Équation de Poisson : programme Python

1. Introduction

Ce document présente une interface Python pour le programme C présenté dans Équation de Poisson : programme C.

Le module (pypoisson) permet d'effectuer la résolution numérique de l'équation de Poisson 2D (applications en électromagnétisme et en thermodynamique) par la méthode des différences finies :

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = s(x, y)$$

où u(x,y) est la fonction inconnue et s(x,y) la fonction source, éventuellement nulle (équation de Laplace).

Pour la méthode de résolution itérative, le module comporte un solveur tournant sur le processeur central (CPU), et un solveur utilisant les capacités de calculs parallèles des cartes graphiques (GPU), via l'interface de programmation OpenCL.

2. Installation du module

La bibliothèque pthreads est utilisée pour le multi-threading sur le processeur central (CPU). Cette bibliothèque est disponible en standard sur les distributions Linux. Pour win32, il faudra installer la version pthreads-win32.

3. Description de l'interface

3.a. Définition du maillage

La classe principale est `Poisson`. Elle se trouve dans le module `poisson.main`. Son constructeur effectue l'allocation de l'espace mémoire pour le maillage :

```
object = Poisson.Poisson(pw,ph,width)
```

Constructeur : définition d'un maillage 2D

- ▷ `pw, ph` : puissances définissant les nombres de points sur les deux dimensions du maillage. Les nombres de points sont 2^{pw} et 2^{ph}
- ▷ `width` : largeur du domaine, utilisé pour le calcul de la taille des mailles

Remarque : une seule instance de cette classe est utilisable, car le programme C sous-jacent ne comporte qu'une instance des données.

```
Poisson.close()
```

Libération de l'espace mémoire réservée pour le maillage. À appeler avant d'ouvrir une nouvelle instance de la classe.

3.b. Équation en coordonnées cartésiennes.

En coordonnées cartésiennes, les mailles sont carrées, de côté h. Le domaine d'intégration est donc un rectangle de taille $h2^{pw}$ par $h2^{ph}$. La fonction suivante permet de discrétiser le laplacien en coordonnées cartésiennes.

```
Poisson.laplacien()
```

3.c. Conditions limites en coordonnées cartésiennes.

Les deux fonctions suivantes permettent de définir les conditions limites sur les bords du domaine rectangulaire.

```
Poisson.dirichlet_borders(value)
```

Condition limite de Dirichlet sur les bords du domaine rectangulaire.

▷ `value` : valeur sur les bords de u

```
Poisson.neumann_borders(source,derivX1,derivX2,derivY1,derivY2)
```

Condition limite de Neumann sur les bords du domaine, c'est-à-dire valeur de la dérivée selon la normale de chaque bord.

▷ `source` : valeur de s(x,y) au voisinage du bord (0 pour l'équation de Laplace)

▷ `derivX1` : valeur de la dérivée par rapport à x à imposer sur le bord gauche

▷ `derivX1` : valeur de la dérivée par rapport à x à imposer sur le bord droit

▷ `derivY1` : valeur de la dérivée par rapport à y à imposer sur le bord inférieur

▷ `derivY2` : valeur de la dérivée par rapport à y à imposer sur le bord inférieur

3.d. Polygones en coordonnées cartésiennes

On peut définir un polygone dont les arêtes sont parallèles aux axes x et y. Pour cela, il faut créer une instance de la classe `Polygon`, qui se trouve dans le module `poisson.main`. Les coordonnées des sommets du polygone peuvent être définies sur une maille réduite. La maille réduite facilite la définition des polygones ; de plus, elle sera nécessaire pour la méthode multigrille. Pour cela, on définit un exposant p de réduction de la maille. La taille de la maille réduite est 2^{pw-p} par 2^{ph-p}. L'utilisateur doit calculer cette taille pour savoir comment définir les polygones sur le domaine.

```
Polygon.Polygon(poisson,p,point0)
```

Construction d'un polygone à partir d'un point initial.

