GAILLOT Jean-Baptiste **M1 CHPS**

Outils pour la visualisation
Compte-rendu des TD

Tout les fichiers crées `vtk` et `hdf5` sont disponibles dans `./Donnees`. Toutes les captures d'écrans sont disponibles dans `./Captures`.

Table des matières

1 Problème de Poisson en dimension 2

Soit le problème suivant :
Trouver $u : \Omega \to \mathbb{R}$, $\Omega =]0,1[\times]0,1[$ telle que :

$$\begin{cases} -\Delta u = f & \text{sur} \quad \Omega \\ u = 0 & \text{sur} \quad \partial\Omega \end{cases}$$

1.1 Choix des données et import des modules

On importe les modules nécessaires :

```
import numpy as np
from numpy import pi, sin
from scipy import linalg
import matplotlib.pyplot as plt
import time
import h5py
from typing import Callable
```

Pour les premiers tests, on utilise $N = 3$ et $h = 1/4$. On écrit les fonctions `N_vers_h` et `h_vers_N` qui nous serviront dans de nombreuses fonctions définies plus tard.

```
N = 3
h = 1 / (N + 1)
```

```
def N_vers_h(N : int) -> float:

    h = 1 / (N + 1)

    return h
```

```
def h_vers_N(h : float) -> int:

    N = int((1 - h) / h)

    return N
```

1.2 Numérotation des noeuds

On écrit les fonctions :
`numeroter` pour avoir le tableau `Noeud` des noeuds du maillage linéarisé.
`connecter` pour avoir les voisins `Connec` de chaque noeud du maillage.

```
def numeroter(N : int) -> np.ndarray:

    Noeud = np.zeros((N * N, 2), dtype = "int")

    for i in range(1, N + 1):
        for j in range(1, N + 1):
            k = (i - 1) + N * (j - 1)
```

```
            Noeud[k, :] = [i, j]

    return Noeud
```

```
def connecter(N : int, Noeud : np.ndarray) -> np.ndarray:

    Connec = np.zeros((N * N, 5), dtype = "int")
    for i in range (N * N):
        m = 0
        x, y = Noeud[i]
        if x > 1:
            Connec[i, m + 1] = i - 1 + 1
            m += 1
        if x < N:
            Connec[i, m + 1] = i + 1 + 1
            m += 1
        if y > 1:
            Connec[i, m + 1] = i - N + 1
            m += 1
        if y < N:
            Connec[i, m + 1] = i + N + 1
            m += 1
        Connec[i, 0] = m

    return Connec
```

```
# Tests

Noeud = numeroter(N)
Connec = connecter(N, Noeud)

res = "Noeud = \n" + str(Noeud) + "\nConnec = \n" + str(Connec)

print(res)
```

```
Noeud =
[[1 1]
 [2 1]
 [3 1]
 [1 2]
 [2 2]
 [3 2]
 [1 3]
 [2 3]
 [3 3]]
Connec =
[[2 2 4 0 0]
 [3 1 3 5 0]
 [2 2 6 0 0]
 [3 5 1 7 0]
```

```
 [4 4 6 2 8]
 [3 5 3 9 0]
 [2 8 4 0 0]
 [3 7 9 5 0]
 [2 8 6 0 0]]
```

1.3 Construction de la matrice d'approximation du Laplacien

On écrit la fonction :
`construire_matrice` pour avoir la matrice A.

```python
def construire_matrice(N : int, h : float, Connec : np.array) -> np.ndarray:

    A = np.zeros((N * N, N * N))

    for k in range (N * N):
        A[k, k] = 4 / (h ** 2)
        voisins, k_p = True, 1
        while (voisins == True and k_p <= 4):
            voisin = Connec[k, k_p]
            if (voisin == 0):
                voisins = False
            else:
                A[k, voisin - 1] = - 1 / (h ** 2)
                k_p += 1

    return A
```

```python
# Test

A = construire_matrice(N, h, Connec)

res = "A =\n" + str(A)

print(res)
```

```
A =
[[ 64. -16.   0. -16.   0.   0.   0.   0.   0.]
 [-16.  64. -16.   0. -16.   0.   0.   0.   0.]
 [  0. -16.  64.   0.   0. -16.   0.   0.   0.]
 [-16.   0.   0.  64. -16.   0. -16.   0.   0.]
 [  0. -16.   0. -16.  64. -16.   0. -16.   0.]
 [  0.   0. -16.   0. -16.  64.   0.   0. -16.]
 [  0.   0.   0. -16.   0.   0.  64. -16.   0.]
 [  0.   0.   0.   0. -16.   0. -16.  64. -16.]
 [  0.   0.   0.   0.   0. -16.   0. -16.  64.]]
```

