Calcul matriciel

Informatique pour tou(te)s - semaines 1 & 2

Guillaume CONNAN

Lycée Clemenceau - MP / MP*

janvier 2016

Sommaire

Les problèmes

Pivot de Gauß

NUMPY

Gauß : le retour

Complexité

UNIVERSITÉ DE NANTES

Sommaire

Les problèmes
Pivot de Gauß
NUMPY
Gauß : le retour
Complexité

UNIVERSITÉ DE NANTES

```
sage: a = matrix(4,[10,7,8,7,7,5,6,5,8,6,10,9,7,5,9,10])
sage: a
[10  7  8  7]
[ 7  5  6  5]
[ 8  6 10  9]
[ 7  5  9 10]
sage: b = matrix(4,1,[32,23,33,31])
sage: b
[32]
[23]
[33]
[31]
sage: bb = matrix(4,1,[32.1,22.9,33.1,30.9])
sage: bb
[32.1000000000000]
[22.9000000000000]
[33.1000000000000]
[30.9000000000000]
```

```
sage: X = a.solve_right(b)
sage: X
[1]
[1]
[1]
[1]
```

```
sage: X = a.solve_right(b)
sage: X
[1]
[1]
[1]
[1]
```

```
sage: X = a.solve_right(bb)
sage: X
[  46/5]
[ -63/5]
[   9/2]
[-11/10]
```

```
sage: a = matrix(4,[0,0,0,1e-4,1,0,0,0,0,1,0,0,0,0,1,0])

sage: a.eigenvalues()
[0.100000000000000, -0.100000000000000, -0.100000000000000*I,
          ↪  0.100000000000000*I]

sage: a = matrix(4,[0,0,0,0,1,0,0,0,0,1,0,0,0,0,1,0])
sage: a.eigenvalues()
[0, 0, 0, 0]
```

Conditionnement : $cond A = \|A\| \times \|A^{-1}\|$

Conditionnement : $condA = \|A\| \times \|A^{-1}\|$

$\|AB\| \leq \|A\|\|B\|$

Conditionnement : $condA = \|A\| \times \|A^{-1}\|$
$\|AB\| \leq \|A\|\|B\|$
Plus c'est petit (proche de 1), plus c'est stable.

Conditionnement : $condA = \|A\| \times \|A^{-1}\|$
$\|AB\| \leq \|A\|\|B\|$
Plus c'est petit (proche de 1), plus c'est stable.
Difficile à savoir.

Système de Cramer : $n^2 \times n!$ opérations élémentaires...

Système de Cramer : $n^2 \times n!$ opérations élémentaires...
$\approx 5 \times 10^{141}$ siècles pour un système 100×100 avec mon processeur i5 de 60 GFLOPs...

Système de Cramer : $n^2 \times n!$ opérations élémentaires...
$\approx 5 \times 10^{141}$ siècles pour un système 100×100 avec mon processeur i5 de 60 GFLOPs...

```
sage: factorial(100)*10^4/(60*10^9*3600*24*365.25*100)
4.928882183897 81e141
```

Sommaire

Les problèmes

Pivot de Gauß

NUMPY

Gauß : le retour

Complexité

Vous connaissez...

Vous connaissez...

$$A = \begin{pmatrix} \varepsilon & -1 & 1 \\ -1 & 2 & -1 \\ 2 & -1 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

Vous connaissez...

$$A = \begin{pmatrix} \varepsilon & -1 & 1 \\ -1 & 2 & -1 \\ 2 & -1 & 0 \end{pmatrix} \qquad B = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$$

On cherche X telle que $AX = B$

Sommaire

Les problèmes
Pivot de Gauß
NUMPY
Gauß : le retour
Complexité

numpy

```
import numpy as np
```

```
a = np.array([[1,2,3],[4,5,6]])

b = (np.array(np.arange(1,7))).reshape(2,3)

c = np.matrix([[1,2,3],[4,5,6]])

d = np.matrix('1 2 3; 4 5 6')
```

```
In [29]: a*a
Out[29]:
array([[ 1,  4,  9],
       [16, 25, 36]])
```

```
In [29]: a*a
Out[29]:
array([[ 1,  4,  9],
       [16, 25, 36]])
```

```
In [30]: a.dot(a)
---------------------------------------------------------------------------

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

```
In [29]: a*a
Out[29]:
array([[ 1,  4,  9],
       [16, 25, 36]])
```

```
In [30]: a.dot(a)
---------------------------------------------------------------------------

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

```
In [31]: c*c
---------------------------------------------------------------------------

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