▷ `poisson` : instance de la classe Poisson utilisée pour le calcul

▷ `p` : exposant de réduction de maille, qui doit être inférieur à pw et à ph

▷ `point0` : point initial, sous la forme [x,y] (coordonnées sur la maille réduite)

La fonction suivante permet d'ajouter un sommet au polygone. Le sommet est défini par sa direction par rapport au dernier sommet et par sa distance. L'ajout d'un sommet définit une arête sur laquelle on applique une condition limite.

```
Polygon.add_vertex(direction,length,limit,source,value,derivX,derivY)
```

Ajout d'un sommet au polygone.

- ▷ `direction` : direction du sommet par rapport au précédent, au choix :[0,1], [0,-1], [1,0] ou [-1,0]
- ▷ `length` : distance par rapport au sommet précédent (longueur de l'arête), définie sur la maille réduite
- ▷ `limit` : type de condition limite : poisson.main.LIMIT$_SOURCE$, $poisson.main.LIMIT_DIRICHL$ $valeurdes(x,y)auvoisinagedubord(pourLIMIT_SOURCEetLIMIT_NEUMANN)$
- ▷ `value` : valeur à imposer pour LIMIT$_D$$IRICHLET$`derivX` : $valeurdeladriveparrapportximposersurles$
- ▷ `derivY` : valeur de la dérivée par rapport à y à imposer sur les bords parallèles à x, pour LIMIT$_N$$EUMANN$

Le polygone peut être ouvert, auquel cas il peut recevoir une condition limite de source ou de Dirichlet. Pour obtenir un polygone fermé, il faut ajouter le point initial comme dernier sommet.

Pour la condition limite de Neumann, il faut savoir où se trouve l'intérieur du domaine (dans lequel se fait le calcul) par rapport au polygone. Cela est fixé par la convention suivante : lorsqu'on est orienté dans le sens de définition du polygone, le domaine de calcul se trouve à gauche. Par exemple, dans un problème d'électrostatique, si l'on veut définir un conducteur plein dans le domaine, il faudra définir le polygone dans le sens horaire.

Les sommets d'un polygone recoivent un traitement spécial lors de la discrétisation de la condition de Neumann. Pour cette raison, le polygone doit être fermé pour recevoir une condition de Neumann.

3.e. Itérations de Gauss-Seidel sur CPU

Le système linéaire obtenu par discrétisation des équations (équation de Poisson et conditions limites) est résolu par la méthode d'itération de Gauss-Seidel. Ce paragraphe présente les fonctions de la classe `Poisson` qui permettent d'effectuer ce calcul sur le processeur central (CPU).

```
Poisson.iterations(niter)
```

Itérations de Gauss-Seidel.

- ▷ `niter` : nombre d'itérations

Pour contrôler la convergence, la fonctions suivante effectue des blocs d'itérations et calcule la norme de la matrice U à chaque bloc.

```
[ni,norm]=Poisson.iterations_norm(niter,nblock)
```

Itérations avec calcul de la norme de la matrice des valeurs de u.

▷ `niter` : nombre d'itérations dans un bloc

▷ `nblock` : nombre de blocs

▷ `ni` : liste des nombres d'itérations

▷ `norm` : liste des normes

3.f. Itérations de Gauss-Seidel sur GPU

Les itérations peuvent être effectuées sur processeur graphique (GPU). Une plateforme openCL doit être installée.

La première fonction permet d'afficher les plateformes openCL présentes sur le système, et pour chaque plateforme les périphériques associés. Dans les cas courants (une seule carte graphique), il y aura une seule plateforme avec un seul périphérique. Les plateformes et les périphériques sont numérotés à partir de 0.

```
Poisson.platforms()
```

Affiche sur la console les plateformes openCL et leurs périphériques associés.

Par défaut, la plateforme 0 et le périphérique 0 sont sélectionnés. La fonction suivante permet de sélectionner une plateforme openCL et un périphérique :

```
Poisson.set_opencl_platform_device(platform,device)
```

Sélection d'une plateforme et d'un périphérique pour effectuer les itérations.

▷ `platform` : numéro de la plateforme, 0 pour la première

▷ `device` : numéro du périphérique, 0 pour le premier

Les deux fonctions suivantes effectuent les itérations, et sont analogues aux fonctions définies plus haut pour le CPU :

```
Poisson.opencl_iterations(niter)
```

Itérations de Gauss-Seidel sur plateforme openCL.

