

# **Modélisation mult-échelle des phénomènes génériques d'imprégnation sous sollicitations mécaniques de mèches pour l'élaboration de composites à matrice organique.**

Point avancement n°02 du 08/11/2022

## Objectifs

Les procédés visés impliquent l'écoulement d'un fluide à travers un milieu poreux déformable qui mettent en jeu des phénomènes :

- mécaniques dus aux sollicitations de la mèche durant son imprégnation et à l'opération de compactage lors du bobinage pour le procédé d'enroulement, ou de l'application d'un rouleau presseur pour le procédé ATL/AFP ;
- convectifs liés à l'écoulement de la résine lors de la phase d'imprégnation des mèches, puis de la phase de compaction ;
- chimiques et thermiques dus à la polymérisation de la résine au cours du procédé.

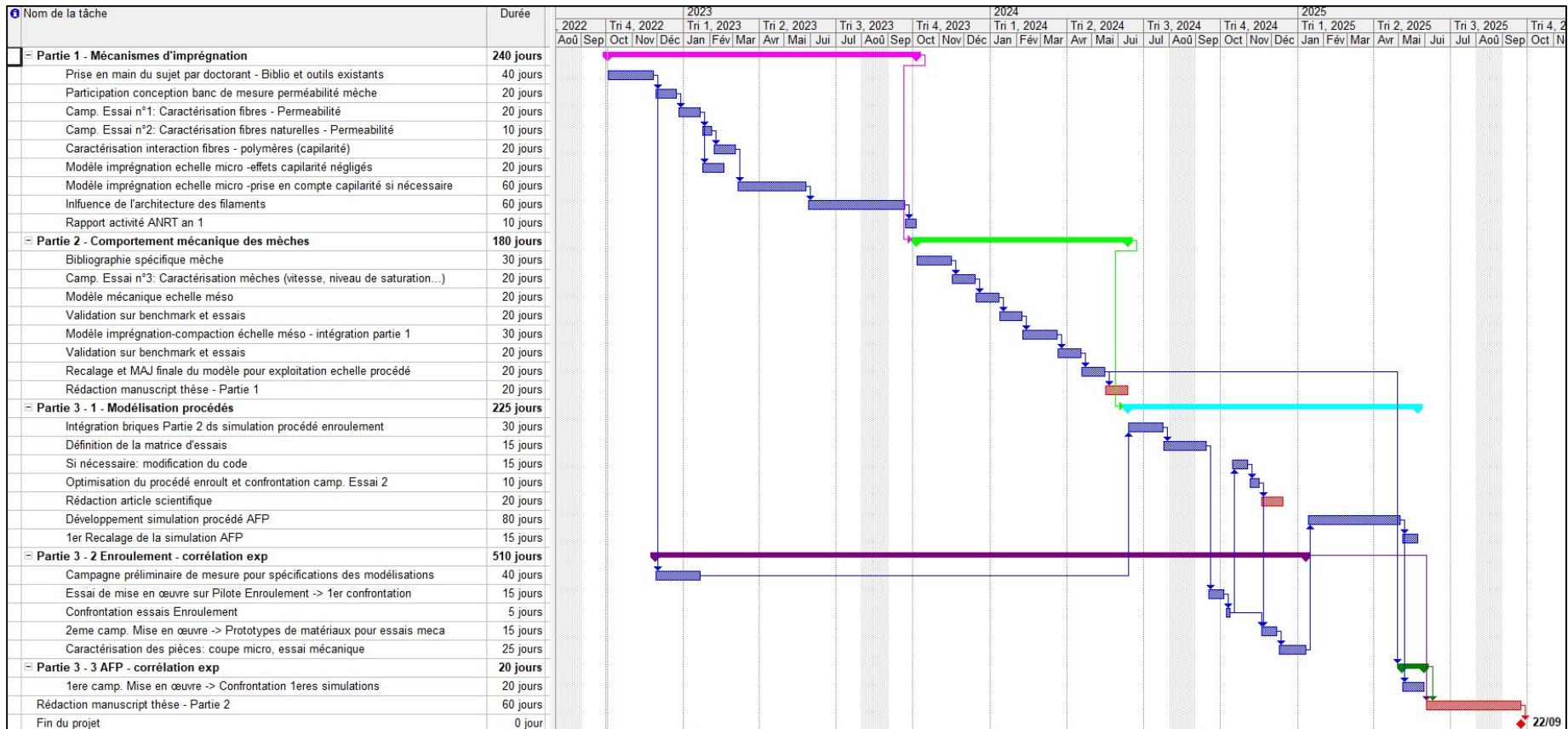
## Rappel du sujet

L'objectif de cette thèse est de proposer et d'identifier un modèle permettant de décrire le couplage entre le comportement thermo mécanique de la mèche et sa saturation en polymère. Ce modèle permettra ainsi de prédire l'état de déformation et donc le taux de renfort pour un niveau de saturation donné.

Le travail est décomposé en 3 parties:

1. Décrire les mécanismes d'imprégnation à différentes échelles.
2. Identifier le comportement mécanique d'une mèche.
3. Intégrer les modèles méso déduits des étapes précédentes dans des modèles de simulation numérique des procédés d'enroulement filamentaire et de dépôse de mèches/bandes (ATL/AFP).

