Corrigé

Résolution numérique de l'équation de Poisson

Durant ce TP nous aurons besoin des importations suivantes :

```
import numpy as np
from numpy.linalg import solve, norm
import matplotlib.pyplot as plt
```

1. Déformation d'une barre élastique

Question 1.

a) D'après la formule de Taylor-Young, $u(x+h) = u(x) + hu'(x) + \frac{h^2}{2}u''(x) + o(h)$ et $u(x-h) = u(x) - hu'(x) + \frac{h^2}{2}u''(x) + o(h)$ donc $u(x+h) + u(x-h) = 2u(x) + h^2u''(x) + o(h^2)$, soit $u''(x) = \frac{1}{h^2}\Big(u(x-h) - 2u(x) + u(x+h)\Big) + o(1)$.

b) Posons :

$$A = \begin{pmatrix} -2 & 1 & 0 & \cdots & 0 \\ 1 & -2 & 1 & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & 1 & -2 & 1 \\ 0 & \cdots & 0 & 1 & -2 \end{pmatrix} \qquad u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{pmatrix} \qquad v = h^2 \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{pmatrix}$$

alors, compte tenu des conditions de Dirichlet sur le bord ($u_0 = u_{n+1} = 0$), le système est équivalent à l'équation matricielle $Au = v$.

Question 2.

a) On définit la fonction suivante :

```
def poisson1(f, n):
    h = 1 / (n+1)
    f = np.vectorize(f)
    x = np.linspace(0, 1, n+2)
    A = np.diag([-2]*n) + np.diag([1]*(n-1), -1) + np.diag([1]*(n-1), 1)
    v = h * h * f(x[1:n+1])
    u = np.zeros(n+2, dtype=float)
    u[1:n+1] = solve(A, v)
    return x, u
```

J'utilise deux fonctions de la bibliothèque `numpy` :

- `diag(lst, k=0)` renvoie une matrice dont la diagonale repérée par l'entier k est égale à la liste `lst` ;
- `vectorize(f)` renvoie la version vectorielle d'une fonction, ce qui permet ensuite de l'appliquer directement à chacune des composantes d'un vecteur (ou d'une matrice).

b) Lorsque $f = 1$ l'équation de Poisson se réduit à : $u''(x) = 1$ soit, compte tenu des conditions aux limites : $u(x) = \frac{x(x-1)}{2}$. Ceci permet de représenter aussi bien la solution approchée de la solution exacte :

```
x, u = poisson1(lambda t: 1, 100)
v = [t*(t-1)/2 for t in x]

plt.subplot(211)
plt.plot(x, u)
plt.axis('equal')
plt.title('Solution approchée')

plt.subplot(212)
plt.plot(x, v)
plt.axis('equal')
plt.title('Solution exacte')

plt.show()
```

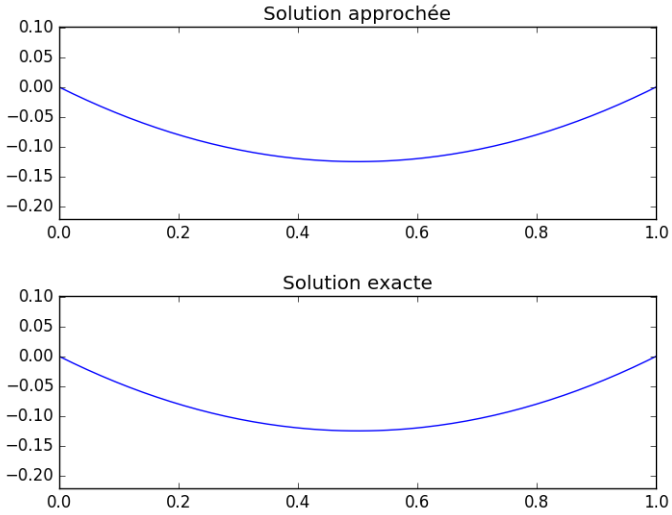


FIGURE 1 – Barre élastique soumise à un champ de gravitation uniforme.

c) L'erreur commise pour $n = 100$ est très faible :

```
>>> print(norm(u - v, np.inf))
1.01307850997e-15
```

2. déformation d'une membrane

Question 3. Nous allons approcher le laplacien de u en (x,y) par l'expression :

$$\frac{1}{h^2}\Big(u(x-h,y)-2u(x,y)+u(x+h,y)\Big)+\frac{1}{h^2}\Big(u(x,y-h)-2u(x,y)+u(x,y+h)\Big) = \frac{1}{h^2}\Big(u(x-h,y)+u(x+h)-4u(x,y)+u(x,y-h)+u(x,y+h)\Big)$$

où h désigne le pas de la subdivision choisie. Autrement dit nous cherchons à résoudre le système d'équations linéaires :

$$\forall i,j \in [\![1,n+1]\!], \qquad \frac{1}{h^2}\Big(u_{i-1,j}+u_{i+1,j}-4u_{i,j}+u_{i,j-1}+u_{i,j+1}\Big) = f(x_i,y_j)$$

système équivalent, compte tenu des conditions sur le bord, à la relation matricielle $Au = v$.

