

L'itération: La boucle `for`

La boucle `for` permet d'appliquer un même bloc d'instructions, successivement, à chacun des éléments d'une liste. La syntaxe d'une boucle `for` est la suivante :

```
for <element> in <list>:  
    <instructions>
```

Exemple, pour sommer les entiers pairs compris entre 0 et 6.

```
somme = 0  
for x in range(0,7,2) :  
    somme = somme + x
```

Comme nous l'avons remarqué précédemment, dans une fonction, la commande `return` termine l'exécution de la fonction. En particulier, le cas échéant, la boucle n'est pas exécutée jusqu'à son terme.

Exemple for

Réaliser le tableau suivant représentant les premiers termes d'une suite de Fibonacci.

i	aux	a	b
		1	1
2	2	1	2
3	3	2	3
4	5	3	5
5	8	5	8

L'itération: La boucle `while`

Si on souhaite répéter une séquence d'instructions un nombre de fois non déterminé à l'avance, on utilise une boucle `while`. La syntaxe est la suivante :

```
while <condition>:  
    <instructions>
```

Le déroulement de la boucle est le suivant :

- on évalue la condition (qui doit être une expression booléenne),
- si la condition renvoie la valeur `True`, on effectue les instructions et on revient au début de la boucle,
- si elle renvoie la valeur `False`, on sort de la boucle.

Exemple `while`

Réaliser le tableau suivant représentant les premiers termes d'une suite de Fibonacci.

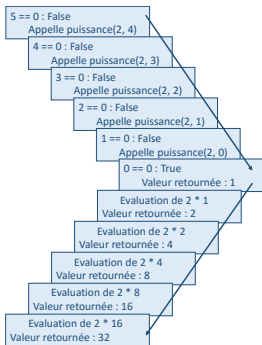
<code>b<10</code>	<code>L</code>	<code>aux</code>	<code>a</code>	<code>b</code>
	<code>[1]</code>		1	1
True	<code>[1,1]</code>	2	1	2
True	<code>[1,1,2]</code>	3	2	3
True	<code>[1,1,2,3]</code>	5	3	5
True	<code>[1,1,2,3,5]</code>	8	5	8
False	<code>[1,1,2,3,5,8]</code>	13	8	13

Algorithmes récursifs

Un programme récursif est un programme qui fait appel à lui-même. Très souvent un algorithme récursif est lié à une relation de récurrence. permettant de calculer la valeur d'une fonction pour un argument n à l'aide des valeurs de cette fonction pour des arguments inférieurs à n .

Soient $x \in \mathbb{R}$ et $n \in \mathbb{N}$. On peut définir x^n par récurrence à partir des relations :

$x^0 = 1$ et $x^n = x \times x^{n-1}$ si $n > 1$:



Exercices: Boucles for

Question 1: Donner le contenu de la liste `L` à l'issue des lignes de commandes suivantes.

```
L = []
n = 3
for i in range(1, 13, 2*n):
    for j in range(n):
        L.append([i, j])
```

```
L = []
n = 3
for i in range(1, n+1):
    for j in range(i):
        if j != n//2:
            L.append([i, j])
```

Exercices: Boucles for

Question 2: Quel est le résultat renvoyé par la fonction suivante ?

```
def f(n) :  
    u = 1  
    for x in range(1,n) :  
        u = u * x  
    return u
```

Question 3: Écrire une fonction `seuil_som_car(M)` qui, pour tout entier M , renvoie le plus petit entier n tel que $1^2 + 2^2 + \dots + n^2 \geq M$.

Question 4: Écrire une fonction récursive `fibo_rec(n)` qui prend en entrée un entier n et retourne le terme de rang n de la suite de Fibonacci.

CORRECTION: Exemple for

Réaliser le tableau suivant représentant les premiers termes d'une suite de Fibonacci.

i	aux	a	b
		1	1
2	2	1	2
3	3	2	3
4	5	3	5
5	8	5	8

```
a = 1
b = 1
for i in range (2,6):
    aux = a + b
    a = b
    b = aux
```


CORRECTION: Exemple `while`

Réaliser le tableau suivant représentant les premiers termes d'une suite de Fibonacci.

b<10	L	aux	a	b
	[1]		1	1
True	[1,1]	2	1	2
True	[1,1,2]	3	2	3
True	[1,1,2,3]	5	3	5
True	[1,1,2,3,5]	8	5	8
False	[1,1,2,3,5,8]	13	8	13

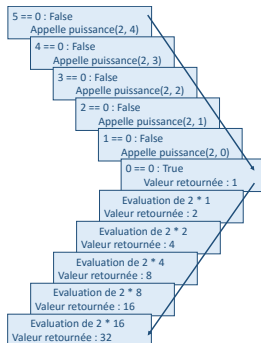
```
# Initialisation suite
a = 1
b = 1
# Initialisation liste des termes
L=[a]
# Construction
while b <= 10;
    L = L + [b]
    aux = a + b
    a = b
    b = aux
```

CORRECTION: Algorithmes récursifs

Un programme récursif est un programme qui fait appel à lui-même. Très souvent un algorithme récursif est lié à une relation de récurrence. permettant de calculer la valeur d'une fonction pour un argument n à l'aide des valeurs de cette fonction pour des arguments inférieurs à n .

Soient $x \in \mathbb{R}$ et $n \in \mathbb{N}$. On peut définir x^n par récurrence à partir des relations :

$x^0 = 1$ et $x^n = x \times x^{n-1}$ si $n > 1$:



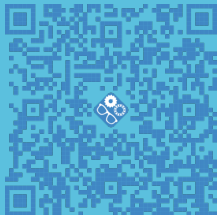
```

def puissance(x,n):
    "calcul récursif de x ** n"
    if n == 0:
        return 1
    else:
        return x * puissance(x,n-1)

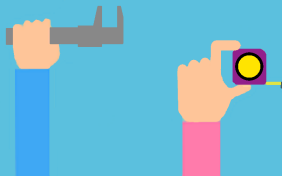
>>> puissance(2,5)
32
  
```



Les Fonctions



Renaud Costadoat
Lycée Dorian



Exemple de fonction

Une *fonction* est un algorithme qui prend des arguments en entrée, effectue une séquence d'instructions et renvoie un résultat. Elle est considérée comme un objet de type *function*.

Un premier exemple consiste à programmer la fonction qui retourne la norme d'un vecteur à 3 coordonnées $\vec{V} = a.\vec{x} + b.\vec{y} + c.\vec{z}$ $\sqrt{a^2 + b^2 + c^2}$.

```
def norme_3(a,b,c):  
    n = (a**2 + b**2 + c**2)**(1/2)  
    return n
```

Attention: Il faut toujours **appeler la fonction** ensuite afin de voir le résultat obtenu.

```
>>> norme_3(3,4,5)  
7.0710678118645755  
>>> norme_3(4,5,6)  
8.774964387392123
```

Forme générale

```
def <nom_de_la_fonction>(<argument_1>,<argument_2>,...,<argument_n>):  
    "Ce commentaire explique à quoi sert cette fonction"  
    <instructions>
```

On n'oubliera pas les : et l'*indentation* qui sont obligatoires en Python !

Remarques:

- les arguments entre parenthèses sont des paramètres formels,
- Les instructions qui forment le corps de la fonction commencent sur la ligne suivante, indentée par quatre espaces (ou une tabulation),
- La première instruction du corps de la fonction peut être un texte dans une chaîne de caractères, cette chaîne est la chaîne de documentation de la fonction et on peut la visualiser dans un terminal en tapant l'instruction `help(nom_de_la_fonction)` ,
- Le retour à la ligne signale la fin de la fonction.

Dès que l'instruction `return` est exécutée (si elle est présente), l'exécution de **la fonction se termine.**

Erreur de programmation

La fonction suivante est sensée retourner la somme des nombres pairs inférieurs à un nombre donné `a` .

```
def somme_pairs(a) :  
    i=1  
    n=0  
    while(i<=a):  
        if i%2==0:  
            n=n+i  
        i=i+1  
    return n
```

```
>>> somme_pairs(5)  
0  
>>> somme_pairs(3)  
0
```

Question 1: Expliquer pourquoi ce programme ne donne pas le résultat attendu.

Question 2: Proposer une modification afin que ce soit le cas.

Dichotomie

Voici le processus complet :

- Au rang 0,
 - soient $a_0 = a$, $b_0 = b$. Il existe une solution x_0 de l'équation $(f(x) = 0)$ dans l'intervalle $[a_0, b_0]$.
- Au rang 1,
 - si $f(a_0).f(\frac{a_0+b_0}{2}) \leq 0$, alors on pose $a_1 = a_0$, $b_1 = \frac{a_0+b_0}{2}$,
 - sinon on pose $a_1 = \frac{a_0+b_0}{2}$ et $b_1 = b$.
 - dans les deux cas, il existe une solution x_1 de l'équation $(f(x) = 0)$ dans l'intervalle $[a_1, b_1]$.
- Au rang n , supposons construit un intervalle $[a_n, b_n]$, de longueur $\frac{b-a}{2^n}$, et contenant une solution x_n de l'équation $(f(x) = 0)$. Alors:
 - si $f(a_n).f(\frac{a_n+b_n}{2}) \leq 0$, alors on pose $a_{n+1} = a_n$, $b_{n+1} = \frac{a_n+b_n}{2}$,
 - sinon on pose $a_{n+1} = \frac{a_n+b_n}{2}$ et $b_{n+1} = b_n$,
 - dans les deux cas, il existe une solution x_{n+1} de l'équation $(f(x) = 0)$ dans l'intervalle $[a_{n+1}, b_{n+1}]$.

