

Systemes d'equations non-lineaires

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

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

VECTEUR DE FONCTIONS INCONNUES

VECTEUR D'INCONNUES

*Notation compacte :
les vecteurs sont en gras.*

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

Que peut-on faire avec les systèmes ?

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

Méthodes numériques itératives

Méthode de bisection

Méthodes du point fixe

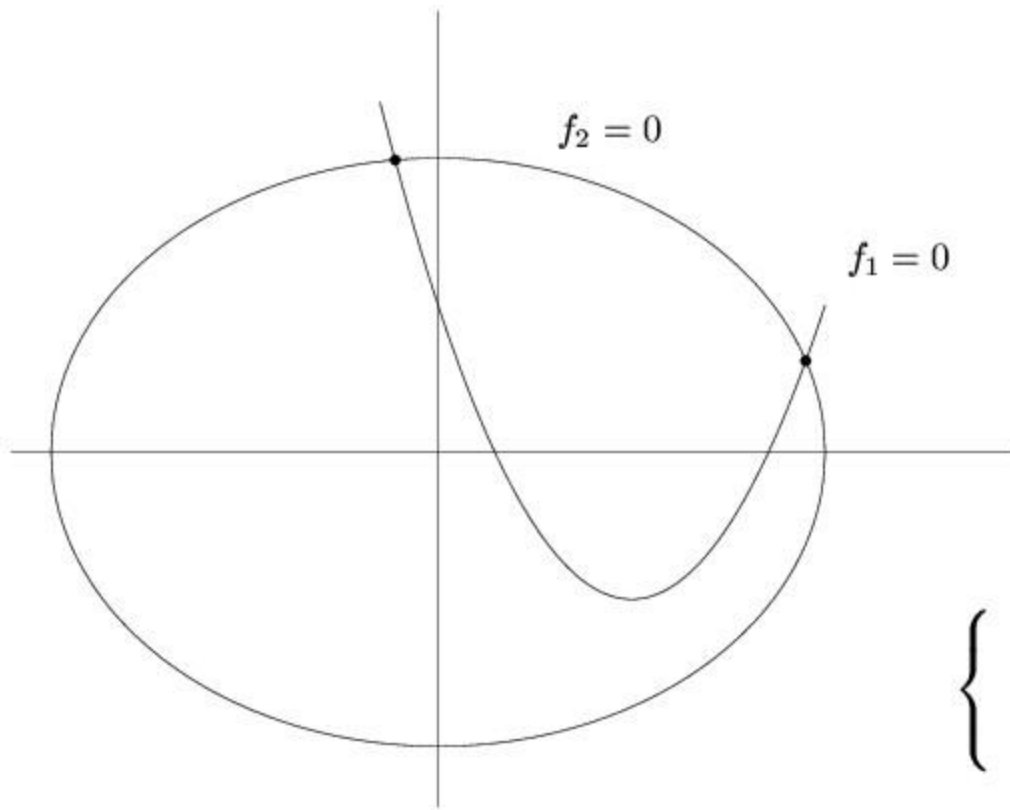
Méthode de Newton-Raphson

Méthode de la sécante

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 !



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

Méthode du point fixe

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

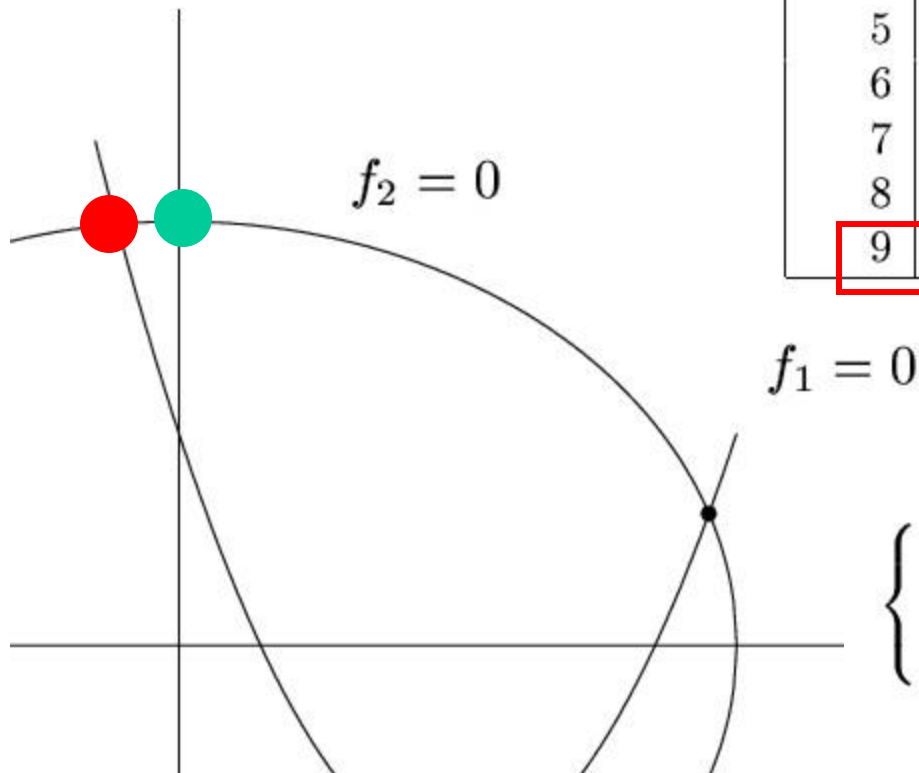
Itération de Paul

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

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

Itération de Pierre

Parfois,
cela marche...

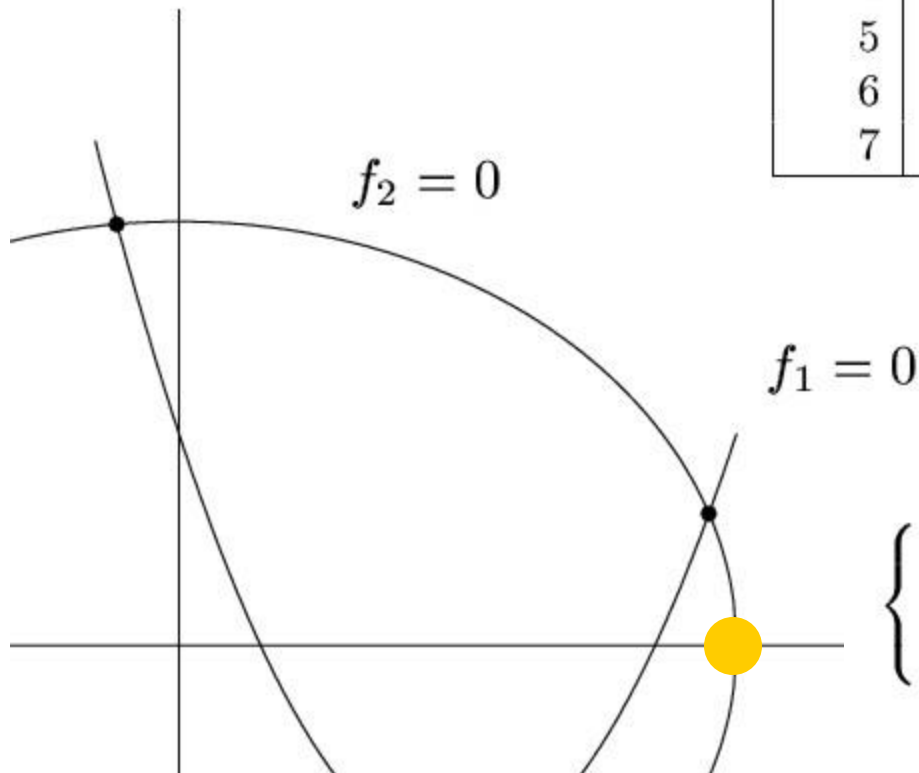


| i | $x_{1,i}$ | $x_{2,i}$ |
|-----|-------------|-----------|
| 0 | 0.0000000 | 1.0000000 |
| 1 | - 0.2500000 | 1.0000000 |
| 2 | - 0.2187500 | 0.9921875 |
| 3 | - 0.2221680 | 0.9939880 |
| 4 | - 0.2223147 | 0.9938121 |
| 5 | - 0.2221941 | 0.9938029 |
| 6 | - 0.2222163 | 0.9938095 |
| 7 | - 0.2222147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

$$\left\{ \begin{array}{l} x_{1,i+1} = \frac{x_{1,i}^2 - x_{2,i} + 0.5}{2} \\ x_{2,i+1} = \frac{-x_{1,i}^2 - 4x_{2,i}^2 + 8x_{2,i} + 4}{8} \end{array} \right.$$

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 |



$$\left\{ \begin{array}{l} x_{1,i+1} = \frac{x_{1,i}^2 - x_{2,i} + 0.5}{2} \\ x_{2,i+1} = \frac{-x_{1,i}^2 - 4x_{2,i}^2 + 8x_{2,i} + 4}{8} \end{array} \right.$$

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

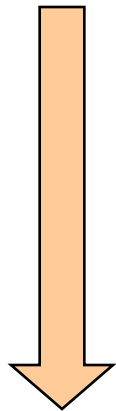
$$\sum_{i=1}^n \left| \frac{\partial g_j}{\partial x_i} \right| < 1 \quad j = 1 \dots n$$

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

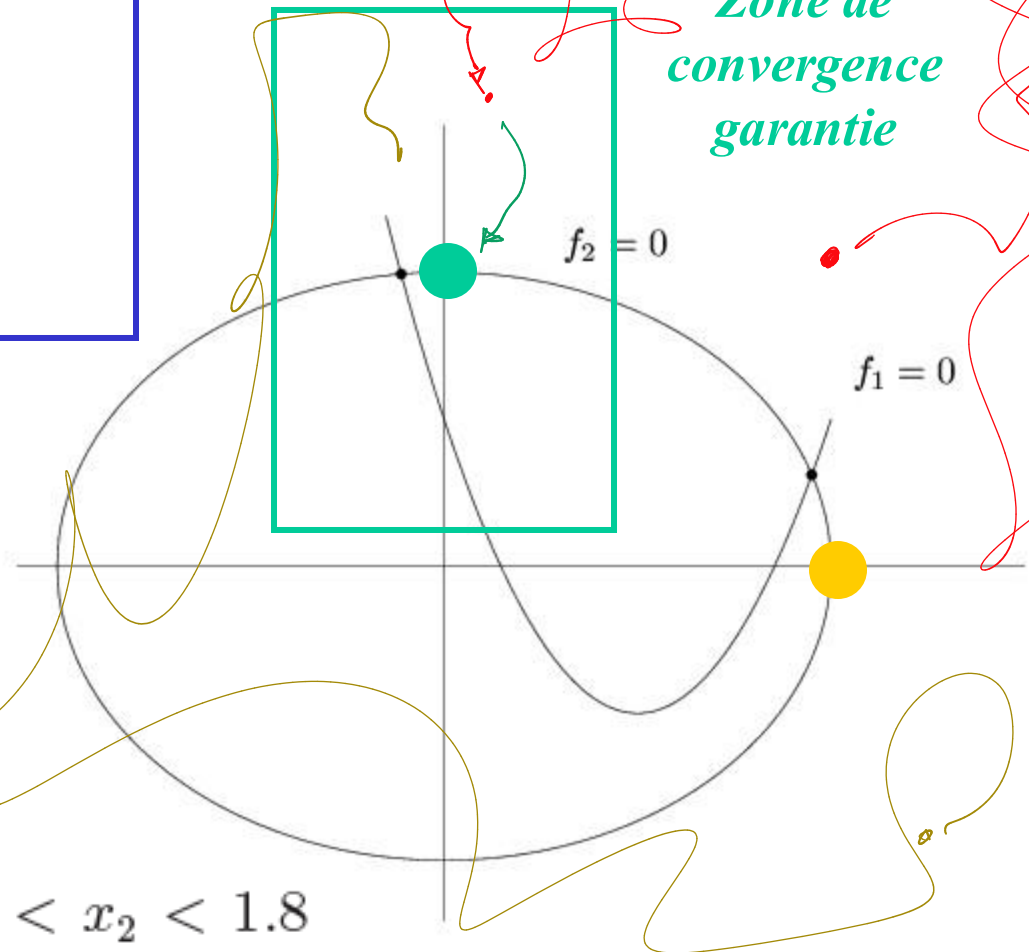
Et la condition de Lipschitz...

