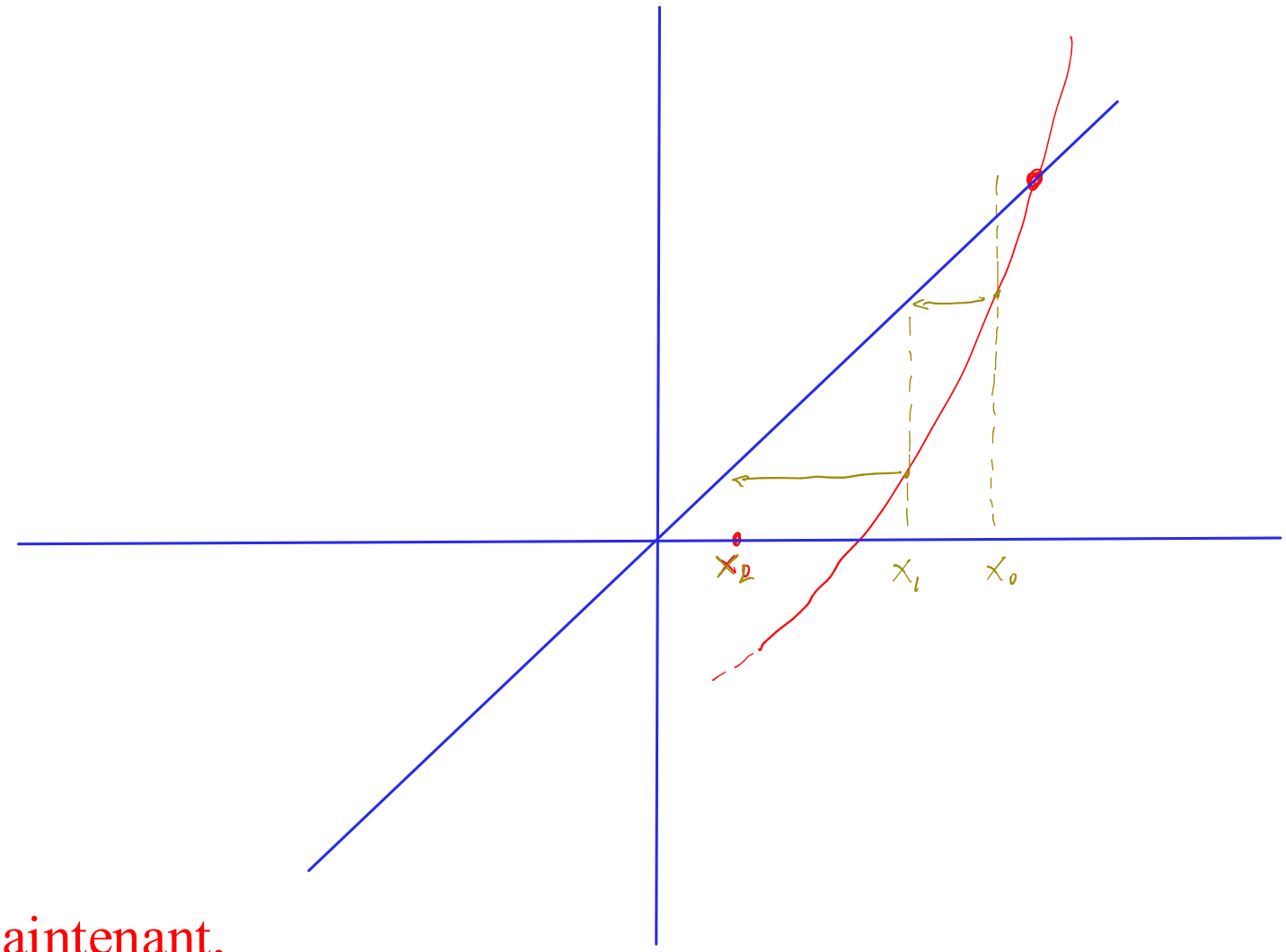
C'est une bonne question…
On va s'y intéresser d'ici peu

Interprétation géométrique



$$x_{i+1} = g(x_i)$$

Intuitivement, on observe que
la méthode converge si |g'(x)| < 1



Et maintenant,
cela ne marche plus !

Exemple

$$\underbrace{x^3 - 3x + 1}_{f(x)} = 0$$

$$x = \underbrace{\frac{x^3+1}{3}}_{g(x)}$$

$$x = \overbrace{x^3 - 2x + 1}^{g(x)} + x$$

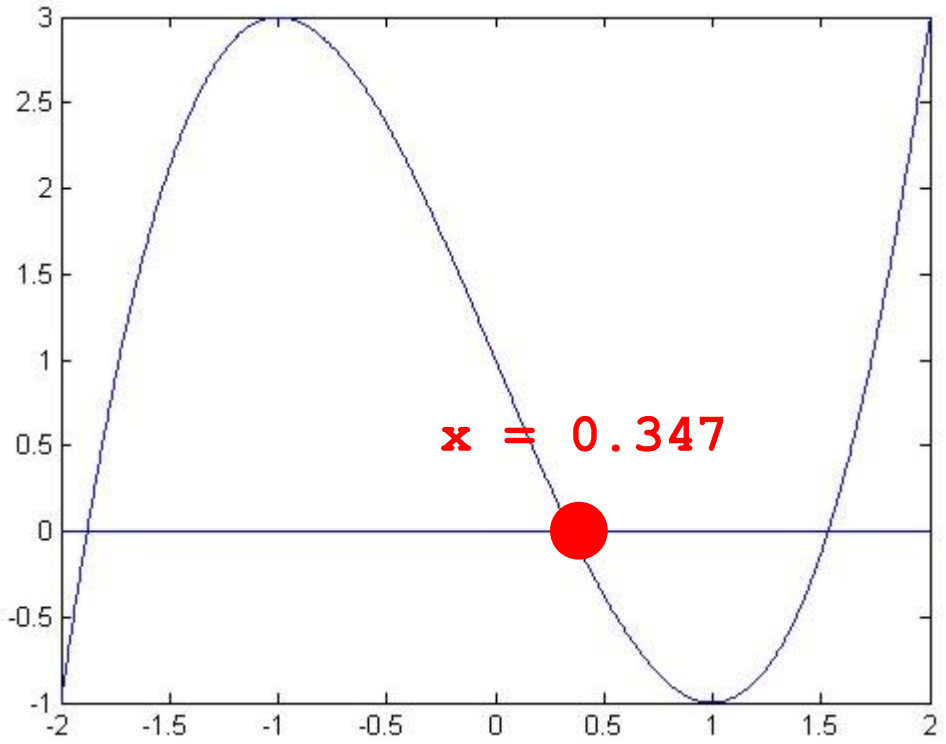
$$x = \underbrace{x^3 - 3x + 1}_{=0} + x$$

