Informatique
TP6 : Implémentations d'algos numériques
CPP 2A

CPP

Septembre - octobre 2015

1 Pivot de Gauss

Pour commencer, téléchargez le fichier `naive_gauss.py` depuis la page web du cours. Créez un fichier `test_gauss.py` avec le contenu suivant dans le même répertoire :

```
import naive_gauss as g
import numpy as np
```

Vérifiez que le fichier s'exécute correctement en Python. Pour l'instant, il n'affiche rien.

Exercice 1 *À l'aide de la fonction* `g.solve` *(la fonction* `solve` *définie dans* ***`naive_gauss.py`****), résolvez le système suivant :*

$$\begin{cases} x + y = 3 \\ x - y = -1 \end{cases}$$

Bien sûr, l'algorithme du pivot de Gauss est déjà implémenté dans la bibliothèque NumPy : c'est la fonction `np.linalg.solve`*. Comparez les résultats obtenus par ces deux fonctions en utilisant la fonction* `g.affiche_ecart`*. Normalement, les deux résultats devraient être identiques.*

Nous allons maintenant voir le comportement de notre algorithme sur le système :

$$\begin{cases} x + \frac{1}{4}y = 0 \\ x + \frac{5}{4}y + 12z = 0 \\ x + \frac{1}{3}y + z + t = 10^{16} \\ x + \frac{5}{4}y + 13z + t = 0 \end{cases}$$

La solution analytique de ce système est : $10^{16}.(-3, 12, -1, 1)$. On peut aisément le vérifier de tête.

Exercice 2

1. *Reprendre l'exercice précédent avec le système ci-dessus.*
2. *Quelle fonction (parmi* `g.solve` *et* `np.linalg.solve`*) donne le résultat le plus proche de la solution analytique ? Utilisez* `g.affiche_ecart` *pour le trouver.*
3. *Comparer les affichages du résultat de* `np.linalg.solve` *et de la solution analytique. Comment expliquer que la différence de ces résultats soient non nulle, alors que leur affichage est identique ?*

Le problème de précision sur `g.solve` vient de l'algorithme de choix de pivot. À un certain moment, un coefficient mathématiquement nul, mais ayant à l'exécution une valeur flottante très petite, est sélectionné comme pivot : ceci aboutit à des grosses erreurs de calculs.

Exercice 3 *Pour mettre ce phénomène en évidence, on demande de modifier la fonction* ***`gauss`*** *de* ***`naive_gauss.py`****, de manière à ce qu'elle affiche :*

- *les* `A` *et* `b` *initiaux*
- *puis, pour après chaque sélection de pivot, la valeur de ce pivot et les* `A` *et* `b` *résultants de l'application des transvections sur ce pivot.*

Pour rendre l'affichage plus interactif, on pourra utiliser l'instruction suivante avant chaque sélection de pivot :

```
input("Appuyez sur une touche pour continuer...")
```

En vérifiant les calculs affichés à la main, indiquez quel pivot "quasi-nul" est sélectionné. Quelles conséquences cela a-t-il sur la suite des calculs ?

Exercice 4 *Modifier l'algorithme en appliquant la méthode du "pivot partiel" vue en cours : on sélectionne la ligne contenant le pivot le plus grand en valeur absolue (essayez de retrouver une solution similaire à celle du cours, sans la regarder).*

- *Vérifier ce nouvel algorithme en suivant son exécution pas-à-pas comme dans l'exercice précédent ;*
- *Comparer le résultat avec celui de* `np.linalg.solve`*.*

En conclusion, même si l'algorithme obtenu donne ici un résultat identique à celui de `np.linalg.solve`, il est en général préférable d'utiliser cette dernière fonction, car elle est plus rapide.

2 Résolution d'équations différentielles

Dans cette partie, nous allons avoir besoin d'écrire des fonctions paramétrées par d'autres fonctions. L'exercice préliminaire suivant montre comment cela fonctionne en Python.