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

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



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



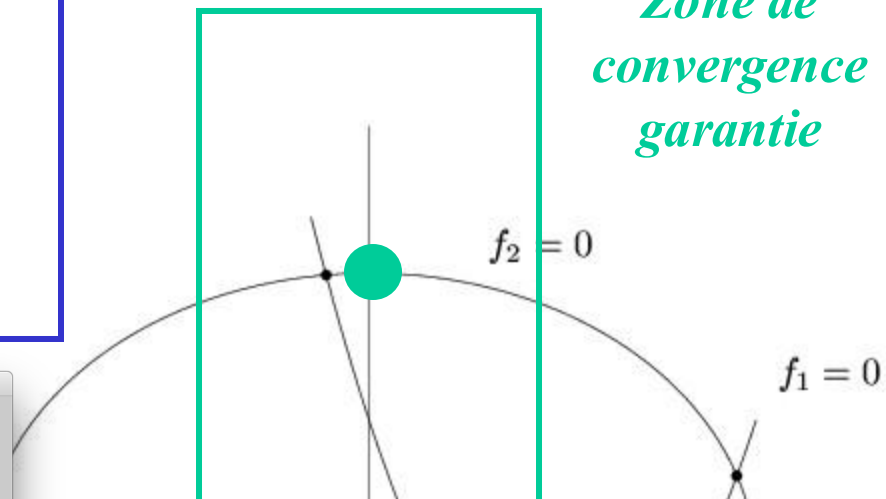
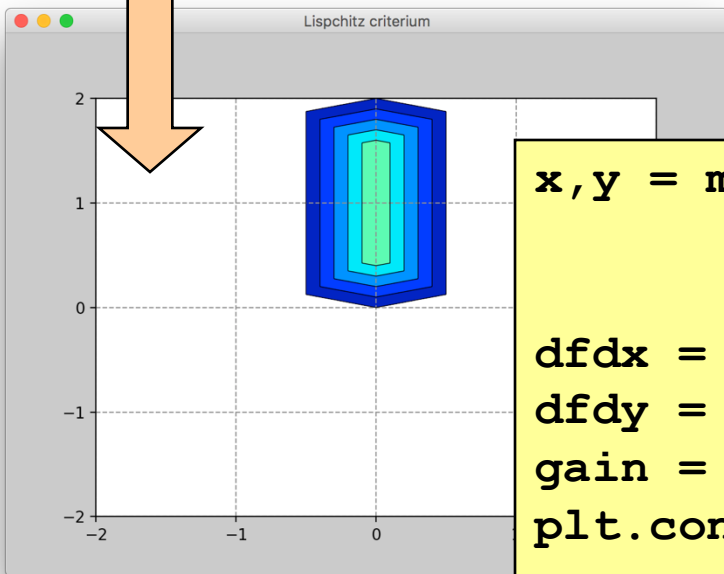
*Zone de
convergence
garantie*

Et la condition de Lipschitz...

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

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

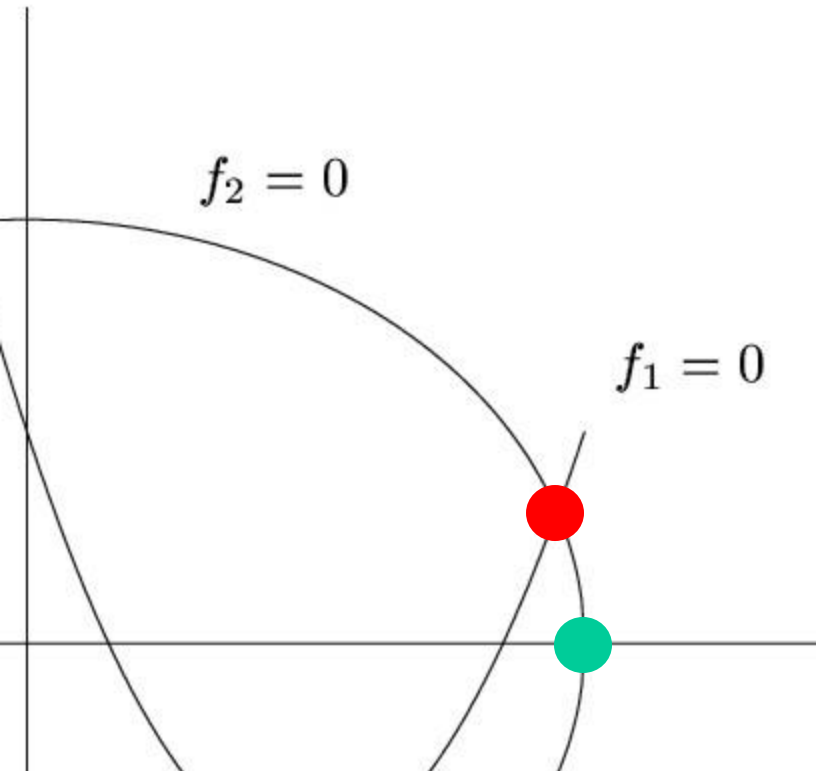
*Zone de
convergence
garantie*



```
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))
```

Une autre
itération
converge...

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



| 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 ?

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

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

Algorithme de Seidel

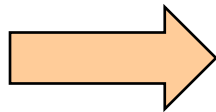
On utilise la dernière valeur disponible pour chaque inconnue

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 ?

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



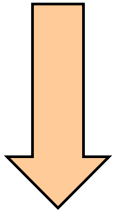
Méthodes itératives (pt fixe)

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)

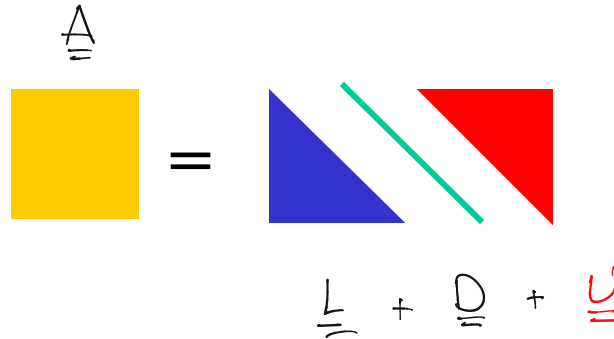
Résultat toujours obtenu en un nombre fini d'opérations

Nombre d'opérations requises = n^3

Un système linéaire

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

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



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

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

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

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

Méthodes de Jacobi et de Gauss-Seidel

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



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

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*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
```

```
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 ?

$$x = g(x)$$

IL FAUT

$\forall k$

$$|\lambda_k| \leq 1$$

RAYON
SPECTRAL

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

$$\begin{aligned} \underbrace{x_{i+1} - x}_{e_{i+1}} &= \underbrace{x_i + e}_{\cancel{e}} - \underbrace{x - e}_{\cancel{e}} \\ &= \underbrace{x_i - x}_{e_i} \\ &= \sum_i e_i \end{aligned}$$

EN ECRIVANT
LES e_i COMME UNE COMBINAISON
LINEAIRE

DES VECTEURS
PROPRIES DE \mathbf{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...

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

Ecrivons l'erreur comme une combinaison 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...

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

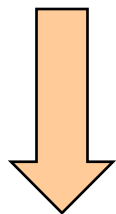
... on doit exiger que le rayon spectral de la matrice M soit inférieur à l'unité

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

Condition pratique pour Jacobi

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

*La dominance diagonale
permet d'avoir
la convergence...*



Soit M une matrice diagonalisable d'ordre n, on a alors :

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

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

Méthode de Newton-Raphson

Matrice jacobienne du système

On fournit \mathbf{x}_0

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

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_i) \overbrace{(\mathbf{x}_{i+1} - \mathbf{x}_i)}^{\Delta \mathbf{x}} = -\mathbf{f}(\mathbf{x}_i)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta \mathbf{x}$$

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

Notation compacte :
les vecteurs sont en gras.

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_i) = \frac{\partial f_j}{\partial x_k} \Big|_{(x_{1,i}, x_{2,i}, \dots, x_{n,i})}$$

Tableau à deux indices

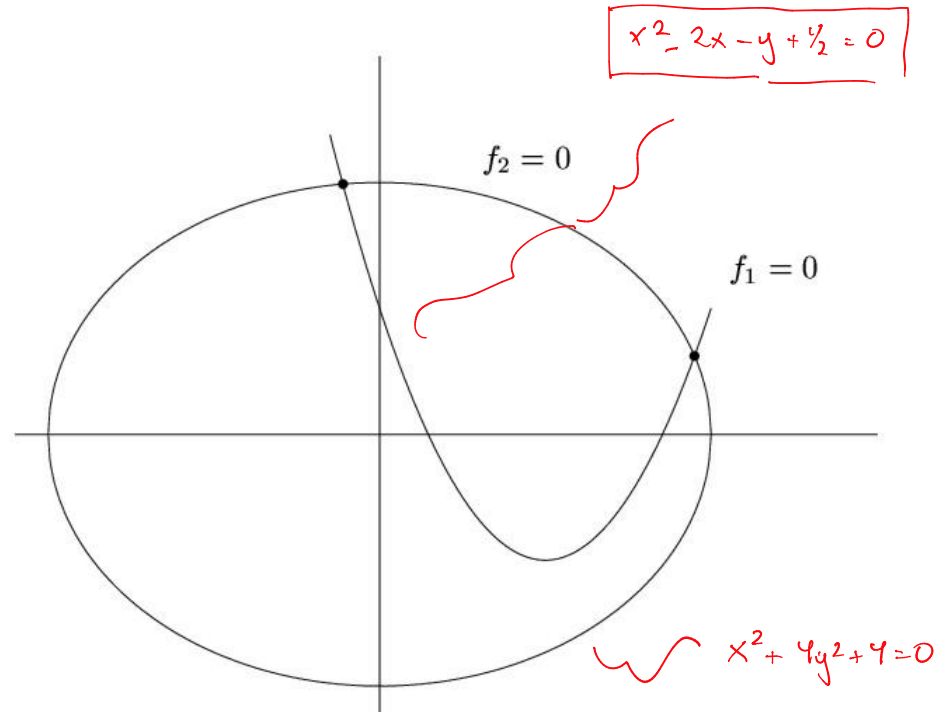
On calcule la dérivée partielle par rapport à la k -ième inconnue de la j -ième composante de f

Tableau à deux indices

Il s'agit de la seconde composante du vecteur Inconnue lors de la i -ème itération

Exemple

$$\left\{ \begin{array}{l} \underbrace{f_1(x_1, x_2)}_{x_1^2 - 2x_1 - x_2 + 0.5} = 0 \\ \underbrace{f_2(x_1, x_2)}_{x_1^2 + 4x_2^2 - 4} = 0 \end{array} \right.$$



$$\begin{aligned} \frac{\partial f_j}{\partial x_k} \Big|_{(x_1, x_2)} &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} \Big|_{(x_1, x_2)} & \frac{\partial f_1}{\partial x_2} \Big|_{(x_1, x_2)} \\ \frac{\partial f_2}{\partial x_1} \Big|_{(x_1, x_2)} & \frac{\partial f_2}{\partial x_2} \Big|_{(x_1, x_2)} \end{bmatrix} \\ &= \begin{bmatrix} 2x_1 - 2 & -1 \\ 2x_1 & 8x_2 \end{bmatrix} \end{aligned}$$

Exemple

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

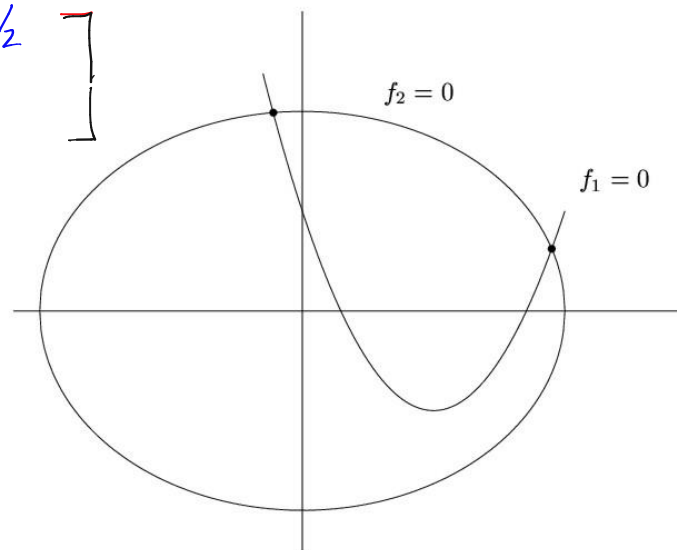
$$\begin{bmatrix} f_1(x) \\ f_2(x) \end{bmatrix} = \begin{bmatrix} x^2 - 2x - y + \frac{1}{2} \\ x^2 + 4y^2 - 4 \end{bmatrix}$$

$$\frac{\partial f}{\partial x} = \begin{bmatrix} 2x - 2 & -1 \\ 2x & 4y \end{bmatrix}$$

$$\underbrace{\begin{bmatrix} 2x - 2 & -1 \\ 2x & 4y \end{bmatrix}}_{\frac{\partial f}{\partial x}} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} =$$

$$\begin{bmatrix} x^2 - 2x - y + \frac{1}{2} \\ x^2 + 4y^2 - 4 \end{bmatrix}$$

↕



Comment appliquer la méthode de Newton-Raphson à ce système d'équations non-linéaires ?