1.4 Construction du second membre

On écrit la fonction :
`construire_vecteur` pour avoir le vecteur B.

```
def construire_vecteur(N : int, f : Callable[[float], float], Noeud : np.ndarray) ->␣
↪np.ndarray:

    B = np.zeros(N * N, dtype = "float")
    h = N_vers_h(N)

    for k in range (N * N):
        x_i, y_j = Noeud[k] * h
        B[k] = f(x_i, y_j)

    return B
```

```
# Test

def f_1(x : float, y : float) -> float:

    res = sin(2 * pi * x) * sin(2 * pi * y)

    return res

B = construire_vecteur(N, f_1, Noeud)

res = "B =\n" + str(B)

print(res)
```

```
B =
[ 1.00000000e+00  1.22464680e-16 -1.00000000e+00  1.22464680e-16
  1.49975978e-32 -1.22464680e-16 -1.00000000e+00 -1.22464680e-16
  1.00000000e+00]
```

1.5 Résolution du système linéaire

On écrit la fonction :
`resoudre_systeme` pour avoir U la solution de $AU = B$.

```
def resoudre_systeme(A : np.ndarray, B : np.ndarray) -> np.ndarray:

    lu, piv = linalg.lu_factor(A)
    U = linalg.lu_solve((lu, piv), B)

    return U
```

```
# Test

U = resoudre_systeme(A, B)

res = "U =\n" + str(U)

print(res)
```

```
U =
[ 1.56250000e-02  1.85037171e-18 -1.56250000e-02  2.22395211e-18
 -3.11065919e-19 -2.25262643e-18 -1.56250000e-02 -2.63814823e-18
  1.56250000e-02]
```

1.6 Estimation de l'erreur

On écrit les fonctions :
`calculer_sol_ex` pour avoir Uex la solution réelle du problème. Commentaire : `u_sol` est la fonction qui calcule Uex pour un point du maillage.
`calculer_norme_erreur` pour avoir `erreur` la norme L^2 de l'erreur entre U et Uex.

```
def calculer_sol_ex(N : int, u_sol : Callable[[float], float], Noeud : np.ndarray) ->␣
 ↪np.ndarray:

    Uex = np.zeros(N * N, dtype = "float")
    h = N_vers_h(N)

    for k in range (N * N):
        x_i, y_j = Noeud[k] * h
        Uex[k] = u_sol(x_i, y_j)

    return Uex
```

```
def calculer_norme_erreur(h : float, U : np.ndarray, Uex : np.ndarray) -> float:

    erreur = linalg.norm(U - Uex) * h

    return erreur
```

```
# Tests

def u_1_sol(x : float, y : float) -> float:

    res = 1 / (8 * (pi ** 2)) * sin(2 * pi * x) * sin(2 * pi * y)

    return res

Uex = calculer_sol_ex(N, u_1_sol, Noeud)
erreur = calculer_norme_erreur(h, U, Uex)

res = "Uex = \n" + str(Uex) + "\nerreur =\n" + str(erreur)

print(res)
```

```
Uex =
[ 1.26651480e-02  1.55103329e-18 -1.26651480e-02  1.55103329e-18
  1.89946795e-34 -1.55103329e-18 -1.26651480e-02 -1.55103329e-18
  1.26651480e-02]
erreur =
```

```
0.0014799260223538886
```

1.7 Estimation du taux de convergence

On écrit la fonction :
`simulation` pour avoir le résultat de la simulation complète avec les paramètres du problème : N, f et u_{sol}.
Commentaire : `sol` est une liste d'entier qui permet de définir si l'on souhaite avoir le champ de valeurs uniquement pour U (`sol = [1]`) (par exemple dans le cas où Uex ne serait pas connu), uniquement pour Uex (`sol = [2]`) ou pour U et Uex (`sol = [1, 2]`).

```
def simulation(N : int, f : Callable[[float], float], u_sol : Callable[[float], float]␣
↪| None, sol : np.ndarray) -> tuple[np.ndarray | None, np.ndarray | None, np.ndarray␣
↪| None]:

    h = N_vers_h(N)
    Noeud = numeroter(N)
    Connec = connecter(N, Noeud)
    A = construire_matrice(N, h, Connec)
    B = construire_vecteur(N, f, Noeud)
    U, Uex, erreur = None, None, None
    if (1 in sol):
        U = resoudre_systeme(A, B)
    if (2 in sol):
        Uex = calculer_sol_ex(N, u_sol, Noeud)
    if (1 in sol and 2 in sol):
        erreur = calculer_norme_erreur(h, U, Uex)

    return (U, Uex, erreur)
```

