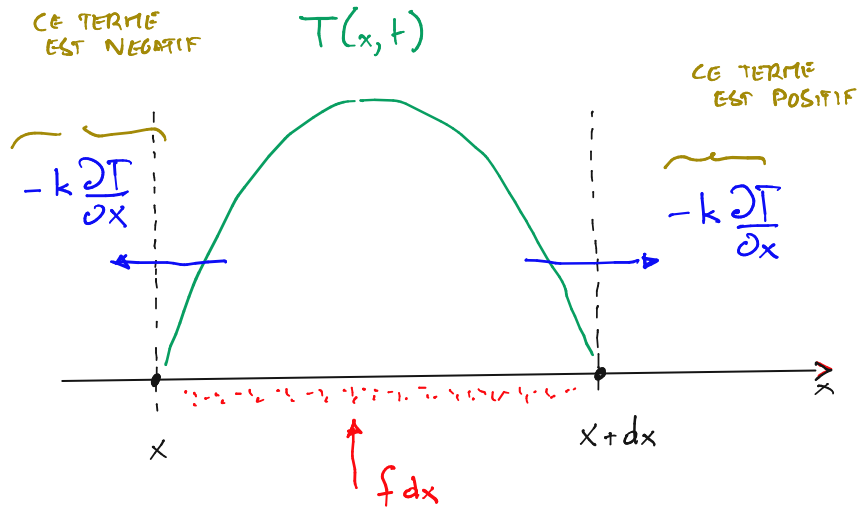


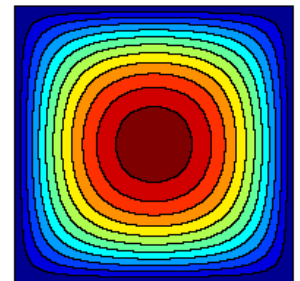
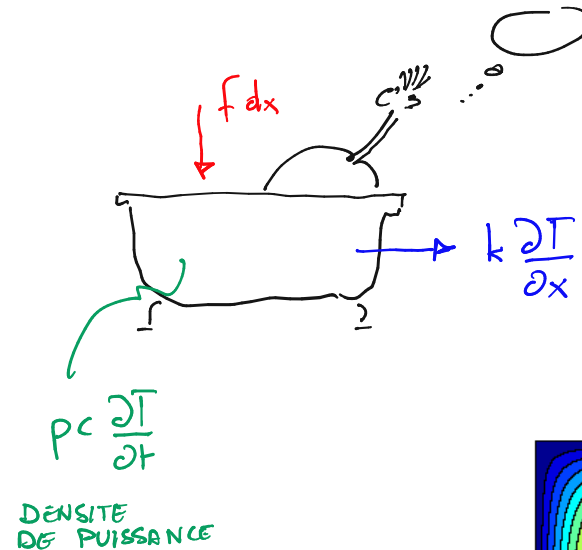
Un peu de physique !

$$dx \underbrace{\rho c}_{\substack{\text{MASSE} \\ \text{VOLUMIQUE}}} \underbrace{\frac{\partial T}{\partial t}}_{\substack{\text{CHALEUR} \\ \text{SPECIFIQUE}}} = \underbrace{f dx}_{\text{CONDUCTIBILITE THERMIQUE}} + k \frac{\partial T}{\partial x} \Big|_{x+dx} - k \frac{\partial T}{\partial x} \Big|_x$$

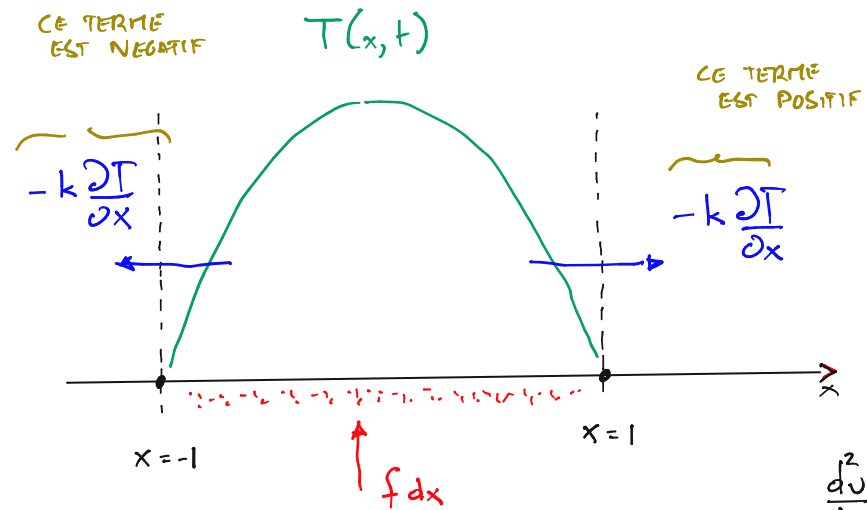


$$\rho c \frac{\partial T}{\partial t} = f + k \frac{\partial^2 T}{\partial x^2}$$

EQUATION DE LA CHALEUR



Solution analytique !



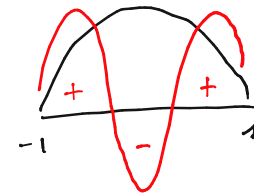
$$u(x) = \sum_{i=1, \text{ IMPAIRS}}^{\infty} C_i \sin\left(\frac{i(x+1)\pi}{2}\right)$$

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

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

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

$$\begin{aligned} &= 0 \quad \text{si } i \neq j \\ &\neq 0 \quad \text{si } i = j \end{aligned}$$



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

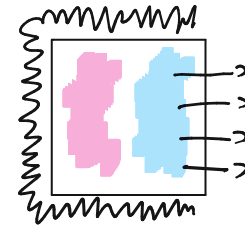
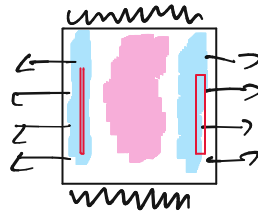
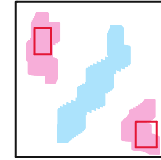
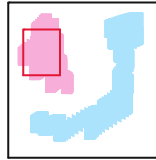
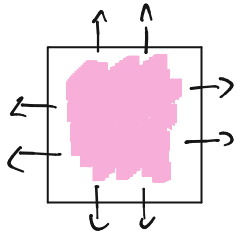
$$u(-1) = 0$$