$$x = \underbrace{-x^3 + 3x - 1}_{=0} + x$$

$$\underbrace{-x^3 + 4x - 1}_{g(x)}$$

$$x = \underbrace{\sqrt[3]{3x-1}}_{g(x)}$$

Exemple

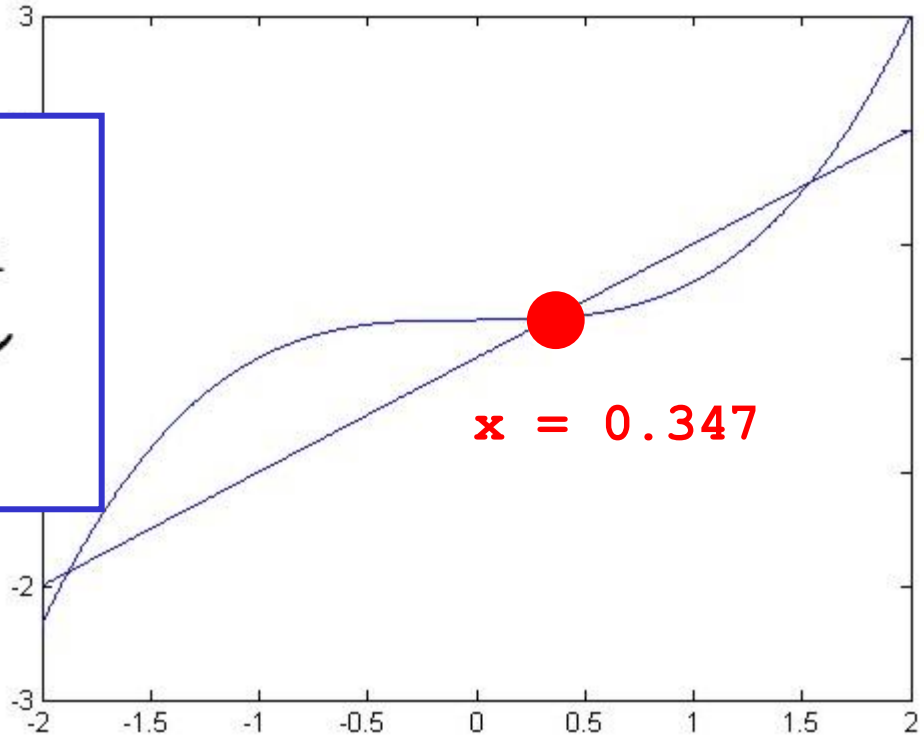


$$\underbrace{x^3 - 3x + 1}_{f(x)} = 0$$

Une itération possible…

$$\underbrace{x^3 - 3x + 1}_{f(x)} = 0$$

$$x_{i+1} = \underbrace{\frac{x_i^3 + 1}{3}}_{g(x_i)}$$



Une implémentation possible…

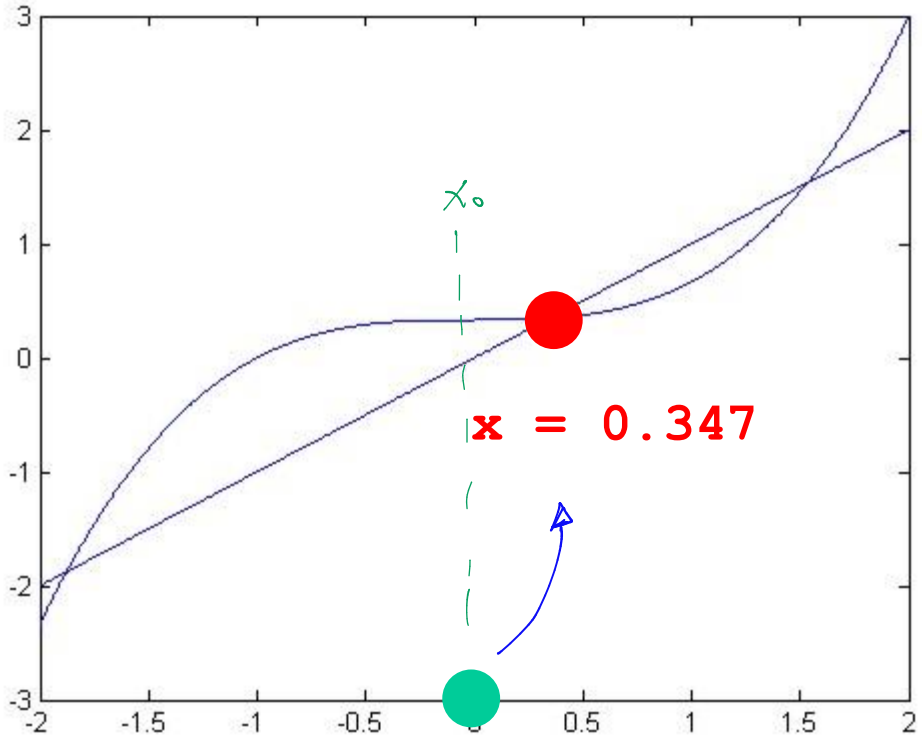
```
def iter(x,tol,nmax,g):
  n = 0; delta = float("inf")
  while abs(delta) > tol and n < nmax :
    n = n + 1
    xold = x
    x = g(x)
    delta = xold - x
    print(" x = %21.14e (%d)" % (x,n))
  return x
```

```
g = lambda x : (x**3 + 1)/3
```

$$x_{i+1} = \underbrace{\frac{x_i^3 + 1}{3}}_{g(x_i)}$$

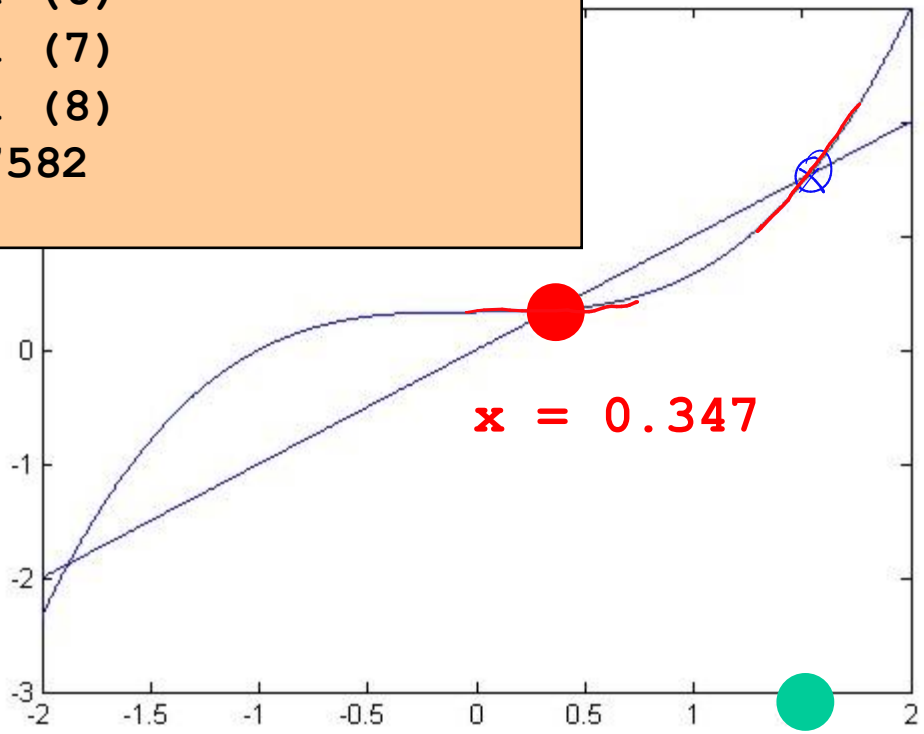
```
>>> print("Found x = ", iter(0.0,10e-3,50,g))
 x =  3.3333333333333e-01 (1)
 x =  3.4567901234567e-01 (2)
 x =  3.4710218694706e-01 (3)
Found x =  0.3471021869470616
```