A chaque étape, on a $a_n \leq x_n \leq b_n$

On arrête le processus dès que $b_n - a_n = \frac{b-a}{2^n}$ est inférieur à la précision souhaitée.

Exemple dichotomie

Question 1: Coder la fonction $f(x)$ qui à la valeur x renvoie la valeur $f(x) = x^3 + 2.x^2 + 3.x - 4$.

Question 2: Coder la fonction $g(x)$ qui à la valeur x renvoie la valeur $g(x) = x^3 + 2.x^2 + 3.x - 6$.

Question 3: Coder la fonction `dichotomie(f, p)` qui à la fonction f renvoie la valeur de la racine du polynôme avec une précision minimale p . La fonction devra retourner la valeur de a_n ainsi que l'erreur finale $b_n - a_n$.

Précision:

Il n'existe qu'une seule racine pour les fonctions $f(x)$ et $g(x)$ et elles sont comprises entre -10 et 10 .



Calcul d'intégrales



Renaud Costadoat
Lycée Dorian



DORIAN



Méthode des rectangles

Dans cette méthode, on calcule l'intégrale numérique en réalisant une somme de surfaces de rectangles. Le domaine d'intégration est découpé en intervalles et on fait comme si la fonction restait constante sur chaque intervalle.

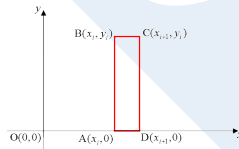
Sur chaque intervalle, on réalise ainsi l'approximation suivante :

$$\int_a^b f(x).dx \approx (b-a).f(\alpha), \text{ où } \alpha \text{ est une abscisse appartenant à l'intervalle limité par } a \text{ et } b.$$

Nous nous limiterons ici aux cas où $\alpha = a$, $\alpha = b$ ou $\alpha = \frac{a+b}{2}$.

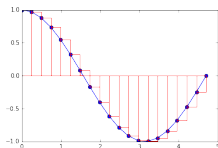
Comme exemple, nous allons réaliser un programme d'intégration pour $\alpha = a$ et nous visualiserons les rectangles.

Pour tracer un rectangle ABCD (voir figure ci-dessous), il suffit de faire un plot avec les coordonnées de A, B, C, D et A. On termine par A pour fermer le tracé.

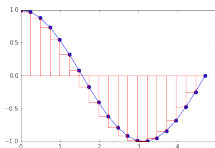


Méthode des rectangles

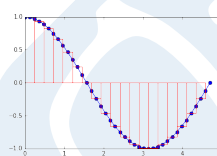
Cas $\alpha = a$



Cas $\alpha = b$



Cas $\alpha = \frac{a+b}{2}$



La méthode $\alpha = \frac{a+b}{2}$ est plus difficile à mettre en œuvre car, il faut alors découper en deux les intervalles d'intégration. Souvent a et b sont des entiers consécutifs et cela empêche cette division.

L'objectif de la méthode des rectangles est de faire la somme de la surface de ces carrés au fur et à mesure de l'avancement du programme.

Méthode des rectangles ($\alpha = a$)

Création de la fonction y et de la variable x .

```
xmin, xmax, nbx = 0, 3*pi/2, 2006  
x = linspace(xmin, xmax, nbx)  
y = cos(x)  
plot(x, y, "bo-")
```

Calcul de l'intégrale int .

```
nbi = nbx - 1 # nombre d'intervalles  
int = linspace(1, 1, nbx)  
int[0]=0  
  
for i in range(nbi):  
    int[i+1] = int[i] + y[i]*(x[i+1]-x[i])
```

Méthode des rectangles ($\alpha = b$)

Modifier le programme pour prendre $\alpha = b$.

```
xmin, xmax, nbx = 0, 3*pi/2, 2006
x = linspace(xmin, xmax, nbx)
y = cos(x)
plot(x,y,"bo-")

nbi = nbx - 1 # nombre d'intervalles
int = linspace(1, 1, nbx)
int[0]=0

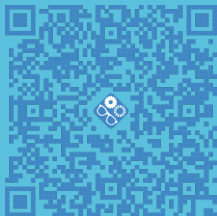
for i in range(nbi):
    int[i+1] = int[i] + y[i+1]*(x[i+1]-x[i])
```

Méthode des trapèzes

Coder la méthode des trapèzes.



Tracer des résultats



Renaud Costadoat
Lycée Dorian



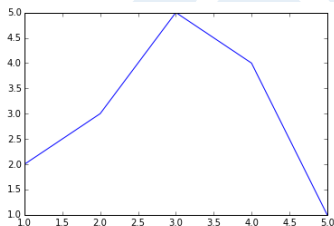
Création de courbes

L'instruction `plot()` permet de tracer des courbes qui relient des points dont les abscisses et ordonnées sont fournies dans des tableaux.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.array(range(1,6))
y = np.array([2, 3, 5, 4, 1])
plt.plot(x, y)

plt.show() # affiche la figure
```



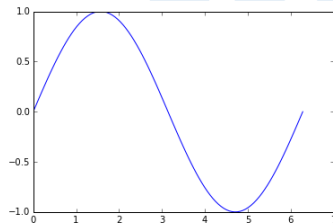
Création de courbes

De la même manière, il est possible de tracer une fonction $\sin(x)$.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)
plt.plot(x, y)

plt.show()
```



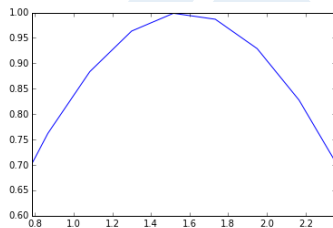
Définition du domaine des axes

Les fonction `xlim(xmin, xmax)` et `ylim(ymin, ymax)` permettent de fixer indépendamment les domaines des abscisses et des ordonnées.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 30)
y = np.sin(x)
plt.plot(x, y)
plt.xlim(np.pi/4, 3*np.pi/4)
plt.ylim(0.6, 1)

plt.show()
```



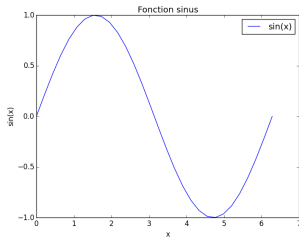
Ajouter un titre, une légende ou des labels

Les courbes tracées doivent être accompagnée d'informations afin de pouvoir être interprétées.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 30)
y = np.sin(x)
plt.plot(x, y, label="sin(x)")
plt.legend()
plt.title("Fonction sinus")
plt.xlabel("x")
plt.ylabel("sin(x)")

plt.show()
```



Style des lignes

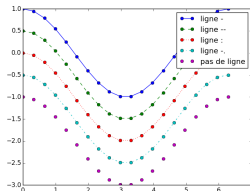
Il est nécessaire de modifier le style des lignes afin de les identifier.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 20)
y = np.cos(x)

plt.plot(x, y, "o-", label="-")
plt.plot(x, y-0.5, "o--", label="--")
plt.plot(x, y-1, "o:", label=":")
plt.plot(x, y-1.5, "o-.", label="-.")
plt.plot(x, y-2, "o", label="no line")
plt.legend()

plt.show()
```

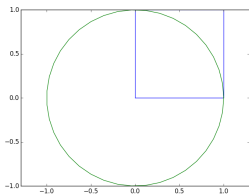


Tracé de formes

Comme la fonction `plot()` ne fait que relier des points, il est possible de lui fournir plusieurs points avec la même abscisse.

```
x = np.array([0, 1, 1, 0, 0])
y = np.array([0, 0, 1, 1, 0])
plt.plot(x, y)
plt.xlim(-1, 2)
plt.ylim(-1, 2)

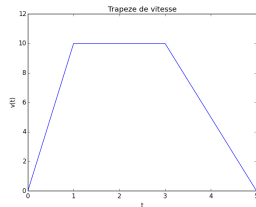
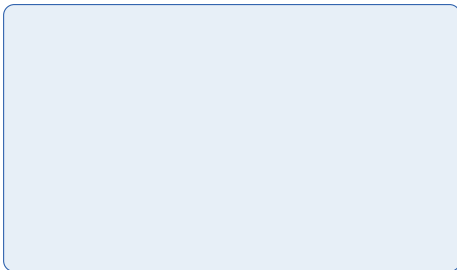
theta = np.linspace(0, 2*np.pi, 40)
x = np.cos(theta)
y = np.sin(theta)
plt.plot(x, y)
plt.axis("equal")
plt.show()
```



Tracé de formes

Coder le tracé du trapèze de vitesse $v(t) = \dot{x}(t)$ suivant:

- $0 \leq t \leq 1 : \ddot{x}(t) = 10m.s^{-2}$,
- $1 < t \leq 3 : \dot{x}(t) = v_{max}$,
- $3 < t \leq 5 : \ddot{x}(t) = a_{frein}$, avec $v(5) = 0m.s^{-1}$.

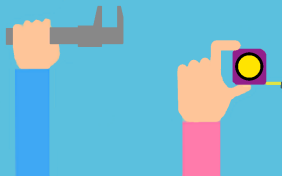




Introduction à Numpy



Renaud Costadoat
Lycée Dorian



Introduction

La bibliothèque NumPy (<http://www.numpy.org/>) permet d'effectuer des calculs numériques avec Python. Elle introduit une gestion facilitée des tableaux de nombres.

Il faut au départ importer le package numpy avec l'instruction suivante :

```
>>> import numpy as np
```

Certaines variables sont prédéfinies comme la variable π .