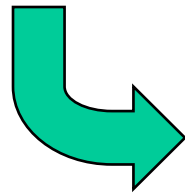
On fournit \mathbf{x}_0

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

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_i) \overbrace{(\mathbf{x}_{i+1} - \mathbf{x}_i)}^{\Delta \mathbf{x}} = -\mathbf{f}(\mathbf{x}_i)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta \mathbf{x}$$

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



$$\begin{bmatrix} \left. \frac{\partial f_1}{\partial x_1} \right|_{(x_{1,i}, x_{2,i})} & \left. \frac{\partial f_1}{\partial x_2} \right|_{(x_{1,i}, x_{2,i})} \\ \left. \frac{\partial f_2}{\partial x_1} \right|_{(x_{1,i}, x_{2,i})} & \left. \frac{\partial f_2}{\partial x_2} \right|_{(x_{1,i}, x_{2,i})} \end{bmatrix} \begin{bmatrix} \Delta x_1 \\ \Delta x_2 \end{bmatrix} = \begin{bmatrix} -f_1(x_{1,i}, x_{2,i}) \\ -f_2(x_{1,i}, x_{2,i}) \end{bmatrix}$$

Que fournit la méthode de Newton-Raphson pour ce système d'équations non-linéaires ?

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

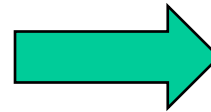
On fournit \mathbf{x}_0

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

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_i) \overbrace{(\mathbf{x}_{i+1} - \mathbf{x}_i)}^{\Delta \mathbf{x}} = -\mathbf{f}(\mathbf{x}_i)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta \mathbf{x}$$

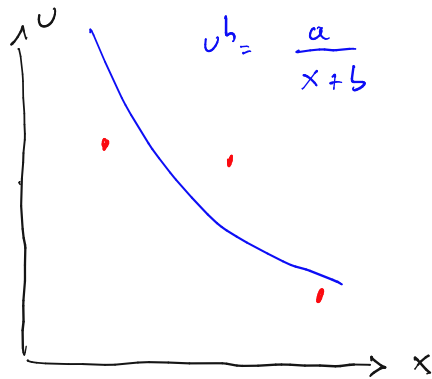
Si on converge, la solution \mathbf{x} est le dernier \mathbf{x}_{i+1} calculé



| i | $x_{1,i}$ | $x_{2,i}$ |
|-----|-----------|-----------|
| 0 | 2.000000 | 0.250000 |
| 1 | 1.906250 | 0.312500 |
| 2 | 1.900691 | 0.311213 |
| 4 | 1.900677 | 0.311219 |

Dans le domaine de convergence asymptotique, on double le nombre de chiffres significatifs à chaque itération

Optimisation non-linéaire



n mesures (X_i, U_i)

$$u^h(x) = \frac{a}{x+b}$$

$$f(a, b) = \sum_{i=1}^n \left(U_i - \frac{a}{(X_i + b)} \right)^2$$

Fonction à minimiser

On ajuste les deux paramètres a et b afin de minimiser le carré des écarts

$$\frac{\partial f}{\partial x} \Delta x = -f \quad x = \begin{bmatrix} a \\ b \end{bmatrix}$$

$$f = \begin{bmatrix} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \end{bmatrix} = 0$$

$$f(a, b) = \sum_{i=1}^n \left(U_i - \frac{a}{(X_i + b)} \right)^2$$

Fonction à minimiser

$$\begin{cases} \frac{\partial f}{\partial a} = 0 \\ \frac{\partial f}{\partial b} = 0 \end{cases}$$

$$\begin{cases} 0 = \frac{\partial f}{\partial a} = 2 \sum \left(U_i - \frac{a}{X_i + b} \right) \left(\frac{-1}{X_i + b} \right) \\ 0 = \frac{\partial f}{\partial b} = 2 \sum \left(U_i - \frac{a}{X_i + b} \right) \left(\frac{a}{(X_i + b)^2} \right) \end{cases}$$

$$\frac{\partial f}{\partial a} = \cancel{2} \sum \left(U_i - \frac{a}{X_i + b} \right) \left(\frac{-1}{X_i + b} \right)$$

$$\frac{\partial f}{\partial b} = \cancel{2} \sum \left(U_i - \frac{a}{X_i + b} \right) \left(\frac{a}{(X_i + b)^2} \right)$$

$$f(a, b) = \sum_{i=1}^n \left(U_i - \frac{a}{(X_i + b)} \right)^2$$

Fonction à minimiser

$$\begin{array}{c}
 \sum \frac{2}{(c)^2} \\
 \left[\begin{array}{c} \frac{\partial^2 f}{\partial a^2} \\ \frac{\partial^2 f}{\partial a \partial b} \end{array} \right] \left[\begin{array}{c} \Delta a \\ \Delta b \end{array} \right]^2 - \left[\begin{array}{c} \frac{\partial f}{\partial a} \\ \frac{\partial f}{\partial b} \end{array} \right] \\
 \sum \frac{2U_i}{(c)^2} - \sum \frac{4a}{(c)^3} \qquad \sum -\frac{4U_i a}{(c)^3} + \sum \frac{6a^2}{(c)^4} \qquad \sum \frac{-2U_i}{(c)^2} + \sum \frac{2a}{(c)^2} \\
 \sum \frac{2U_i a}{(c)^2} - \sum \frac{2a^2}{(c)^3}
 \end{array}$$

$$f(a, b) = \sum_{i=1}^n \left(U_i - \frac{a}{(X_i + b)} \right)^2$$

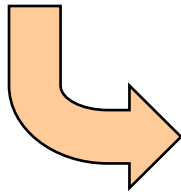
Fonction à minimiser

Extréma d'une fonction à deux variables...

$$0 = \frac{\partial f}{\partial a}$$
$$0 = \frac{\partial f}{\partial b}$$

*Conditions nécessaires
pour obtenir un minimum !*

Deux équations non-linéaires



$$0 = \sum_{i=1}^n 2 \left(U_i - \frac{a}{(X_i + b)} \right) \frac{-1}{(X_i + b)}$$

$$0 = \sum_{i=1}^n 2 \left(U_i - \frac{a}{(X_i + b)} \right) \frac{a}{(X_i + b)^2}$$

Utilisons Newton-Raphson !

On calcule (a_{k+1}, b_{k+1}) à partir de (a_k, b_k) avec

$$\begin{bmatrix} \frac{\partial^2 f}{\partial a^2}(a_k, b_k) & \frac{\partial^2 f}{\partial a \partial b}(a_k, b_k) \\ \frac{\partial^2 f}{\partial b \partial a}(a_k, b_k) & \frac{\partial^2 f}{\partial b^2}(a_k, b_k) \end{bmatrix} \begin{bmatrix} \Delta a \\ \Delta b \end{bmatrix} = - \begin{bmatrix} \frac{\partial f}{\partial a}(a_k, b_k) \\ \frac{\partial f}{\partial b}(a_k, b_k) \end{bmatrix}$$

$$\begin{aligned} a_{k+1} &= a_k + \Delta a \\ b_{k+1} &= b_k + \Delta b \end{aligned}$$

$$\frac{\partial^2 f}{\partial a^2} = 2 \sum_{i=1}^n \frac{1}{(X_i + b)^2}$$

$$\frac{\partial^2 f}{\partial b^2} = -4a \sum_{i=1}^n \frac{U_i}{(X_i + b)^3} + 6a^2 \sum_{i=1}^n \frac{1}{(X_i + b)^4}$$

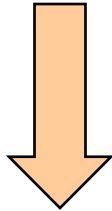
$$\frac{\partial^2 f}{\partial a \partial b} = \frac{\partial^2 f}{\partial b \partial a} = 2 \sum_{i=1}^n \frac{U_i}{(X_i + b)^2} - 4a \sum_{i=1}^n \frac{1}{(X_i + b)^3}$$

$$\frac{\partial f}{\partial a} = -2 \sum_{i=1}^n \frac{U_i}{(X_i + b)} + 2a \sum_{i=1}^n \frac{1}{(X_i + b)^2}$$

$$\frac{\partial f}{\partial b} = 2a \sum_{i=1}^n \frac{U_i}{(X_i + b)^2} - 2a^2 \sum_{i=1}^n \frac{1}{(X_i + b)^3}$$

Et espérer que cela converge...

Il y aura un problème.... si on obtient une valeur de b identique à l'opposé des données !

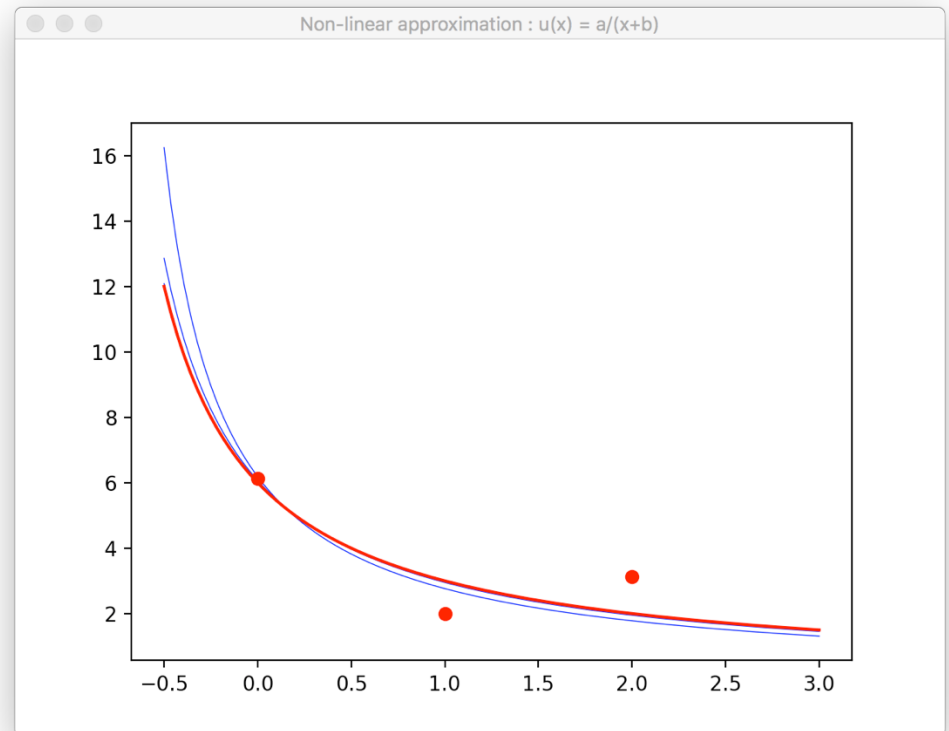


Warning: Divide by zero.
ans = Inf

$$\sum_{i=1}^n 2 \left(U_i - \frac{a}{(X_i + b)} \right) \frac{-1}{(X_i + b)}$$

$$\sum_{i=1}^n 2 \left(U_i - \frac{a}{(X_i + b)} \right) \frac{a}{(X_i + b)^2}$$

En pratique...



```
Iteration 1 : 5.0476842e-01 (a = 4.9952884e+00 b = 8.0757337e-01)
Iteration 2 : 7.6236375e-01 (a = 5.7448233e+00 b = 9.4684249e-01)
Iteration 3 : 2.3523119e-01 (a = 5.9751743e+00 b = 9.9450952e-01)
Iteration 4 : 2.5103378e-02 (a = 5.9996858e+00 b = 9.9992860e-01)
Iteration 5 : 3.2217774e-04 (a = 5.9999999e+00 b = 9.9999999e-01)
Iteration 6 : 5.8254259e-08 (a = 6.0000000e+00 b = 1.0000000e+00)
Iteration 7 : 5.4721560e-16 (a = 6.0000000e+00 b = 1.0000000e+00)
Observed rate of convergence : 2.44
Theoretical rate : 2.00
```