Essayons…



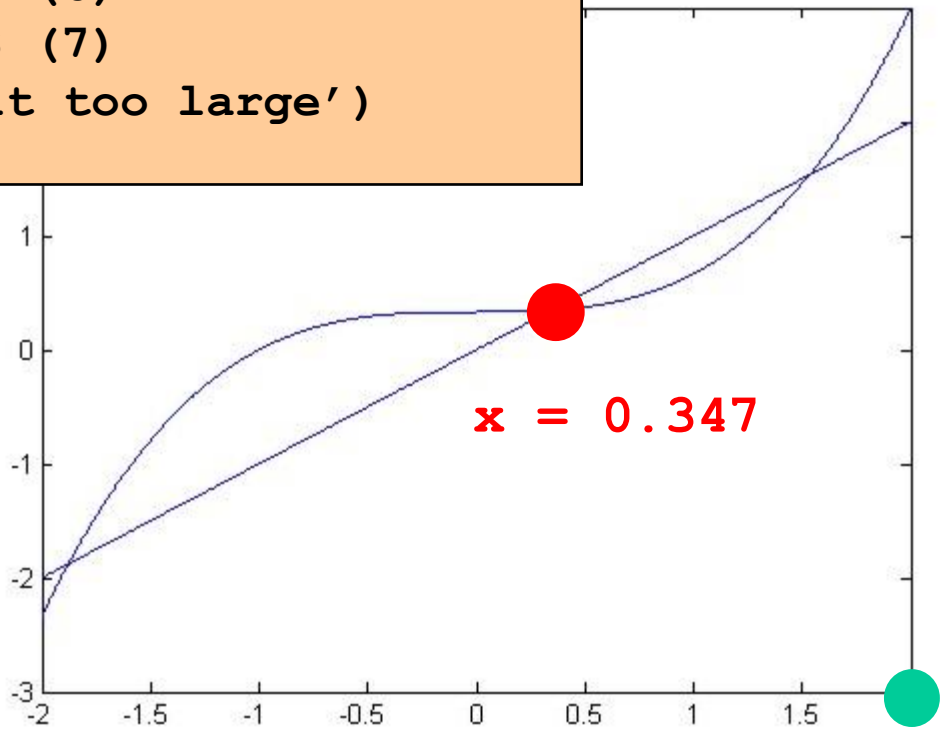
```
>>> print("Found x = ", iter(1.5,10e-3,50,g))
 x =  1.4583333333333e+00 (1)
 x =  1.3671633873456 8e+00 (2)
 x =  1.1851379776297 1e+00 (3)
 x =  8.8195982531111e-01 (4)
 x =  5.6689693229218 2e-01 (5)
 x =  3.9406162522186 4e-01 (6)
 x =  3.5373056261596 7e-01 (7)
 x =  3.4808688221075 8e-01 (8)
Found x =  0.34808688221075 82
```

Et en partant d'un autre point…



```
>>> print("Found x = ", iter(2.0,10e-3,50,g))
 x =  3.00000000000000e+00 (1)
 x =  9.33333333333333e+00 (2)
 x =  2.71345679012346e+02 (3)
 x =  6.65959007564967e+06 (4)
 x =  9.84512506785963e+19 (5)
 x =  3.18084464276016e+59 (6)
 x = 1.07276876343287e+178 (7)
OverflowError: (34, 'Result too large')
```

Et encore plus loin…



Et de manière plus rigoureuse ?

Théorème 5.1.

Supposons que $g(x)$ et $g'(x)$ sont continues sur l'intervalle $[a, b]$ qui contient le point fixe unique x de la fonction g. Si la valeur de départ x_0 est choisie dans cet intervalle et si la condition suivante (dite condition de Lipschitz) est satisfaite

$$|g'(x)| \leq K < 1 \qquad \forall x \in [a, b],$$

alors l'itération $x_{i+1} = g(x_i)$ converge vers x.

Pour les fonctions lipschitziennes…

$$x_{i+1} = g(x_i)$$

$$x = g(x)$$

$$x_{i+1} - x = g(x_i) - g(x)$$

e_{i+1} ; $g'(\xi)(x_i - x)$; e_i

$$|e_{i+1}| \leq K |e_i| \qquad |g'(\xi)| \leq K < 1$$

$$\lim_{i \to \infty} |e_{i+1}| \leq \lim_{i \to 0} K^i |e_0|$$

$\lim_{i \to 0} K^i \to 0$

Pour les fonctions lipschitziennes…

$$\underbrace{(x_{i+1} - x)}_{e_{i+1}} = g(x_i) - g(x)$$

$\downarrow$ En vertu du théorème de la moyenne.

$$= g'(\xi)\ (x_i - x)$$

$\downarrow$ En vertu de la condition de Lipschitz.

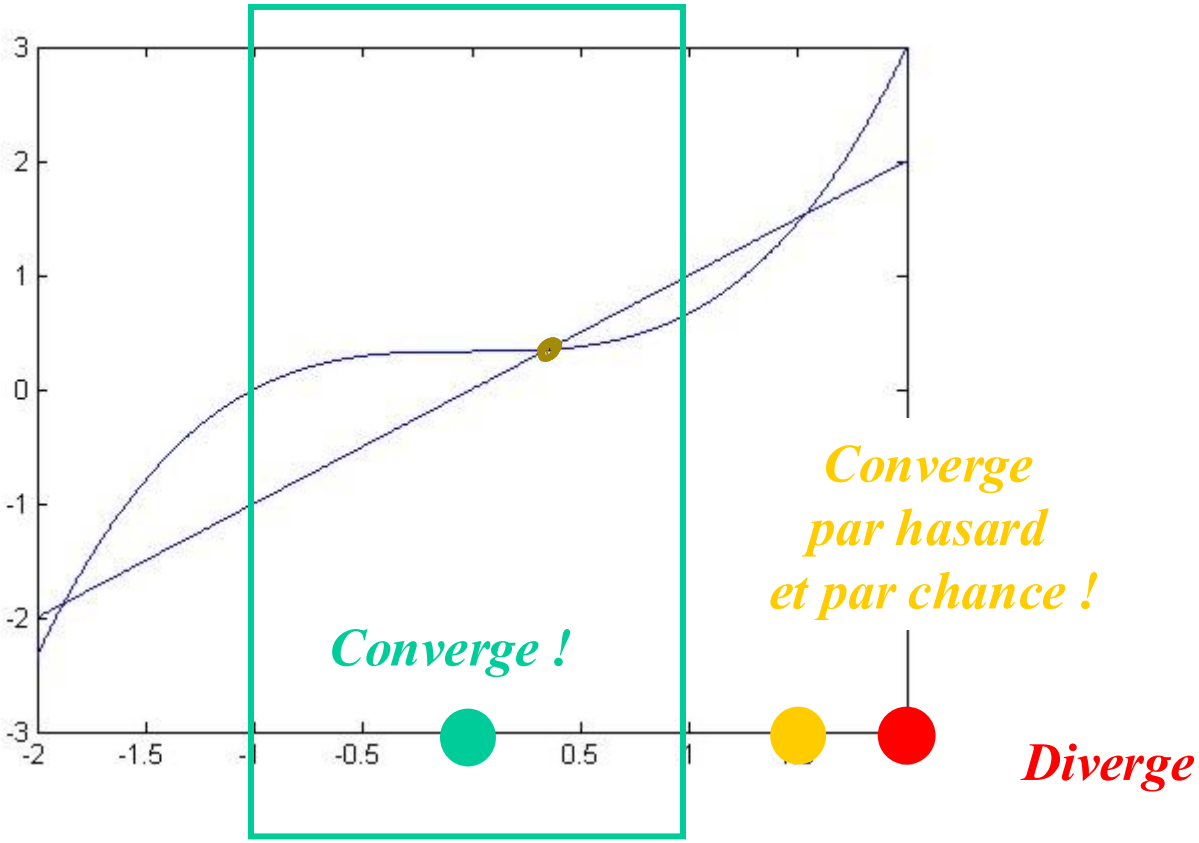
$$|e_{i+1}| \leq K\ |e_i|$$

$\Rightarrow$

$$0 \leq \lim_{i \to \infty} |x_i - x| \leq \lim_{i \to \infty} K^i\ |x_0 - x| = 0$$

Est-ce logique ?

$$x_{i+1} = \underbrace{\frac{x_i^3 + 1}{3}}_{g(x_i)}$$



Zone de convergence garantie : g'(x) = x^2 dans l'intervalle –1, 1 !