$$u(1) = 0$$

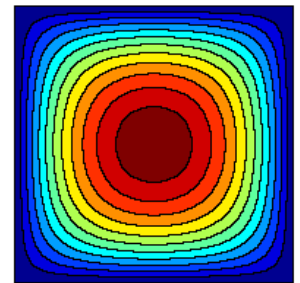
$$C_j = \frac{16}{j^3 \pi^3}$$

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

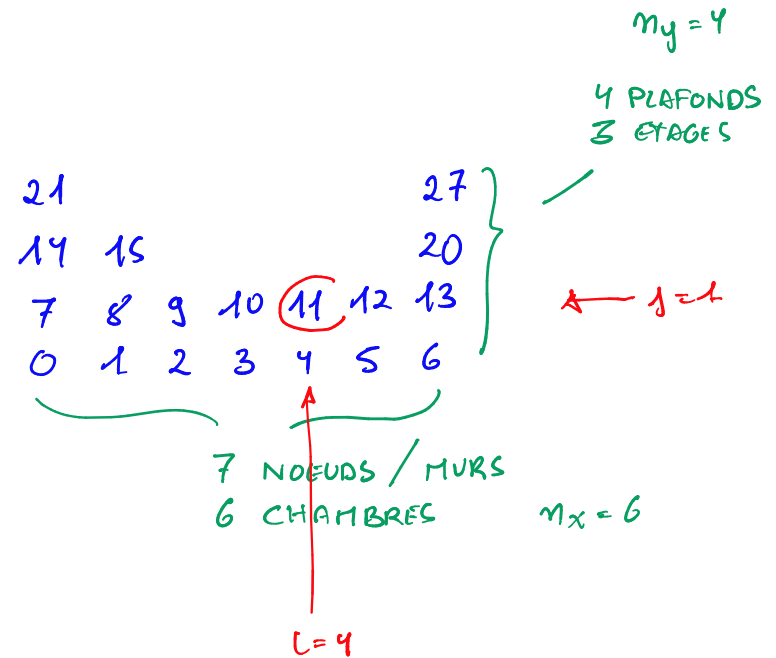
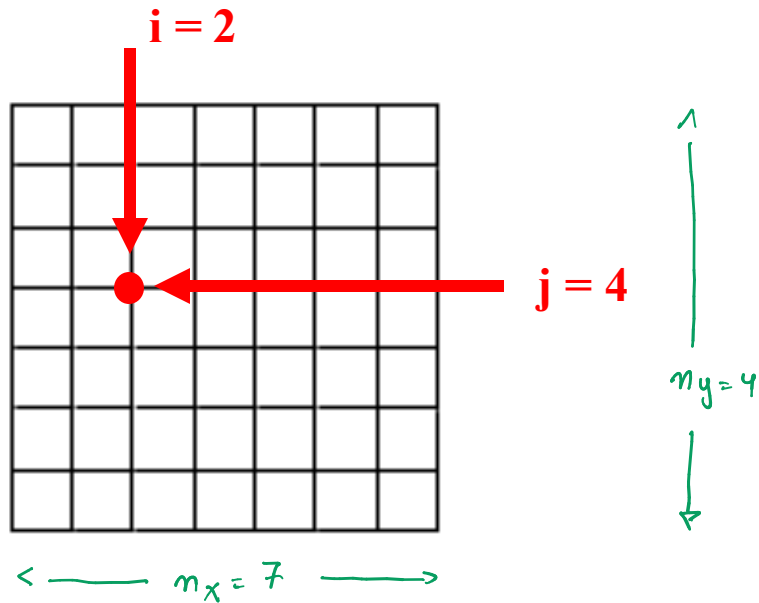
$$k \left[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right] + 1 = 0$$



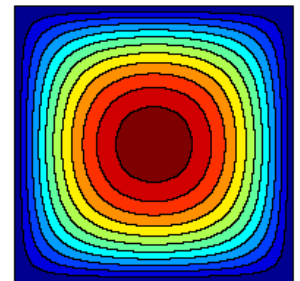
En
deux
dimension !



Un peu de numérotation

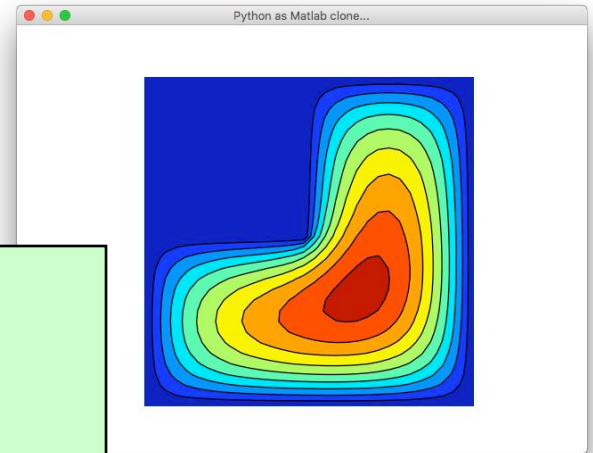


$$\text{INDEX} = \underset{i}{4} + \underset{j}{1} * n_x$$



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 résoudre
numériquement un
problème aux
conditions frontières ?

Comment intégrer
numériquement
une fonction ?

Et les équations non-
linéaires ?

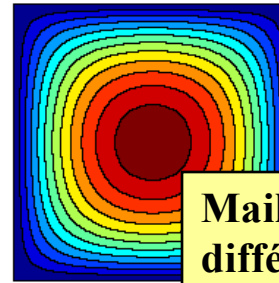
Et les méthodes itératives ?

*Comment résoudre numériquement
une équation différentielle ordinaire ?*

*Comment résoudre numériquement
une équation aux dérivées partielles ?*

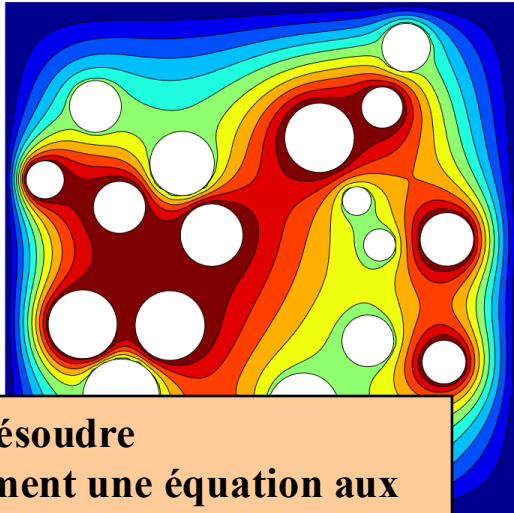
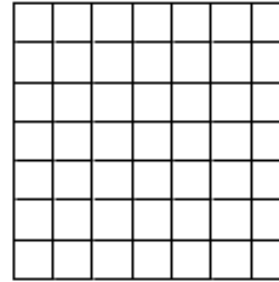
Comment résoudre
numériquement une
équation aux dérivées
partielles ?

A quoi servent les méthodes numériques ?



LEPL1104

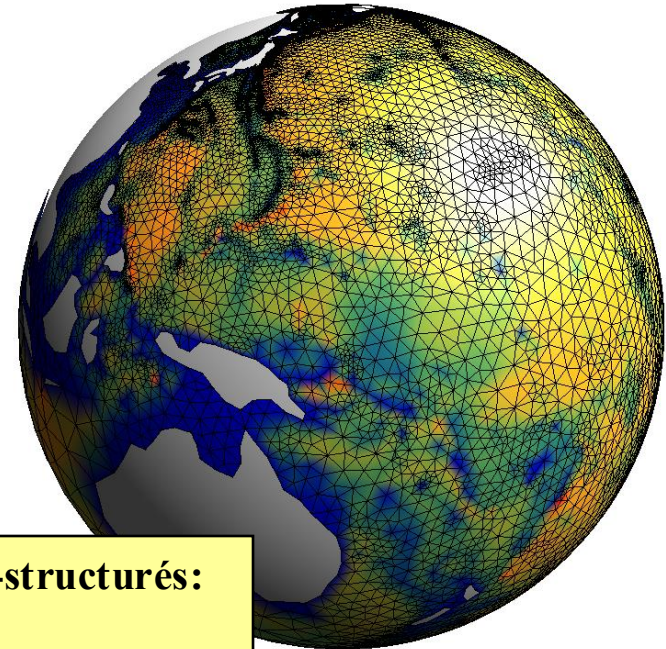
**Maillages structurés:
différences finies**



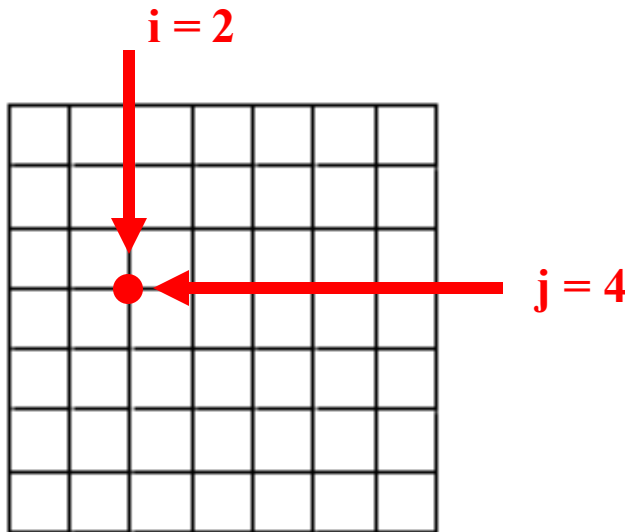
**Comment résoudre
numériquement une équation aux
dérivées partielles avec des
conditions aux limites ?**