Systemes d'equations differentielles non-lineaires

$$\begin{cases} u_1'(x) = f_1(x, u_1(x), u_2(x), \dots, u_n(x)) \\ u_2'(x) = f_2(x, u_1(x), u_2(x), \dots, u_n(x)) \\ \vdots \\ u_n'(x) = f_n(x, u_1(x), u_2(x), \dots, u_n(x)) \end{cases}$$

Notation compacte :
les vecteurs sont en gras.

$$\begin{cases} u_1(a) = \bar{u}_1 \\ u_2(a) = \bar{u}_2 \\ \vdots \\ u_n(a) = \bar{u}_n \end{cases}$$

Trouver $\mathbf{u}(x)$ tel que

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

C'est exactement la même histoire !

$$U_{ji} = u_j^h(X_i) \approx u_j(X_i)$$

$$\mathbf{U}_i = \mathbf{u}^h(X_i) \approx \mathbf{u}(X_i)$$

*Notation compacte :
les vecteurs sont en gras.*

Méthode d'Euler explicite

$$U_{ji+1} = U_{ji} + hf_j(X_i, U_{1i}, \dots, U_{ni})$$

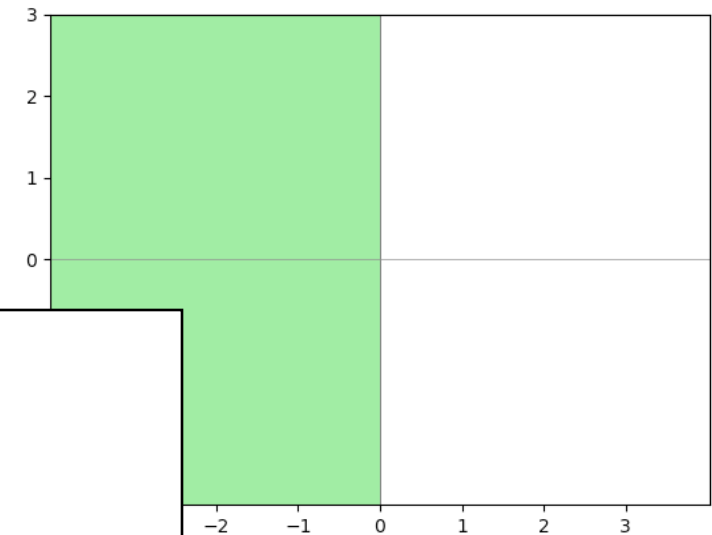
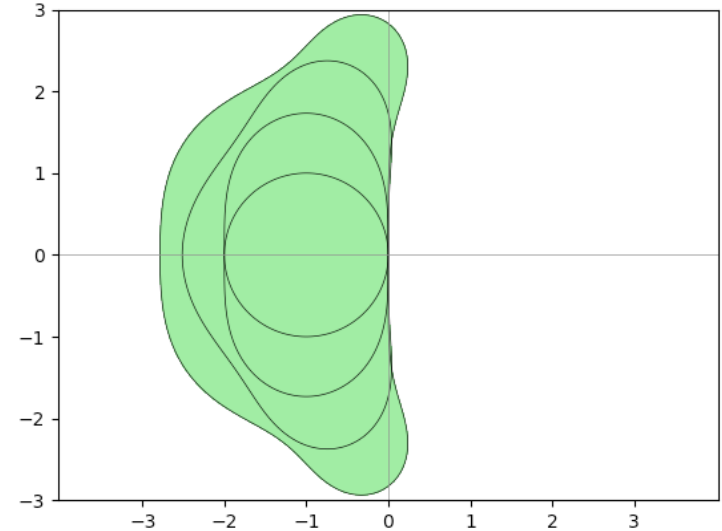
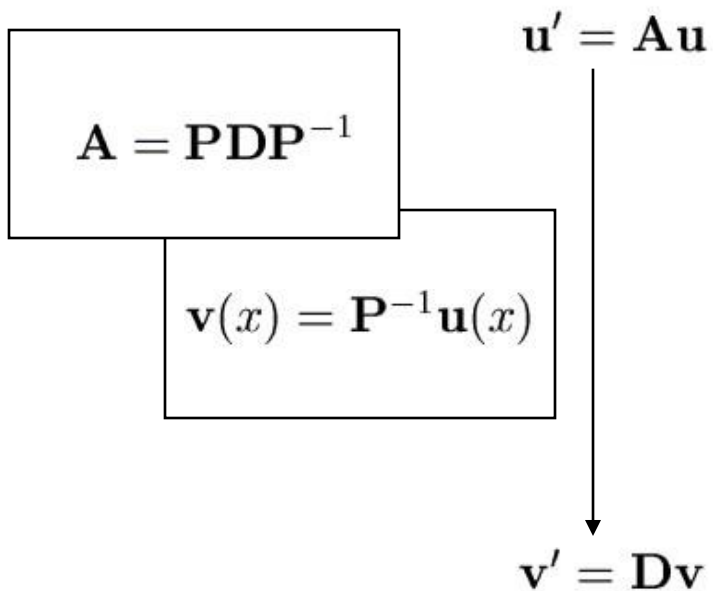
Tableau à deux indices

*On calcule la j-ème composante du vecteur
des inconnues à la i-ème abscisse temporelle*

Vecteur

*On calcule la j-ème
composante du vecteur des
inconnues.*

Stabilité des systèmes



Le problème différentiel initial est équivalent à n équations scalaires

$$v'_i(x) = \lambda_i v_i(x), \quad i = 1, \dots, n$$

Exemple

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

$$\begin{cases} u'(x) = -u^2(x) + v(x) \\ v'(x) = -50v^2(x) + x, & x \in [0, 4] \\ u(0) = 1 \\ v(0) = 1 \end{cases}$$

$$u' = f_1(x, u, v)$$

$$v' = f_2(x, u, v)$$

$$\begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} \\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} \end{bmatrix} = \begin{bmatrix} -2u(x) & 1 \\ 0 & -100v(x) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial v} \\ \frac{\partial f_2}{\partial u} & \frac{\partial f_2}{\partial v} \end{bmatrix} = \begin{bmatrix} -2v(x) & 1 \\ 0 & -100v(x) \end{bmatrix} \stackrel{\text{A}}{=} \underline{\underline{\text{A}}}$$

$$\lambda_1(x) = -100v(x)$$

$$\lambda_2(x) = -2v(x)$$

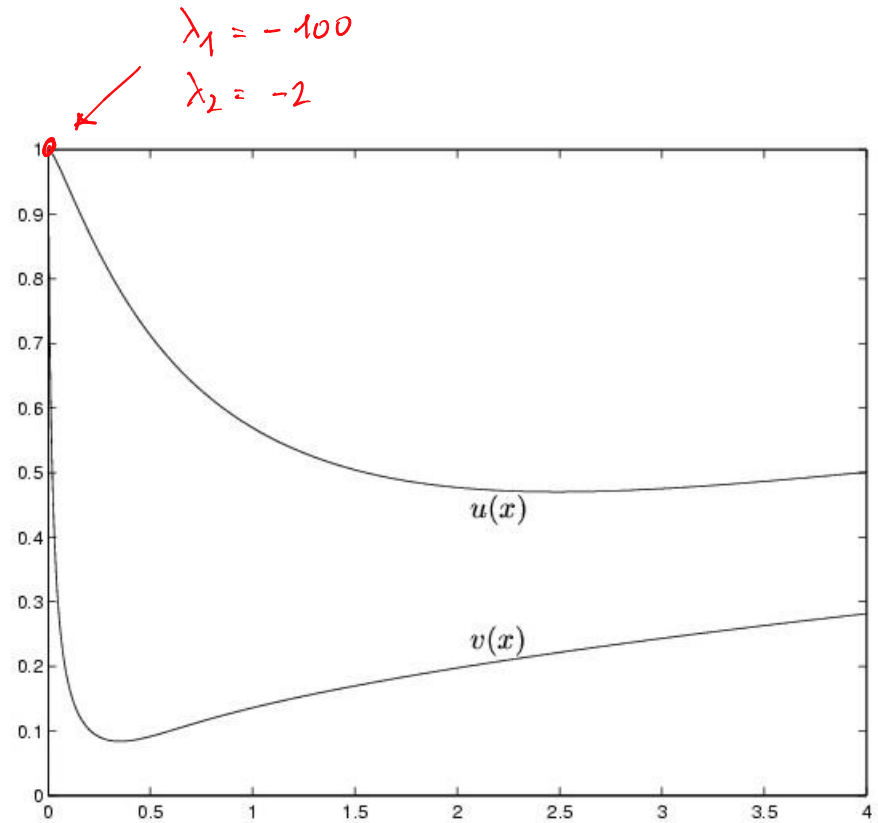
$$\det [\underline{\underline{\text{A}}} - \lambda \underline{\underline{\text{I}}}] = 0$$

$$\det \begin{bmatrix} -2v - \lambda & 1 \\ 0 & -100v - \lambda \end{bmatrix} = 0$$

$$(-2v - \lambda)(-100v - \lambda)$$

$$\begin{bmatrix} \left. \frac{\partial f_1}{\partial u} \right|_x & \left. \frac{\partial f_1}{\partial v} \right|_x \\ \left. \frac{\partial f_2}{\partial u} \right|_x & \left. \frac{\partial f_2}{\partial v} \right|_x \end{bmatrix} = \begin{bmatrix} -2u & 1 \\ 0 & -100v \end{bmatrix}$$

Il s'agit d'un
problème raide



$$\det \begin{bmatrix} -2u(x) - \lambda & 1 \\ 0 & -100v(x) - \lambda \end{bmatrix} = 0$$

$$\begin{aligned} \lambda_1(x) &= -100v(x) \\ \lambda_2(x) &= -2u(x) \end{aligned}$$

Euler explicite

$$\begin{cases} U_{i+1} = U_i + h (-U_i^2 + V_i) \\ V_{i+1} = V_i + h (-50 V_i^2 + X_i) \end{cases}$$

$$\begin{cases} U_{i+1} = U_i + h (-U_{i+1}^2 + V_{i+1}) \\ V_{i+1} = V_i + h (-50 V_{i+1}^2 + X_{i+1}) \end{cases}$$

Euler implicite

Newton-Raphson et Euler implicite

$$\left\{ \begin{array}{l} \overbrace{U_{i+1} - U_i - h (-U_{i+1}^2 + V_{i+1})}^{g_1(U_{i+1}, V_{i+1})} = 0 \\ \underbrace{V_{i+1} - V_i - h (-50 V_{i+1}^2 + X_{i+1})}_{g_2(U_{i+1}, V_{i+1})} = 0 \end{array} \right.$$

*Notation compacte :
les vecteurs sont en gras.*

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

Comment appliquer la méthode de Newton-Raphson à ce système d'équations non-linéaires ?