On effectue la simulation pour des valeurs de N et de h différentes. On calcule, en fonction de h : U, Uex, l'erreur et le temps CPU. Finalement, on fait la regression linéaire pour obtenir α et C :

```
h_tab = [1 / x for x in range (5, 101, 5)]
N_tab = [h_vers_N(h) for h in h_tab]
erreur_tab = np.zeros(len(N_tab))
temps_tab = np.zeros(len(N_tab))

for i in range (len(h_tab)):
    N = N_tab[i]
    debut = time.process_time()
    U, Uex, erreur = simulation(N, f_1, u_1_sol, [1, 2])
    fin = time.process_time()
    temps = fin - debut
    temps_tab[i] = temps
    erreur_tab[i] = erreur

ln_h_tab = np.log(h_tab)
ln_erreur_tab = np.log(erreur_tab)
alpha, ln_C = np.polyfit(ln_h_tab, ln_erreur_tab, 1)
```

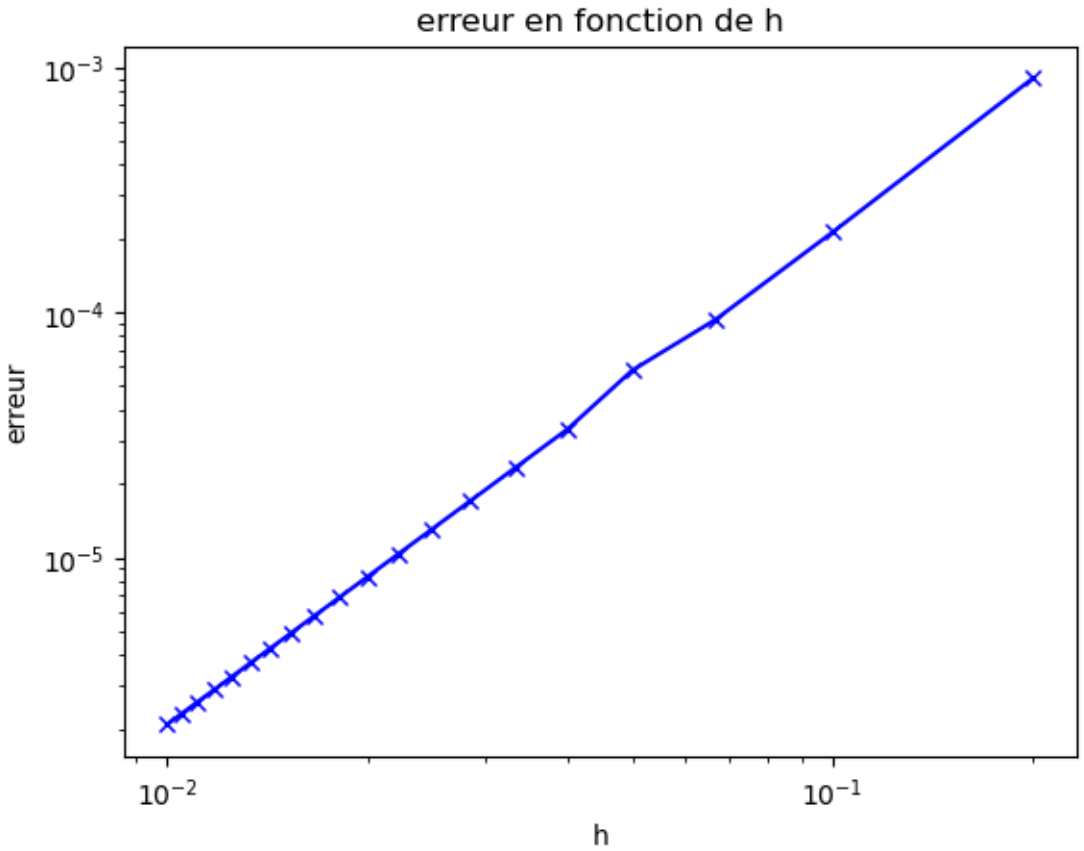
```
C = np.exp(ln_C)

res = "alpha = \n" + str(alpha) + "\nC = \n" + str(C)

print(res)
```

```
alpha =
2.022927603082109
C =
0.02295524653719487
```

On obtient $\alpha \approx 2$ et C petit ($C \neq 0$), ce qui est attendu.

On trace le graphique de l'erreur en fonction de h :