Question 4. On en déduit la fonction :

```
def poisson2(f, n):
    h = 1/(n+1)
    x = np.linspace(0, 1, n+2)
    A = np.zeros((n*n, n*n), dtype=float)
    for i in range(n*n):
        A[i, i] = -4
    for i in range(n):
        for j in range(n-1):
            A[n*i+j, n*i+j+1] = 1
            A[n*i+j+1, n*i+j] = 1
    for i in range(n-1):
        for j in range(n):
            A[n*(i+1)+j, n*i+j] = 1
            A[n*i+j, n*(i+1)+j] = 1
    v = np.zeros(n*n, dtype=float)
    for i in range(n):
        for j in range(n):
            v[n*i+j] = h*h*f(x[i+1], x[j+1])
    return solve(A, v).reshape((n, n))
```

Question 5.

a) On obtient la représentation de la membrane soumise à un champ de pesanteur uniforme en réalisant le script suivante :

```
def f_pesanteur(x, y):
    return 1

plt.matshow(poisson2(f_pesanteur, 50))
plt.show()
```

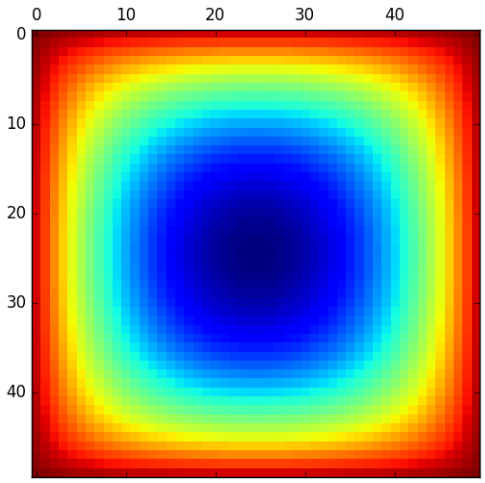


FIGURE 2 – Membrane soumise à un champ de pesanteur uniforme.

b) La force exercée par la pression de cinq doigts se modélise ainsi :

```
def doigt(x, y, x0, y0):
    r = .05
    if (x-x0)**2 + (y-y0)**2 < r**2:
        return 1/r**2
    else:
        return 0

def f_doigts(x, y):
    s = 0
    for (x0, y0) in [(.74, .12), (.3, .22), (.21, .44), (.24, .66), (.35, .85)]:
        s += doigt(x, y, x0, y0)
    return s

plt.matshow(poisson2(f_doigts, 50))
plt.show()
```

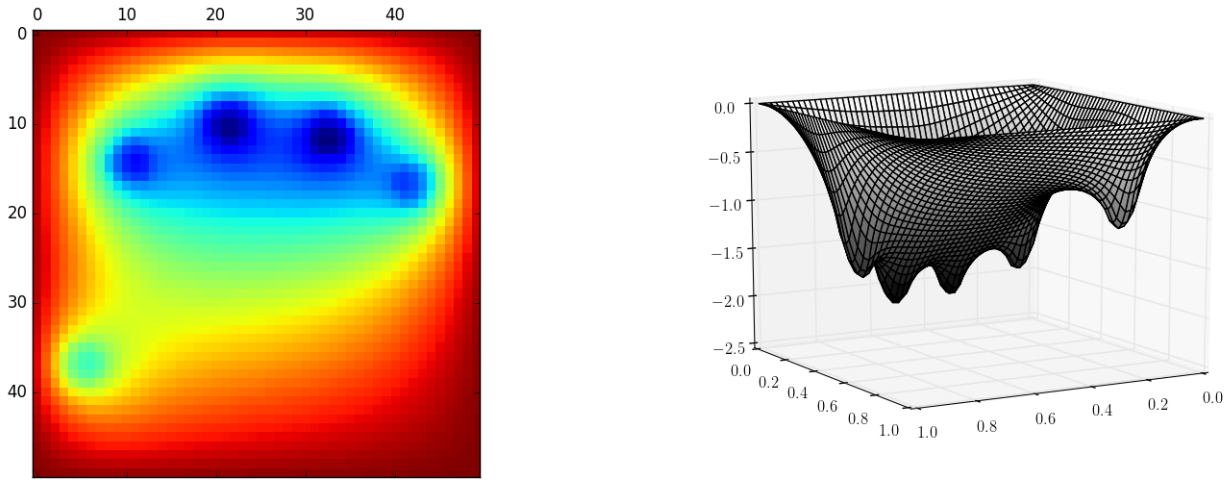


FIGURE 3 – Membrane soumise à la pression de cinq doigts.