LEPL1110

**Maillages non-structurés:
éléments finis**



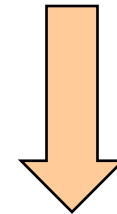
Grille / Maillage



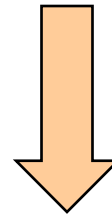
$$h = (b - a)/(m)$$

$$X_i = a + ih,$$

$i = 0, \dots, m$



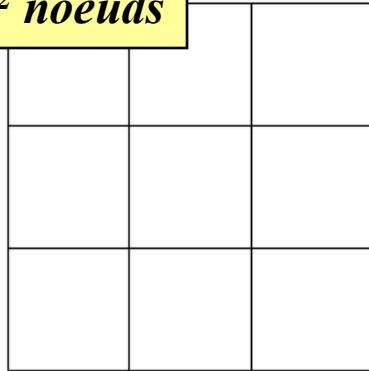
$$\mathbf{X}_{ij} = (X_i, Y_j) = (ih, jh)$$



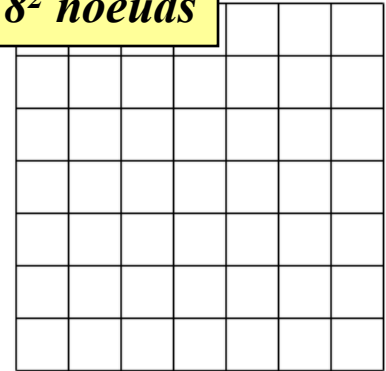
$$U_{ij} = u^h(\mathbf{X}_{ij}) \approx u(\mathbf{X}_{ij})$$

Méthode des différences finies

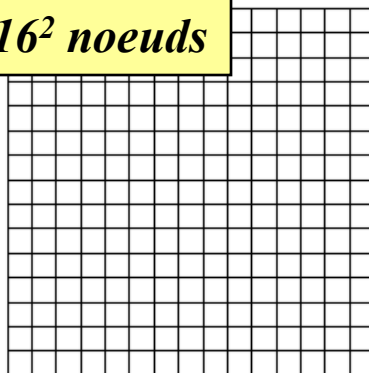
4² noeuds



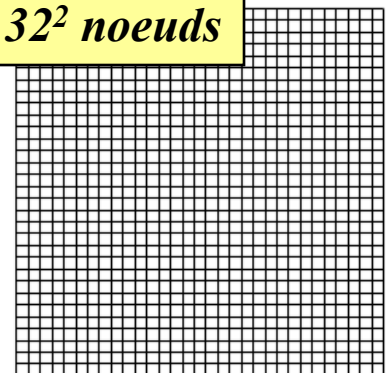
8² noeuds



16² noeuds

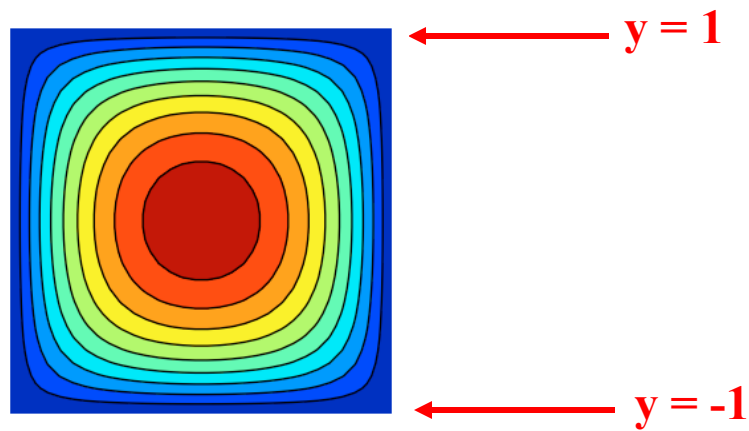


32² noeuds



Exemple

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



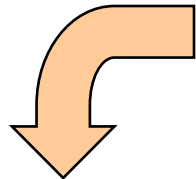
$$u(x, y) = \sum_{i, j \text{ impairs}} C_{ij} \sin\left(\frac{i\pi(x+1)}{2}\right) \sin\left(\frac{j\pi(y+1)}{2}\right)$$

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,


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

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

Différences finies

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

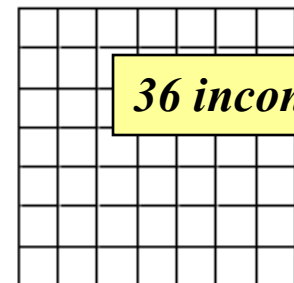


$$\frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2} + 1 = 0$$

(n-2)(n-2) équations linéaires

(n-2)(n-2) inconnues

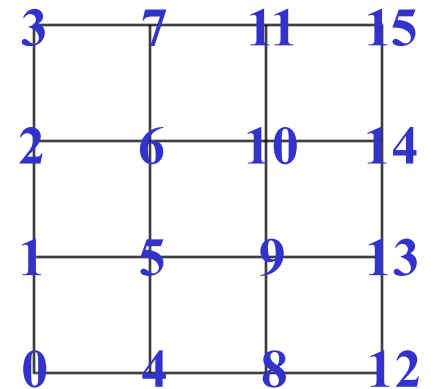
Il suffit donc de résoudre un grand système linéaire



36 inconnues

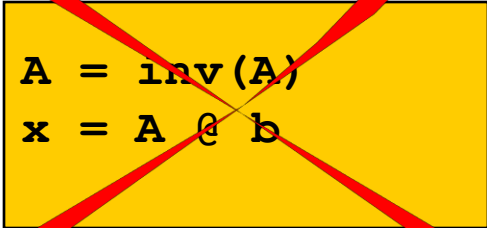
Programme Python....

```
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)
```

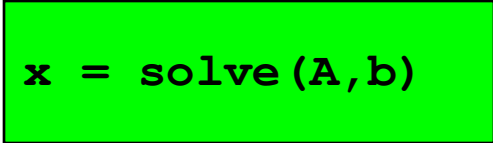


$$\frac{U_{i+1,j} + U_{i-1,j} + U_{i,j+1} + U_{i,j-1} - 4U_{i,j}}{h^2} + 1 = 0$$

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

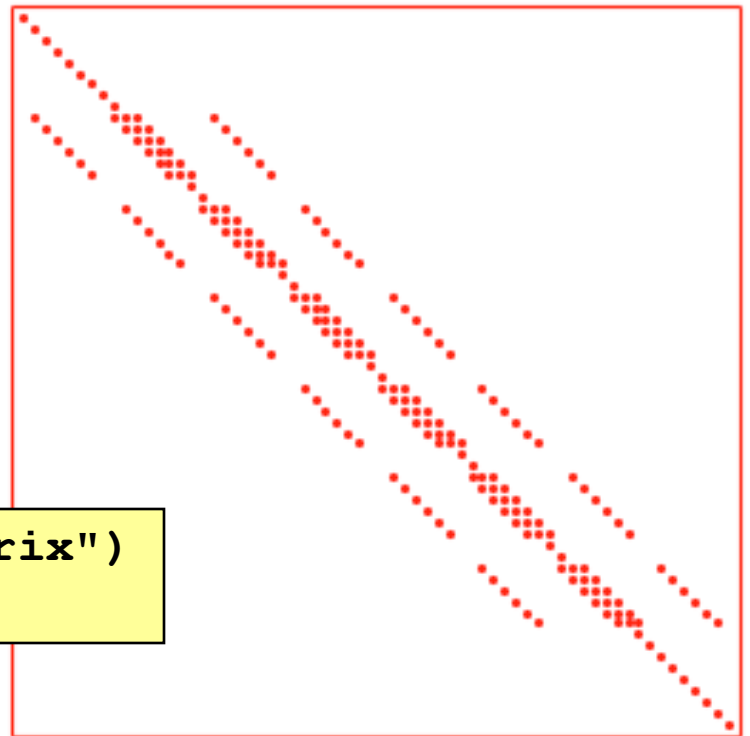
```
x = solve(A,b)
```

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)
```

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

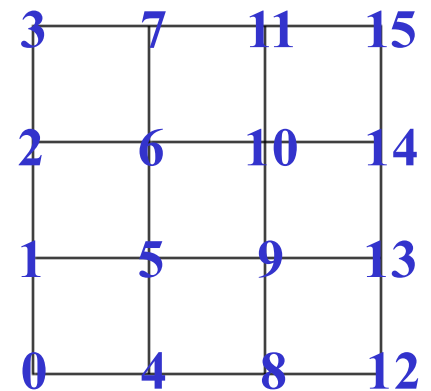
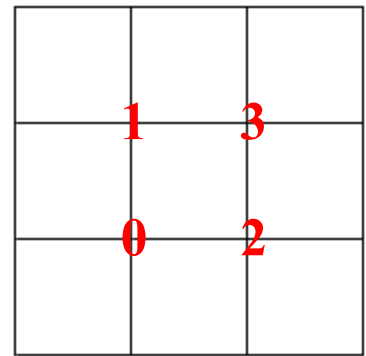
```
bash-3.2$ python poissonSimple.py
=== Considering nx=ny=150
=== Full solver      : elapsed time is 68.746733 seconds.
=== Sparse solver   : elapsed time is 1.635676 seconds.
```



<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)
```