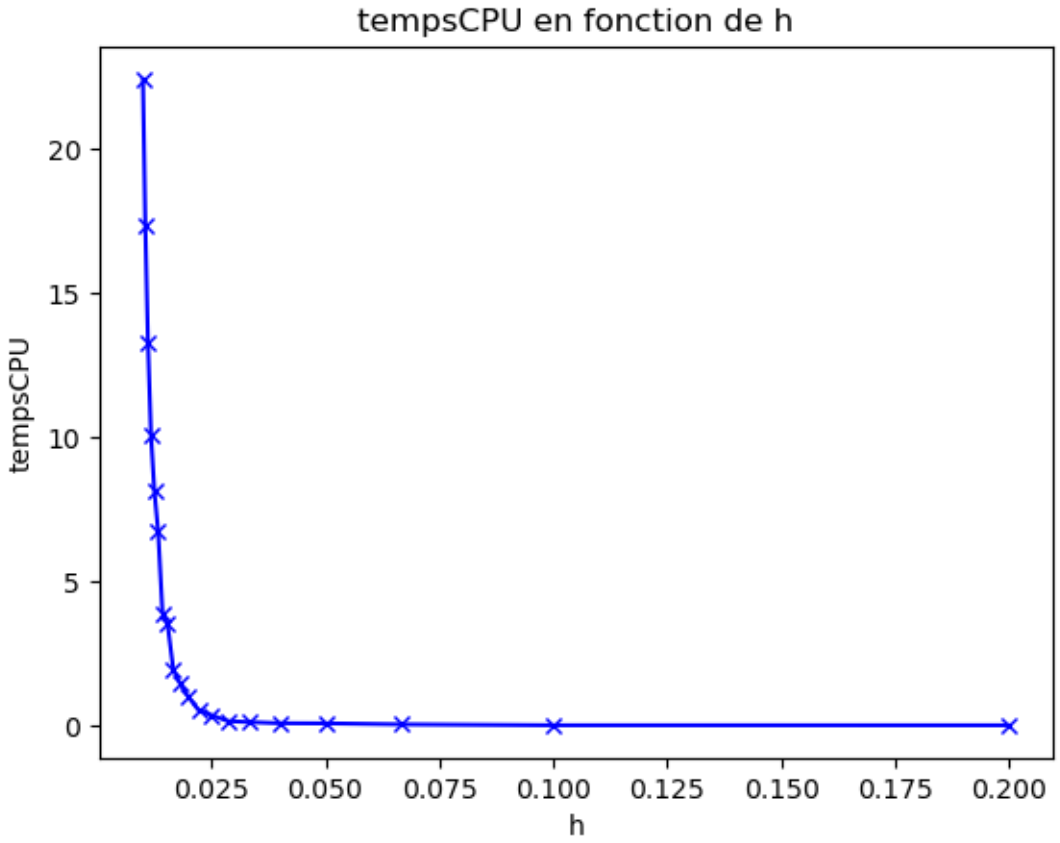
```
plt.loglog(h_tab, erreur_tab, marker = 'x', linestyle = '-', color = 'b')
plt.xlabel("h")
plt.ylabel("erreur")
plt.title("erreur en fonction de h")
plt.show()
```



L'erreur semble quadratique pour h.

On trace le graphique du temps CPU en fonction de h :

```
plt.plot(h_tab, temps_tab, marker = 'x', linestyle = '-', color = 'b')
plt.xlabel("h")
plt.ylabel("tempsCPU")
plt.title("tempsCPU en fonction de h")
plt.show()
```



Le temps CPU semble exponentiel pour h.

2 Visualisation de données à l'aide de `Paraview` et format de fichier `vtk`

2.1 Fonctions intermédiaires pour créer le fichier

Le but est de créer un fichier `vtk` pour stocker le résultat de la simulation et pouvoir le visualiser avec `Paraview`.

On écrit les fonctions qui permettent de construire le fichier `vtk` :
`creer_vtk_entete` pour avoir l'entête.
`creer_vtk_dataset` pour avoir le système de coordonnées.
`creer_vtk_cells` pour avoir le système de cellules.
`creer_vtk_cell_types` pour avoir le type de chaque cellule.
`creer_vtk_point_data` pour avoir le nombre de valeurs / vecteurs par champ.
`creer_vtk_fielddata` pour avoir les champs de valeurs / vecteurs.

Commentaires :

— `z` est un entier qui permet de définir si l'on souhaite afficher un vecteur de la valeur de la U selon deux plans pour le voir en 3D ($z = 0.00$ et $z = 0.01$) (`z != 0`) ou non (`z = 0`).