▷ `niter` : nombre d'itérations

```
[ni,norm]=Poisson.opencl_iterations_norm(niter,nblock)
```

Itérations avec calcul de la norme de la matrice des valeurs de u.

▷ `niter` : nombre d'itérations dans un bloc

▷ `nblock` : nombre de blocs

▷ `ni` : liste des nombres d'itérations

▷ `norm` : liste des normes

3.g. Récupération des données

La matrice U contient les valeurs de $u(x, y)$ aux points de la maille. Elle est fournie sous forme d'ue matrice numpy par la fonction suivante :

```
U=Poisson.get_array()
```

▷ `U` : tableau numpy contenant les valeurs de u

Les dérivées par rapport à x et y sont obtenues avec les deux fonctions suivantes :

```
DX=Poisson.get_derivX()
```

▷ `DX` : tableau numpy contenant les valeurs de $\frac{\partial u}{\partial x}$

```
DY=Poisson.get_derivY()
```

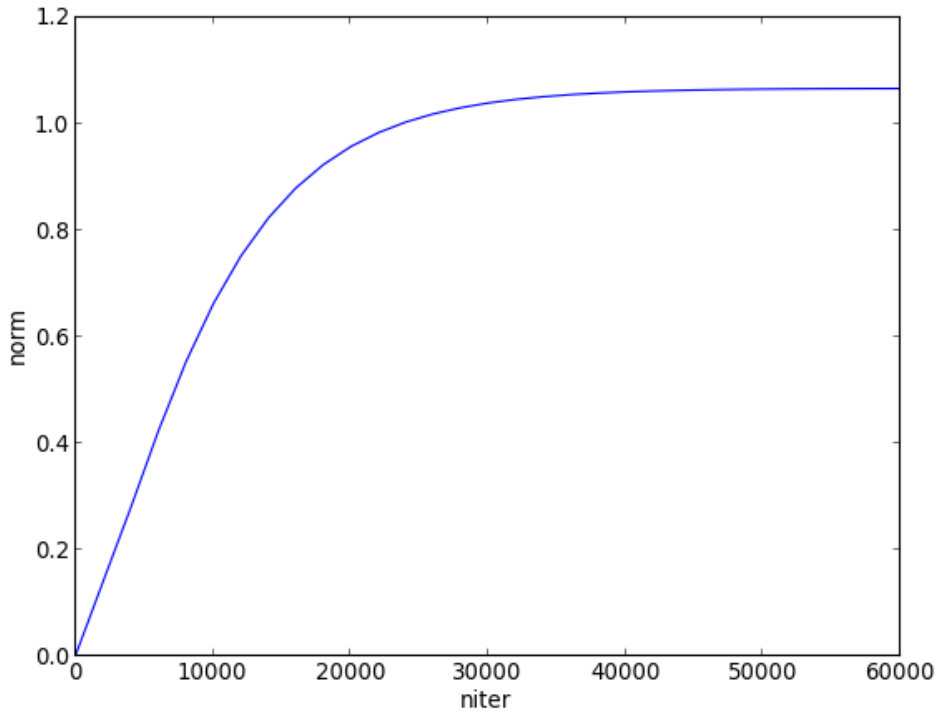
▷ `DY` : tableau numpy contenant les valeurs de $\frac{\partial u}{\partial y}$