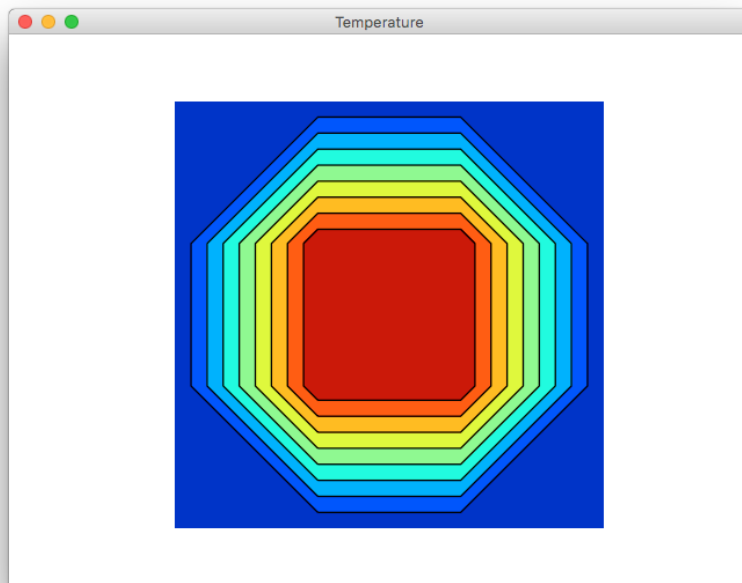


```

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")

```

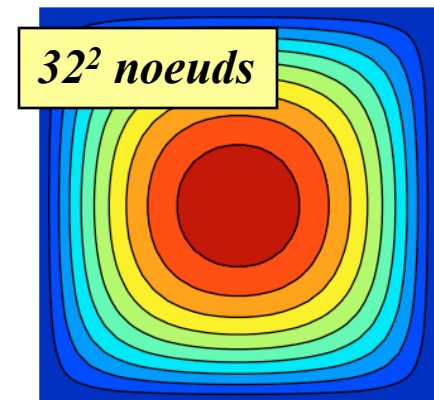
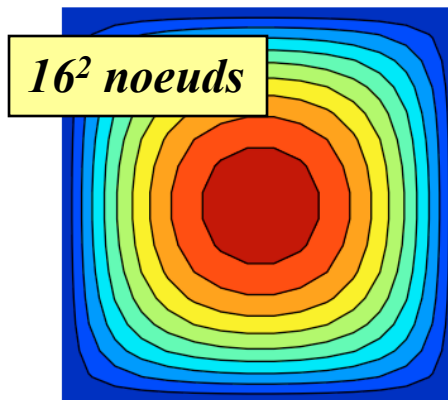
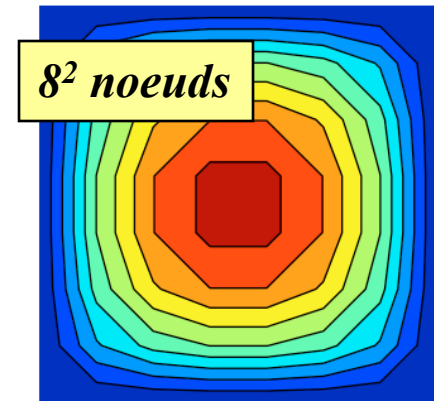
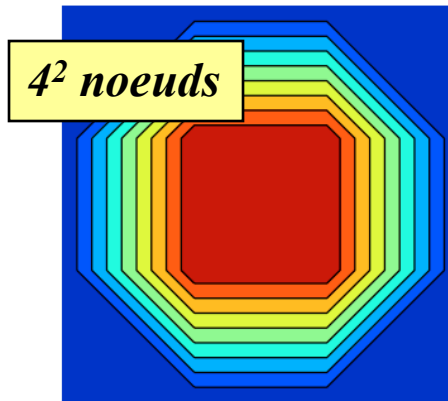


Et faire un joli plot

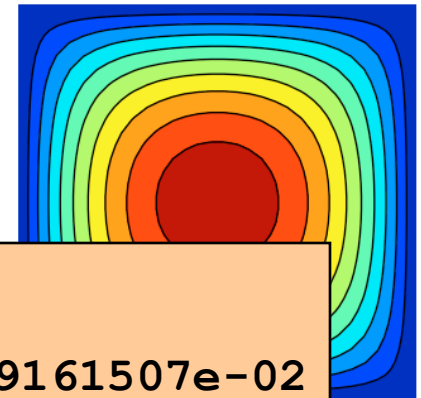
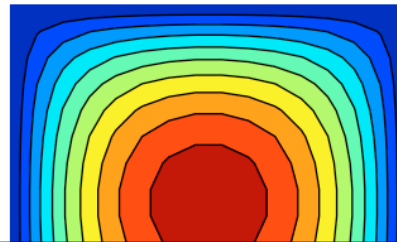
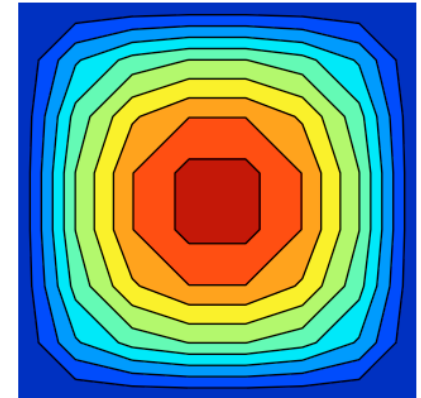
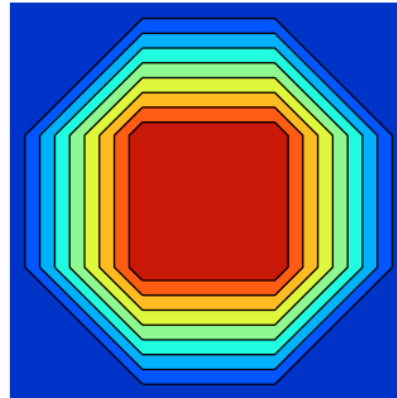
3	7	11	15
2	6	10	14
1	5	9	13
0	4	8	12