On fournit \mathbf{x}_0

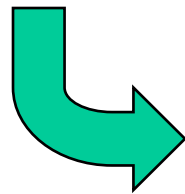
Tant que $\Delta \mathbf{x} > \epsilon$, on calcule \mathbf{x}_{i+1} à partir de \mathbf{x}_i avec

$$\frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_i) \overbrace{(\mathbf{x}_{i+1} - \mathbf{x}_i)}^{\Delta \mathbf{x}} = -\mathbf{f}(\mathbf{x}_i)$$

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \Delta \mathbf{x}$$

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

$$\underline{\underline{I}} - h \underline{\underline{A}}$$



$$\begin{bmatrix} \left. \frac{\partial g_1}{\partial x_1} \right|_{(x_1, x_2)} & \left. \frac{\partial g_1}{\partial x_2} \right|_{(x_1, x_2)} \\ \left. \frac{\partial g_2}{\partial x_1} \right|_{(x_1, x_2)} & \left. \frac{\partial g_2}{\partial x_2} \right|_{(x_1, x_2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - h \begin{bmatrix} -2x_1 & 1 \\ 0 & -100x_2 \end{bmatrix}$$

$\underline{\underline{A}}$

Convergence de Newton-Raphson ?

$$\begin{bmatrix} \left. \frac{\partial g_1}{\partial x_1} \right|_{(x_1, x_2)} & \left. \frac{\partial g_1}{\partial x_2} \right|_{(x_1, x_2)} \\ \left. \frac{\partial g_2}{\partial x_1} \right|_{(x_1, x_2)} & \left. \frac{\partial g_2}{\partial x_2} \right|_{(x_1, x_2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - h \begin{bmatrix} -2x_1 & 1 \\ 0 & -100x_2 \end{bmatrix}$$

CECI EST QUASIMENT
LA CONDITION DE STABILITE
DE LA METHODE
D'EULER EXPLICITE

Euler implicite

On fournit \mathbf{U}_0

Pour chaque i , on calcule $\mathbf{x} = \mathbf{U}_{i+1}$ à partir de \mathbf{U}_i en résolvant le problème

$$\underbrace{\mathbf{U}_{i+1} - \mathbf{U}_i - hf(\mathbf{U}_{i+1}, X_{i+1})}_{\mathbf{g}(\mathbf{x})} = 0$$

par la méthode de Newton-Raphson en partant de \mathbf{U}_i

On fournit $\mathbf{x}_0 = \mathbf{U}_i$

Newton-Raphson

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

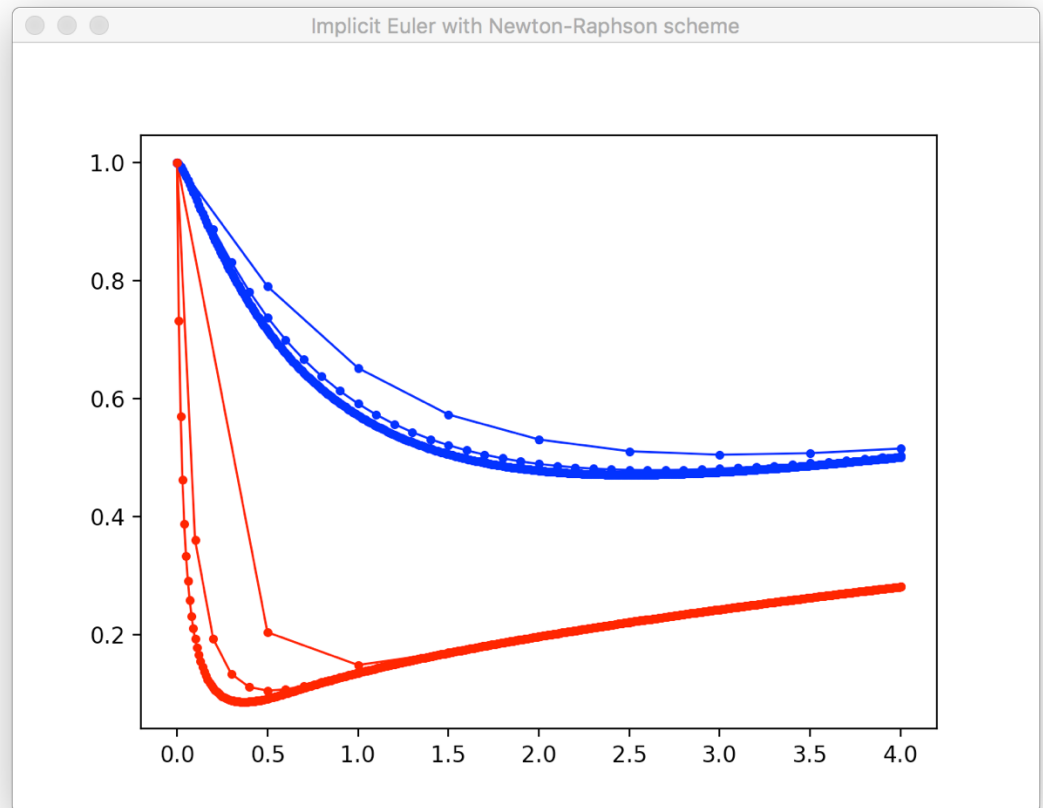
$$\frac{\partial \mathbf{g}}{\partial \mathbf{x}}(\mathbf{x}_j) \overbrace{(\mathbf{x}_{j+1} - \mathbf{x}_j)}^{\Delta \mathbf{x}} = -\mathbf{g}(\mathbf{x}_j)$$

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \Delta \mathbf{x}$$

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

Schémas
imbriqués

Utilisation conjointe des méthodes d'Euler implicite et de Newton-Raphson

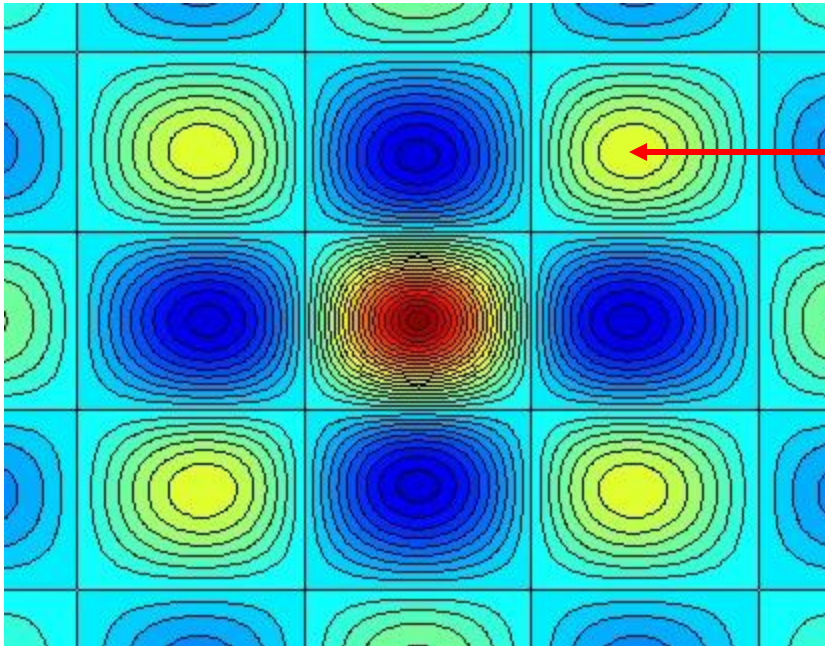
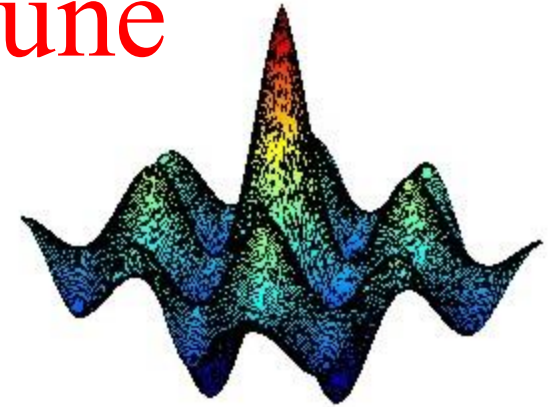


**Minimiser une fonction
non-linéaire est très très difficile !**

Les méthodes numériques... existantes

**Des propos dignes de l'un de ces fous
énigmatiques qui hantent les forêts enchantées
dans les pièces de Shakespeare**

Trouver le maximum d'une fonction non-linéaire est une tâche très très difficile...



Un maximum local ?

Comment savoir la présence ou l'absence d'une fosse plus profonde sur base d'informations locales....

Impossible ?

Minimiser une fonction non-linéaire est très très difficile !

Le message est que rien n'est su de manière sûre. Il y a des choses dont nous savons que nous les savons. Il y a des inconnues connues, c'est-à-dire des choses dont nous savons maintenant que nous ne les savons pas.

Mais, il y a aussi des inconnues inconnues. Il y a des choses dont nous savons que nous ne les savons pas Et chaque année, nous découvrons un peu plus de ces inconnues inconnues.

... Il y a une autre façon de dire cela, c'est que l'inexistence de preuves n'est pas une preuve d'inexistence

Secrétaire à la Défense US
à l'OTAN à Bruxelles, juin 2002 !

(Lewis Lapham)

Les méthodes numériques... existantes

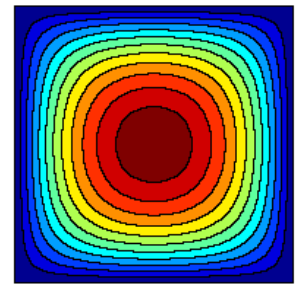
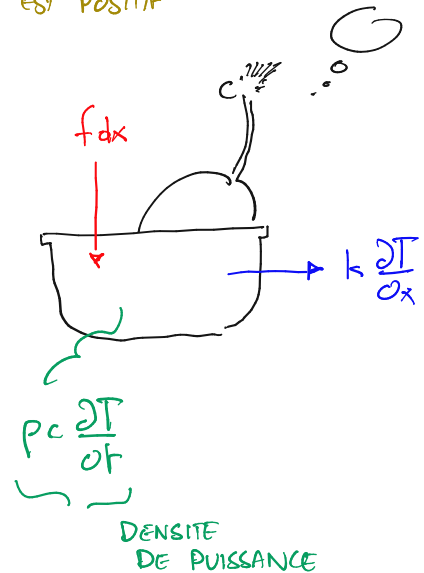
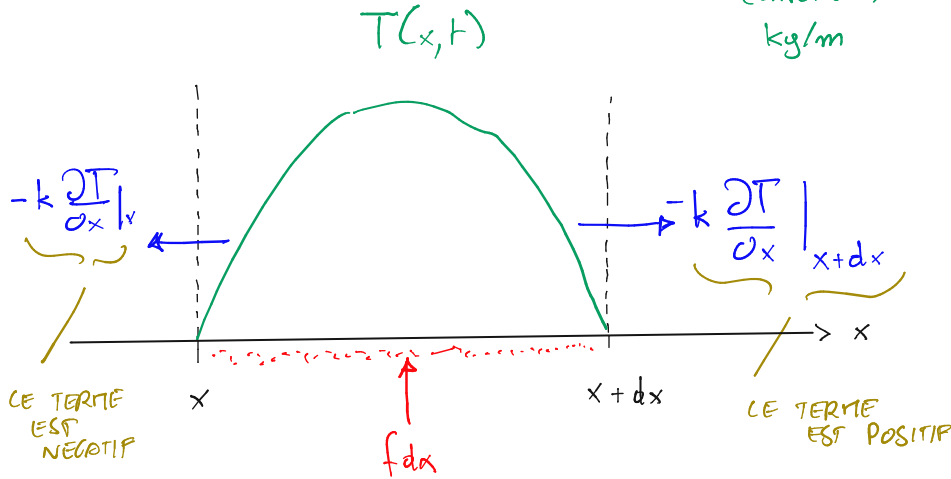
**Des propos dignes de l'un de ces fous
énigmatiques qui hantent les forêts enchantées
dans les pièces de Shakespeare**

Un peu de physique !