```
>>> np.pi  
3.141592653589793
```


Tableaux et matrices

Le premier point fondamental de Numpy (et de Scipy) est la gestion des tableaux et des matrices grâce à la fonction objet `array()`.

```
>>> np.array([1,2,3,4,5,6])
array([1, 2, 3, 4, 5, 6])
>>> np.array([1,2,3,4,5,6], 'd')
array([ 1.,  2.,  3.,  4.,  5.,  6.])
>>> np.array([1,2,3,4,5,6], 'D')
array([ 1.+0.j,  2.+0.j,  3.+0.j,  4.+0.j,  5.+0.j,  6.+0.j])
```

Pour créer une matrice, vous pouvez utiliser `array()` avec des listes de listes Python.

```
>>> np.array([[0,1],[1,0]], 'd')
array([[ 0.,  1.],
       [ 1.,  0.]])
```

Tableaux et matrices

Vous pouvez aussi générer des matrices vides (remplies de zéros) de tailles arbitraires.

```
np.zeros((3,3),'d')  
array([[ 0.,  0.,  0.],  
       [ 0.,  0.,  0.],  
       [ 0.,  0.,  0.]])
```

Ainsi que des matrices identité avec la fonction `identity()`.

```
>>> np.identity(4,'d')  
array([[ 1.,  0.,  0.,  0.],  
       [ 0.,  1.,  0.,  0.],  
       [ 0.,  0.,  1.,  0.],  
       [ 0.,  0.,  0.,  1.]])
```

Opérations sur les matrices

Les objets matrice font ce qu'il faut lorsqu'ils sont multipliés par des scalaires :

```
>>> 0.125*np.identity(3,'d')  
array([[ 0.125,  0.    ,  0.    ],  
       [ 0.    ,  0.125,  0.    ],  
       [ 0.    ,  0.    ,  0.125]])
```

Addition de deux matrices (sous réserve toutefois qu'elles soient de mêmes dimensions).

```
>>> np.identity(2,'d') + np.array([[1,1],[1,2]])  
array([[ 2.,  1.],  
       [ 1.,  3.]])
```

Opérations sur les matrices

Multiplication de matrice et produit scalaire.

```
>>> np.dot(np.identity(2), np.ones((2,2)))  
array([[ 1.,  1.],  
       [ 1.,  1.]])  
>>> v = np.array([3,4], 'd')  
>>> np.sqrt(np.dot(v,v))  
5.0
```

Produit vectoriel.

```
>>> u = np.array([1,2])  
>>> v = np.array([3,4])  
>>> np.cross(v,v)  
5.0
```

Opérations sur les matrices

Multiplication de matrices membres à membres

```
>>> a=np.array([1,2,3])  
>>> b=np.array([3,2,1])  
>>> np.multiply(a,b)  
array([3, 4, 3])
```

Opérations sur les matrices

La fonction `shape` permet de déterminer la dimension d'une matrice.

```
>>> m = np.array([[1,2],[3,4]])  
>>> m.shape  
(2,2)
```

Les fonctions `determinant()`, `inverse()` et `transpose()` produisent les résultats attendus. Une transposition peut être abrégée en plaçant `.T` à la fin d'un nom d'objet matrice.

```
>>> m = np.array([[1,2],[3,4]])  
>>> m.T  
array([[1, 3],  
       [2, 4]])
```

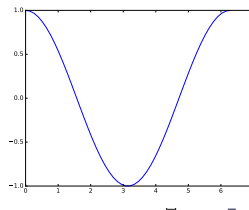
Espace linéaire, fonctions matricielles et tracés

La fonction objet `linspace(start, end, nb)` crée un tableau espace linéaire de points de valeurs comprises entre `start` (inclus) et `end` (inclus) en un nombre `nb` de points.

```
>>> np.linspace(0,1,20)
array([ 0.          ,  0.05263158,  0.10526316,  0.15789474,  0.21052632,
        0.26315789,  0.31578947,  0.36842105,  0.42105263,  0.47368421,
        0.52631579,  0.57894737,  0.63157895,  0.68421053,  0.73684211,
        0.78947368,  0.84210526,  0.89473684,  0.94736842,  1.          ])
```

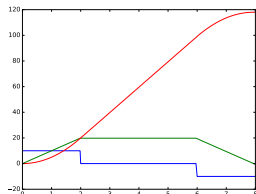
Cette fonction permet de tracer des données pour un graphique.

```
x = np.linspace(0,2*np.pi)
plt.plot(x,np.cos(x))
```



Espace linéaire: Exercice

Proposer le code permettant de tracer la figure suivante présentant, pour un solide en translation:



- l'accélération (en bleu),
- la vitesse (en vert),
- la position (en rouge).

Solveurs matriciels

Il est possible de résoudre des systèmes d'équations linéaires avec la fonction `solve()`.

$$\begin{cases} x + y + z = 6 \\ 2 * y + 5 * z = -4 \\ 2 * x + 5 * y - z = 27 \end{cases}$$

```
>>> A = np.array([[1,1,1],[0,2,5],[2,5,-1]])  
>>> b = np.array([6,-4,27])  
>>> np.linalg.solve(A,b)  
array([ 5.,  3., -2.])
```

Solveurs matriciels

Plusieurs fonctions pour calculer des valeurs propres ainsi que des vecteurs propres :

- `eigvals()` retourne les valeurs propres d'une matrice
- `eigvalsh()` retourne les valeurs propres d'une matrice hermitienne,
- `eig()` retourne les valeurs propres et les vecteurs propres d'une matrice,
- `eigh()` retourne les valeurs propres et les vecteurs propres d'une matrice hermitienne.

```
>>> A = np.array([[13,-4],[-4,7]], 'd')
>>> np.linalg.eigvalsh(A)
array([ 5., 15.])
>>> np.linalg.eigh(A)
(np.array([ 5., 15.]), np.array([[ -0.4472136 , -0.89442719],
                                [-0.89442719,  0.4472136 ]]))
```

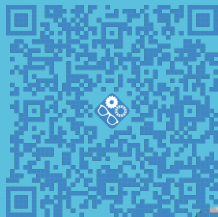
Solveurs matriciels: Exercice

Proposer un code utilisant la bibliothèque Numpy permettant de résoudre le système d'équations suivant.

$$\begin{cases} x - 3 * y + 8 * z = -16 \\ 2 * x + 3 * y - 4 * z = 24 \\ 2 * y - 3 * z = 7 \\ 8 * x - 3 * z = 17 \end{cases}$$



Numpy/Scipy: Fonctions complémentaires



Renaud Costadoat
Lycée Dorian



Introduction

Les bibliothèques NumPy (<http://www.numpy.org/>) et SciPy (<http://www.scipy.org/>) permettent d'effectuer des calculs numériques avec Python.

Il faut au départ importer ces packages avec l'instruction suivante :

```
>>> import numpy as np
>>> import scipy.optimize as resol
>>> import scipy.integrate as integr
```

Résolution approchée d'équations

Pour résoudre une équation du type $f(x) = 0$ où f est une fonction d'une variable réelle, on peut utiliser la fonction `fsolve`. Lorsqu'il existe plusieurs solutions, il est possible de donner une estimation d'une solution afin de tendre vers ce résultat.

```
def f(x) :  
    return x**2 - 3  
  
>>> resol.fsolve(f, -1.)  
[-1.73205081]  
>>> resol.fsolve(f, 1.)  
[ 1.73205081]
```

Résolution approchée d'équations

Dans le cas d'une fonction f à valeurs vectorielles, on utilise la fonction `root`. Par exemple, pour résoudre le système non linéaire:
$$\begin{cases} x^2 - y^2 = 2 \\ 2x - y - 1 = 2 \end{cases}$$

```
def f(v):  
    return v[0]**2-v[1]**2-2, 2*v[0]-v[1]-3  
>>> sol=resol.root(f,[0,0])  
>>> sol.success  
True  
>>> sol.x  
[ 1.42264973 -0.15470054]  
>>> sol=resol.root(f,[3,3])  
>>> sol.success  
True  
>>> sol.x  
[ 2.57735027  2.15470054]
```

Calcul approché d'intégrales

La fonction `quad` du module `scipy.integrate` permet de calculer la valeur approchée d'intégrales. Elle affiche l'erreur d'approximation.

```
def f(x) :  
    return x**(-3)  
  
>>> integr.quad(f, 1, 3)  
(0.4444444444444444, 4.784195109973667e-11)  
>>> integr.quad(f, 1, np.inf)  
(0.5, 5.551115123125783e-15)
```


Calcul approché d'équations différentielles

Pour résoudre une équation différentielle du premier ordre ($x'(t) = f(x, t)$), on peut utiliser la fonction `odeint` du module `scipy.integrate`.

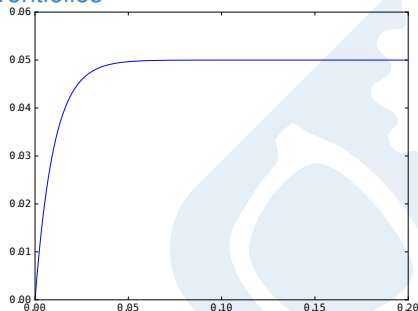
Cette fonction nécessite une liste de valeurs de t , commençant en t_0 , et une condition initiale x_0 . La fonction renvoie les valeurs approchées des solutions.

Par exemple, dans le but de déterminer la tension aux bornes d'un condensateur dans un circuit RC. Il est possible d'utiliser le code suivant.