Méthode de Newton-Raphson

$$f(x) = f(x_0) + (x - x_0) f'(x_0) + \frac{(x - x_0)^2}{2} f''(x_0) + \dots$$

On fournit x_0

Tant que $|\Delta x| > \epsilon$, on calcule x_{i+1} à partir de x_i avec

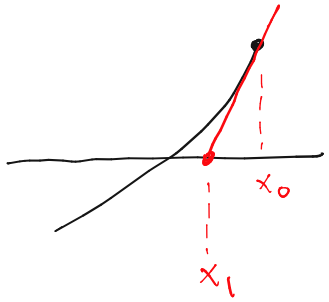
$$f'(x_i) \overbrace{(x_{i+1} - x_i)}^{\Delta x} = -f(x_i)$$

$$x_{i+1} = x_i + \Delta x$$

Si on converge, la solution x est le dernier x_{i+1} calculé

$$(x_1 - x_0) f'(x_0) + f(x_0) = 0$$

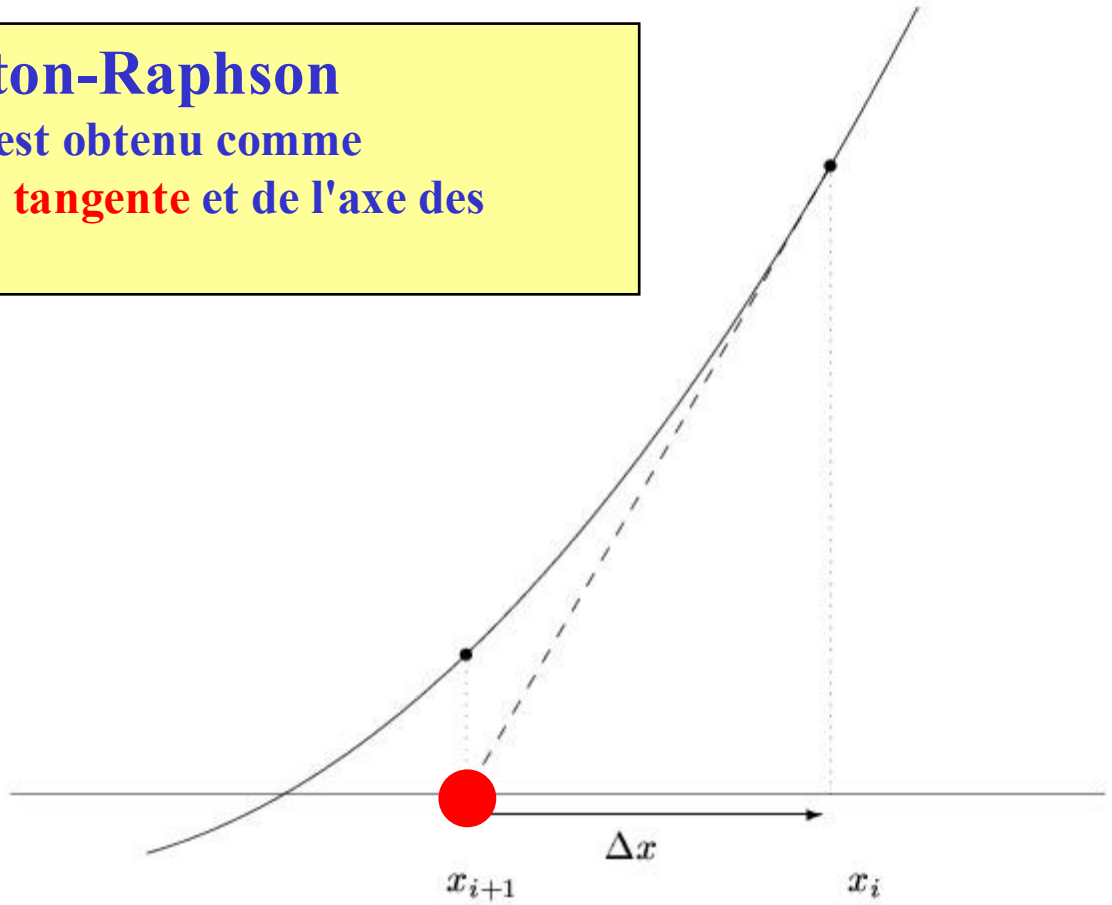
$$x_1 - x_0 = -\frac{f(x_0)}{f'(x_0)}$$



Interprétation géométrique

Méthode de Newton-Raphson

Le nouveau point est obtenu comme l'intersection de la tangente et de l'axe des abscisses



Taux de convergence de Newton-Raphson

$$f(x) = \underbrace{f(x_i)}_{=0} + \underbrace{(x - x_i)}_{e_i} f'(x_i) + \dots$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$0 = \underbrace{f(x_i)}_{f} + e_i \underbrace{f'(x_i)}_{f'} + e_i^2 \frac{f''(\xi)}{2}$$

$$f = -e_i f' - \frac{e_i^2}{2} f''$$

$$e_{i+1} = x - x_{i+1}$$

$$= \underbrace{x - (x_i}_{e_i} - \frac{f}{f'})$$

$$e_{i+1} = \frac{\cancel{e_i f'} - \cancel{e_i f'} - e_i^2 f''/2}{f'}$$

$$e_{i+1} = \frac{e_i f' + f}{f'}$$

$$e_{i+1} = e_i^2 \underbrace{\left[\frac{-f''}{2f'} \right]}_{C}$$

Newton-Raphson : Taux de convergence quadratique

Propagation de l'erreur dans un schéma de Newton-Raphson

$$
\begin{aligned}
e_{i+1} &= x - x_{i+1} \\
&= x - \left(x_i - \frac{f(x_i)}{f'(x_i)} \right) \\
&= \frac{e_i f'(x_i) + f(x_i)}{f'(x_i)}
\end{aligned}
$$

Développement en série de Taylor

$$
\begin{aligned}
0 &= f(x) \\
&= f(x_i) + \underbrace{(x - x_i)}_{e_i} f'(x_i) + \underbrace{(x - x_i)^2}_{e_i^2} \frac{f''(\xi)}{2}
\end{aligned}
$$

$$
e_{i+1} = \underbrace{-\frac{1}{2} \frac{f''(\xi)}{f'(x_i)}}_{C} e_i^2
$$

Convergence si cette constante est comprise entre [-1,1]…

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i - h)}{2h}$$

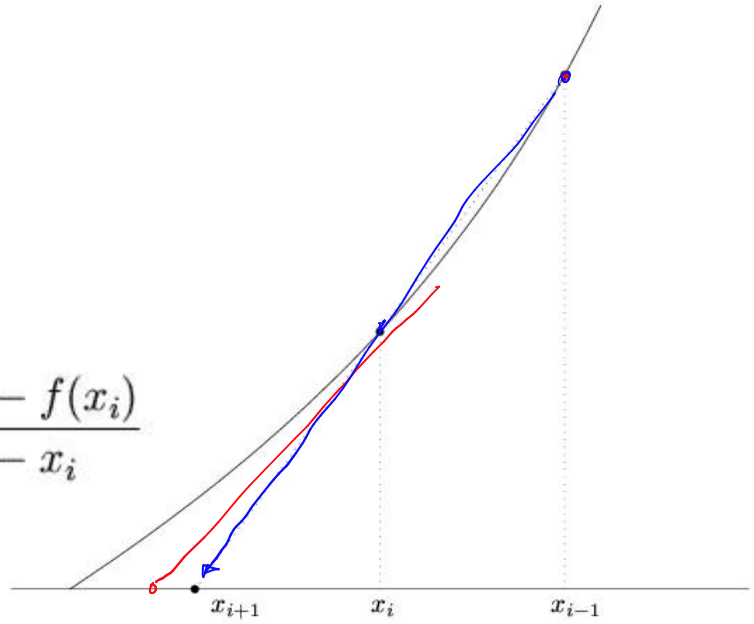
Evaluation numérique de f'

Deux estimations de f requises.
Difficulté de sélectionner h…

Une idée particulière

Une seule estimation de f requise.
Pas de paramètre à choisir !

$$f'(x_i) \approx \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$



Méthode de la sécante

$$|e_{i+1}| = C|e_i|^{1.618}$$

$$x_{i+1} = x_i - \frac{f(x_i)(x_i - x_{i-1})}{f(x_i) - f(x_{i-1})}$$

Quelle est la méthode qui converge le plus rapidement ?