```
In [32]: c * c.transpose()
Out[32]:
matrix([[14, 32],
        [32, 77]])
```

```
In [33]: np.eye(4,4)
Out[33]:
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

```
In [33]: np.eye(4,4)
Out[33]:
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
```

```
In [34]: np.eye(4,4, dtype=bool)
Out[34]:
array([[ True, False, False, False],
       [False,  True, False, False],
       [False, False,  True, False],
       [False, False, False,  True]], dtype=bool)
```

```
In [42]: a = np.zeros((2,3), dtype=int)

In [43]: a
Out[43]:
array([[0, 0, 0],
       [0, 0, 0]])
```

```
In [47]: np.diag(np.arange(5))
Out[47]:
array([[0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0],
       [0, 0, 0, 0, 4]])
```

```
In [47]: np.diag(np.arange(5))
Out[47]:
array([[0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 2, 0, 0],
       [0, 0, 0, 3, 0],
       [0, 0, 0, 0, 4]])
```

```
In [49]: a > 0
Out[49]:
array([[False, False, False, False, False],
       [False,  True, False, False, False],
       [False, False,  True, False, False],
       [False, False, False,  True, False],
       [False, False, False, False,  True]], dtype=bool)
```

```
In [51]: a = np.random.rand(3,3)

In [52]: a
Out[52]:
array([[ 0.08468479,  0.01769339,  0.89631521],
       [ 0.90388521,  0.71191769,  0.60721673],
       [ 0.16991842,  0.26063882,  0.08871752]])
```

```
In [53]: np.linalg.det(a)
Out[53]: 0.09508812509461911 4
```

```
In [53]: np.linalg.det(a)
Out[53]: 0.09508812509461911 4
```

```
In [54]: np.linalg.inv(a)
Out[54]:
array([[-1.00017403,  2.44031334, -6.597658  ],
       [ 0.24174255, -1.52266579,  7.97937737],
       [ 1.20540439, -0.2005057 ,  0.46583953]])
```

```
In [53]: np.linalg.det(a)
Out[53]: 0.09508812509461911 4
```

```
In [54]: np.linalg.inv(a)
Out[54]:
array([[-1.00017403,  2.44031334, -6.597658  ],
       [ 0.24174255, -1.52266579,  7.97937737],
       [ 1.20540439, -0.2005057 ,  0.46583953]])
```

```
In [55]: a.dot(np.linalg.inv(a))
Out[55]:
array([[  1.00000000e+00,   0.00000000e+00,   1.11022302e-16],
       [  0.00000000e+00,   1.00000000e+00,   2.77555756e-16],
       [  4.16333634e-17,  -1.45716772e-16,   1.00000000e+00]])
```

```
In [64]: b = np.random.rand(3,1)
In [65]: x = np.linalg.solve(a,b)
```

```
In [64]: b = np.random.rand(3,1)
In [65]: x = np.linalg.solve(a,b)
```

```
In [68]: a.dot(x) == b
Out[68]:
array([[False],
       [False],
       [False]], dtype=bool)
```

```
In [64]: b = np.random.rand(3,1)
In [65]: x = np.linalg.solve(a,b)
```

```
In [68]: a.dot(x) == b
Out[68]:
array([[False],
       [False],
       [False]], dtype=bool)
```

```
In [69]: np.array_equal(a.dot(x), b)
Out[69]: False
```

```
In [64]: b = np.random.rand(3,1)
In [65]: x = np.linalg.solve(a,b)
```

```
In [68]: a.dot(x) == b
Out[68]:
array([[False],
       [False],
       [False]], dtype=bool)
```

```
In [69]: np.array_equal(a.dot(x), b)
Out[69]: False
```

```
In [70]: np.allclose(a.dot(x), b)
Out[70]: True
```

Sommaire

Les problèmes
Pivot de Gauß
NUMPY
Gauß : le retour
Complexité

Choix du pivot maximum

Choix du pivot maximum

```
def trouve_pivot(A,dep) :
    ind_max = dep
    for k in range(dep + 1, len(A)) :
        if abs(A[k,dep]) > abs(A[ind_max,dep]) :
            ind_max = k
    return ind_max
```

Transvection : $L_i \leftarrow L_i + k \times L_j$

Transvection : $L_i \leftarrow L_i + k \times L_j$

```
def transvection(A,i,j,k) :
    A[i] += k * A[j]
```