4. Exemples

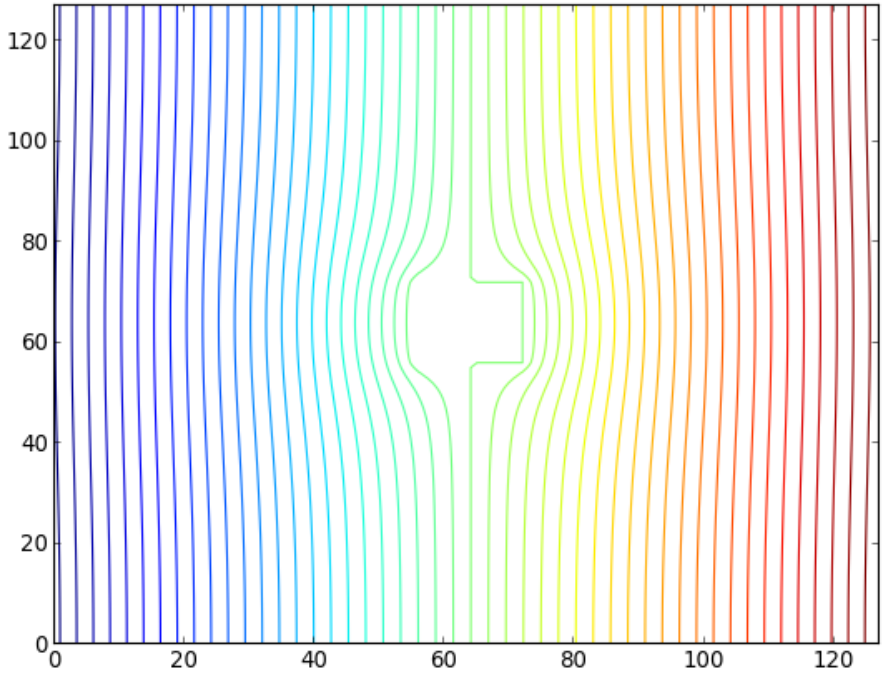
4.a. Carré conducteur dans un champ uniforme

On considère un problème d'électrostatique (équation de Laplace) avec un champ électrique sur les bords à gauche et à droite, et un conducteur carré au milieu.

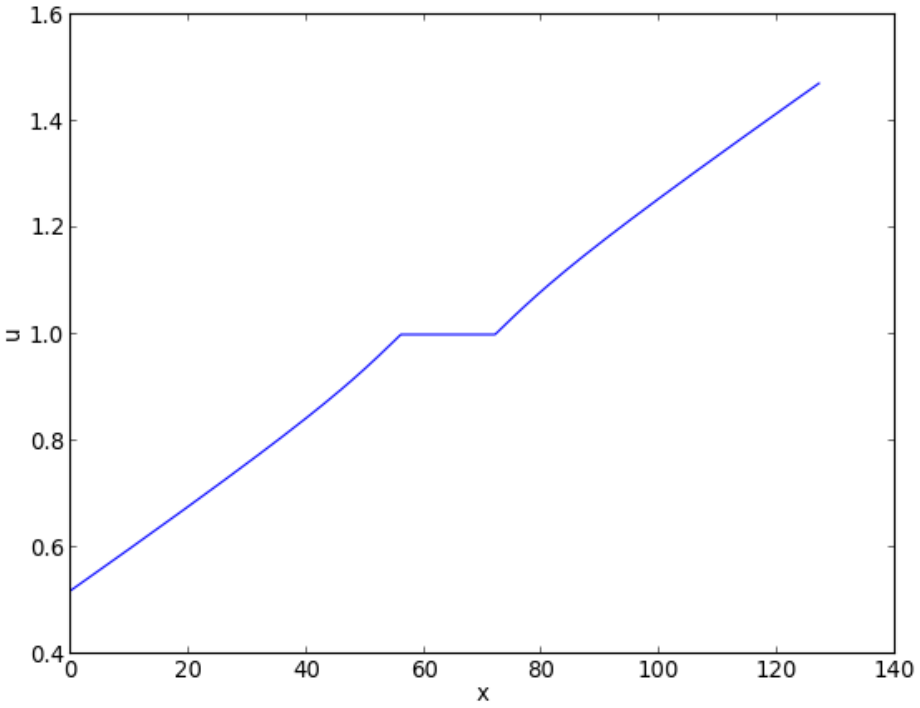
```
from pylab import *
import numpy
import poisson.main
n=7
laplace=poisson.main.Poisson(n,n,1)
laplace.laplacien()
laplace.neumann_borders(0,1,1,0,0)
p=n-4 # maille reduite 4x4
conducteur = poisson.main.Polygon(laplace,p,[7,7])
conducteur.add_vertex([0,1],2,poisson.main.LIMIT_DIRICHLET,0,1,0,0)
conducteur.add_vertex([1,0],2,poisson.main.LIMIT_DIRICHLET,0,1,0,0)
conducteur.add_vertex([0,-1],2,poisson.main.LIMIT_DIRICHLET,0,1,0,0)
conducteur.add_vertex([-1,0],2,poisson.main.LIMIT_DIRICHLET,0,1,0,0)
conducteur.apply_limit()
laplace.set_opencl_platform_device(0,0)
result=laplace.opencl_iterations_norm(2000,30)
plot(result[0],result[1])
xlabel('niter')
ylabel('norm')
```