— Pour construire le fichier, on rajoute les bords (sur lesquels $u = 0$) qui étaient absents dans les résultats de la simulation (dans toute la suite, la valeur de N représente toujours le nombre de points dans une direction sans compter les bords, les bords ne sont présents que dans les fichiers `vtk`).

```
def creer_vtk_entete(titre : str) -> str:

    contenu = "# vtk DataFile Version 3.0\n"
    contenu += titre + "\n"
    contenu += "ASCII\n\n"

    return contenu
```

```
def creer_vtk_dataset(N : int, z : int) -> str:

    h = N_vers_h(N)

    contenu = "DATASET UNSTRUCTURED_GRID\n"
    points = (N + 2) ** 2
    if (z != 0):
        points *= 2
    contenu += "POINTS " + str(points) + " FLOAT\n"
    if (z != 0):
        k = 0
    for i in range (N + 2):
        for j in range (N + 2):
            contenu += str(i * h) + " " + str(j * h)
            contenu += " 0.00\n"

    if (z != 0):
        k = 0
        for i in range (N + 2):
            for j in range (N + 2):
                contenu += str(i * h) + " " + str(j * h)
                contenu += " 0.01\n"
                k += 1
    contenu += "\n"

    return contenu
```

```
def creer_vtk_cells(N : int, z : int) -> str:

    contenu = ""

    cells = (N + 1) ** 2
    facteur_1, facteur_2 = 5, 4
    if (z != 0):
```

```
        facteur_1, facteur_2 = 9, 8

    contenu += "CELLS " + str(cells) + " " + str(facteur_1 * cells) + "\n"
    for i in range (N + 1):
        for j in range (N + 1):
            k = i * (N + 2) + j
            contenu += str(facteur_2) + " " + str(k) + " " + str(k + 1) + " " + str(k␣
↪+ N + 3) + " " + str(k + N + 2)
            if (z != 0):
                k_2 = (N + 2) ** 2 + i * (N + 2) + j
                contenu += " " + str(k_2) + " " + str(k_2 + 1) + " " + str(k_2 + N +␣
↪3) + " " + str(k_2 + N + 2)
            contenu += "\n"

    contenu += "\n"

    return contenu
```

```
def creer_vtk_cell_types(N : int, z : int) -> str:

    contenu = ""

    cell_types = (N + 1) ** 2
    num_cell_types = 9
    if (z != 0):
        num_cell_types = 12

    contenu += "CELL_TYPES " + str(cell_types) + "\n"
    for i in range (cell_types):
        contenu += str(num_cell_types) + "\n"
    contenu += "\n"

    return contenu
```

```
def creer_vtk_point_data(N : int, sol : list[int], z : int) -> str:

    contenu = ""

    point_data = (N + 2) ** 2
    if (z != 0):
        point_data *= 2

    contenu += "POINT_DATA " + str(point_data) + "\n"
    if (z == 0):
        contenu += "FIELD FieldData " + str(len(sol)) + "\n"

    return contenu
```

```python
def creer_vtk_fielddata(nom : str, N : int, U : np.ndarray, z : int) -> str:

    contenu = ""

    point_data = (N + 2) ** 2
    if (z == 0):
        contenu += nom + " 1 " + str(point_data) + " FLOAT\n"
    else:
        contenu += "\nVECTORS " + nom + " FLOAT\n"

    boucles = 1
    if (z != 0):
        boucles = 2

    for boucle in range (boucles):
        k = 0
        for i in range (N + 2):
            for j in range (N + 2):
                if (i == 0 or i == N + 1 or j == 0 or j == N + 1):
                    if (z == 0):
                        contenu += "0.00\n"
                    else:
                        contenu += "0.00 0.00 0.00\n"
                else:
                    if (z == 0):
                        contenu += str(U[k]) + "\n"
                    else:
                        contenu += "0.00 0.00 " + str(U[k]) + "\n"
                    k += 1

    return contenu
```

2.2 Fonction pour créer le fichier

On écrit la fonction :
`creer_vtk` pour écrire le résultat de la simulation dans le fichier `fichier` sous le format `vtk` avec les paramètres du problème N, f, u_{sol} et les options `fichier`, `titre`, `sol` et `z`.

```python
def creer_vtk(fichier : str, titre : str, N : int, f : Callable[[float], float], u_sol␣
 ↪: Callable[[float], float] | None, sol : list[int], z : int) -> None:

    U, Uex, _ = simulation(N, f, u_sol, sol)

    contenu = ""
    contenu += creer_vtk_entete(titre)
    contenu += creer_vtk_dataset(N, z)
    contenu += creer_vtk_cells(N, z)
    contenu += creer_vtk_cell_types(N, z)
    contenu += creer_vtk_point_data(N, sol, z)
```

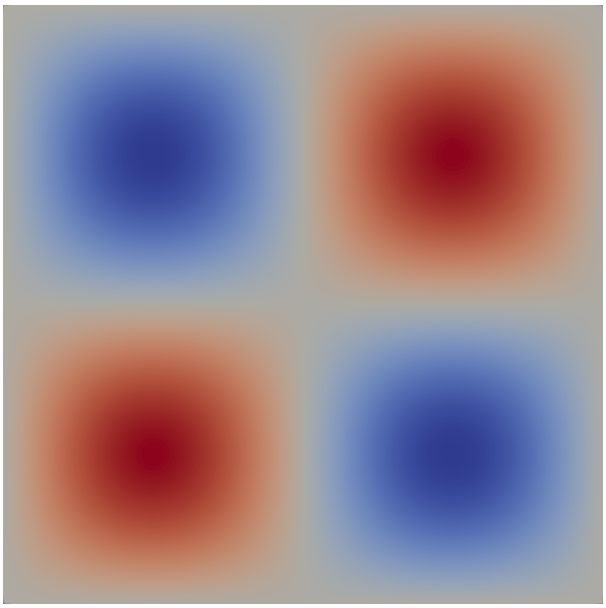
```
    if (1 in sol):
        contenu += creer_vtk_fielddata("U", N, U, z)
    if (2 in sol):
        contenu += creer_vtk_fielddata("Uex", N, Uex, z)

    fichier = open(fichier, "w")
    fichier.write(contenu)
    fichier.close()

    return None
```