```
E=5 #Tension en volt
R=200 #Résistance en ohm
C=50*10**(-6) #Capacité du condensateur en Coulomb
def f(u, t) :
    return E-u/(R*C)
```

Calcul approché d'équations différentielles

```
>>> T=np.arange(0,0.2,0.01)
>>> X= integr.odeint(f,0,T)
>>> X
array([[ 0.          ],
       [ 0.03160602],
       [ 0.04323323],
       [ 0.04751065],....
```



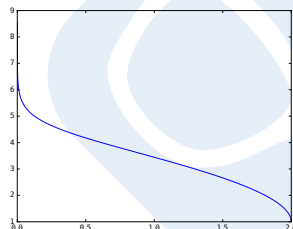
Question: Coder la résolution du mouvement oscillatoire d'une masse liée à un ensemble amortisseur/ressort.

Calcul approché d'équations différentielles

Il est alors possible de résoudre des systèmes d'équations différentielles.

$$\begin{cases} x'(t) = -t * x(t) \\ y'(t) = x(t) + 0.2 * y(t) \end{cases}, \text{ avec la condition initiale } x(0) = 2, y(0) = 1.$$

```
def f(x, t) :
    f=np.array([-t*x[0], x[0]+0.2*x[1]])
    return f
T = np.arange(0, 5.01, 0.01)
X = integr.odeint(f, np.array([2.,1.]), T)
>>> X
[[ 2.00000000e+00  1.00000000e+00]
 [ 1.99990000e+00  1.02202166e+00]
 ...,
```



Le système d'ordre 1 satisfait par $X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}$ permettra de résoudre une équation différentielle scalaire d'ordre 2 de solution x .

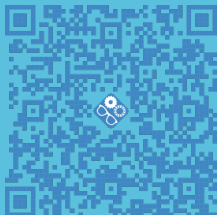
Nombres complexes

Avec Python le nombre imaginaire pur i se note `1j`. Les attributs `real` et `imag` permettent d'obtenir la partie réelle et la partie imaginaire. La fonction `abs` calcule son module.

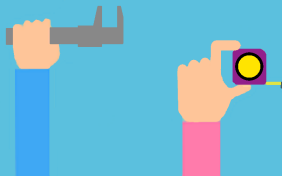
```
>>> a = 6 + 2j
>>> b = 9 - 1j
>>> a*b
(56+12j)
>>> a.real
6.0
>>> a.imag
2.0
>>> abs(a)
6.324555320336759
```



Méthode de Newton



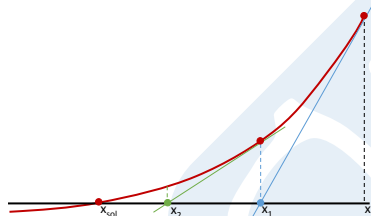
Renaud Costadoat
Lycée Dorian



Développement de Taylor

Hypothèses de départ:

- f est continue sur l'intervalle $[a, b]$,
- f s'annule en un unique point de $[a, b]$, que l'on notera x_{sol} ,
- f une fonction dérivable sur $[a, b]$,
- f' ne s'annule pas sur $[a, b]$.



Definition

Théorème du développement de Taylor à l'ordre 1.

Soit f une fonction définie sur un intervalle I et $u \in I$. Le développement de Taylor à l'ordre 1 de f est donné par:

$$f(x) = f(u) + f'(u) \cdot (x - u) + o(x - u)$$

Géométriquement, lorsqu'on néglige le reste, le développement de Taylor donne l'équation de la tangente en u . L'intersection de cette tangente avec l'axe (O, \vec{x}) tend vers x_{sol} .

Démarche

Notons $\Delta(x)$ cette équation, l'abscisse c de l'intersection de la tangente avec l'axe des abscisses est donnée par la résolution de $\Delta(c) = 0() \Leftrightarrow f(u) + f'(u).(c - u) = 0 \Leftrightarrow c = u - \frac{f(u)}{f'(u)}$.

Pseudo code:

1. Soit x_0 , un réel dans I ,
2. La tangente à Cf au point d'abscisse x_0 , a pour équation $y = f'(x_0).(x - x_0) + f(x_0)$,
3. x_1 est le réel vérifiant: $f'(x_0).(x_1 - x_0) + f(x_0) = 0$, ainsi $x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$,
4. Récurrence: le réel x_n étant construit, calculer $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.

Conditions d'arrêt

- lorsque l'écart entre deux termes consécutifs est inférieur à une valeur définie,
- lorsque l'image du milieu de notre intervalle par f est inférieur à une valeur définie,
- lorsque le nombre d'itérations est supérieur à un nombre défini.

Le dernier terme x_n sera une approximation de x_{sol} .

Code python

Proposer le code python de la fonction `methode_Newton` .

Code python

Proposer le code python de la fonction `methode_Newton` .

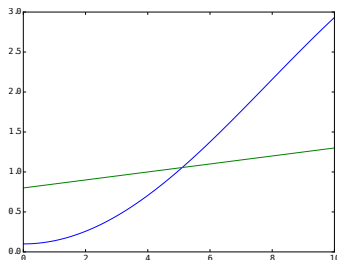
```
def methode_Newton(f,df,a,p) :  
    x0=a  
    while abs(f(x0))>p : # ou abs(e)>p  
        x1=x0-f(x0)/df(x0)  
        e=x1-x0  
        x0=x1  
    return x1
```

Exercice

Déterminer en utilisant Python le point d'intersection des deux fonctions suivantes sur l'intervalle $[0, 5]$.

• $f(x) = -2.\cos\left(\frac{x}{5}\right) + 2,1,$

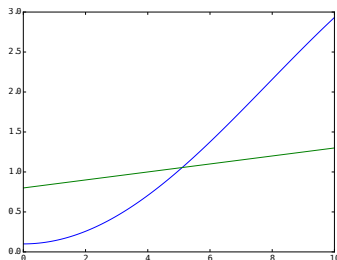
• $f(x) = 0,05.x + 0,8.$



Exercice

Déterminer en utilisant Python le point d'intersection des deux fonctions suivantes sur l'intervalle $[0, 5]$.

- $f(x) = -2.\cos(\frac{x}{5}) + 2,1$,
- $f(x) = 0,05.x + 0,8$.



```
def f(x):  
    return -2*np.cos(x/5)+2.1-(0.05*x+0.8)  
def df(x):  
    return 2/5.*np.sin(x/5)-(0.05)  
  
print(methode_Newton(f,df,a,p))
```

Comparaison des deux méthodes

Deux critères permettent de déterminer l'efficacité des deux méthodes.

- **Rapidité de convergence:** Dans les hypothèses où les deux méthodes convergent, la méthode de Newton converge plus « rapidement ». Exemple dans le cas précédent (résultat, nombre d'itérations) :

(5.1059522149400793, 5)
(5.105952024459839, 25)

- **Hypothèses et fonctions étudiées:** Sur ce critère, la méthode est moins contraignante, de plus, il n'est pas nécessaire avec la méthode de la dichotomie de déterminer la dérivée de la fonction.



Méthode d'Euler



Renaud Costadoat
Lycée Dorian

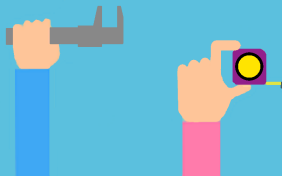


Schéma d'Euler

Soit une équation différentielle écrite sous la forme :

$$\forall x \in I, u'(x) = f(x, u(x)), \text{ avec } u(x_0) = y_0$$

L'objectif de l'utilisation de la méthode d'Euler est de calculer une approximation de la fonction u sur l'intervalle I . Cette approximation peut être:

- **locale**: si seule la valeur en un point nous intéresse,
- **globale**: calcul des valeurs de u à intervalles réguliers.

Objectif

Schéma d'Euler

D'après le calcul du taux d'accroissement:

$$\forall x \in I, u'(x) = \lim_{x \rightarrow a} \frac{u(x) - u(a)}{x - a}$$

Ce qui peut s'écrire, après un changement de variable:

$$\forall x \in I, u'(x) = \lim_{dx \rightarrow 0} \frac{u(x + dx) - u(x)}{dx}$$

En reprenant l'équation différentielle du 1er ordre vu précédemment, on obtient:

$$\forall x \in I, f(x, u(x)) = \lim_{dx \rightarrow 0} \frac{u(x + dx) - u(x)}{dx}$$

Démarche

En discrétisant le problème précédent, il est alors possible d'écrire:

C'est la récurrence suivante qui va nous permettre de résoudre l'équation différentielle:

Pseudo code:

1. Soit x , un vecteur représentant la variable de dérivation,
2. Soit y_0 , un réel représentant la condition initiale $u(x_0) = y_0$
3. $u(x_1) = (x_1 - x_0) * f(x, u(x_0)) + u(x_0)$
4. Récurrence: le réel x_n étant construit, calculer $u(x_{i+1}) = (x_{i+1} - x_i) * f(x, u(x_i)) + u(x_i)$

Démarche

En discrétisant le problème précédent, il est alors possible d'écrire:

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

C'est la récurrence suivante qui va nous permettre de résoudre l'équation différentielle:

$$u(x_{i+1}) = (x_{i+1} - x_i) * f(x, u(x_i)) + u(x_i)$$

Pseudo code:

1. Soit x , un vecteur représentant la variable de dérivation,
2. Soit y_0 , un réel représentant la condition initiale $u(x_0) = y_0$
3. $u(x_1) = (x_1 - x_0) * f(x, u(x_0)) + u(x_0)$
4. Récurrence: le réel x_n étant construit, calculer $u(x_{i+1}) = (x_{i+1} - x_i) * f(x, u(x_i)) + u(x_i)$

Code python

Proposer le code python de la fonction `methode_Euler` .

Code python

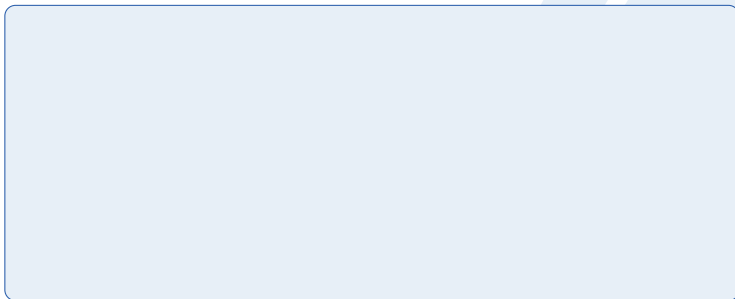
Proposer le code python de la fonction `methode_Euler` .