1	3	
0	2	

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.]]
```

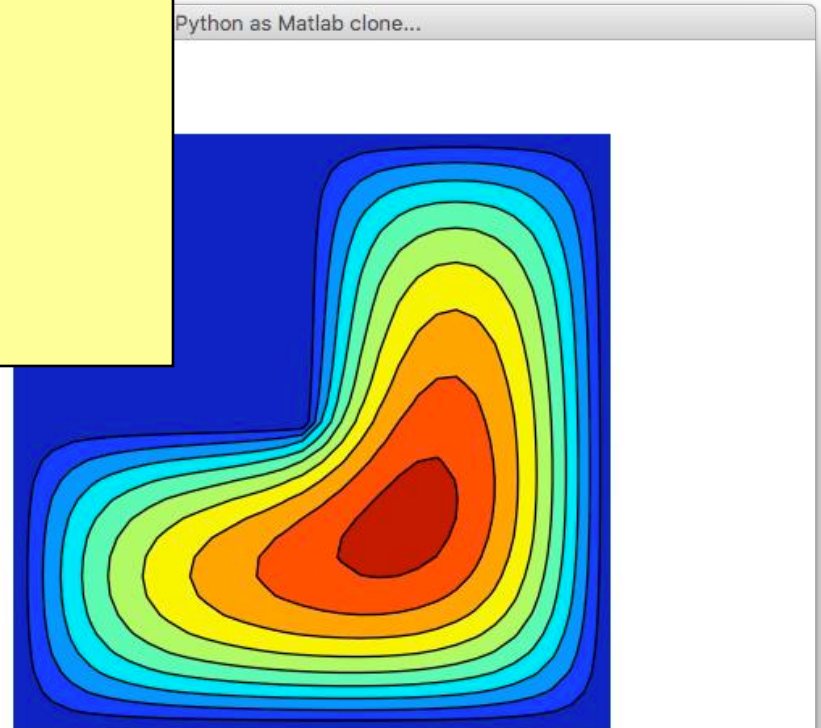
Et finalement...

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

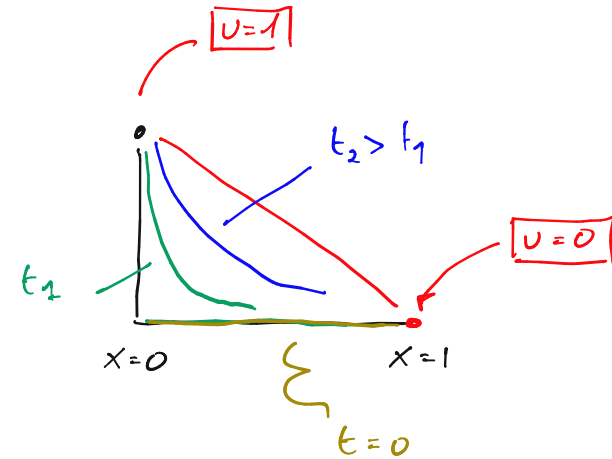
A = delsq(map)
B = ones(size(index))
U = zeros(shape(map))
U[map>0] = spsolve(A,B)[index-1]

plt.contourf(U,10)
```

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



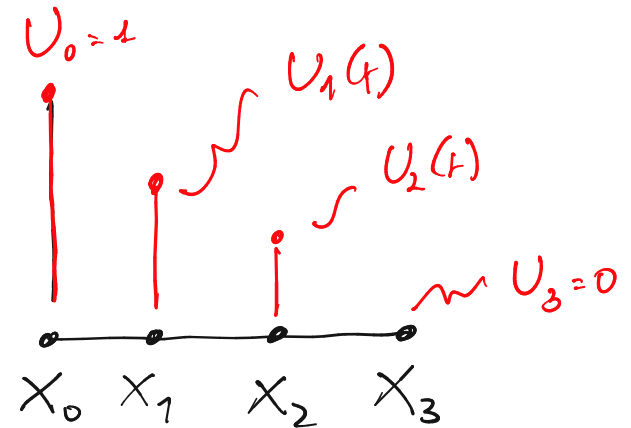
$$\rho c \frac{\partial u}{\partial t} = k \frac{\partial^2 u}{\partial x^2}$$



$$\frac{\partial u}{\partial t} = \underbrace{\frac{k}{\rho c}}_{\alpha} \frac{\partial^2 u}{\partial x^2}$$

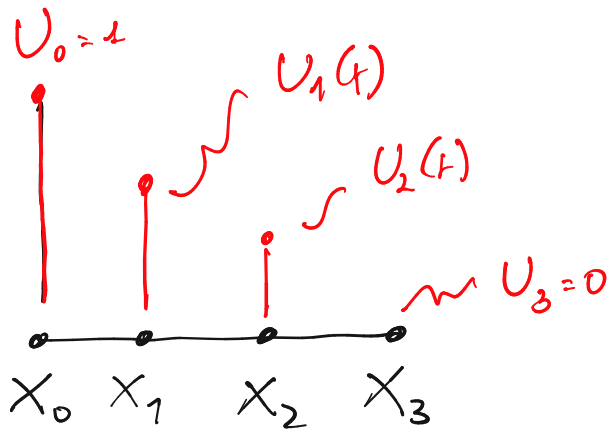
α DIFFUSIVITE THERMIQUE

$\left[\frac{m^2}{s} \right]$



Et
faisons
varier le temps

Example



$$\underline{u}'(t) = \beta \underline{A} \underline{u}(t)$$

$$\frac{dU_0}{dt} = 0$$

$$\frac{dU_1}{dt} = \alpha \left[\frac{U_0 - 2U_1 + U_2}{(\Delta x)^2} \right]$$

$$\frac{dU_2}{dt} = \alpha \left[\frac{U_1 - 2U_2 + U_3}{(\Delta x)^2} \right]$$

$$\frac{dU_3}{dt} = 0$$

$$\underbrace{\frac{d}{dt} \begin{bmatrix} U_0(t) \\ U_1(t) \\ U_2(t) \\ U_3(t) \end{bmatrix}}_{\underline{u}'(t)} = \underbrace{\alpha}_{\beta} \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\underline{A}} \underbrace{\begin{bmatrix} U_0(t) \\ U_1(t) \\ U_2(t) \\ U_3(t) \end{bmatrix}}_{\underline{u}(t)}$$

Différences finies (espace)

Euler explicite (temps)

En définissant $\alpha = k/(\rho c)$

$$\left(\frac{U_i^{n+1} - U_i^n}{\Delta t}\right) = \alpha \left(\frac{U_{i+1}^n - 2U_i^n + U_{i-1}^n}{(\Delta x)^2}\right)$$

En définissant $\beta = \frac{\alpha \Delta t}{(\Delta x)^2}$,

$$U_i^{n+1} = U_i^n + \beta (U_{i+1}^n + U_{i-1}^n - 2U_i^n)$$

C'est une itération pour un vecteur qui doit converger vers la solution de régime
C'est quelque chose qu'on a déjà rencontré...

En fait, on intègre
un système
linéaire...

$$\mathbf{u}' = \frac{\alpha}{(\Delta x)^2} \mathbf{A} \mathbf{u}$$

$$\mathbf{u}_n = \begin{bmatrix} U_1^n \\ U_2^n \\ U_3^n \\ U_4^n \\ U_5^n \\ \\ U_{n_x}^n \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} -2 & 1 & & & & & \\ 1 & -2 & 1 & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 & 1 \\ & & & & & & 1 & -2 \end{bmatrix}$$

Stabilité des systèmes

$$\mathbf{A} = \mathbf{PDP}^{-1}$$

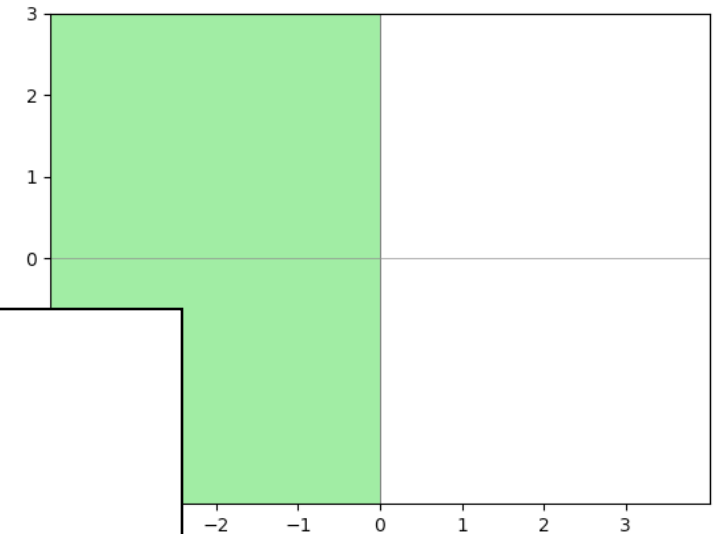
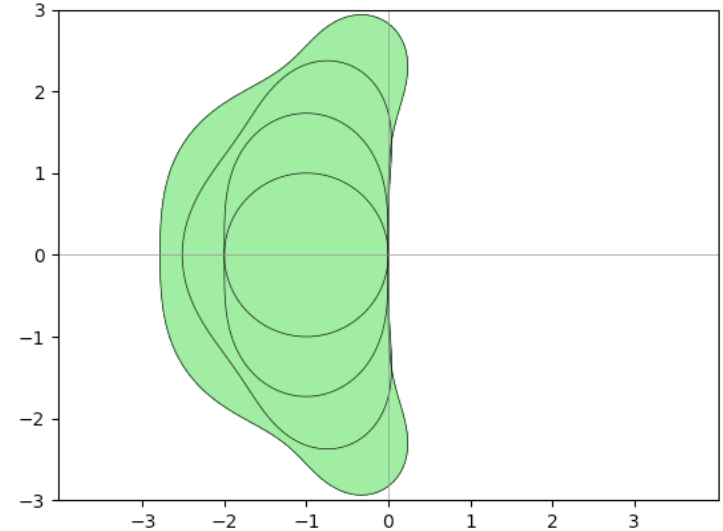
$$\mathbf{v}(x) = \mathbf{P}^{-1}\mathbf{u}(x)$$

$$\mathbf{u}' = \mathbf{A}\mathbf{u}$$

$$\mathbf{v}' = \mathbf{D}\mathbf{v}$$

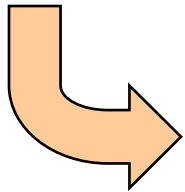
Le problème différentiel initial est équivalent à n équations scalaires