Taux de convergence quadratique

$$e_{i+1} = \underbrace{-\frac{1}{2}\frac{f''(\xi)}{f'(x_i)}}_{C} e_i^2$$

Méthode de Newton-Raphson

Taux de convergence superlinéaire mais pas quadratique !

$$|e_{i+1}| = C|e_i|^{1.618}$$

Méthode de la sécante

Quelle est la méthode qui converge le plus rapidement ?

Taux de convergence quadratique

$$e_{i+1} = \underbrace{-\frac{1}{2}\frac{f''(\xi)}{f'(x_i)}}_{C} e_i^2$$

Méthode de Newton-Raphson
1 itération revient à calculer
1 estimation de f
1 estimation de f'

Taux de convergence superlinéaire mais pas quadratique !

$$|e_{i+1}| = C|e_i|^{1.618}$$

Méthode de la sécante
1 itération revient à calculer
1 estimation de f

$$1.618^2 = 2.6179 > 2$$

Systèmes d'équations non-linéaires

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ \qquad \ldots & \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{cases}$$

Notation compacte :
les vecteurs sont en gras.

$$\mathbf{f}(\mathbf{x}) = 0$$

Que peut-on faire avec les systèmes ?

Méthodes numériques itératives

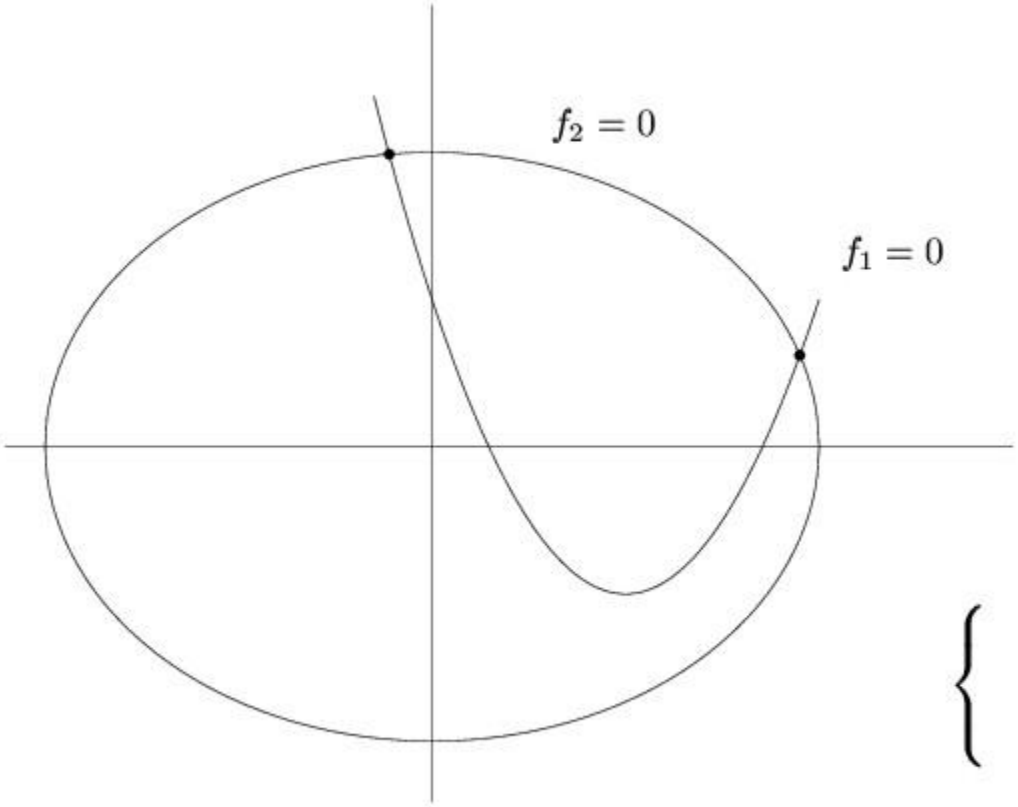
- **Méthode de bissection**
- **Méthodes du point fixe**
- **Méthode de Newton-Raphson**

Robuste, converge toujours si on a un intervalle de départ !

Mais pas généralisable aux systèmes !

Généralisables de manière immédiate aux systèmes..
Ne convergent que sous conditions…
Nécessitent un candidat initial proche de la solution…

Trouver la solution d'un système non-linéaire est très très difficile !



$$\begin{cases} \overbrace{x_1^2 - 2x_1 - x_2 + 0.5}^{f_1(x_1,x_2)} = 0 \\ \underbrace{x_1^2 + 4x_2^2 - 4}_{f_2(x_1,x_2)} = 0 \end{cases}$$

Méthode du point fixe

On fournit $\mathbf{x}_0$

Tant que $\| \Delta \mathbf{x} \| > \epsilon$, on calcule $\mathbf{x}_{i+1}$ à partir de $\mathbf{x}_i$ avec

$$\mathbf{x}_{i+1} = \mathbf{g}(\mathbf{x}_i)$$

Si on converge, la solution $\mathbf{x}$ est le dernier $\mathbf{x}_{i+1}$ calculé

Notation compacte :
les vecteurs sont en gras.

Condition de Lipschitz

$$\sum_{i=1}^{n} \left| \frac{\partial g_j}{\partial x_i} \right| < 1 \qquad j = 1 \ldots n$$

La méthode du point fixe convergera vers la racine si la condition de Lipschitz est satisfaite !

Méthode du point fixe

$$\begin{cases} x_{1,i+1} = \dfrac{x_{1,i}^2 - x_{2,i} + 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i}^2 - 4x_{2,i}^2 + 8x_{2,i} + 4}{8} \end{cases}$$

Itération de Paul

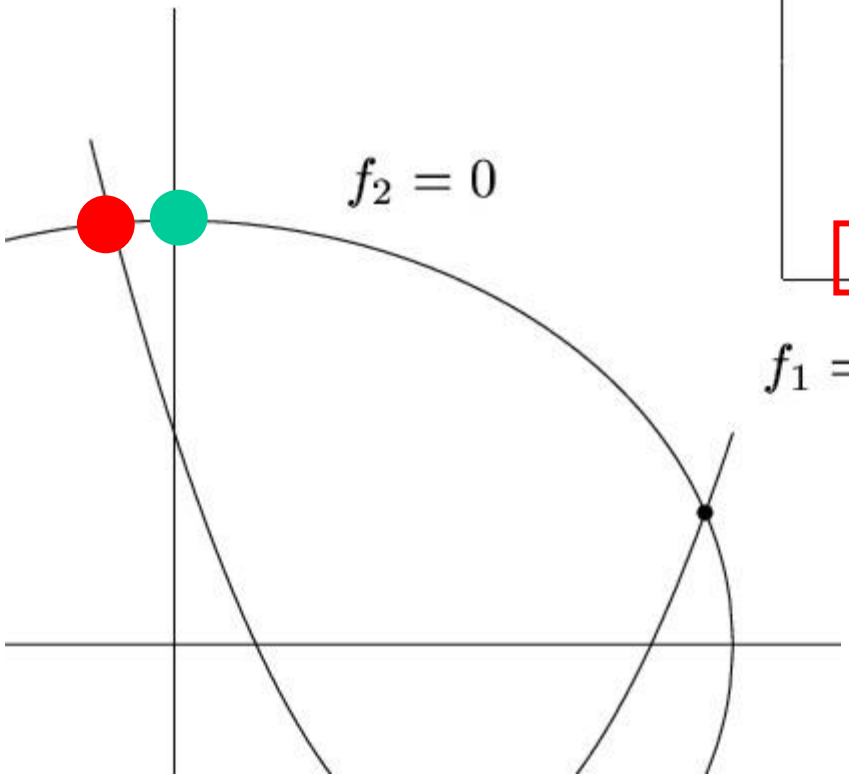
$$\begin{cases} \overbrace{x_1^2 - 2x_1 - x_2 + 0.5}^{f_1(x_1,x_2)} = 0 \\ \underbrace{x_1^2 + 4x_2^2 - 4}_{f_2(x_1,x_2)} = 0 \end{cases}$$

$$\begin{cases} x_{1,i+1} = \dfrac{-x_{1,i}^2 + 4x_{1,i} + x_{2,i} - 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i}^2 - 4x_{2,i}^2 + 11x_{2,i} + 4}{11} \end{cases}$$