2.3 Applications

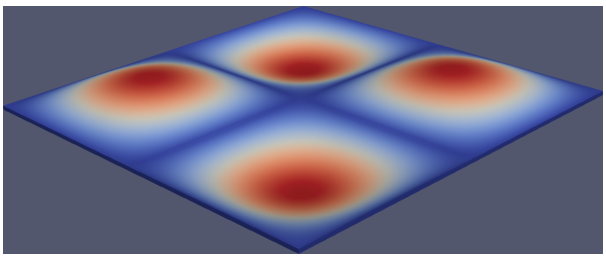
Test d'une simulation pour $f(x, y) = \sin(2\pi x)\sin(2\pi y)$ avec l'écriture de U et Uex dans un fichier `vtk` :

```
N = 99

fichier = "./Donnees/1-2D.vtk"
titre = "Laplacien 2D"
creer_vtk(fichier, titre, N, f_1, u_1_sol, [1, 2], 0)

fichier = "./Donnees/1-3D.vtk"
titre = "Laplacien 3D"
creer_vtk(fichier, titre, N, f_1, u_1_sol, [1, 2], 1)
```

On peut visualiser les fichiers avec `Paraview` :



Laplacien (option 2D)

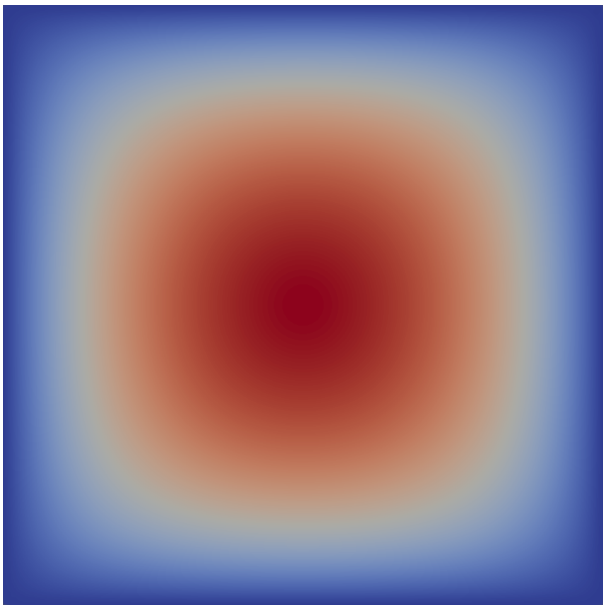


Laplacien (option 3D)

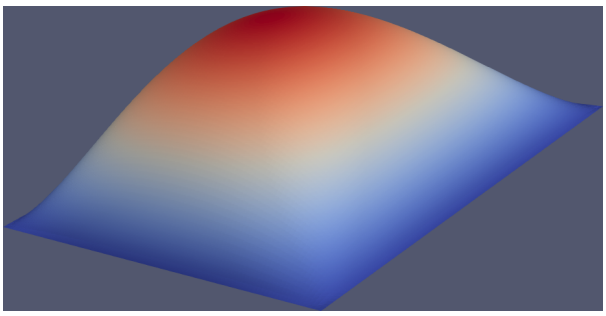
Test d'une simulation pour $f(x, y) = 1$ avec l'écriture de U dans un fichier `vtk` :

```
def f_2(x : float, y : float) -> float:

    res = 1

    return res

N = 99

fichier = "./Donnees/2-2D.vtk"
titre = "Laplacien 2D"
creer_vtk(fichier, titre, N, f_2, None, [1], 0)

fichier = "./Donnees/2-3D.vtk"
titre = "Laplacien 3D"
creer_vtk(fichier, titre, N, f_2, None, [1], 1)
```

On peut visualiser les fichiers avec `Paraview` :



Laplacien (option 2D)



Laplacien (option 3D)

3 Sauvegarde de données sous le format `hdf5`

3.1 Sauvegarde des résultats

Le but est de créer un fichier `hdf5` pour stocker le résultat de plusieurs simulations (pour $N_{tab} = \{N_1, ..., N_n\}$) sous une forme de répertoires.

On écrit la fonction :
`creer_hdf5` pour écrire le résultat des simulations dans le fichier `fichier` sous le format `hdf5`. Son contenu comprend :

— Un répertoire / groupe pour `N=*` (pour $N = \star \in N_{tab}$) qui contient les datasets `U` et `Uex`. Le seul attribu de chaque répertoire / groupe est `N` qui prend la valeur N.