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



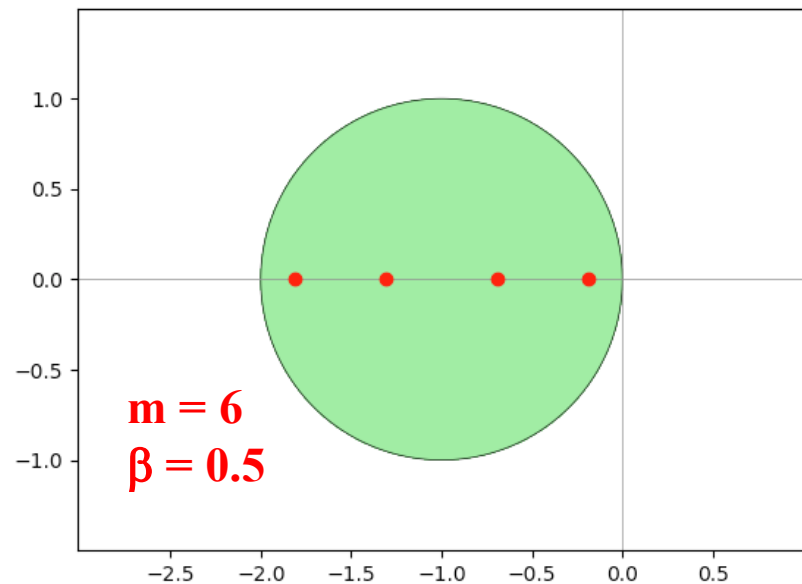
$$\mathbf{u}_{n+1} = \mathbf{u}_n + \beta \mathbf{A} \mathbf{u}_n$$

Euler
explicite



$$|1 + \beta \lambda_i| < 1$$

$$\Delta x = 0.2, \Delta t = 0.02$$

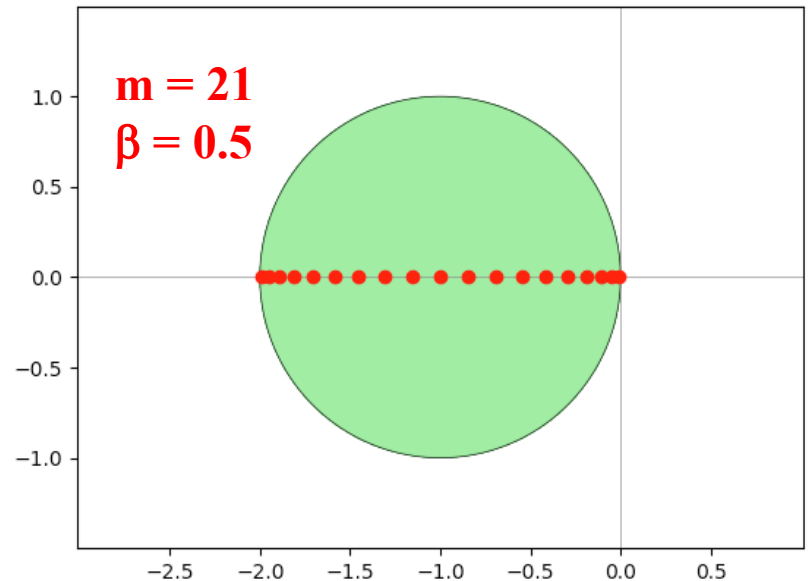
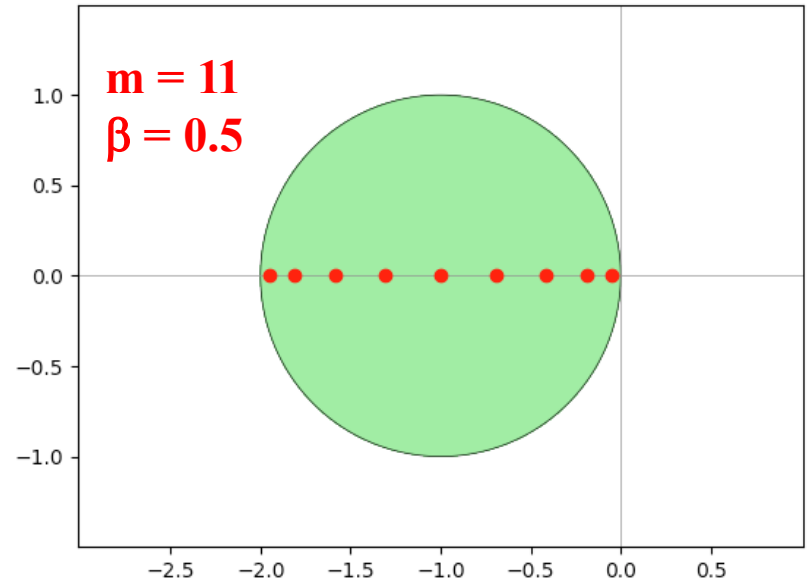


$$\Delta x = 0.1, \Delta t = 0.005$$

$$\beta = \frac{\alpha \Delta t}{(\Delta x)^2} < \frac{1}{2}$$

En raffinant
le maillage...

$$\Delta x = 0.05, \Delta t = 0.00125$$



Adaptons...

$$\mathbf{u}_{n+1} = \underbrace{(\mathbf{I} + \beta \mathbf{A})}_{\mathbf{M}} \mathbf{u}_n$$

Condition pour qu'une méthode itérative converge ?

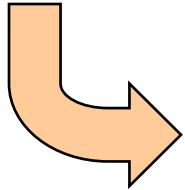
$$\begin{aligned} \underbrace{(\mathbf{x}_{i+1} - \mathbf{x})}_{\mathbf{e}_{i+1}} &= \mathbf{M}\mathbf{x}_i + \mathbf{c} - \mathbf{M}\mathbf{x} - \mathbf{c} & \mathbf{x}_{i+1} &= \mathbf{M}\mathbf{x}_i + \mathbf{c} \\ &\downarrow \\ &= \mathbf{M}(\mathbf{x}_i - \mathbf{x}) \\ &\downarrow \text{ En procédant de la même manière pour chaque étape,} \\ &= \mathbf{M}^{i+1} \underbrace{(\mathbf{x}_0 - \mathbf{x})}_{\mathbf{e}_0} \end{aligned}$$

Il faut que...

$$\lim_{i \rightarrow \infty} \mathbf{M}^i \mathbf{e}_0 = 0$$

$$\mathbf{u}_n = \begin{bmatrix} U_1^n \\ U_2^n \\ U_3^n \\ U_4^n \\ U_5^n \\ \\ U_{n_x}^n \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & \\ & & 1 & -2 & 1 & & \\ & & & 1 & -2 & 1 & \\ & & & & 1 & -2 & 1 \\ & & & & & & 1 & -2 \end{bmatrix}$$

$$\mathbf{u}_{n+1} = \mathbf{u}_n + \beta \mathbf{A} \mathbf{u}_n$$



$$|1 + \beta \lambda_i| < 1$$

Et toujours,
les valeurs
propres...