3. Méthode de relaxation

Question 6.

a) On définit la fonction suivante :

```
def itere(u, x, f, n, h, omega):
    uu = np.zeros_like(u)
    for i in range(1, n+1):
        for j in range(1, n+1):
            uu[i, j] = (1 - omega) * u[i, j] + omega / 4 * (uu[i, j-1] + uu[i-1, j] + u[i+1, j] + u[i,
    return uu
```

b) Puis :

```python
def poisson3(f, n, epsilon, kmax = 500):
    h = 1/(n+1)
    omega = 2 / (1 + np.sin(np.pi * h))
    x = np.linspace(0, 1, n+2)
    u = np.zeros((n+2, n+2), dtype=float)
    for k in range(kmax):
        uu = itere(u, x, f, n, h, omega)
        if norm(uu - u, np.inf) / norm(uu, np.inf) < epsilon:
            return uu
        u = uu
    return None
```

Le paramètre `kmax` permet d'indiquer un rang au delà duquel l'itération est stoppée.

c) On peut maintenant réaliser le script :

```
plt.imshow(poisson3(f_doigts, 100, .01))
plt.show()
```

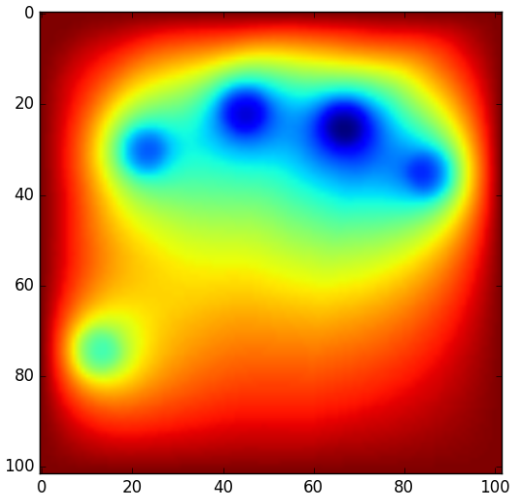
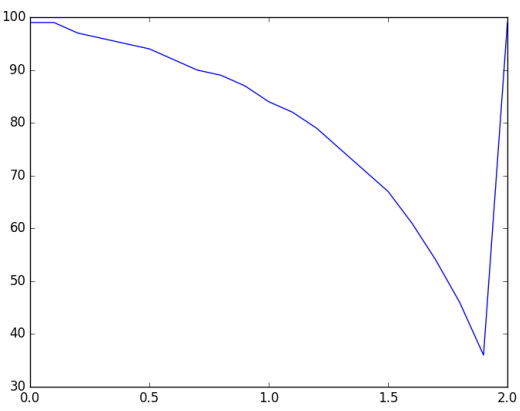


FIGURE 4 – Membrane soumise à la pression de cinq doigts.

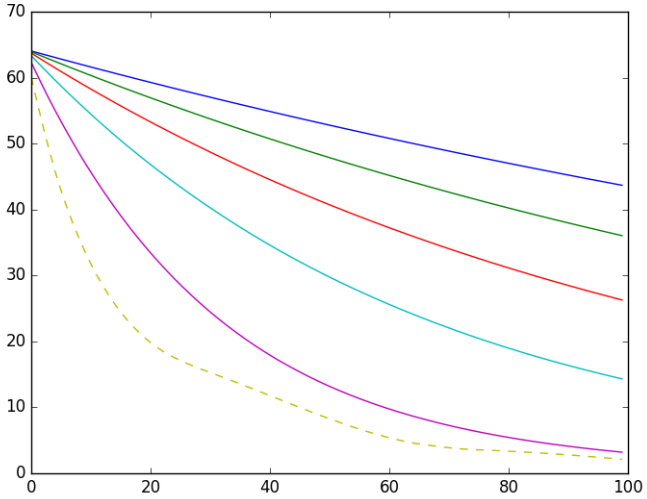
Question 7.

a) On calcule sans peine $f = -8\pi^2 u$. L'influence du facteur ω est visible sur le graphe ci-dessous, qui représente le nombre d'étapes nécessaires pour atteindre la condition d'arrêt en fonction de ω :



La meilleure valeur pour ω observée sur ce graphe : $\omega = 0,9$ est très proche de la valeur théorique $\dfrac{1}{1+\sin(\pi h)} \approx 1,94$.

b) Enfin, on observe pour différentes valeurs de ω la vitesse de convergence vers la solution exacte :



(en pointillés, la solution correspondant au facteur optimal ω).
On peut observer que plus ω est proche de sa valeur optimale, plus la vitesse de convergence est importante.