```
def methode_euler(F,y0,t):  
    y = [0]*len(t)  
    y[0] = y0  
    for i in range(len(t)-1):  
        y[i+1] = y[i]+(t[i+1]-t[i])*F(y[i],t[i])  
    return y
```

Application

On considère une équation différentielle d'ordre 1 avec condition initiale (problème de Cauchy) :
$$\begin{cases} y'(t) = y(t) \\ y(0) = 1 \end{cases}$$

Résoudre cette équation à l'aide de la méthode d'Euler.



Application

On considère une équation différentielle d'ordre 1 avec condition initiale (problème de Cauchy) :
$$\begin{cases} y'(t) = y(t) \\ y(0) = 1 \end{cases}$$

Résoudre cette équation à l'aide de la méthode d'Euler.

```
def F(y,t):  
    return y  
  
def methode_euler(F,y0,t):  
    y = [0]*len(t)  
    y[0] = y0  
    for i in range(len(t)-1):  
        y[i+1] = y[i]+(t[i+1]-t[i])*F(y[i],t[i])  
    return y
```

Application: Le circuit R.C.

L'objet de l'étude suivante est le calcul de la tension aux bornes du condensateur dans un circuit R.C. lorsqu'il est en charge.

Donner les équations électriques qui régissent son comportement.

Donc,

Données:

- $R = 300\Omega$,
- $e(t) = 6V$,
- $C = 10mF$.

Application: Le circuit R.C.

L'objet de l'étude suivante est le calcul de la tension aux bornes du condensateur dans un circuit R.C. lorsqu'il est en charge.

Donner les équations électriques qui régissent son comportement.

$$\begin{cases} e(t) = U_R(t) + U_C(t) \\ U_R(t) = R.i(t) \\ i(t) = C. \frac{dU_C(t)}{dt} \end{cases}$$

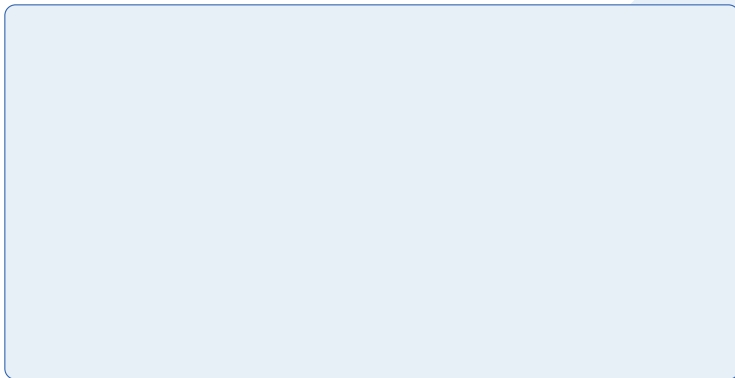
Donc,
$$\begin{cases} \frac{dU_C(t)}{dt} = \frac{e(t) - U_C(t)}{R.C} \\ U_C(0) = 0 \end{cases}$$

Données:

- $R = 300\Omega$,
- $e(t) = 6V$,
- $C = 10mF$.

Application: Le circuit R.C.

Résoudre cette équation à l'aide de la méthode d'Euler.



Application: Le circuit R.C.

Résoudre cette équation à l'aide de la méthode d'Euler.

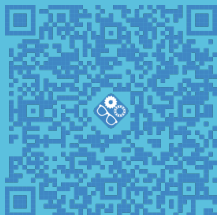
```
e=6
R=300
C=10*10**(-3)

def F(y,t):
    return (e-y)/(R*C)

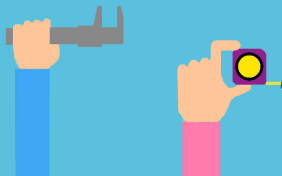
def methode_euler(F,y0,t):
    y = [0]*len(t)
    y[0] = y0
    for i in range(len(t)-1):
        y[i+1] = y[i]+(t[i+1]-t[i])*F(y[i],t[i])
    return y
```



Pivot de Gauss



Renaud Costadoat
Lycée Dorian



Système d'équations

Soit le système d'équations suivant :

$$\begin{cases} y_0 = a_{0,0} \cdot x_0 + a_{0,1} \cdot x_1 + a_{0,2} \cdot x_2 + \dots + a_{0,n-1} \cdot x_{n-1} \\ y_1 = a_{1,0} \cdot x_0 + a_{1,1} \cdot x_1 + a_{1,2} \cdot x_2 + \dots + a_{1,n-1} \cdot x_{n-1} \\ \dots \\ y_{n-1} = a_{n-1,0} \cdot x_0 + a_{n-1,1} \cdot x_1 + a_{n-1,2} \cdot x_2 + \dots + a_{n-1,n-1} \cdot x_{n-1} \end{cases}$$

Ce système d'équations peut s'écrire sous la forme suivante: $Y = A \cdot X$, avec:

$$X = \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{n-1} \end{pmatrix}, Y = \begin{pmatrix} y_0 \\ y_1 \\ \dots \\ y_{n-1} \end{pmatrix}, A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & a_{1,2} & \dots & a_{1,n-1} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \dots & a_{n-1,n-1} \end{pmatrix}$$

Ces systèmes sont dit de Cramer, c'est à dire qu'ils admettent une unique solution. La matrice A est donc inversible.

Cas triangulaire

Si la matrice A est triangulaire, la résolution du problème est simple.

Ce système d'équations peut s'écrire sous la forme suivante: $Y = A.X$, avec:

$$A = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \dots & a_{0,n-1} \\ 0 & a_{1,1} & a_{1,2} & \dots & a_{1,n-1} \\ 0 & 0 & a_{2,2} & \dots & a_{2,n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-1,n-1} \end{pmatrix}$$

La dernière équation devient: $y_{n-1} = a_{n-1,n-1} \cdot x_{n-1}$, ainsi $x_{n-1} = \frac{a_{n-1,n-1}}{y_{n-1}}$. En remontant tout le système l'ensemble des x_i peuvent être déterminés.

$$\text{Ainsi, } \forall i \in [0, n-2], x_i = \frac{y_i - \sum_{j=i+1}^{n-1} a_{i,j} \cdot x_j}{a_{i,i}}$$

Cas triangulaire

Soit la matrice triangulaire A et le vecteur Y suivant :

$$A = \begin{pmatrix} 2 & 5 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 4 \end{pmatrix}, Y = \begin{pmatrix} 6 \\ 4 \\ 9 \end{pmatrix}$$

Cela correspond au système d'équations suivant :

$$\begin{cases} 6 = 2.x_0 + 5.x_1 + 3.x_2 \\ 4 = 1.x_1 + 2.x_2 \\ 9 = 4.x_2 \end{cases}$$

Cela mène à la résolution suivante :

$$\begin{cases} x_0 = \\ x_1 = \\ x_2 = \end{cases}$$

Cas triangulaire

Soit la matrice triangulaire A et le vecteur Y suivant :

$$A = \begin{pmatrix} 2 & 5 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 4 \end{pmatrix}, Y = \begin{pmatrix} 6 \\ 4 \\ 9 \end{pmatrix}$$

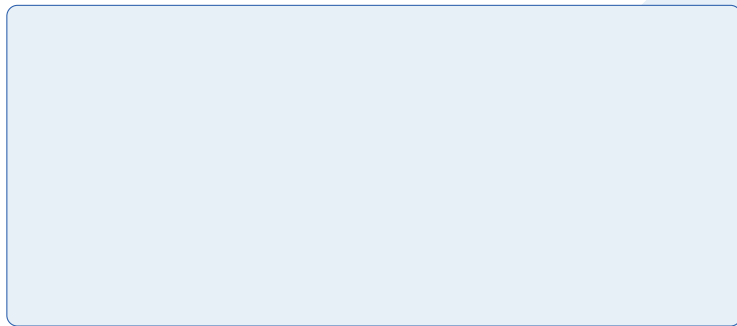
Cela correspond au système d'équations suivant :

$$\begin{cases} 6 = 2.x_0 + 5.x_1 + 3.x_2 \\ 4 = 1.x_1 + 2.x_2 \\ 9 = 4.x_2 \end{cases}$$

Cela mène à la résolution suivante :

$$\begin{cases} x_0 = 1.75 \\ x_1 = -0.5 \\ x_2 = 2.25 \end{cases}$$

Cas triangulaire: Code Python



Cas triangulaire: Code Python