Exercice 5 *Écrire une fonction* `it(f, n, x)` *qui pour tout entier naturel n calcule $f^n(x)$ défini par récurrence sur n avec $f^0(x) = x$ et $f^{n+1}(x) = f(f^n(x))$.*

```
def it(f, x, n):
    r = x
    # à compléter
    return r
```

NB *dans le corps de la fonction* `it`*, le paramètre* `f` *s'utilise comme une fonction ordinaire. Par exemple, on peut faire le calcul "*`r = f(x)`*".*

Vérifier que "`it(succ, 10, 3) == 13`*" et "*`it(double, 10, 3) == 3 * (2 ** 10)`*" où* `succ` *et* `double` *sont définis par :*

```
def succ(x):
    return x+1

def double(x):
    return it(succ, x, x)
```

2.1 Introduction à la résolution d'équation différentielle ordinaire (EDO)

On considère l'équation différentielle ordinaire (EDO) du premier ordre suivante

$$y'(x) = f(y(x), x), \quad x \in [a, b].$$

Si l'on se donne en plus une condition initiale de la forme $y(a) = y_0$, on obtient un *problème de Cauchy*. Sous certaines hypothèses de régularité sur la fonction f [1], on sait qu'il existe une unique solution y sur $[a, b]$. Même si dans certains cas (*e.g.* f linéaire à coefficients constants) on connaît explicitement cette solution, il est la plupart du temps impossible de l'écrire. On utilise alors des méthodes permettant d'approcher cette solution en calculant des approximations de proches en proches.

L'intervalle $[a, b]$ est découpé en N intervalles de taille $h = \frac{b-a}{N}$ et la solution y est approchée aux points $x_n = a + nh$, $n = 0, ..., N$. On note $y_n \approx y(x_n)$ l'approximation de y au point x_n.

2.2 Méthode d'Euler

Dans la suite, on se propose d'étudier la *méthode d'Euler* qui consiste à approcher la solution en utilisant la formule de la tangente

$$y(x + h) \approx y(x) + hy'(x) \approx y(x) + hf(y(x), x).$$

Concrètement, on calcule la suite $(y_n)_n$ à partir de la valeur initiale y_0 en utilisant la récurrence $y_{n+1} = y_n + hf(y_n, x_n)$.

1. f globalement lipschitzienne par rapport à y.

Exercice 6 (Implémentation) *Dans un fichier nommé* `edo.py`*, écrire une fonction* `euler(f, a, b, y0, N)` *qui retourne la liste des* y_n*,* $n = 0, ..., N$ *approchant la solution de notre EDO par la méthode d'Euler.*

Exercice 7 (Tests) *Quelle est la solution attendue pour* $f(y, x) = \lambda y$ *et* $y(0) = 1$ *? Utiliser ce résultat pour tester l'implémentation de l'exercice précédent : tracer la solution exacte et la solution approchée avec Matplotlib. Quelle est l'influence du paramètre* N *?*

2.3 Comparaison avec SciPy (en bonus)

Le module SciPy de Python est consacré au calcul scientifique avancé. En particulier, en utilisant la fonction `scipy.integrate.odeint`, il est possible d'approcher la solution d'une EDO grâce à des méthodes plus performantes (mais bien plus complexes aussi) que la méthode d'Euler. Il s'utilise comme ceci :

```
import scipy.integrate as scint
import numpy as np
import matplotlib.pyplot as plt


def f(y, x):
    return np.cos(10 * y) * np.cos(x)  # f(y(x), x) = cos(10y(x)) cos(x)

x = np.linspace(0, 10, 100)
y = scint.odeint(f, 0, x)  # f, y_0, points ou la solution est approchee

plt.plot(x, y)
plt.show()
```

Exercice 8 (Comparaison avec SciPy) *Utiliser SciPy ainsi que la méthode d'Euler pour approcher la solution du problème de Cauchy associée à* $f(y, x) = \cos(10y)\cos(x)$ *et* $y(0) = 0$*. On prendra soin d'utiliser de même nombre de pas* N *dans les deux méthodes. Comparer visuellement les deux solutions à l'aide de Matplotlib.*