Itération de Pierre

Parfois,
cela marche…

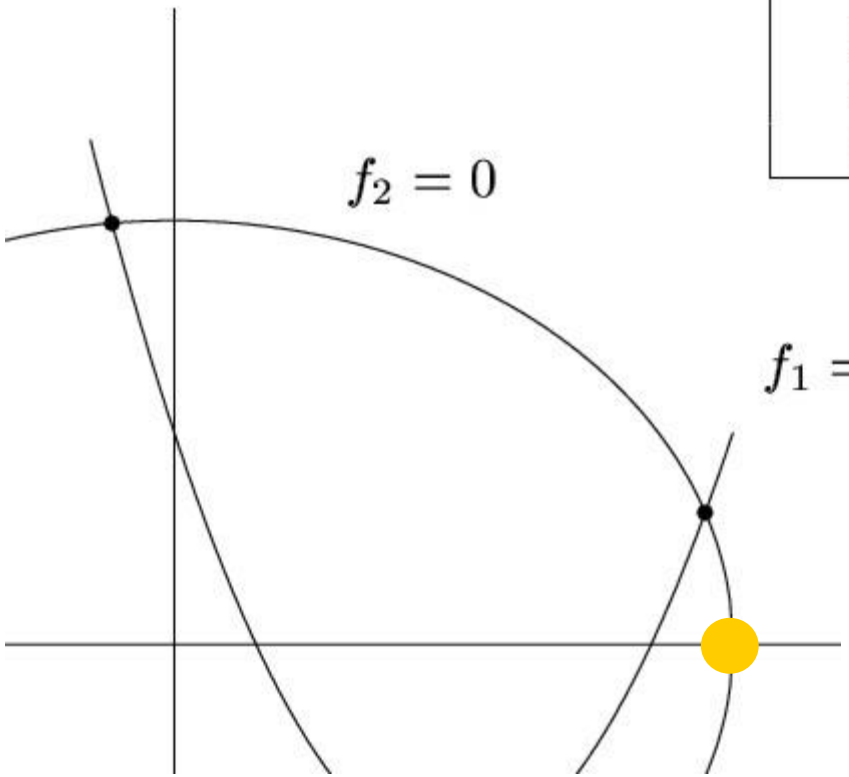
i	$x_{1,i}$	$x_{2,i}$
0	0.0000000	1.0000000
1	- 0.2500000	1.0000000
2	- 0.2187500	0.9921875
3	- 0.2221680	0.9939880
4	- 0.2223147	0.9938121
5	- 0.2221941	0.9938029
6	- 0.2222163	0.9938095
7	- 0.2222147	0.9938083
8	- 0.2222145	0.9938084
9	- 0.2222146	0.9938084



$$\begin{cases} x_{1,i+1} = \dfrac{x_{1,i}^2 - x_{2,i} + 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i}^2 - 4x_{2,i}^2 + 8x_{2,i} + 4}{8} \end{cases}$$

Parfois, cela diverge..

i	$x_{1,i}$	$x_{2,i}$
0	2.0000000	0.0000000
1	2.2500000	0.0000000
2	2.7812500	- 0.1328125
3	4.1840820	- 0.6085510
4	9.3075467	- 2.4820360
5	44.8062311	- 15.8910907
6	1 011.9947186	- 392.6042650
7	512263.2073904	-205477.8225378

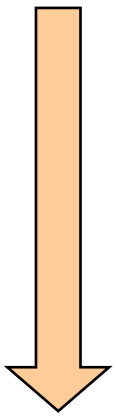


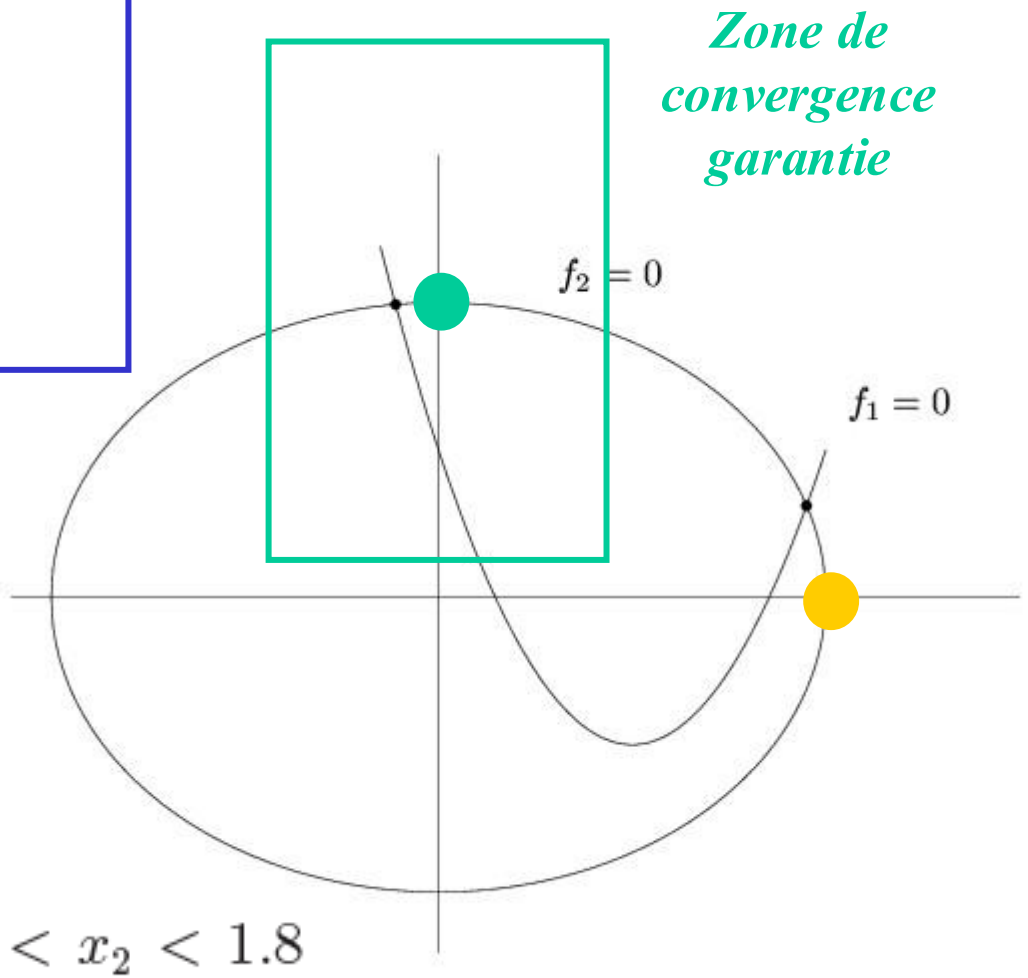
$$\begin{cases} x_{1,i+1} = \dfrac{x_{1,i}^2 - x_{2,i} + 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i}^2 - 4x_{2,i}^2 + 8x_{2,i} + 4}{8} \end{cases}$$