Échange $L_i \leftrightarrow L_j$

Échange $L_i \leftrightarrow L_j$

```
def echange(A,i,j) :
    tmp = copy(A[i])
    A[i], A[j] = A[j], tmp
```

```
def gauss(A,B) :
    T = np.concatenate((A,B), axis = 1)
```

```
def gauss(A,B) :
    T = np.concatenate((A,B), axis = 1)
```

```
    ind_max = len(A)
    assert ind_max == len(A[0]), "Matrice non carrée"
```

```
def gauss(A,B) :
    T = np.concatenate((A,B), axis = 1)
```

```
    ind_max = len(A)
    assert ind_max == len(A[0]), "Matrice non carrée"
```

```
   for row_piv in range(ind_max) :
        ligne_piv = trouve_pivot(T, row_piv)
        if ligne_piv > row_piv :
            echange(T, row_piv, ligne_piv)
```

```
def gauss(A,B) :
    T = np.concatenate((A,B), axis = 1)
```

```
    ind_max = len(A)
    assert ind_max == len(A[0]), "Matrice non carrée"
```

```
   for row_piv in range(ind_max) :
        ligne_piv = trouve_pivot(T, row_piv)
        if ligne_piv > row_piv :
            echange(T, row_piv, ligne_piv)
```

```
        pivot = T[row_piv, row_piv]
        for sub_row in range(row_piv + 1, ind_max):
            coeff = T[sub_row, row_piv] / pivot
            transvection(T, sub_row, row_piv, -coeff)
```

Le tableau est maintenant triangulaire. Il ne reste plus qu'à effectuer la phase de remontée.

Le tableau est maintenant triangulaire. Il ne reste plus qu'à effectuer la phase de remontée.

```
    Sol = np.arange(ind_max, dtype=np.float)
    for i in range(ind_max - 1, -1, -1):
        Sol[i] = (T[i, ind_max] - sum(T[i,j]*Sol[j] for j in range(i+1,
                 ↪  ind_max))) / T[i,i]
    return Sol
```

```
In [24]: a = np.random.rand(1000,1000)

In [25]: b = np.random.rand(1000,1)
```

```
In [24]: a = np.random.rand(1000,1000)

In [25]: b = np.random.rand(1000,1)
```

```
In [26]: %timeit gauss(a,b)
1 loops, best of 3: 3.27 s per loop
```

```
In [24]: a = np.random.rand(1000,1000)

In [25]: b = np.random.rand(1000,1)
```

```
In [26]: %timeit gauss(a,b)
1 loops, best of 3: 3.27 s per loop
```

```
In [27]: %timeit np.linalg.solve(a,b)
1 loops, best of 3: 421 ms per loop
```

Exercice 1

1. *Comment modifier la fonction* `gauss` *pour calculer le déterminant d'une matrice ?*
2. *pour calculer l'inverse d'une matrice ?*

Sommaire

Les problèmes
Pivot de Gauß
NUMPY
Gauß : le retour
Complexité

```python
def trouve_pivot(A,dep) :
    ind_max = dep
    for k in range(dep + 1, len(A)) :
        if abs(A[k,dep]) > abs(A[ind_max,dep]) :
            ind_max = k
    return ind_max
```

```
def transvection(A,i,j,k) :
    A[i] += k * A[j]
```

```
def echange(A,i,j) :
    tmp = copy(A[i])
    A[i], A[j] = A[j], tmp
```

```python
def gauss(A,B) :
    T = np.concatenate((A,B), axis = 1)
    ind_max = len(A)
    assert ind_max == len(A[0]), "Matrice non carrée"
    for row_piv in range(ind_max) :
        ligne_piv = trouve_pivot(T,row_piv)
        if ligne_piv > row_piv :
            echange(T, row_piv, ligne_piv)
        pivot = T[row_piv, row_piv]
        for sub_row in range(row_piv + 1, ind_max):
            coeff = T[sub_row, row_piv] / pivot
            transvection(T, sub_row, row_piv, -coeff)
```

```
Sol = np.arange(ind_max, dtype=np.float)
    for i in range(ind_max-1, -1, -1):
        Sol[i] = (T[i,ind_max] - sum(T[i,j]*Sol[j] for j in range(i+1,
                 ↪  ind_max))) / T[i,i]
    return Sol
```

```
In [87]: %timeit gauss(a,b)
10 loops, best of 3: 21.9 ms per loop

In [88]: a = np.random.rand(200,200)

In [89]: b = np.random.rand(200,1)

In [90]: %timeit gauss(a,b)
10 loops, best of 3: 93.8 ms per loop
```

```
In [87]: %timeit gauss(a,b)
10 loops, best of 3: 21.9 ms per loop

In [88]: a = np.random.rand(200,200)

In [89]: b = np.random.rand(200,1)

In [90]: %timeit gauss(a,b)
10 loops, best of 3: 93.8 ms per loop
```

??????????????????????

```
In [79]: a = np.random.rand(100,100)

In [80]: %timeit transvection(a,1,10,3)
100000 loops, best of 3: 3.27 µs per loop

In [81]: a = np.random.rand(200,200)

In [82]: %timeit transvection(a,1,10,3)
100000 loops, best of 3: 3.67 µs per loop

In [83]: a = np.random.rand(400,400)

In [84]: %timeit transvection(a,1,10,3)
100000 loops, best of 3: 3.81 µs per loop
```