Correction exercice 1 *Voir listing* `test_gauss.py` *ci-dessous.*

Correction exercice 2

1. *Voir listing* `test_gauss.py` *ci-dessous.*
2. *Pour* `g.solve`*, sur la plupart des composantes, seul le premier chiffre du résultat est correct ! Au contraire, pour* `np.linalg.solve`*, sur chaque composante, on a une erreur relative légèrement inférieure à* 10^{-15}*, ce qui correspond aux erreurs d'arrondis sur les flottants (52 bits significatifs).*
3. *L'affichage du résultat* `np.linalg.solve` *est identique à celui de la solution analytique, alors que leur différence en flottant est non nulle. Cela vient du fait que l'affichage se fait en décimal, alors que les valeurs calculées sont en binaire dans la mémoire de l'ordinateur. L'affichage fait un arrondi et n'affiche pas exactement la valeur en mémoire...*

Correction exercice 3 *Voir listing* `test_gauss.py` *ci-dessous pour l'implémentation des affichages.*
À l'issue des transvection sur le deuxième pivot, on voit :

```
A =
[[  1.00000000e+00   2.50000000e-01   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   1.20000000e+01   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   2.22044605e-16   1.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   1.00000000e+00   1.00000000e+00]]
```

La valeur `2.22044605e-16` *est le résultat d'une erreur d'arrondi. Mathématiquement, on aurait du obtenir la valeur 0. C'est cette valeur qui sera choisie comme pivot suivant. On va donc* diviser *par cette valeur, ce qui donne les coefficients énormes* `-4.50359963e+15` *dans* `A` *et* `-4.50359963e+31` *dans* `b` *:*

```
A =
[[  1.00000000e+00   2.50000000e-01   0.00000000e+00   0.00000000e+00]
 [  0.00000000e+00   1.00000000e+00   1.20000000e+01   0.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   2.22044605e-16   1.00000000e+00]
 [  0.00000000e+00   0.00000000e+00   0.00000000e+00  -4.50359963e+15]]
b =  [  0.00000000e+00   0.00000000e+00   1.00000000e+16  -4.50359963e+31]
```

Correction exercice 4 *Voir listing* `test_gauss.py` *ci-dessous.*
La triangulation de la matrice donne ici des coefficients qui s'affichent de manière identique à ceux qu'on aurait calculé à la main (pas d'anomalie visible).

```
A =
[[  1.     0.25   0.     0.  ]
 [  0.     1.    12.     0.  ]
 [  0.     0.     1.     1.  ]
 [  0.     0.     0.     1.  ]]
b =  [  0.00000000e+00   0.00000000e+00   0.00000000e+00   1.00000000e+16]
```