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...

$$\lim_{i \rightarrow \infty} \mathbf{M}^i \mathbf{e}_0 = 0$$

*... on doit exiger que le rayon spectral
de la matrice $\mathbf{M}$ soit inférieur à l'unité*

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

Mais, il faut calculer les valeurs propres d'un opérateur laplacien discret quelconque....

$$\beta = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

Courant, Friedrichs et Lewy (1928)

Eux, ils ne disposaient pas de Python....

Analyse de stabilité

$$U_i^m = \boxed{U^m} \exp \left(i k X_i \right)$$

↑
AMPLITUDE

↑
NBR
D'ONDE
ARBITRAIRE

↑
NBR
COMPLEXE

PERTURBATION
QUELCONQUE

ON VA LA
DECOMPOSER....

$$\boxed{\text{smooth wave}} + \boxed{\text{high frequency wave}} + \boxed{\text{high frequency wave}}$$

↑
ON CHOISIT
ARBITRAIREMENT
UNE COMPOSANTE

*Considérons une perturbation quelconque de la forme suivante
et analysons son évolution....*

On souhaite que son amplitude diminue.

Analyse de stabilité

Amplitude quelconque de la perturbation

Nombre imaginaire

$$U_i^n = \boxed{U^n} e^{ikX_i}$$

Indice spatial

k quelconque

Indice spatial

*Considérons une perturbation quelconque de la forme suivante
et analysons son évolution....*

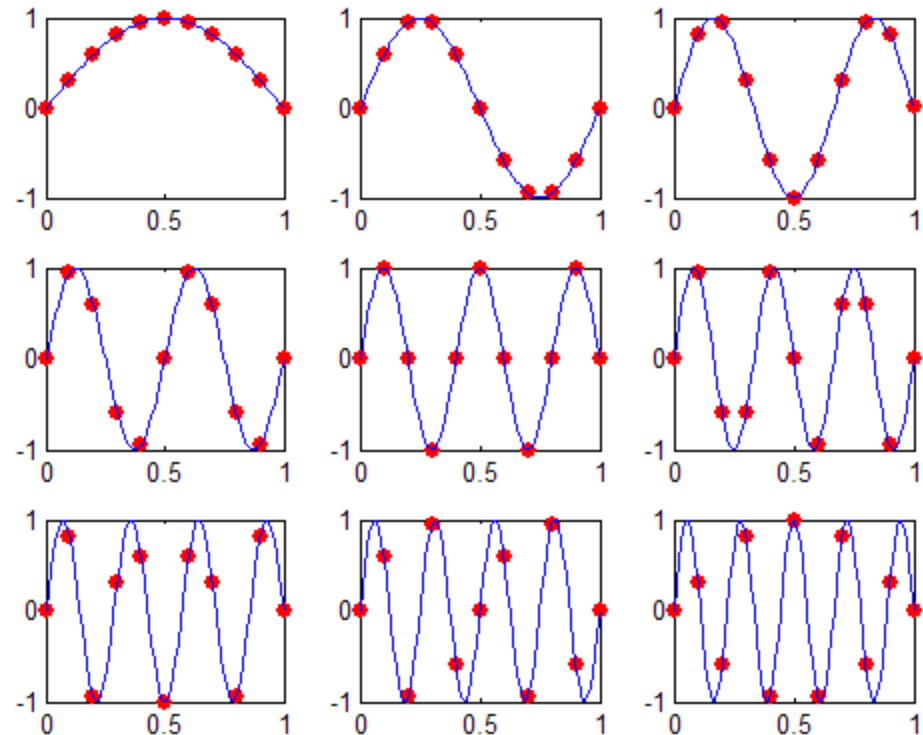
On souhaite que son amplitude diminue.

Quelques k bien choisis...

m = 11
 $\beta = 0.5$

$$U_i^n = U^n \sin\left(\frac{\hat{k}\pi X_i}{L}\right)$$

$$\Delta x = 0.1$$

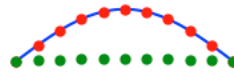


Des perturbations bien particulières et propres...

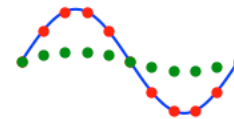
$m = 11$
 $\beta = 0.5$

$$U_i^n = U^n \sin\left(\frac{\hat{k}\pi X_i}{L}\right)$$

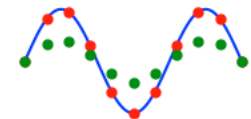
0.0489



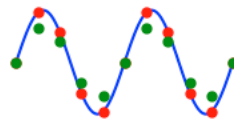
0.1910



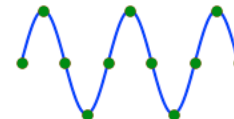
0.4122



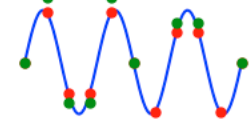
0.6910



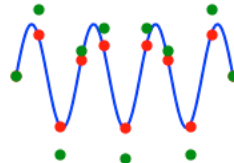
1.0000



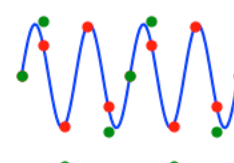
1.3090



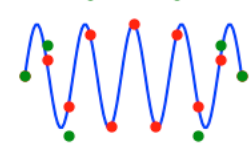
1.5878



1.8090



1.9511



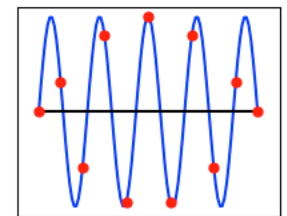
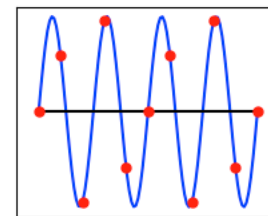
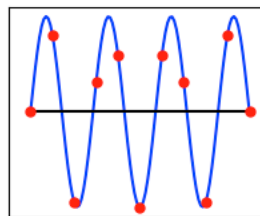
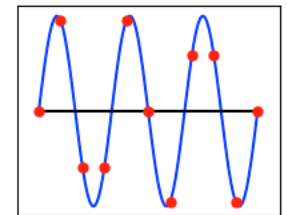
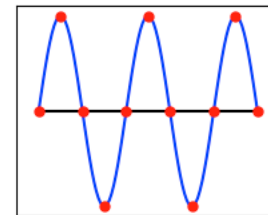
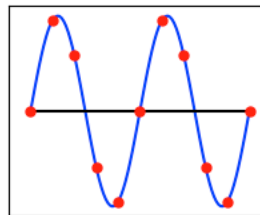
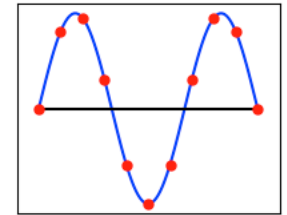
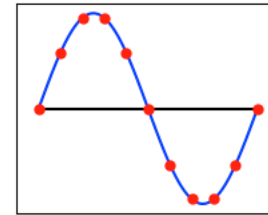
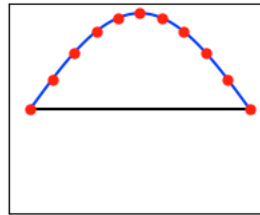
u

$-\beta \mathbf{A} \mathbf{u}$

C'est quoi ces perturbations ?

*N'importe quelle perturbation peut être écrite
comme une combinaison linéaire de ces
vecteurs propres de notre opérateur....*

$$m = 11$$
$$\beta = 0.5$$



Dimension de l'espace discret = 9
Nombre de vecteurs de base = 9

Calculons
les valeurs
et
les vecteurs
propres...