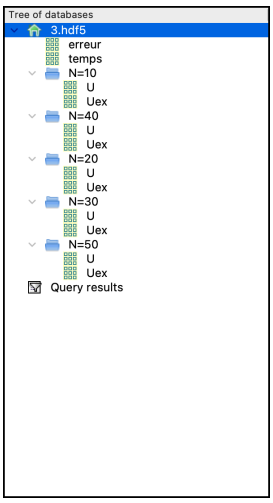
— Les datasets `erreur_tab` et `temps_tab`.

```
def creer_hdf5(fichier : str, N_tab : np.ndarray, f : Callable[[float], float], u_sol :
 ↪ Callable[[float], float] | None, sol : list[int]) -> None:

    fichier = h5py.File(fichier, "a")

    erreur_tab = np.zeros(len(N_tab))
    temps_tab = np.zeros(len(N_tab))

    for i in range (len(N_tab)):
        debut = time.process_time()
        U, Uex, erreur = simulation(N_tab[i], f, u_sol, sol)
        fin = time.process_time()
        temps = fin - debut
        temps_tab[i] = temps
        erreur_tab[i] = erreur
        groupe = fichier.create_group("N=" + str(N_tab[i]))
        donnee_U = groupe.create_dataset("U", data = U)
        donnee_Uex = groupe.create_dataset("Uex", data = Uex)
        groupe.attrs["N"] = N_tab[i]

    fichier.create_dataset("erreur", data = erreur_tab)
    fichier.create_dataset("temps", data = temps_tab)

    fichier.close()

    return None
```

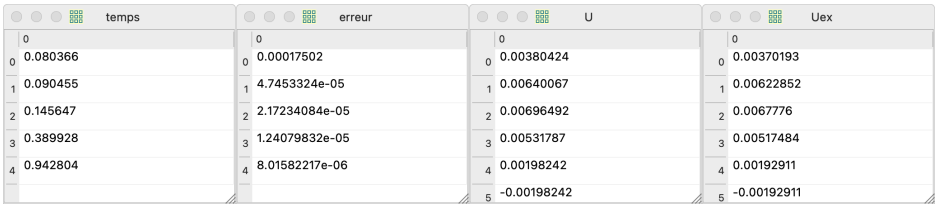
Test d'une écriture de la simulation dans un fichier `hdf5` :

```
# Tests

N_tab = [10 * x for x in range (1, 6)]
fichier = "./Donnees/3.hdf5"

creer_hdf5(fichier, N_tab, f_1, u_1_sol, [1, 2])
```

On peut voir la structure du fichier avec `vitables` :



Structure du fichier



Datasets

Test d'une lecture du fichier `hdf5` crée :

Lecture des clés :

```
# Tests

fichier = h5py.File("./Donnees/3.hdf5", "r")

res = "Clés :\n"
for k in fichier.keys():
    res += k + "\n"

print(res)
```

```
Clés :
N=10
N=20
N=30
N=40
N=50
erreur
temps
```

Lecture du contenu du groupe `N=10` (ses datasets et son attribut) :

```
# Tests

res = ""

res += "Contenu du groupe N=10:\n"
groupe = fichier["N=10"]
res += "U =\n"
res += str(groupe["U"][()]) + "\n"
```

```
res += "Uex =\n"
res += str(groupe["Uex"][()]) + "\n"
res += "Attribut N =\n"
res += str(groupe.attrs["N"]) + "\n"

print(res)
```

```
Contenu du groupe N=10:
U =
[ 0.00380424  0.00640067  0.00696492  0.00531787  0.00198242 -0.00198242
 -0.00531787 -0.00696492 -0.00640067 -0.00380424  0.00640067  0.01076916
  0.01171853  0.00894735  0.00333544 -0.00333544 -0.00894735 -0.01171853
 -0.01076916 -0.00640067  0.00696492  0.01171853  0.01275159  0.00973611
  0.00362948 -0.00362948 -0.00973611 -0.01275159 -0.01171853 -0.00696492
  0.00531787  0.00894735  0.00973611  0.00743372  0.00277118 -0.00277118
 -0.00743372 -0.00973611 -0.00894735 -0.00531787  0.00198242  0.00333544
  0.00362948  0.00277118  0.00103306 -0.00103306 -0.00277118 -0.00362948
 -0.00333544 -0.00198242 -0.00198242 -0.00333544 -0.00362948 -0.00277118
 -0.00103306  0.00103306  0.00277118  0.00362948  0.00333544  0.00198242
 -0.00531787 -0.00894735 -0.00973611 -0.00743372 -0.00277118  0.00277118
  0.00743372  0.00973611  0.00894735  0.00531787 -0.00696492 -0.01171853
 -0.01275159 -0.00973611 -0.00362948  0.00362948  0.00973611  0.01275159
  0.01171853  0.00696492 -0.00640067 -0.01076916 -0.01171853 -0.00894735
 -0.00333544  0.00333544  0.00894735  0.01171853  0.01076916  0.00640067
 -0.00380424 -0.00640067 -0.00696492 -0.00531787 -0.00198242  0.00198242
  0.00531787  0.00696492  0.00640067  0.00380424]
Uex =
[ 0.00370193  0.00622852  0.0067776   0.00517484  0.00192911 -0.00192911
 -0.00517484 -0.0067776  -0.00622852 -0.00370193  0.00622852  0.01047953
  0.01140336  0.00870671  0.00324573 -0.00324573 -0.00870671 -0.01140336
 -0.01047953 -0.00622852  0.0067776   0.01140336  0.01240863  0.00947425
  0.00353187 -0.00353187 -0.00947425 -0.01240863 -0.01140336 -0.0067776
  0.00517484  0.00870671  0.00947425  0.00723379  0.00269665 -0.00269665
 -0.00723379 -0.00947425 -0.00870671 -0.00517484  0.00192911  0.00324573
  0.00353187  0.00269665  0.00100527 -0.00100527 -0.00269665 -0.00353187
 -0.00324573 -0.00192911 -0.00192911 -0.00324573 -0.00353187 -0.00269665
 -0.00100527  0.00100527  0.00269665  0.00353187  0.00324573  0.00192911
 -0.00517484 -0.00870671 -0.00947425 -0.00723379 -0.00269665  0.00269665
  0.00723379  0.00947425  0.00870671  0.00517484 -0.0067776  -0.01140336
 -0.01240863 -0.00947425 -0.00353187  0.00353187  0.00947425  0.01240863
  0.01140336  0.0067776  -0.00622852 -0.01047953 -0.01140336 -0.00870671
 -0.00324573  0.00324573  0.00870671  0.01140336  0.01047953  0.00622852
 -0.00370193 -0.00622852 -0.0067776  -0.00517484 -0.00192911  0.00192911
  0.00517484  0.0067776   0.00622852  0.00370193]
Attribut N =
10
```