# Planning prévisionnel : global



# Planning prévisionnel : local



Pt 2

Pt 2

# Bilan de la dernière réunion

VIDE.



## Ordre du jour

- A. Eléments finis from scratch
- B. Maillage volumique d'un empilement TIFF

## A. Elements finis from scratch

Workflow d'un programme de résolution d'un problème mécanique par éléments finis

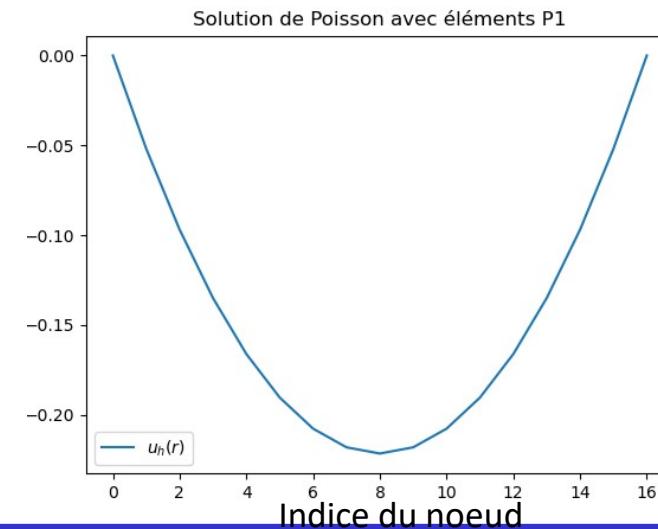
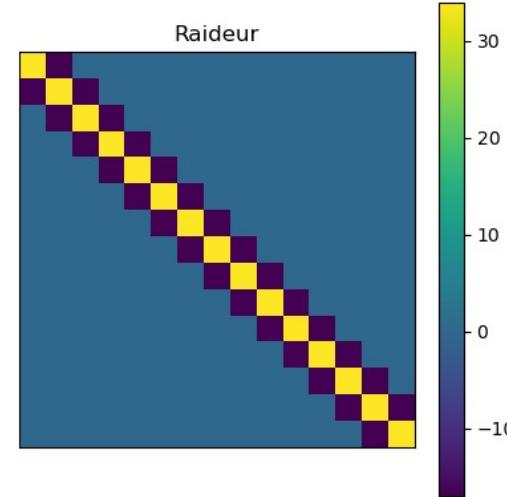
1. Définition de fonctions de formes
2. Maillage
3. Définition des paramètres matériau
4. Construction d'une matrice de raideur locale
5. Assemblage de la matrice de raideur globale
6. Assemblage du vecteur forces extérieures
7. Prise en compte des conditions aux limites dans la matrice de raideur et le second membre
8. Résolution du système linéaire
9. Export / Post-traitement

## A. Démonstrateurs : Laplacien 1D, éléments P1

```

1  #! -*- coding : utf-8 -*-
2  import matplotlib.pyplot as plt
3  import numpy as np
4  # FONCTIONS
5  def f(r):
6      return -2
7
8  # DISCRETISATION SPATIALE
9  r_max = 1                      # rayon extérieur
10 r_min = 0                       # rayon intérieur
11 nr = 16                         # nombre de noeuds
12 ns = nr-1                       # taille du système linéaire
13 dr = (r_max-r_min)/(nr+1)       # pas d'espace
14 r = np.arange(nr+1)*dr          # positions des noeuds
15
16 # print(f"Positions des noeuds\n{r}")
17 # print(f"Distances entre les noeuds\n{r[1]-r[0]}")
18
19 # ASSEMBLAGE DE LA MATRICE DE RAIDEUR
20 K = np.zeros((ns,ns))           # allocation de mémoire et initialisation à 0
21
22 Ke = 1/dr*np.asarray([
23     [1, -1],
24     [-1, 1]
25 ])
26
27 for k in range(ns-1):
28
29     for i in range(2):
30         for j in range(2):
31             m = k + i
32             n = k + j
33             K[m, n] += Ke[i, j]
34
35 K[0 , 0]  += 1/dr
36 K[-1, -1] += 1/dr
37
38 plt.matshow(K)                 # affiche les composantes de la matrice de raideur
39 plt.colorbar()                  # affiche l'échelle des couleurs
40 plt.grid(False)                # efface la grille
41 plt.title("Raideur")          # paramètre le titre du graphe
42 plt.xticks([])                 # efface les étiquettes des abscisses
43 plt.yticks([])                 # efface les étiquettes des ordonnées
44 plt.show()                     # affiche la fenêtre
45
46 # ASSEMBLAGE DU SECONDE MEMBRE : CARRES
47 b = np.zeros(ns)
48
49 for k in range(ns):
50     b[k] = dr*f(r[k+1]-r[k])/2
51
52 # RESOLUTION DU SYSTEME
53 u = np.linalg.solve(K, b)
54 u = np.hstack(([0], u, [0]))
55
56 # CONTROLE
57 plt.plot(u, label="$u_h(r)$")
58 plt.legend()
59 plt.title("Solution de Poisson avec éléments P1")
60 plt.show()