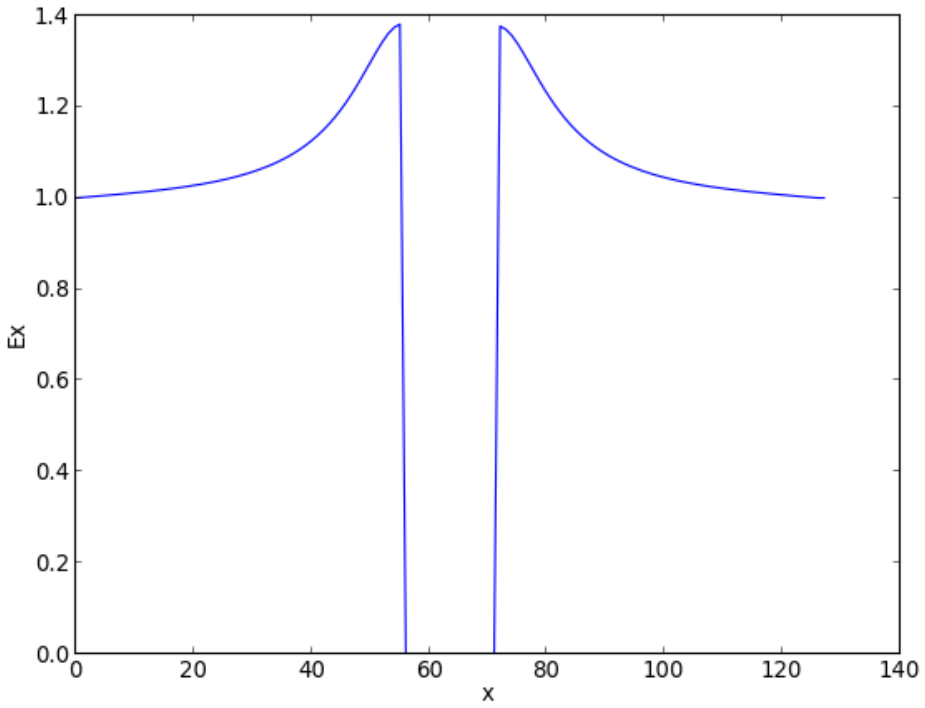

```
U=laplace.get_array()
Ex=laplace.get_derivX()
Ey=laplace.get_derivY()
laplace.close()
figure()
contour(U,50)
```




```
figure()
plot(U[math.pow(2,n-1),:])
xlabel('x')
ylabel('u')
```



```
figure()
plot(Ex[math.pow(2,n-1),:])
xlabel('x')
ylabel('Ex')
```