Lecture des dataset `erreur` et `temps` :

```
# Tests

res = ""

erreur_tab, temps_tab = fichier["erreur"][()], fichier["temps"][()]
res += "erreur_tab =\n"
res += str(erreur_tab) + "\n"
res += "temps_tab = \n"
res += str(temps_tab)

fichier.close()

print(res)
```

```
erreur_tab =
[1.75021874e-04 4.74533240e-05 2.17234084e-05 1.24079832e-05
 8.01582217e-06]
temps_tab =
[0.080366 0.090455 0.145647 0.389928 0.942804]
```

3.2 Récupération des résultats

Le but est de convertir un fichier `hdf5` qui possède la structure crée précédemment en fichiers `vtk` (un fichier `vtk` par valeur de N).

On écrit la fonction :
`hdf5_vers_vtk` pour lire chaque groupe `N=*` d'un fichier `hdf5` et créer le fichier `vtk` correspondant à `N`. (Commentaire : elle est très ressemblante à `creer_vtk` pour l'écriture, mais la partie lecture est obtenue avec le fichier `hdf5` et non la simulation.)

```
def hdf5_vers_vtk(fichier_hdf5 : str, fichier_vtk : str, titre : str, N_tab : np.
 ↪ndarray, sol : list[int], z : int) -> None:

    fichier_hdf5 = h5py.File(fichier_hdf5, "r")

    for i in range (len(N_tab)):
        N = N_tab[i]
        h = N_vers_h(N)
        groupe_nom = "N=" + str(N)
        groupe = fichier_hdf5[groupe_nom]
        if (1 in sol):
            U = groupe["U"][()]
        if (2 in sol):
            Uex = groupe["Uex"][()]

        contenu = ""
        contenu += creer_vtk_entete(titre)
        contenu += creer_vtk_dataset(N, z)
        contenu += creer_vtk_cells(N, z)
        contenu += creer_vtk_cell_types(N, z)
```

```
        contenu += creer_vtk_point_data(N, sol, z)
        if (1 in sol):
            contenu += creer_vtk_fielddata("U", N, U, z)
        if (2 in sol):
            contenu += creer_vtk_fielddata("Uex", N, Uex, z)

        fichier_vtk_N = fichier_vtk[:-4] + "-N=" + str(N) + ".vtk"
        fichier_ec = open(fichier_vtk_N, "w")
        fichier_ec.write(contenu)
        fichier_ec.close()

    fichier_hdf5.close()

    return None
```

Test de la conversion du fichier `hdf5` précédemment crée vers des fichiers `vtk` :

```
# Tests

hdf5_vers_vtk("./Donnees/3.hdf5", "./Donnees/3-2D.vtk", "Laplacien 2D", N_tab, [1, 2],␣
 ↪0)
hdf5_vers_vtk("./Donnees/3.hdf5", "./Donnees/3-3D.vtk", "Laplacien 3D", N_tab, [1, 2],␣
 ↪1)
```