-0.0489



-0.1910



-0.4122



-0.6910



-1.0000



-1.3090



-1.5878



-1.8090

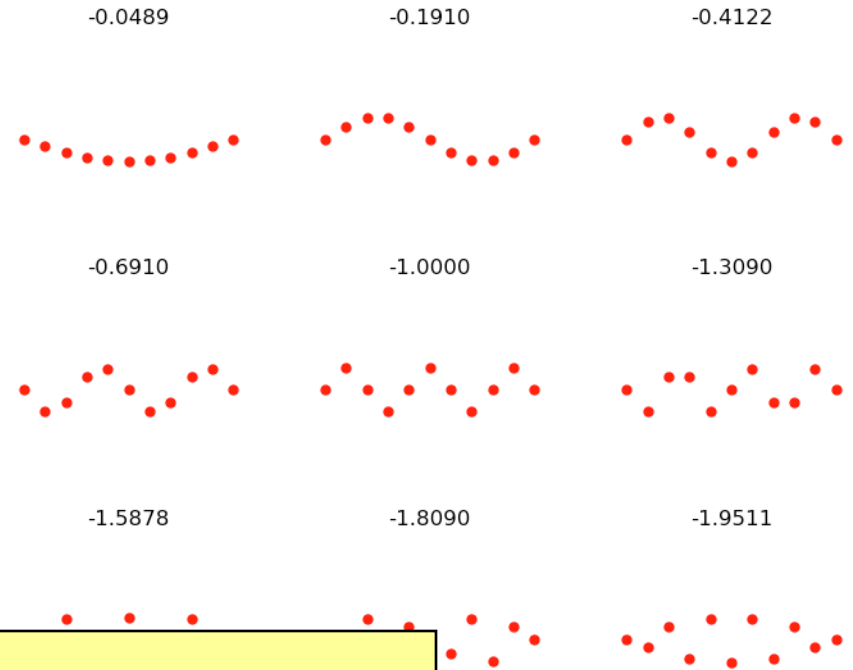


-1.9511

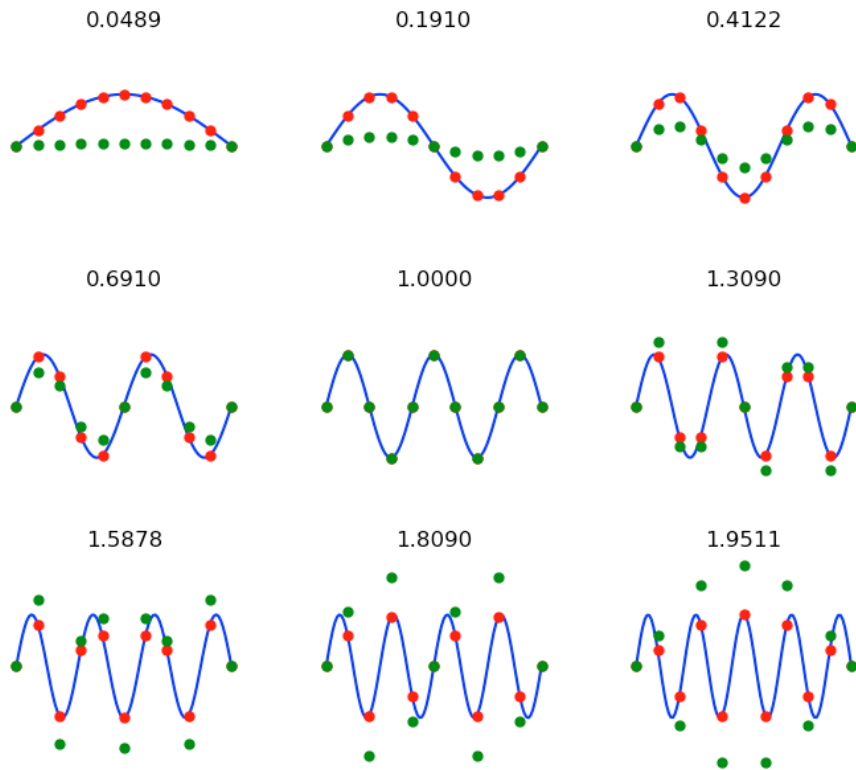


```
m = 11; dx = 1.0/(m-1); beta = (dx*dx)/2  
e = array([0,*ones(m-2),0])  
D = spdiags([e,-2*e,e],[-1,0,1],m,m)  
D = D.T/(dx*dx)  
lam,eigen = eig(beta * D.toarray()[1:-1,1:-1])
```

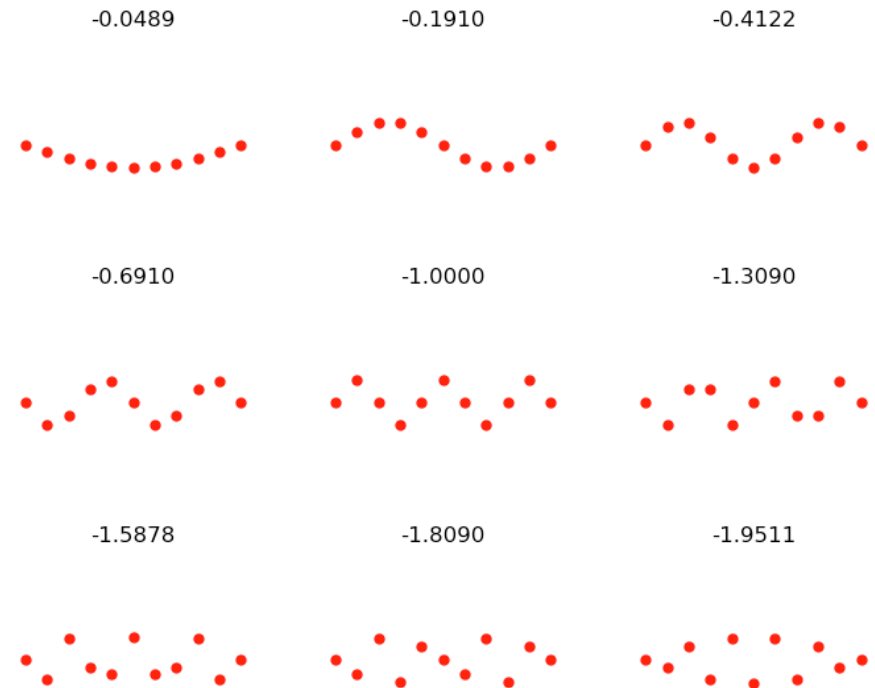
Trier et dessiner les vecteurs propres



```
order = argsort(-lam.real)
X = linspace(0,1,m); V = zeros(m)
for i in range(m-2):
    plt.subplot(msqrt,msqrt,i+1)
    V[1:-1] = eigen[:,order[i]]
    plt.plot(X,V, '.r', markersize=10)
    plt.title("%6.4f" % lam[order[i]].real)
    plt.xlim((-0.1,1.1))
    plt.ylim((-2.1,2.1))
    plt.axis('off')
```



$$U_i^n = U^n \sin\left(\frac{\hat{k}\pi X_i}{L}\right)$$



Et c'est
vraiment cela ?

Propagation des erreurs

$$U_i^{n+1} = U_i^n + \beta (U_{i-1}^n - 2U_i^n + U_{i+1}^n)$$

$$U_i^{n+1} e^{ikX_i} = U_i^n \left[\cancel{e^{ikX_i}} + \beta \left(\cancel{e^{ikX_{i-1}}} e^{-ik\Delta x} - 2 \cancel{e^{ikX_i}} + \cancel{e^{ikX_{i+1}}} e^{ik\Delta x} \right) \right]$$

$$U_i^{n+1} = U_i^n \left[1 + \beta \left(\underbrace{e^{ik\Delta x} + e^{-ik\Delta x}}_{2\cos(k\Delta x)} - 2 \right) \right]$$



$$|U_i^{n+1}| = |U_i^n| |1 - 4\beta|$$

JE CHOISIS LE
CAS LE PLUS
DEFAVORABLE

LORSQUE $\cos = -1$!!

$$-1 < 1 - 4\beta < 1$$

$$\begin{cases} 4\beta < 2 \\ \beta < 1/2 \end{cases}$$

Un peu d'algèbre

$$\beta < \frac{1}{2}$$

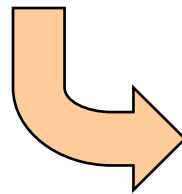
$$|1 + \beta(2 \cos(k\Delta x) - 2)| \leq 1$$

Un peu
d'algèbre ...

En prenant le cas le plus défavorable
où $k\Delta x = \pi$,

$$-1 \leq 1 - 4\beta$$

$$2\beta \leq 1$$



Condition de stabilité
très pénalisante sur le pas de temps....

$$\beta = \frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

Courant, Friedrichs et Lewy (1928)

The so-called Stupid Single Student Behaviour Law