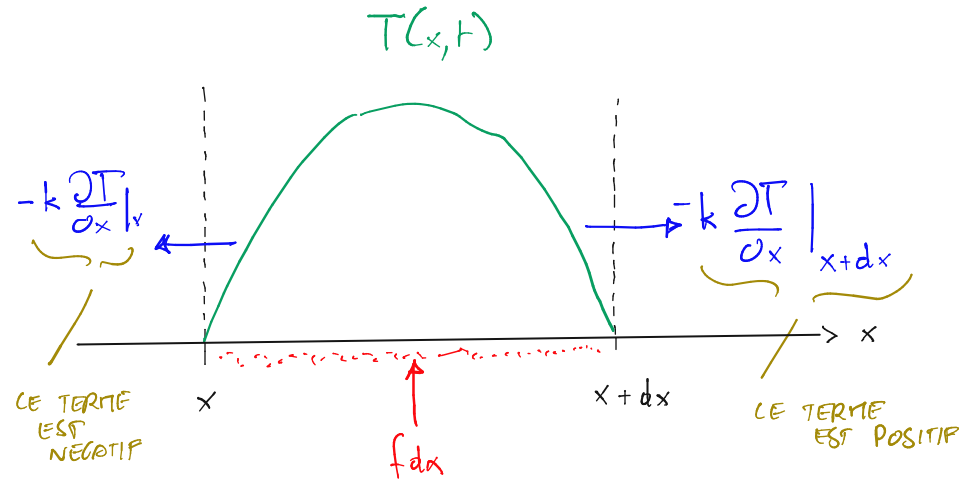
$$\rho c \frac{\partial T}{\partial t} \phi x = f dx + k \frac{\partial T}{\partial x} \Big|_{x+dx} - k \frac{\partial T}{\partial x} \Big|_x$$

MASSE VOLUMIQUE (LINEIQUE) kg/m CHALEUR SPECIFIQUE

$$\rho c \frac{\partial T}{\partial t} = f + k \frac{\partial^2 T}{\partial x^2}$$

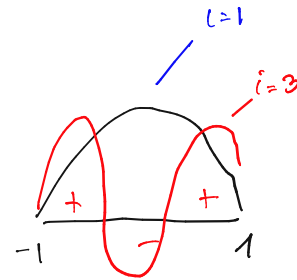


Solution analytique !



$$u(x) = \sum_{i=1}^{\infty} C_i \sin\left(\frac{i(x+1)\pi}{2}\right)$$

i=1
IMPAIRS



$$\frac{d^2 u}{dx^2} + 1 = 0$$

$$u(-1) = 0$$

$$u(1) = 0$$

$$-1 - \sum C_i \sin\left[\dots\right] \frac{i^2 \pi^2}{4} = 0$$

$$\int_{-1}^1 \sin[\dots] \sum C_i \frac{i^2 \pi^2}{4} \sin[i\dots] = \int_{-1}^1 \sin[\dots] dx \quad \forall j$$

$$\sum C_i \frac{i^2 \pi^2}{4} \int_{-1}^1 \underbrace{\sin[\dots] \sin[i\dots]}_{=0 \quad i \neq j} = \int_{-1}^1 \sin[\dots] dx$$

$$= 0 \quad i \neq j$$

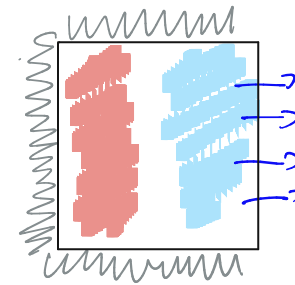
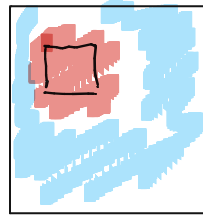
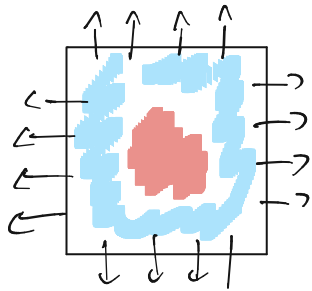
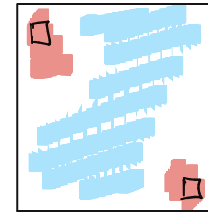
$$\neq 0 \quad i = j$$

$$C_j = \frac{16}{j^3 \pi^3}$$

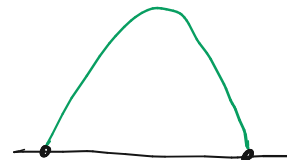
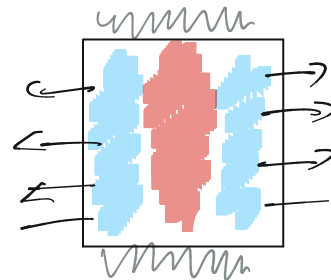
$$\frac{4}{j^3 \pi^3}$$

$$j^2 \frac{\pi^2}{4} C_j = \frac{\int_{-1}^1 \sin(j\dots) dx}{\int_{-1}^1 \sin^2(j\dots) dx}$$

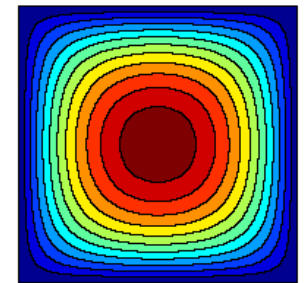
$$k \left[\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right] + 1 = 0$$



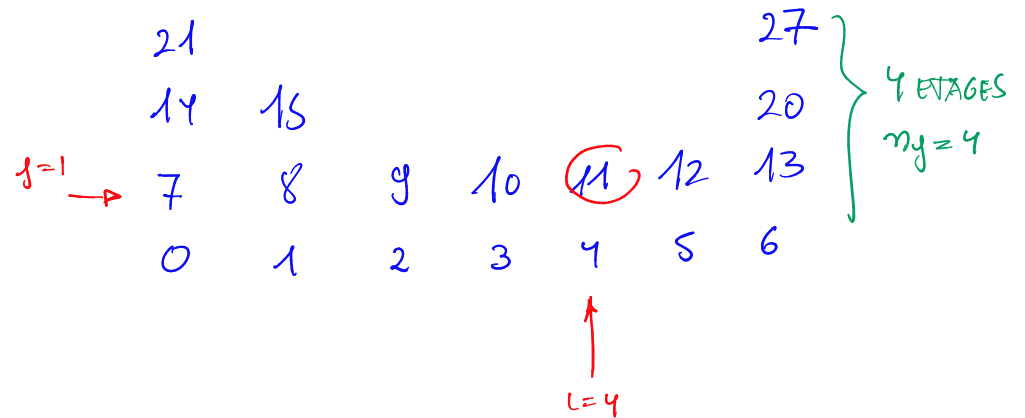
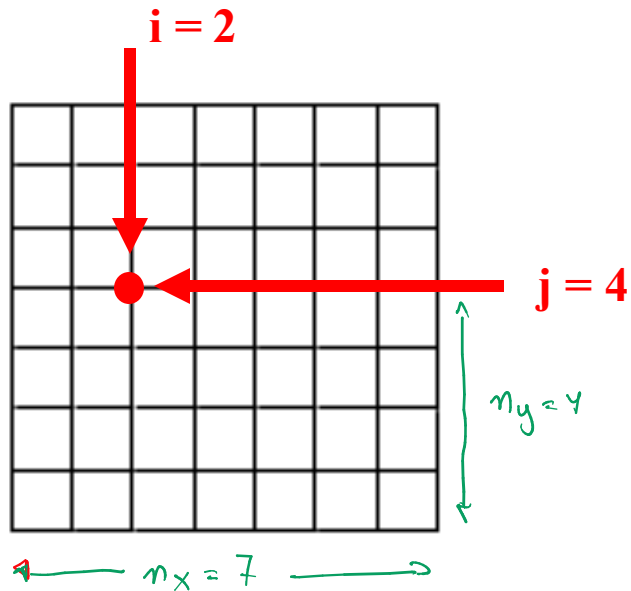
En
deux
dimension !



$$k \frac{d^2 u}{dx^2} = -1$$

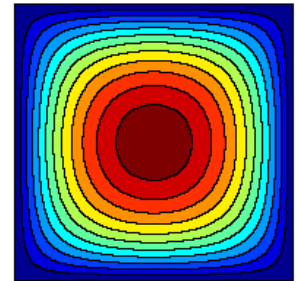


Un peu de numérotation



$$\text{INDEX} = 4 + 1 * m_x = 11$$

i j



Plan du cours de méthodes numériques

Comment résoudre
numériquement un
problème aux
valeurs initiales ?

Comment interpoler
une fonction ?

Comment dériver
numériquement
une fonction ?

Comment approximer
une fonction ?

Comment intégrer
numériquement
une fonction ?

Comment résoudre
numériquement un
problème aux
conditions frontières ?

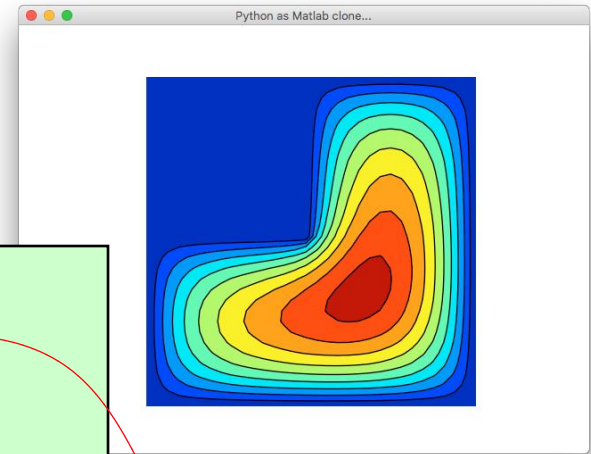
Et les équations non-
linéaires ?

Et les méthodes itératives ?

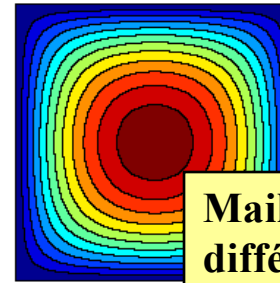
*Comment résoudre numériquement
une équation différentielle ordinaire ?*

*Comment résoudre numériquement
une équation aux dérivées partielles ?*

Comment résoudre
numériquement une
équation aux dérivées
partielles ?

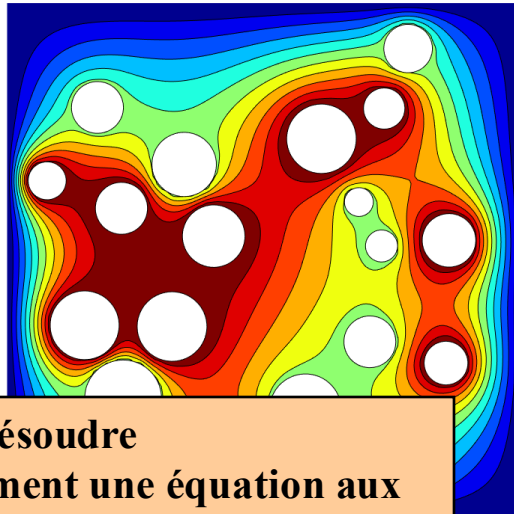
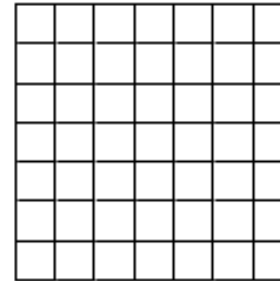


A quoi servent les méthodes numériques ?



LEPL1104

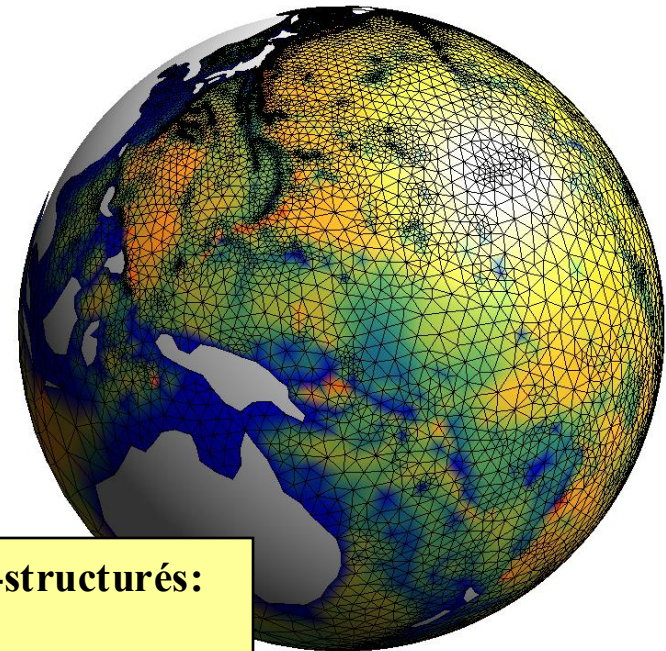
Maillages structurés:
différences finies



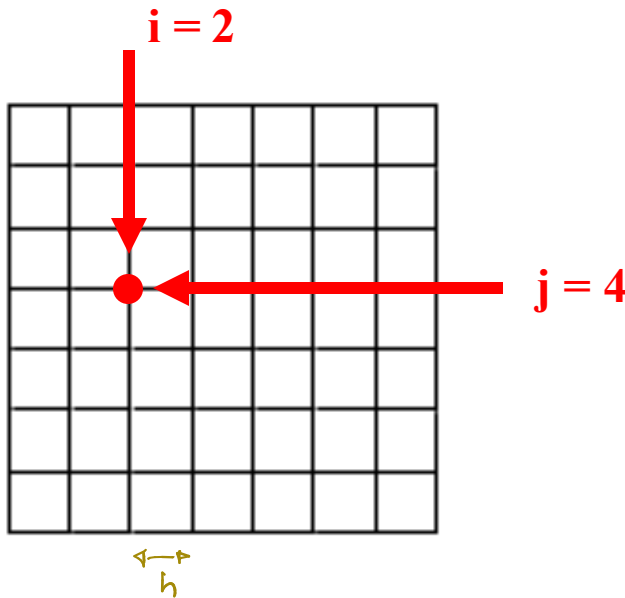
Comment résoudre numériquement une équation aux dérivées partielles avec des conditions aux limites ?