Et la condition de Lipschitz…

$$|x_1| + |-0.5| < 1$$

$$\left|\frac{-x_1}{4}\right| + |-x_2 + 1| < 1$$



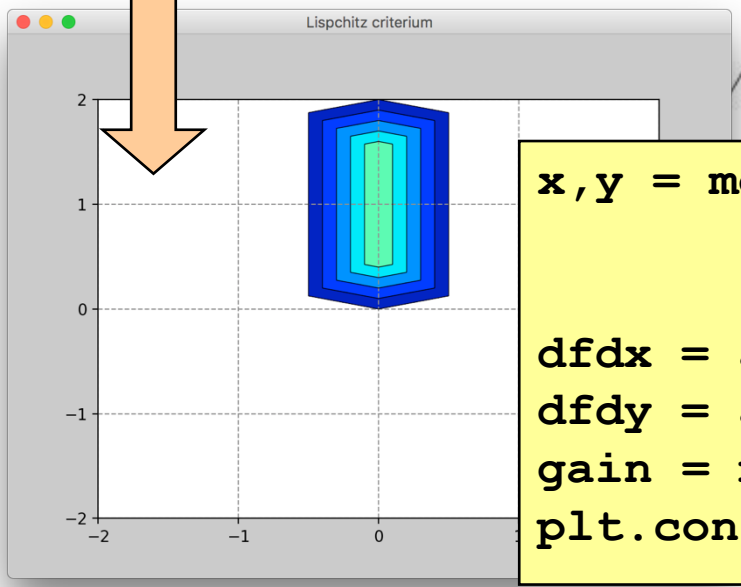


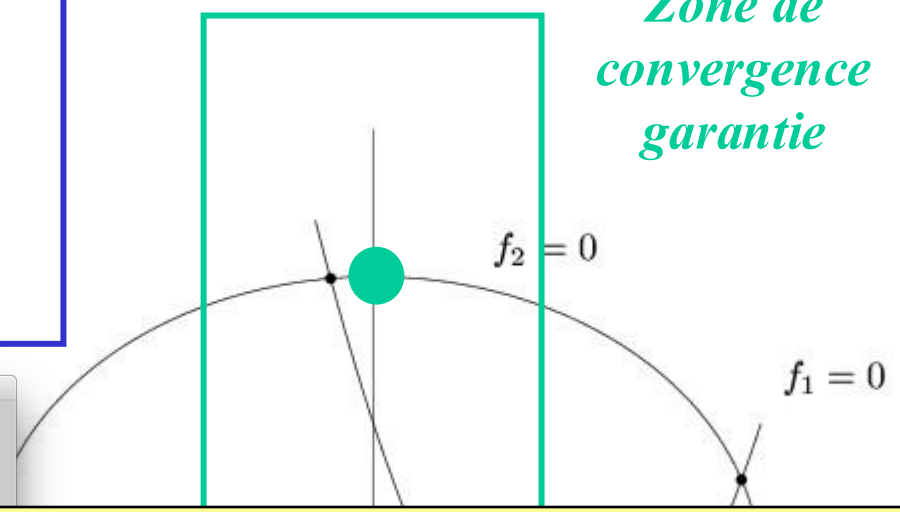
$$-0.5 < x_1 < 0.5 \text{ et } 0.2 < x_2 < 1.8$$

Et la condition de Lipschitz…

$$|x_1| + |-0.5| < 1$$

$$\left|\frac{-x_1}{4}\right| + |-x_2 + 1| < 1$$



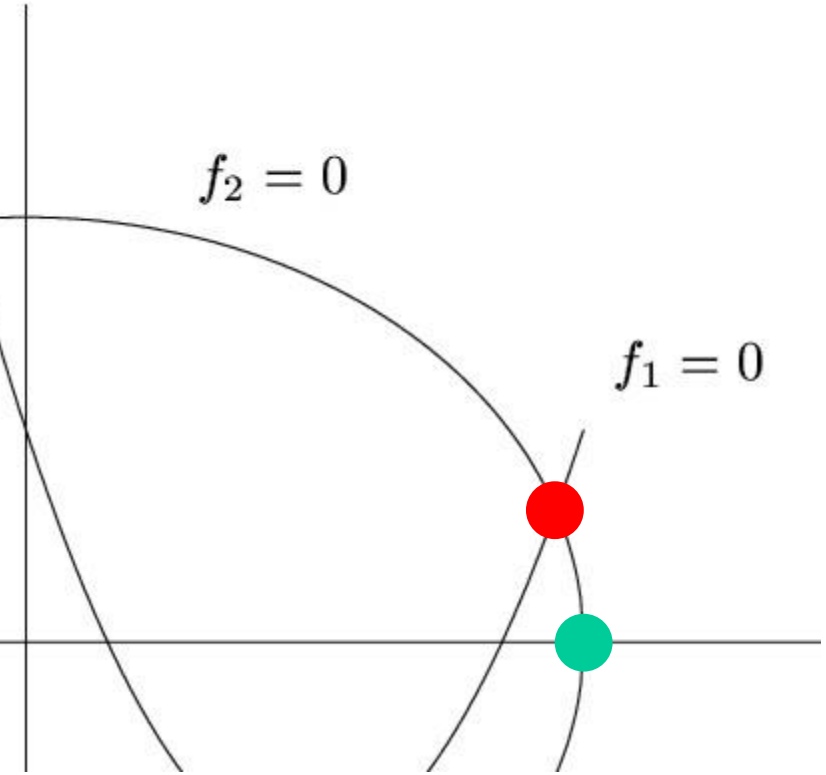


Zone de convergence garantie

```
x,y = meshgrid(linspace(-2,2,1000),
               linspace(-2,2,1000))

dfdx = abs(x)    + 0.5
dfdy = abs(x/4)  + abs(-y+1)
gain = maximum(dfdx,dfdy)
plt.contourf(x,y,gain,arange(0,1.1,0.1)
```

$$\begin{cases} x_{1,i+1} = \dfrac{-x_{1,i}^2 + 4x_{1,i} + x_{2,i} - 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i}^2 - 4x_{2,i}^2 + 11x_{2,i} + 4}{11} \end{cases}$$

Une autre itération converge...



i	$x_{1,i}$	$x_{2,i}$
0	2.0000000	0.0000000
1	1.7500000	0.0000000
2	1.7187500	0.0852273
3	1.7530629	0.1776676
4	1.8083448	0.2504410
8	1.9035947	0.3160782
12	1.9009241	0.3112267
16	1.9006516	0.3111994
20	1.9006771	0.3112196
24	1.9006768	0.3112185

Est-il possible d'améliorer la vitesse de convergence ?

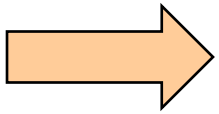
$$\begin{cases} x_{1,i+1} = \dfrac{-x_{1,i}^2 + 4x_{1,i} + x_{2,i} - 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i}^2 - 4x_{2,i}^2 + 11x_{2,i} + 4}{11} \end{cases}$$

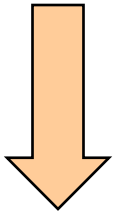
$$\begin{cases} x_{1,i+1} = \dfrac{-x_{1,i}^2 + 4x_{1,i} + x_{2,i} - 0.5}{2} \\ x_{2,i+1} = \dfrac{-x_{1,i+1}^2 - 4x_{2,i}^2 + 11x_{2,i} + 4}{11} \end{cases}$$

Algorithme de Seidel
On utilise la dernière valeur disponible pour chaque inconnue
Parfois, cela converge plus vite,
Parfois, cela converge moins vite, parfois cela diverge…

Peut-on appliquer la méthode du point fixe à un système linéaire ?

$$\underbrace{Ax - b}_{f(x)} = 0$$





Méthodes itératives (pt fixe)

- Parfois plus rapides
- Mais, ne convergent pas toujours
- Moins gourmandes en mémoire
- Utiles pour les très grands systèmes

Méthodes directes

- Elimination gaussienne (technique d'échelonnement du cours d'algèbre)
- Résultat toujours obtenu en un nombre fini d'opérations
- Nombre d'opérations requises = n^3