Le résultat est vraiment identique à celui de `np.linalg.solve`*.*

Fichier `text_gauss.py`

```
import naive_gauss as g
import numpy as np

print("---EXO 1---")
a = np.array([[1, 1],
              [1, -1]])
b = np.array([3, -1])
g.affiche_ecart("naive", g.solve(a, b), "linalg",  np.linalg.solve(a, b))

print("\n---EXO 2---")
a = np.array([[1, 1/4,  0, 0],
              [1, 5/4, 12, 0],
              [1, 1/3,  1, 1],
              [1, 5/4, 13, 1]])
b = np.array([0, 0, 1e16, 0])
x_naive = g.solve(a, b)
x_linalg = np.linalg.solve(a, b)
g.affiche_ecart("naive", x_naive, "linalg",  x_linalg)
```

```
x_analytique = np.array([-3e16, 1.2e17, -1e16, 1e16])
g.affiche_ecart("naive", x_naive, "analytique",  x_analytique)
g.affiche_ecart("linalg",  x_linalg, "analytique",  x_analytique)


print("\n---EXO 3---")
def trace_gauss(A0, b0):
    A = A0.copy() # pour ne pas détruire A
    b = b0.copy()
    n = len(b)  # on pourrait aussi verifier que a a les bonnes dimensions
    print("A orig = ")
    print(A)
    print("b orig = ", b)
    for i in range(n - 1):
        input("Appuyez sur une touche pour continuer...")
        # choix du pivot
        ipiv = g.pivot_index(A, i)
        # echanges
        if ipiv != i:
            g.swap_lines(A, i, ipiv)
            tmp = b[ipiv]
            b[ipiv] = b[i]
            b[i] = tmp
        # pivotage
        for k in range(i + 1, n):
            factor = -A[k, i] / A[i, i]
            g.transvection_lines(A, i, k, factor)
            b[k] = b[k] + b[i] * factor
        print("piv =", A[i, i], " ==> A = ")
        print(A)
        print("b = ", b)
    return [A, b]

g.gauss = trace_gauss
g.solve(a, b)

print("\n---EXO 4---")
def pivot_index(A, i):
    n = A.shape[0]  # nombre de lignes
    j = i
    for k in range(i+1,n):
        if abs(A[k,i]) > abs(A[j,i]):
            j = k
    return j

g.pivot_index = pivot_index
x_fix = g.solve(a, b)
g.affiche_ecart("fix", x_fix, "linalg",  x_linalg)
g.affiche_ecart("fix", x_fix, "analytique",  x_analytique)
```

Correction exercice 5

```
def it(f, n, x):
    r = x
    for i in range(n):
        r = f(r)
    return r
```

Correction exercice 6

```
# fichier edo.py

def euler(f, a, b, y0, N):
    y = y0
    x = a
    h = (b - a) / N
    res = [y]
    for i in range(N):
        y = y + h * f(y, x)
```

```
        x = x + h
        res.append(y)
    return res
```

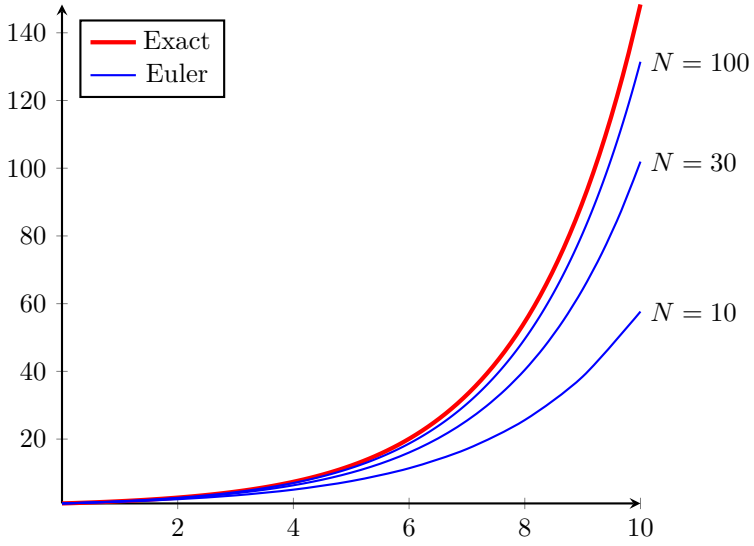
Correction exercice 7 *La solution est* $y(x) = e^{\lambda x}$.

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

def f(y, x):
    return 0.5 * y  # on choisit λ = 1/2

N = 100
a = 0
b = 10
x = np.linspace(a, b, N + 1)
y = edo.euler(f, a, b, 1, N)

plt.plot(x, y)
# truc : y(x) = e^{λx} = e^{f(x,x)}
plt.plot(x, np.exp(f(x, x)))
plt.show()
```



Correction exercice 8

```
import scipy.integrate as scint
import edo
import numpy as np
import matplotlib.pyplot as plt

def f(y, x):
    return np.cos(10 * y) * np.cos(x)

a = 0
b = 10
N = 150
y0 = 0

x = np.linspace(a, b, N + 1)
y_scipy = scint.odeint(f, y0, x)
y_euler = edo.euler(f, a, b, y0, N)

plt.plot(x, y_scipy)
plt.plot(x, y_euler)
plt.show()
```