```
def triangle(A,b):  
    n =len(b)  
    x = [0 for i in range(n)]  
    x[n-1] = b[n-1]/A[n-1][n-1]  
    for i in range(n-2,-1,-1):  
        s = 0  
        for j in range(i+1,n):  
            s = s + A[i][j]*x[j]  
        x[i] = (b[i] - s) / A[i][i]  
    return x
```


Etape 1: Placer le pivot sur la première ligne

Soit la matrice $A = \begin{pmatrix} 2 & 2 & -3 & 4 \\ -2 & -1 & -3 & 2 \\ 6 & 4 & 4 & 5 \\ 1 & 5 & 3 & 2 \end{pmatrix}$, $Y = \begin{pmatrix} 2 \\ -5 \\ 16 \\ 11 \end{pmatrix}$

1. Recherche de la plus grande valeur $A[0]_{max}$ de la première colonne de la matrice: ici $A[2][0] = 6$,
2. Inverser cette ligne et la première de la matrice et faire de même pour le vecteur.

$$A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ -2 & -1 & -3 & 2 \\ 2 & 2 & -3 & 4 \\ 1 & 5 & 3 & 2 \end{pmatrix}, Y = \begin{pmatrix} 16 \\ -5 \\ 2 \\ 11 \end{pmatrix}$$

Etape 2: Mettre des 0 sur le reste de la première colonne

Soit la matrice $A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ -2 & -1 & -3 & 2 \\ 2 & 2 & -3 & 4 \\ 1 & 5 & 3 & 2 \end{pmatrix}$, $Y = \begin{pmatrix} 16 \\ -5 \\ 2 \\ 11 \end{pmatrix}$

Pour chaque ligne sauf la première

1. Calcul du coefficient $x = A[i][0]/A[0][0]$,
2. Calculer les nouveaux coefficients qui permettent d'annuler le premier:

$$A[i][k] = A[i][k] - x * A[0][k]$$

$$A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ -2 & -1 & -3 & 2 \\ 2 & 2 & -3 & 4 \\ 1 & 5 & 3 & 2 \end{pmatrix}, Y = \begin{pmatrix} 16 \\ 0.33 \\ -3.33 \\ 8.33 \end{pmatrix}$$

Etape 2: Mettre des 0 sur le reste de la première colonne

Soit la matrice $A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ -2 & -1 & -3 & 2 \\ 2 & 2 & -3 & 4 \\ 1 & 5 & 3 & 2 \end{pmatrix}$, $Y = \begin{pmatrix} 16 \\ -5 \\ 2 \\ 11 \end{pmatrix}$

Pour chaque ligne sauf la première

1. Calcul du coefficient $x = A[i][0]/A[0][0]$,
2. Calculer les nouveaux coefficients qui permettent d'annuler le premier:

$$A[i][k] = A[i][k] - x * A[0][k]$$

$$A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ 0 & 0.33 & -1.66 & 3.66 \\ 0 & 0.66 & -4.33 & 2.33 \\ 0 & 4.33 & 2.33 & 1.16 \end{pmatrix}, Y = \begin{pmatrix} 16 \\ 0.33 \\ -3.33 \\ 8.33 \end{pmatrix}$$

Etape 3: Mettre des 0 sur le reste de la première colonne

La procédure continue avec la matrice de dimension juste inférieure

$$A = \begin{pmatrix} 0.33 & -1.66 & 3.66 \\ 0.66 & -4.33 & 2.33 \\ 4.33 & 2.33 & 1.16 \end{pmatrix}, Y = \begin{pmatrix} 0.33 \\ -3.33 \\ 8.33 \end{pmatrix}$$

Le résultat de l'étape de mise sous forme triangulaire est alors:

$$A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ & & & \\ & & & \\ & & & \end{pmatrix}, Y = \begin{pmatrix} 16 \\ 8.33 \\ -4.61 \\ 1.5 \end{pmatrix}$$

Etape 3: Mettre des 0 sur le reste de la première colonne

La procédure continue avec la matrice de dimension juste inférieure

$$A = \begin{pmatrix} 0.33 & -1.66 & 3.66 \\ 0.66 & -4.33 & 2.33 \\ 4.33 & 2.33 & 1.16 \end{pmatrix}, Y = \begin{pmatrix} 0.33 \\ -3.33 \\ 8.33 \end{pmatrix}$$

Le résultat de l'étape de mise sous forme triangulaire est alors:

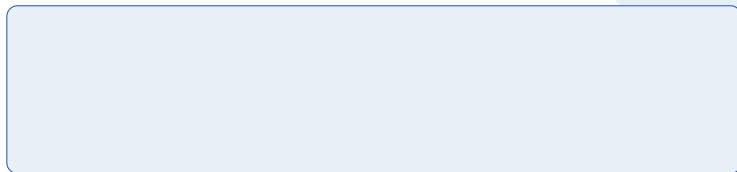
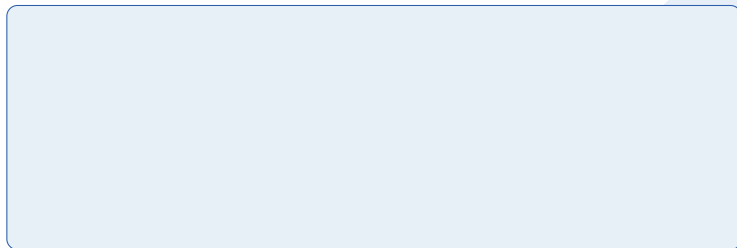
$$A = \begin{pmatrix} 6 & 4 & 4 & 5 \\ 0 & 4.33 & 2.33 & 1.16 \\ 0 & 0 & -4.69 & 2.15 \\ 0 & 0 & 0 & 2.73 \end{pmatrix}, Y = \begin{pmatrix} 16 \\ 8.33 \\ -4.61 \\ 1.5 \end{pmatrix}$$

Il ne reste plus qu'à utiliser la méthode décrite précédemment afin de déterminer le résultat.

Le résultat final pour ces valeurs est alors: $X = \begin{pmatrix} 0.642642642643 \\ 1.10810810811 \\ 1.23723723724 \\ 0.552552552553 \end{pmatrix}$

Etude 1: Code Python

Recherche du pivot et permutation des lignes:



Etude 1: Code Python

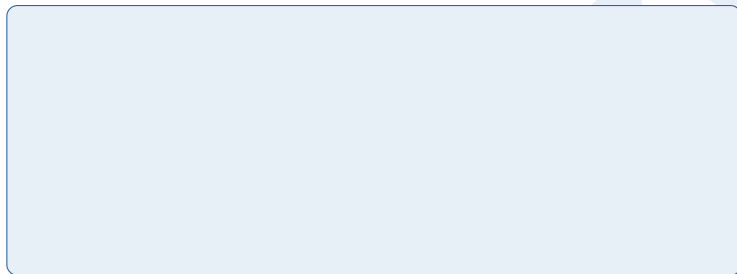
Recherche du pivot et permutation des lignes:

```
def recherche_pivot(A, j0):  
    n = len(A)  
    imax = j0  
    for i in range(j0+1, n):  
        if abs(A[i][j0]) > abs(A[imax][j0]):  
            imax = i  
    return imax
```

```
def permutation(A,i,j):  
    t = A[i]  
    A[i] = A[j]  
    A[j] = t
```


Etude 2: Code Python

Transvection:



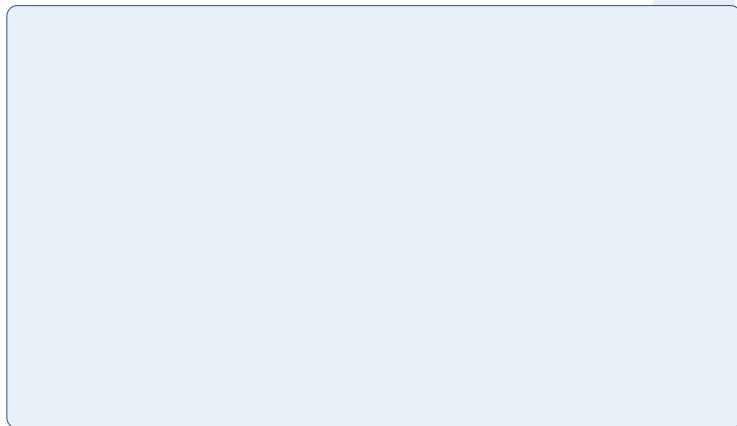
Etude 2: Code Python

Transvection:

```
def transvection(A,i,j,x):  
    # si vecteur  
    if type(A[0]) != list:  
        A[i] = A[i] + x*A[j]  
    else:  
        n = len(A[0])  
        for k in range(n):  
            A[i][k] = A[i][k] + x*A[j][k]
```

Etude 3: Code Python

Résolution du système:



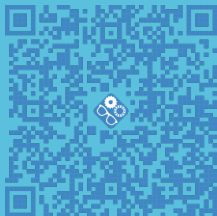
Etude 3: Code Python

Résolution du système:

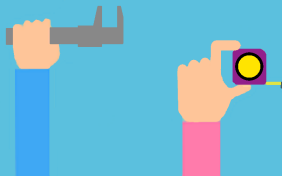
```
def resolution_systeme(A0,b0):  
    n = len(A)  
  
    for i in range(n-1):  
        i0 = recherche_pivot(A,i)  
        permutation(A,i,i0)  
        permutation(b,i,i0)  
  
        for k in range(i+1,n):  
            x = A[k][i]/A[i][i]  
            transvection(A,k,i,-x)  
            transvection(b,k,i,-x)  
  
    return triangle(A,b)
```



Bases de données



Renaud Costadoat
Lycée Dorian



Bases de données

Definition

Une Base de données est un gros ensemble d'**informations** structurées **mémorisées** sur un support permanent.

Un système de fichier correspond à cette définition, cependant, l'utilisation directe de fichiers soulève des problèmes :

1. Lourdeur d'accès aux données,
2. Manque de sécurité,
3. Pas de contrôle de concurrence.

C'est pour cela qu'un logiciel de gestion est utilisé.

Definition

Un Système de Gestion de Bases de Données (SGBD) est un **logiciel** de haut niveau qui permet de manipuler les **informations** stockées dans une base de données.

Utilisation d'un SGBD

L'utilisation d'un SGBD suppose de comprendre (et donc de savoir utiliser) les fonctionnalités suivantes :

1. Définition du schéma de données en utilisant les modèles de données du SGBD,
2. Opérations sur les données : recherche, mises-à-jour, etc,
3. Partager les données entre plusieurs utilisateurs. (Mécanisme de transaction),
4. Optimiser les performances, par le réglage de l'organisation physique des données. Cet aspect relève plutôt de l'administration et ne sera évoqué que dans l'introduction.

Exemple de données

Extrait d'une base de données sur le cinéma.

Film	Année	Nom Réal	Prénom Réal	Année Réal
Fargo	1996	Coen	Joel	1953
Fargo	1996	Coen	Ethan	1957
Pulp Fiction	1994	Tarantino	Quentin	1963
Inglourious Basterds	2009	Tarantino	Quentin	1963
Inside Llewyn Davis	2013	Coen	Joel	1953
Inside Llewyn Davis	2013	Coen	Ethan	1957
RockNRolla	2008	Ritchie	Guy	1968

Avec un cas aussi simple, un ensemble de problèmes potentiels apparaît. Une des principales causes de problème vient du fait qu'il est possible de représenter la même information **plusieurs fois**.

Anomalies liées à l'utilisation d'une base de données

Anomalies lors d'une insertion

Il est possible de faire apparaître plusieurs fois le même film (*Fargo* apparaît deux fois). Une bonne question consiste d'ailleurs à se demander ce qui distingue deux films l'un de l'autre, et à quel moment on peut dire que la même information a été répétée. Peut-il y avoir deux films différents avec le même titre par exemple ?

Anomalies lors d'une modification

La redondance d'information entraîne également des anomalies de mise à jour. En modifiant l'année de naissance de *Quentin Tarantino* sur la ligne *Inglourious Basterds*, cela ne modifie pas la ligne *Pulp Fiction*. Des informations incohérentes apparaissent.

Anomalies lors d'une destruction

Il est impossible de supprimer un film sans supprimer du même coup son metteur en scène. Si on souhaite, par exemple, ne plus voir le film *RockNRolla* figurer dans la base de données, on va effacer du même coup les informations sur *Guy Ritchie*.

Distinctions des tables

Une bonne méthode évitant les anomalies ci-dessus consiste à :

1. Être capable de représenter individuellement les films et les réalisateurs, de manière à ce qu'une action sur l'un n'entraîne pas systématiquement une action sur l'autre,
2. Définir une méthode d'identification d'un film ou d'un réalisateur, qui permette d'assurer que la même information est représentée une seule fois,
3. Préserver le lien entre les films et les réalisateurs, mais sans introduire de redondance.

Table des films

Id	Film	Année
1	Fargo	1996
2	Pulp Fiction	1994
3	Inglourious Basterds	2009
4	Inside Llewyn Davis	2013
5	RockNRolla	2008

Table des réalisateurs

Id	Nom	Prénom	Année
1	Coen	Joel	1953
2	Coen	Ethan	1957
3	Tarantino	Quentin	1963
4	Ritchie	Guy	1968

Lien entre les tables

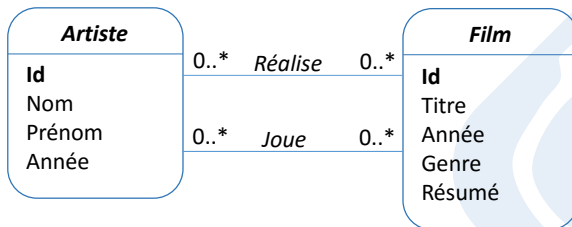
Il reste à représenter le lien entre les films et les metteurs en scène, sans introduire de redondance. Maintenant que nous avons défini les identifiants, il existe un moyen simple pour indiquer quel est le metteur en scène qui a réalisé un film : associer l'identifiant du metteur en scène au film. On ajoute des attributs *IdReal1* et *IdReal2* dans la *Table des films*, et on obtient la représentation suivante.

Table des films

Id	Film	Année	IdReal1	IdReal2
1	Fargo	1996	1	2
2	Pulp Fiction	1994	3	
3	Inglourious Basterds	2009	3	
4	Inside Llewyn Davis	2013	1	2
5	RockNRolla	2008	4	

Schéma base de données

Le schéma suivant permet de décrire la base de donnée et les liens entre les bases.



Les associations sont caractérisées par des cardinalités. La notation **0..*** sur le lien *Réalise*, du côté de l'entité **Artiste**, signifie qu'un film peut être réalisé par plusieurs artistes, ou aucun. Une notation **0..1** signifierai en revanche qu'un film ne peut être réalisé que par au plus un artiste, **1..1** signifierai un seul artiste.

Les **clés** sont indiqués en gras, elles permettent d'identifier de manière unique une instance de la table.

Tables de jointure

La mise en place de tables de jointure permet de retrouver les relations entre les instances des tables.

Table des réalisateurs

Id	IdFilm	IdReal1	IdReal2
1	1	1	2
2	2	3	
3	3	3	
4	5	4	
5	4	1	2

Définition d'un schéma relationnel

Une relation est caractérisée par un **nom** et se compose d'un ensemble de colonnes désignées par un nom d'**attribut**. Dans chaque colonne on trouve des valeurs d'un certain **domaine** (chaînes de caractères, nombres). Enfin on constate que chaque ligne (ou tuple) correspond à une **entité**.

Ainsi pour le schéma relationnel suivant:

Film (titre: string, année: number, genre : string)

- *Film* est le **nom**,
- *titre*, *année* et *genre* sont des **attributs**,
- *string* et *number* sont des **domaines**.

('Fargo', 1996, 'Thriller') est un **tuple**.

Types du langage de données SQL

Type	Description	Taille
INTEGER	Type des entiers relatifs	4 octets
SMALLINT	Idem.	2 octets
BIGINT	Idem.	8 octets
FLOAT	Flottants simple précision	4 octets
DOUBLE PRECISION	Flottants double précision	8 octets
REAL	Synonyme de FLOAT	4 octets
NUMERIC (M, D)	Numérique avec précision fixe.	M octets
DECIMAL (M, D)	Idem.	M octets
CHAR(M)	Chaînes de longueur fixe	M octets
VARCHAR(M)	Chaînes de longueur variable	$L + 1$ avec $L \leq M$
BIT VARYING	Chaînes d'octets	Longueur de la chaîne
DATE	Date (jour, mois, an)	env. 4 octets
TIME	Horaire (heure, minutes, secondes)	env. 4 octets
DATETIME	Date et heure	8 octets
YEAR	Année	2 octets

Langage de définition de données SQL

Langage de définition de données SQL

```
CREATE TABLE eleve (id INT (4) NOT NULL,  
                    email VARCHAR (50) NOT NULL,  
                    nom VARCHAR (20) NOT NULL,  
                    prenom VARCHAR (20),  
                    motDePasse VARCHAR (60) NOT NULL,  
                    classe VARCHAR (4) DEFAULT 'PTSI',  
                    lvl INT (1) NOT NULL,  
                    PRIMARY KEY (id),  
                    FOREIGN KEY (lvl))
```

Clés primaire (PRIMARY KEY)

Une clé est un attribut qui identifie de manière unique un tuple d'une relation.

Clés étrangères (FOREIGN KEY)

La norme SQL ANSI permet d'indiquer quelles sont les clés étrangères dans une table, autrement dit, quels sont les attributs qui font référence à une ligne dans une autre table.

Langage de définition de données SQL

Modification d'une table: Ajout d'un champ

```
ALTER TABLE Internaute ADD region VARCHAR(10)
```

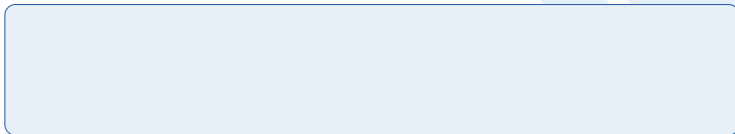
Modification d'une table: Modification d'un champ

```
ALTER TABLE Internaute MODIFY region VARCHAR(30) NOT NULL
```

Application

Proposer le code de création d'une table permettant de gérer la lv1. Les attributs seront:

- un *id* (clé primaire),
- un *nom*,
- une *salle*.



Application

Proposer le code de création d'une table permettant de gérer la lv1. Les attributs seront:

- un *id* (clé primaire),
- un *nom*,
- une *salle*.