Un système linéaire

$$\underbrace{\mathbf{A}\mathbf{x} - \mathbf{b}}_{\mathbf{f}(\mathbf{x})} = 0$$

$$\underline{\underline{x}} = \underline{\underline{A}}^{-1} \underline{b}$$

$\underline{\underline{A}}$

$$\underline{\underline{L}} + \underline{\underline{D}} + \underline{\underline{U}}$$

$$(\underline{\underline{A}} - \underline{\underline{D}})\,\underline{x} + \underline{\underline{D}}\,\underline{x} - \underline{b} = 0$$

$$\underline{x} = \underline{\underline{D}}^{-1}\left[(\underline{\underline{D}} - A)\,\underline{x} + \underline{b}\right]$$

JACOBI

$g(\underline{x})$

$$(\underline{\underline{A}} - \underline{\underline{D}} - \underline{\underline{L}})\,\underline{x} + (\underline{\underline{D}} + \underline{\underline{L}})\,\underline{x} - \underline{b} = 0$$

$$\underline{x} = (\underline{\underline{D}} + \underline{\underline{L}})^{-1}\left[(\underline{\underline{D}} + \underline{\underline{L}} - \underline{\underline{A}})\,\underline{x} + \underline{b}\right]$$

GAUSS SEIDEL

$g(\underline{x})$

Méthodes
de Jacobi et de Gauss-Seidel

$$\underbrace{\mathbf{A}\mathbf{x} - \mathbf{b}}_{\mathbf{f}(\mathbf{x})} = 0$$

$$\mathbf{x}_{i+1} = \underbrace{\mathbf{D}^{-1}(\mathbf{D} - \mathbf{A})\mathbf{x}_i + \mathbf{D}^{-1}\mathbf{b}}_{\mathbf{g}_{Jacobi}(\mathbf{x}_i)}$$

Approximation de A^{-1}

$$\mathbf{x}_{i+1} = \underbrace{(\mathbf{D}+\mathbf{L})^{-1}\overbrace{(\mathbf{D}+\mathbf{L}-\mathbf{A})}^{-\mathbf{U}}\,\mathbf{x}_i + (\mathbf{D}+\mathbf{L})^{-1}\,\mathbf{b}}_{\mathbf{g}_{Gauss-Seidel}(\mathbf{x}_i)}$$

Exemple concret

$$\begin{bmatrix} 5 & 3 \\ 4 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

On remplace la deuxième équation par « equation(2) – 2 equation(1) » pour obtenir la convergence !

```
def g(x):
  y = copy(x)
  y[0] =  (-3*x[1]+6)/5
  y[1] =  -(6*x[0]-4)/8
  return y

def jacobi(x,tol,nmax):
  n = 0; delta = tol+1;
  x = array(x,dtype=float)
  while (norm(delta) > tol and n < nmax):
    n = n+1; xold = x.copy()
    x = g(x)
    delta = x - xold
  return x

print(jacobi([0,0],10e-6,50))
```

Et Gauss-Seidel ?

$$\begin{bmatrix} 5 & 3 \\ 4 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \end{bmatrix}$$

L'implémentation de Gauss-Seidel est plus simple que celle de Jacobi !

```
def g(x):
  y = copy(x)
  y[0] =  (-3*x[1]+6)/5
  y[1] =  -(6*
  return y

def jacobi(x,t
  n = 0; delta
  x = array(x,
  while (norm(
    n = n+1; x
    x = g(x)
    delta = x
  return x
```

```
def g(x):

  x[0] =  (-3*x[1]+6)/5
  x[1] =  -(6*x[0]-4)/8
  return x

def gaussseidel(x,tol,nmax):
  n = 0; delta = tol+1;
  x = array(x,dtype=float)
  while (norm(delta) > tol and n < nmax):
    n = n+1; xold = x.copy()
    x = g(x)
    delta = x - xold
return x
```

Jacobi

```
>>>print(jacobi([0,0],10e-6,50))
 Estimated error  1.3000000e+00 at iteration 1
 Estimated error  9.4868330e-01 at iteration 2
 Estimated error  5.8500000e-01 at iteration 3
 Estimated error  4.2690748e-01 at iteration 4
 Estimated error  2.6325000e-01 at iteration 5


 Estimated error  2.9436348e-05 at iteration 28
 Estimated error  1.8151752e-05 at iteration 29
 Estimated error  1.3246357e-05 at iteration 30
 Estimated error  8.1682883e-06 at iteration 31
[ 1.63636089 -0.72726502]
```

Gauss-Seidel

```
>>>print(gaussseidel([0,0],10e-6,50))
 Estimated error  1.2649111e+00 at iteration 1
 Estimated error  3.0000000e-01 at iteration 2
 Estimated error  1.3500000e-01 at iteration 3


 Estimated error  1.0215189e-04 at iteration 12
 Estimated error  4.5968349e-05 at iteration 13
 Estimated error  2.0685757e-05 at iteration 14
 Estimated error  9.3085907e-06 at iteration 15
[ 1.63635754 -0.72726816]
```

Convergence ?

$\underline{x} = \underline{g}(\underline{x})$

$$\mathbf{x}_{i+1} = \underbrace{\mathbf{M}\mathbf{x}_i}_{\underline{g}(\underline{x}_i)} + \mathbf{c}$$

IL FAUT

$\forall k \quad \underbrace{|\lambda_k|}_{\text{RAYON SPECTRAL}} < 1$

$\underline{x}_{i+1} - \underline{x} = \underline{\underline{M}}\,\underline{x}_i + \underline{c} - \underline{\underline{M}}\,\underline{x} - \underline{c}$

$\underline{e}_{i+1} = \underline{\underline{M}}\,(\underbrace{\underline{x}_i - \underline{x}}_{\underline{e}_i})$

$= \underline{\underline{M}}^i\,\underline{e}_0$

EN ECRIVANT LES $\underline{e}_i$ COMME UNE COMBINAISON LINEAIRES DES VECTEURS PROPRES DE $\underline{\underline{M}}$

Convergence d'une méthode itérative ?

$$\underbrace{(\mathbf{x}_{i+1} - \mathbf{x})}_{\mathbf{e}_{i+1}} = \mathbf{M}\mathbf{x}_i + \mathbf{c} - \mathbf{M}\mathbf{x} - \mathbf{c}$$

$$= \mathbf{M}\,(\mathbf{x}_i - \mathbf{x})$$

En procédant de la même manière pour chaque étape,

$$= \mathbf{M}^{i+1}\,\underbrace{(\mathbf{x}_0 - \mathbf{x})}_{\mathbf{e}_0}$$

Il faut que…

$$\lim_{i \to \infty} \mathbf{M}^i\,\mathbf{e}^{(0)} = 0$$

Ecrivons l'erreur comme une combili de vecteurs propres…

$$\mathbf{e}_0 = \sum_{j=1}^{n} \alpha_j \mathbf{v}_j$$

↓ Puisque $\mathbf{M}\ \mathbf{v}_i = \lambda_i \mathbf{v}_i$,

$$\mathbf{e}_i = \sum_{j=1}^{n} \alpha_j \lambda_j^i \mathbf{v}_j$$

Pour obtenir…

$$\lim_{i \to \infty} \mathbf{M}^i \mathbf{e}^{(0)} = 0$$

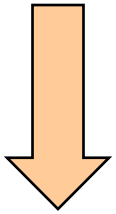
… on doit exiger que le rayon spectral de la matrice M soit inférieur à l'unité

$$|\lambda_i| < 1 \qquad \forall\ i$$

Condition pratique pour Jacobi

$$\sum_{j=1,\neq i}^{n} \left| \frac{a_{ij}}{a_{ii}} \right| < 1 \qquad i = 1, \ldots, n$$

La dominance diagonale permet d'avoir la convergence…



Soit $\mathbf{M}$ *une matrice diagonalisable d'ordre* n*, on a alors :*

$$|\lambda_i| \leq \sum_{j=1}^{n} |m_{ij}| \qquad i = 1, \ldots, n$$

$$|\lambda_i| < 1 \qquad \forall\ i$$