LEPL1110

Maillages non-structurés:
éléments finis

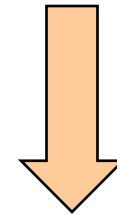


Grille / Maillage

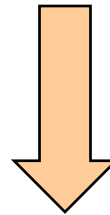


$$X_i = a + ih,$$

$i = 0, \dots, m$



$$\mathbf{X}_{ij} = (X_i, Y_j) = (ih, jh)$$

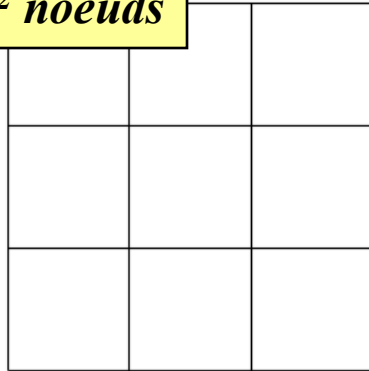


$$U_{ij} = u^h(\mathbf{X}_{ij}) \approx u(\mathbf{X}_{ij})$$

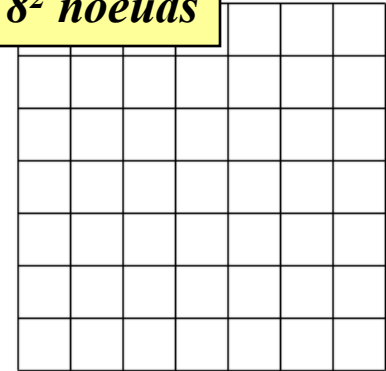
$$h = (b - a)/(m)$$

Méthode des différences finies

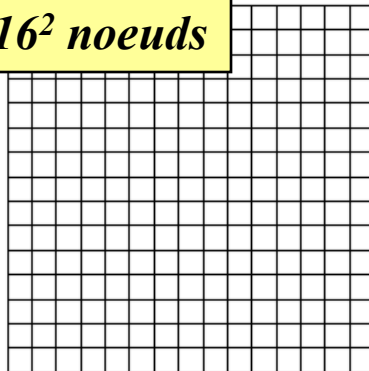
4^2 noeuds



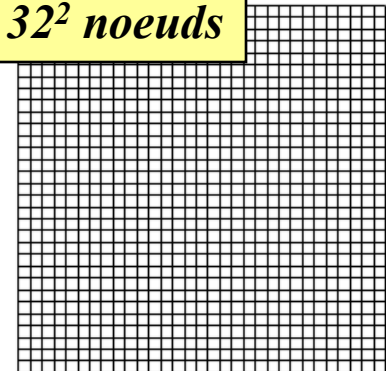
8^2 noeuds



16^2 noeuds

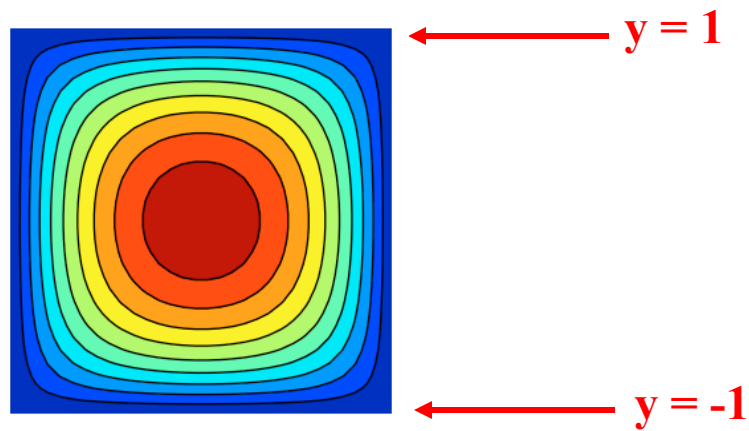


32^2 noeuds



Exemple

$$\begin{aligned}\nabla^2 u(x, y) + 1 &= 0, & (x, y) \in \Omega, \\ u(x, y) &= 0, & (x, y) \in \partial\Omega,\end{aligned}$$



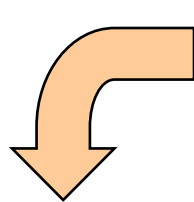
$$u(x, y) = \sum_{i, j \text{ impairs}} C_{ij} \sin\left(\frac{i\pi(x+1)}{2}\right) \sin\left(\frac{j\pi(y+1)}{2}\right)$$

Comment trouver C_{ij} ?

$$\nabla^2 u(x, y) + 1 = 0,$$

$$\sum_{i,j \text{ impairs}} \frac{-\pi^2(i^2 + j^2)}{4} C_{ij} \sin\left(\frac{i\pi(x+1)}{2}\right) \sin\left(\frac{j\pi(y+1)}{2}\right) + 1 = 0,$$

En vertu de l'orthogonalité des sinus,



$$\frac{\pi^2(i^2 + j^2)}{4} C_{ij} - \underbrace{\int_{\Omega} \sin\left(\frac{i\pi(x+1)}{2}\right) \sin\left(\frac{j\pi(y+1)}{2}\right) d\Omega}_{16/(ij\pi^2)} = 0,$$

$$u(x, y) = \sum_{i,j \text{ impairs}} \frac{64}{\pi^4(i^2 + j^2)ij} \sin\left(\frac{i\pi(x+1)}{2}\right) \sin\left(\frac{j\pi(y+1)}{2}\right)$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + 1 = 0$$

Différences finies

$$\left(\frac{U_{i+1,j} - 2U_{i,j} + U_{i-1,j}}{h^2} + \frac{U_{i,j+1} - 2U_{i,j} + U_{i,j-1}}{h^2} \right) + 1 = 0$$

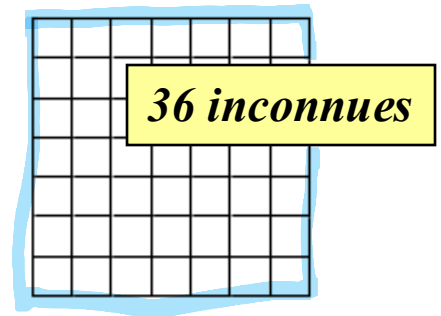


$$\frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2} + 1 = 0$$

(n-2)(n-2) équations linéaires

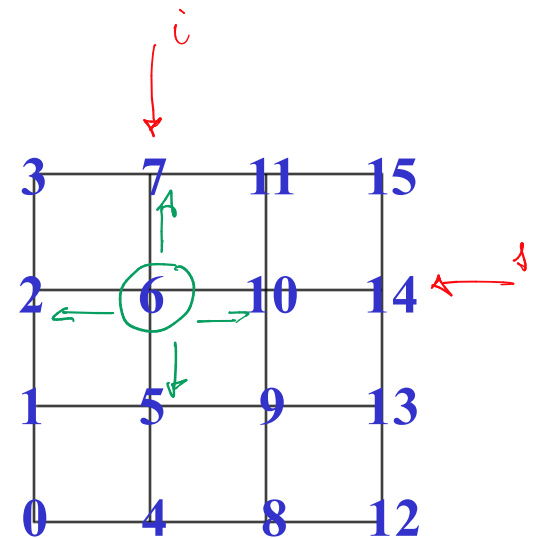
(n-2)(n-2) inconnues

Il suffit donc de résoudre un grand système linéaire



Programme Python....

```
def poissonSolve(nx,ny):
    n = nx*ny; h = 2/(ny-1)
    A = eye(n); B = zeros(n)
    for i in range(1,nx-1):
        for j in range(1,ny-1):
            index = i + j*nx
            A[index,index] = 4.0
            A[index,index-1] = -1.0
            A[index,index+1] = -1.0
            A[index,index-nx] = -1.0
            A[index,index+nx] = -1.0
            B[index] = 1
    return solve(A,B).reshape(ny,nx)
```



$$\frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2} + 1 = 0$$

Comment résoudre le système discret avec python...

```
A = inv(A)  
x = A @ b
```

```
x = solve(A,b)
```

A frequent misuse of `inv` arises when solving the system of linear equations. One way to solve this is with

```
x = inv(A) @ b
```

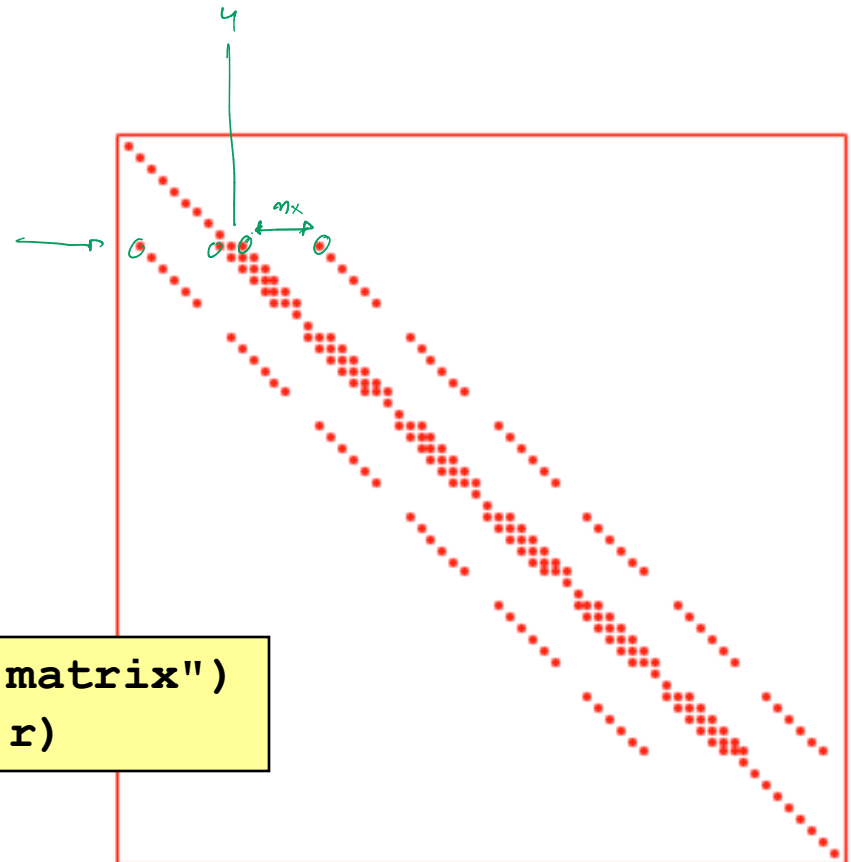
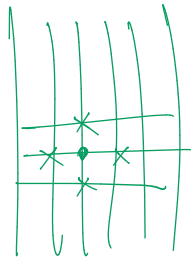
A better way, from both an execution time and numerical accuracy standpoint, is to use the `solve` function

```
x = solve(A,b)
```

This produces the solution using Gaussian elimination, without forming the inverse.

*On résout un système linéaire,
on ne l'inverse jamais....
(J. Meinguet)*

Les différences finies produisent une matrice creuse...



```
plt.figure("Sparsity of the matrix")  
plt.spy(A,marker='o',color='r')
```

Utiliser scipy.sparse !



```
from scipy.sparse import dok_matrix
from scipy.sparse.linalg import spsolve

n = nx*ny; h = 2/(ny-1)
A = dok_matrix((n,n), dtype=float32);
B = zeros(n)
for i in range(n):
    A[i,i] = 1.0
for i in range(1,nx-1):
    for j in range(1,ny-1):
        index = i + j*nx
        A[index,index] = 4.0
        A[index,index-1] = -1.0
        A[index,index+1] = -1.0
        A[index,index-nx] = -1.0
        A[index,index+nx] = -1.0
        B[index] = 1
U = spsolve(A/(h*h).tocsr(), B).reshape(ny, nx)
```

MATRICE
CREUSE

SOLVEUR
CREUX

Pour les matrices
de grande taille
c'est plus rapide !