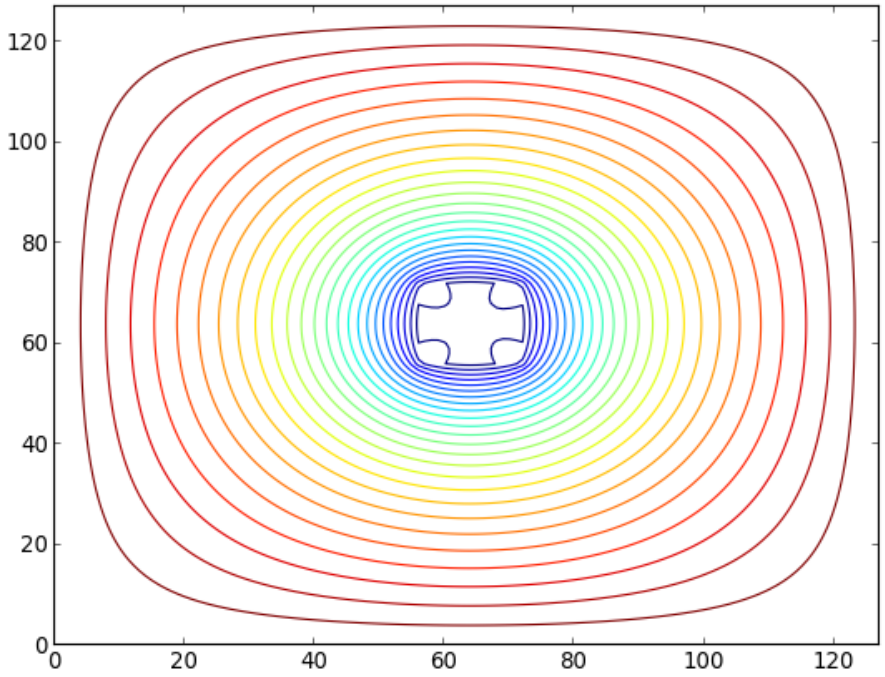


4.b. Carré chargé

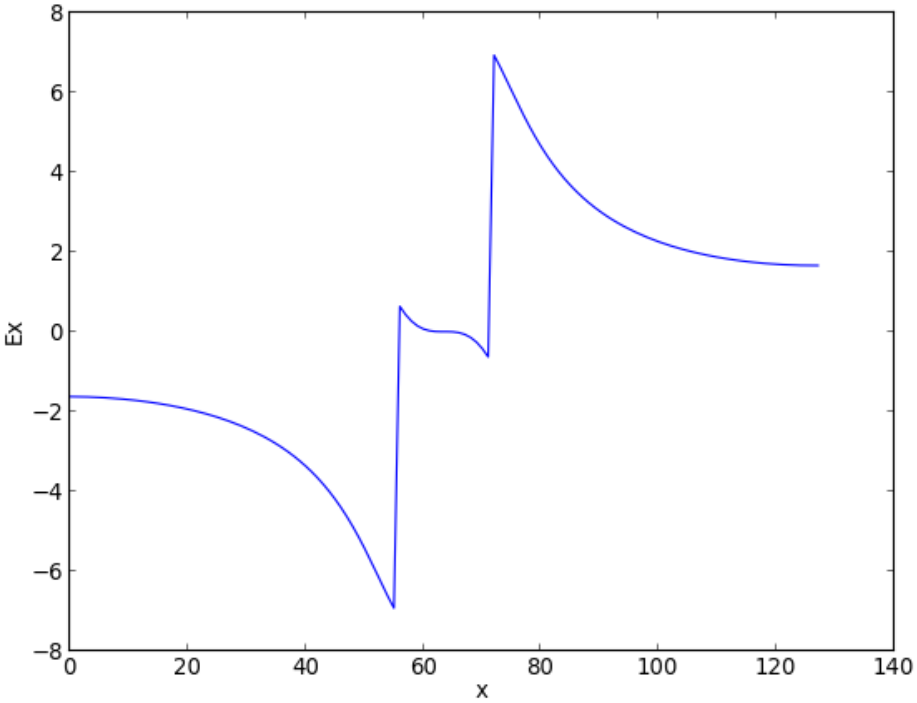
Une forme carrée placée au centre porte une charge sur son contour. Le potentiel est nul sur les bords du domaine.

```
n=7
laplace = poisson.main.Poisson(n,n,1)
laplace.laplacien()
laplace.dirichlet_borders(0.0)
p=n-4 # maille reduite 4x4
conducteur = poisson.main.Polygon(laplace,p,[7,7])
s=1000
conducteur.add_vertex([0,1],2,poisson.main.LIMIT_SOURCE,s,0,0,0)
conducteur.add_vertex([1,0],2,poisson.main.LIMIT_SOURCE,s,0,0,0)
conducteur.add_vertex([0,-1],2,poisson.main.LIMIT_SOURCE,s,0,0,0)
conducteur.add_vertex([-1,0],2,poisson.main.LIMIT_SOURCE,s,0,0,0)
conducteur.apply_limit()
laplace.set_opencl_platform_device(0,0)
result=laplace.opencl_iterations_norm(2000,30)
U=laplace.get_array()
Ex=laplace.get_derivX()
```

```
Ey=laplace.get_derivY()
laplace.close()
figure()
contour(U,30)
```




```
figure()
plot(Ex[math.pow(2,n-1),:])
xlabel('x')
ylabel('Ex')
```


8
6
4
2
0
−2
−4
−6
−8
Ex
0
20
40
60
80
100
120
140
x