```
CREATE TABLE lv1 (id INT (1) NOT NULL,  
                  nom VARCHAR (30) NOT NULL,  
                  salle VARCHAR (30) NOT NULL,  
                  PRIMARY KEY (id))
```

Requêtes SQL

La suite va prendre pour exemple la base de données des *Villes de France* dont le schéma relationnel est le suivant:

Ville (ville_id:int, ville_departement:int, ville_nom_reel=: char, ville_code_postal=int, ville_population_2010=int, ville_longitude_dms:int , ville_latitude_dms:int) (d'autres colonnes sont présentes mais elles ne nous intéressent pas ici).

- **FROM** indique la (ou les) tables dans lesquelles on trouve les attributs utiles à la requête,
- **SELECT** indique la liste des attributs constituant le résultat (* pour tout sélectionner),
- **WHERE** indique les conditions que doivent satisfaire les n-uplets de la base pour faire partie du résultat.

Requêtes SQL

Lister les éléments de la base

```
SELECT ville_nom_reel, ville_population_2010  
FROM villes_france_free  
WHERE ville_code_postal=40990
```

ville_nom_reel	ville_population_2010
Saint-Vincent-de-Paul	2997
Téthieu	656
Gourbera	356
Herm	1044
Mées	1736
Saint-Paul-lès-Dax	12409
Angoumé	288

Requêtes SQL

Modifier l'affichage des attributs.

```
SELECT ville_nom_reel as 'Ville',  
ville_population_2010/1000 AS 'Population en milliers'  
FROM villes_france_free  
WHERE ville_code_postal=40990 and ville_population_2010>1000
```

Ville	Population en milliers
Herm	1.0440
Mées	1.7360
Saint-Vincent-de-Paul	2.9970
Saint-Paul-lès-Dax	12.4090

Requêtes SQL

Recherche dans une table.

```
SELECT ville_nom_reel as 'Ville', ville_code_postal as 'Code Postal'  
FROM 'villes_france_free'  
WHERE 'ville_nom_reel'='Saint-Vincent-de-Paul'
```

Ville	Code Postal
Saint-Vincent-de-Paul	33440
Saint-Vincent-de-Paul	40990

Éviter les doublons.

```
SELECT DISTINCT ville_nom_reel as 'Ville'  
FROM 'villes_france_free'  
WHERE 'ville_nom_reel'='Saint-Vincent-de-Paul'
```

Ville
Saint-Vincent-de-Paul

Requêtes SQL

Affichage trié:

```
SELECT ville_nom_reel, ville_population_2010  
FROM villes_france_free  
WHERE ville_code_postal=40990 ORDER BY ville_population_2010
```

ville_nom_reel	ville_population_2010
Angoumé	288
Gourbera	356
Téthieu	656
Herm	1044
Mées	1736
Saint-Vincent-de-Paul	2997
Saint-Paul-lès-Dax	12409

Le tri dans l'ordre descendant s'effectue en ajoutant DESC, à la suite des paramètres de classement.

Requêtes SQL

Calcul de somme:

```
SELECT SUM(ville_population_2010) AS 'Population Totale 40990'  
FROM villes_france_free  
WHERE ville_code_postal=40990
```

Population Totale 40990
19486

Calcul de moyenne:

```
SELECT AVG(ville_population_2010) AS 'Population Moyenne 40990'  
FROM villes_france_free  
WHERE ville_code_postal=40990
```

Population Moyenne 40990
2783.7143

Requêtes SQL

Compter les valeurs:

```
SELECT COUNT(ville_nom_reel) AS 'Nombre de villes'  
FROM villes_france_free  
WHERE ville_code_postal=40990
```

Nombre de villes
7

```
SELECT ville_departement AS 'Département',  
COUNT(ville_id) as 'Nb villes'  
FROM villes_france_free  
WHERE ville_departement > 40  
      AND ville_departement < 45  
GROUP BY ville_departement
```

Département	Nb villes
41	291
42	327
43	260
44	221

Requêtes SQL

Limiter le nombre de réponse

```
SELECT ville_nom_reel AS Ville,  
       ville_population_2010 AS Population  
FROM villes_france_free  
ORDER BY ville_population_2010 DESC LIMIT 1
```

Ville	Population
Paris	2243833

Rechercher le maximum d'une liste

```
SELECT ville_nom_reel AS Ville,  
       ville_population_2010 AS Population  
FROM villes_france_free  
WHERE ville_population_2010=  
      (SELECT max(ville_population_2010)  
       FROM villes_france_free)
```

Ville	Population
Paris	2243833

Requêtes SQL

Recherche de plusieurs maximums.

```
SELECT ville_departement AS Département, ville_nom_reel AS Ville,  
       ville_population_2010 AS Population  
FROM villes_france_free,  
     (SELECT max(ville_population_2010) AS Max FROM villes_france_free  
      WHERE ville_departement >=40 AND ville_departement <45  
      GROUP BY ville_departement) m  
WHERE v.ville_population_2010=m.Max
```

Département	Ville	Population
40	Mont-de-Marsan	31225
41	Blois	46492
42	Saint-Étienne	171260
43	Le Puy-en-Velay	18521
44	Nantes	284970

Requêtes SQL

Utilisation de `HAVING` pour limiter les résultats avec des `SUM` , `AVG` ,...

```
SELECT ville_departement AS Département,  
       SUM(ville_population_2010) AS Population  
FROM villes_france_free  
GROUP BY ville_departement  
HAVING SUM(ville_population_2010)<100000
```

Département	Population
48	77082
975	6080

Requêtes SQL

Des tables sont ajoutées afin de gérer les régions et les départements.

```
SELECT departement_nom as Département, departement_code as Numéro  
FROM 'departement'  
WHERE departement_code < 50 AND departement_code >= 40
```

Département	Numéro
Landes	40
Loir-et-Cher	41
Loire	42
Haute-Loire	43
Loire-Atlantique	44
Loiret	45
Lot	46
Lot-et-Garonne	47
Lozère	48
Maine-et-Loire	49

Requêtes SQL

```
SELECT 'Id_region', 'NCCENR' AS Nom  
FROM 'regions'  
WHERE 'Id_region'>40
```

Id_region	Nom
44	Alsace-Champagne-Ardenne-Lorraine
52	Pays de la Loire
53	Bretagne
75	Aquitaine-Limousin-Poitou-Charentes
76	Languedoc-Roussillon-Midi-Pyrénées
84	Auvergne-Rhône-Alpes
93	Provence-Alpes-Côte d'Azur
94	Corse

Requêtes SQL

```
SELECT * FROM 'jointure departement-region' WHERE id_region=75
```

id_departement	id_region
16	75
17	75
19	75
23	75
24	75
33	75
40	75
47	75
64	75
79	75
86	75
87	7

Requêtes SQL

Des tables sont ajoutées afin de gérer les régions et les départements.

```
SELECT v.ville_nom_reel, d.departement_nom, v.ville_population_2010
FROM villes_france_free v
JOIN departement d ON d.departement_code=v.ville_departement
WHERE v.ville_population_2010>400000
ORDER BY v.ville_population_2010 DESC
```

ville_nom_reel	departement_nom	ville_population_2010
Paris	Paris	2243833
Marseille	Bouches-du-Rhône	850726
Lyon	Rhône	484344
Toulouse	Haute-Garonne	441802

Requêtes SQL

```
SELECT v.ville_nom_reel AS Ville, d.departement_nom AS Département,  
r.NCCENR AS Région, v.ville_population_2010 AS Population  
FROM villes_france_free v  
JOIN departement d JOIN regions r JOIN jointure_departement_region j  
  ON d.departement_code=v.ville_departement  
  AND d.departement_code=j.id_departement  
  AND r.Id_region=j.id_region  
WHERE v.ville_population_2010>400000  
ORDER BY v.ville_population_2010 DESC
```

Ville	Département	Région	Population
Paris	Paris	Ile de France	2243833
Marseille	Bouches-du-Rhône	Provence-Alpes-Côte d'Azur	850726
Lyon	Rhône	Auvergne-Rhône-Alpes	484344
Toulouse	Haute-Garonne	Languedoc-Roussillon-Midi-Pyrénées	441802

Requêtes SQL

Modification d'une table (gestion de la LV1).

Insérer une ligne.

```
INSERT INTO lv1 VALUES (1, 'Anglais', 'B302')
```

Modifier une ligne.

```
UPDATE lv1 SET salle='B306')
```

Supprimer une ligne.

```
DELETE FROM lv1 WHERE Id=1
```

Bibliographie

- Cours de bases de données: Philippe Rigaux, 13 juin 2001,
- Liste des villes de France en SQL: <http://sql.sh/736-base-donnees-villes-francaises>.