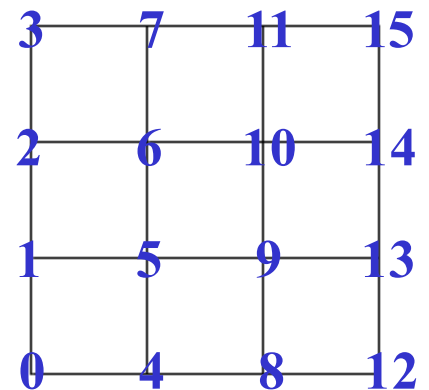
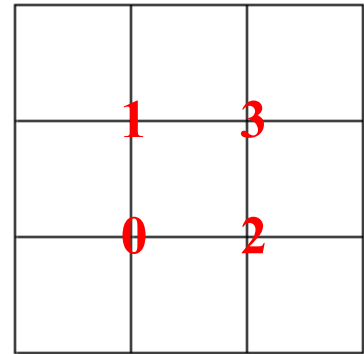
```
bash-3.2$ python poissonSimple.py
=== Considering nx=ny=150
=== Full solver      : elapsed time is 68.746733 seconds.
=== Sparse solver   : elapsed time is 1.635676 seconds.
```



<https://docs.scipy.org/doc/scipy-0.9.0/reference/sparse.html>

Petits problèmes de numérotation...

```
n = nx*ny; h = 2/(ny-1)
A = zeros((n,n))
B = ones((nx-2)*(ny-2))
map = zeros((nx-2)*(ny-2),dtype=int)
for i in range(1,nx-1):
    for j in range(1,ny-1):
        index = i + j*nx
        map[i-1 + (j-1)*(nx-2)] = index
        A[index,index] = 4.0
        A[index,index-1] = -1.0
        A[index,index+1] = -1.0
        A[index,index-nx] = -1.0
        A[index,index+nx] = -1.0
A = A / (h*h)
V = solve(A[map,:][:,map],B)
```

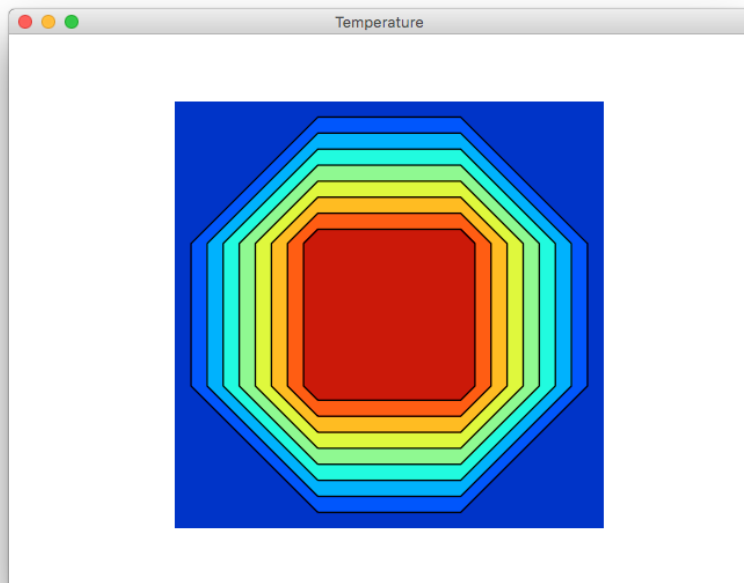


```

U = zeros((ny,nx))
U[1:ny-1,1:nx-1] = V.reshape(ny-2,nx-2)

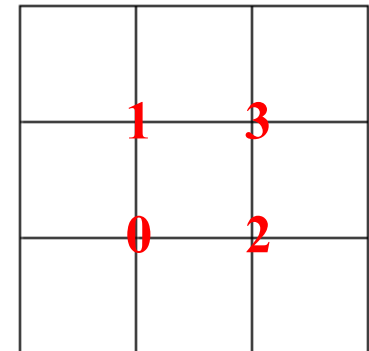
X,Y = meshgrid(linspace(-1,1,nx),linspace(-1,1,ny))
myColorMap = matplotlib.cm.jet
plt.contour(X,Y,U,10,cmap=myColorMap)
plt.contour(X,Y,U,10,colors='k',linewidths=1)
plt.axis("equal"); plt.axis("off")

```

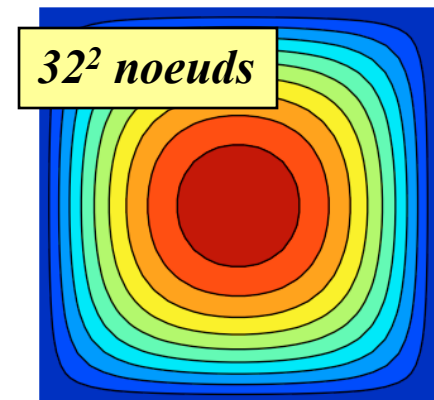
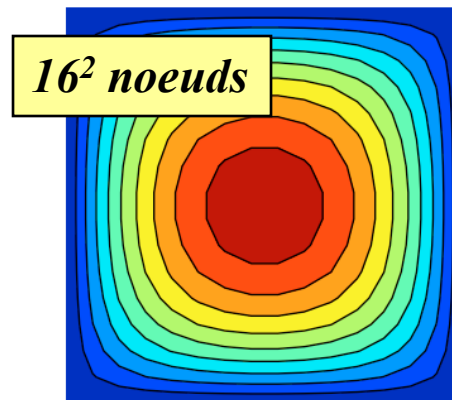
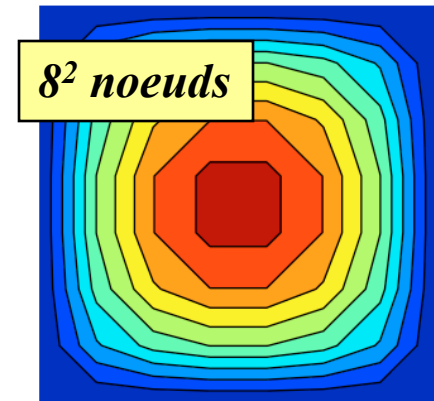
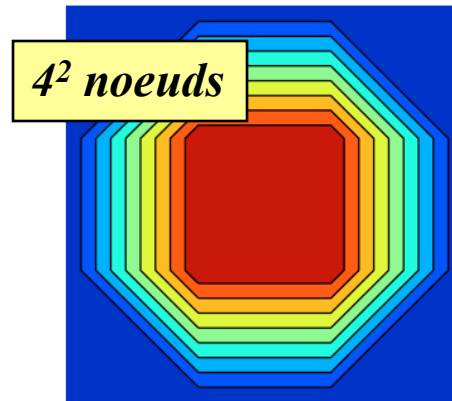


Et faire un joli plot

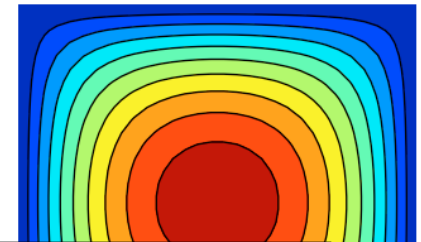
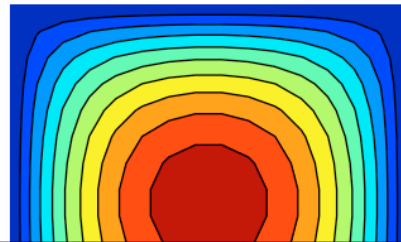
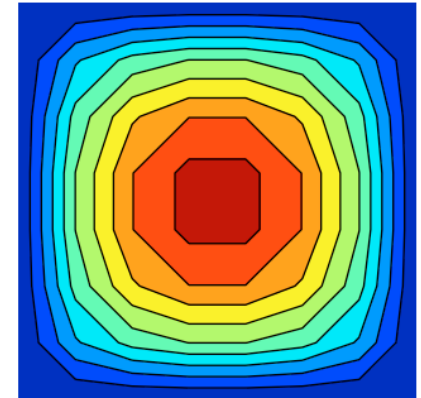
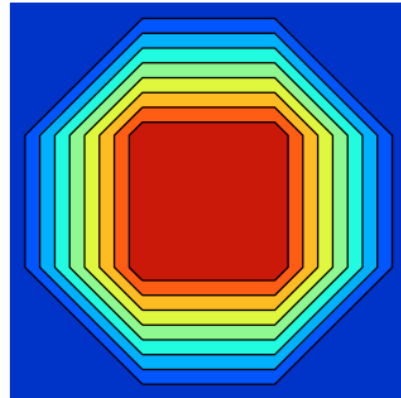
| | | | |
|---|---|----|----|
| 3 | 7 | 11 | 15 |
| 2 | 6 | 10 | 14 |
| 1 | 5 | 9 | 13 |
| 0 | 4 | 8 | 12 |



Est-ce que cela converge ?



Quelle est
la précision ?



```
bash-3.2$ python poissonSuperBasic.py
=== Discretization nx =      4 : error = 1.9161507e-02
=== Discretization nx =      8 : error = 4.4728014e-03
=== Discretization nx =     16 : error = 1.0188850e-03
=== Discretization nx =     32 : error = 2.4094496e-04

===== Estimated order of convergence = 2.1044545e+00
```

Théoriquement...

Soit $u(x, y)$ la solution exacte d'un problème de Poisson aux conditions aux limites sur un domaine $\Omega \subset \mathbb{R}^2$.

Si la fonction u et toutes ses dérivées partielles jusqu'au quatrième ordre sont continues sur le domaine fermé $\bar{\Omega}$, alors il existe une constante positive telle que :

Théorème 6.1.

$$\max |u(X_i, Y_j) - \underbrace{u^h(X_i, Y_j)}_{U_{ij}}| \leq C M h^2$$

où M est la valeur maximale atteinte par une des dérivées quatrièmes de u sur $\bar{\Omega}$ et u^h est la solution discrète obtenue au moyen d'un schéma à 5 points basé sur des différences finies centrées du second ordre.

Ce résultat n'est pas évident a priori...

Démonstration :

Isaacson and Keller, Analysis of Numerical Methods (1966)

Plus compliqué :-)

```
m = 5  
map = numgrid(2*m+1)  
print(map)
```

```
[[ 0  0  0  0  0  0  0  0  0  0  0]  
 [ 0  1  2  3  4  5  6  7  8  9  0]  
 [ 0 10 11 12 13 14 15 16 17 18  0]  
 [ 0 19 20 21 22 23 24 25 26 27  0]  
 [ 0 28 29 30 31 32 33 34 35 36  0]  
 [ 0  0  0  0  0  0 37 38 39 40  0]  
 [ 0  0  0  0  0  0 41 42 43 44  0]  
 [ 0  0  0  0  0  0 45 46 47 48  0]  
 [ 0  0  0  0  0  0 49 50 51 52  0]  
 [ 0  0  0  0  0  0 53 54 55 56  0]  
 [ 0  0  0  0  0  0  0  0  0  0  0]]
```

Et pourtant si simple...

```
m = 3;  
map = numgrid(2*m+1)  
A = delsq(map) ; print(A.toarray())
```

```
[[ 4. -1.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
 [-1.  4. -1.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
 [ 0. -1.  4. -1.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.]  
 [ 0.  0. -1.  4. -1.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.]  
 [ 0.  0.  0. -1.  4.  0.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.]  
 [-1.  0.  0.  0.  0.  4. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
 [ 0. -1.  0.  0.  0. -1.  4. -1.  0.  0.  0.  0.  0.  0.  0.  0.]  
 [ 0.  0. -1.  0.  0.  0. -1.  4. -1.  0.  0.  0.  0.  0.  0.  0.]  
 [ 0.  0.  0. -1.  0.  0.  0. -1.  4. -1. -1.  0.  0.  0.  0.  0.]  
 [ 0.  0.  0.  0. -1.  0.  0.  0. -1.  4.  0. -1.  0.  0.  0.  0.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  4. -1. -1.  0.  0.  0.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. -1. -1.  4.  0. -1.  0.  0.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  4. -1. -1.  0.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1. -1.  4.  0. -1.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  0.  4. -1.]  
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1. -1.  4.]]
```

Et finalement...

```
map = numgrid(32)
index = map[map>0]

A = delsq(map)
B = ones(size(index))
U = zeros(shape(map))
U[map>0] = spsolve(A,B)[index-1]

plt.contourf(U,10)
```

*Malheureusement,
les fonctions numgrid et delsq
de MATLAB ne font pas encore
partie de numpy et de scipy !*