```

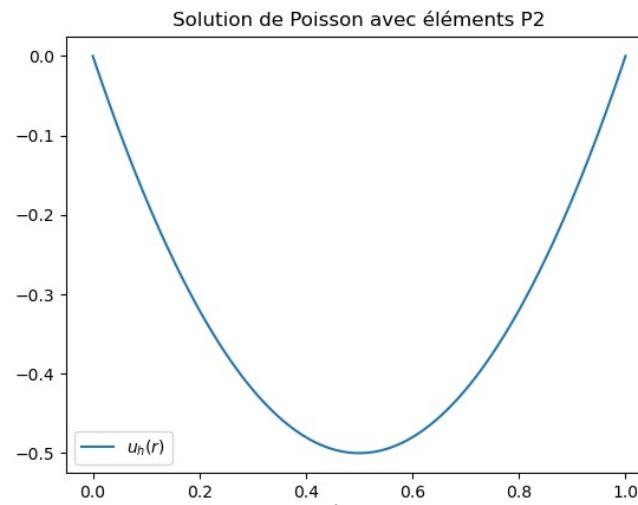
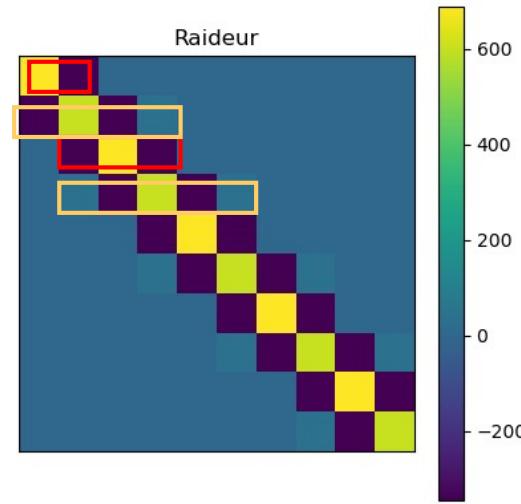


## A. Démonstrateurs : Laplacien 1D, éléments P2

```

1 #! -*- coding : utf-8 -*-
2 import matplotlib.pyplot as plt
3 import numpy as np
4
5 # FONCTIONS
6 def f(r):
7     return -2
8
9 # DISCRETISATION SPATIALE
10 r_max = 1           # rayon extérieur
11 r_min = 0           # rayon intérieur
12 nr = 128           # nombre de noeuds
13 ns = 2*nr+1         # taille du système linéaire
14 dr = (r_max-r_min)/(nr+1) # pas d'espace
15 r = np.arange(ns+2)*dr/2 # positions des noeuds
16
17 # ASSEMBLAGE DE LA MATRICE DE RAIDEUR
18 K_diag = (nr*[16/3, 14/3]) + [16/3]
19 K_sdiag = (2*nr+1)*[-8/3]
20 K_ssdiag = (nr*[0, 1/3]) + [0]
21
22 K = np.diag(K_diag) \
|+ np.diag(K_sdiag[:-1], 1) \
|+ np.diag(K_sdiag[:-1], -1) \
|+ np.diag(K_ssdiag[:-2], 2) \
|+ np.diag(K_ssdiag[:-2], -2)
23
24 K /= dr
25
26 plt.matshow(K[:10,:10]) # affiche les composantes de la matrice de raideur
27 plt.colorbar() # affiche l'échelle des couleurs
28 plt.grid(False) # efface la grille
29 plt.title("Raideur") # paramètre le titre du graphe
30 plt.xticks([]) # efface les étiquettes des abscisses
31 plt.yticks([]) # efface les étiquettes des ordonnées
32 plt.show() # affiche la fenêtre
33
34 # ASSEMBLAGE DU SECOND MEMBRE
35 b = np.zeros(ns)
36
37 for k in range(ns):
38     b[k] = dr*(1/6*f(r[k+1])+1/6*f(r[k])+2/3*f((r[k+1]+r[k])/2))
39
40 # RÉSOLUTION DU SYSTEME
41 u = np.linalg.solve(K, b)
42 u = np.hstack(([0], u, [0]))
43
44 # CONTRÔLE
45 plt.plot(r, u, label="$u_h(r)$")
46 plt.legend()
47 plt.title("Solution de Poisson avec éléments P2")
48 plt.show()
49
50
51
52

```



## A. Setup pour un problème non linéaire

$$\bar{\Omega} = [0, 1]$$

$$u = \bar{\Omega} \rightarrow \mathbb{R}$$

$$x \mapsto u(x)$$

$$Q = \mathbb{R} \rightarrow \mathbb{R}$$

$$y \mapsto 1 + y^2$$

$$\partial_{xx} (Q(u(x)) \partial_x u(x)) = f(x)$$

$$u(0) = 0$$

$$u(1) = 1$$

## A. Algorithme de Newton adapté aux éléments finis

- Calcul du résidu  $F(u)$
- Dérivation formelle : calcul de la dérivée de Fréchet du résidu
- Choix de fonctions de forme
- Maillage
- Itérer sur le nombre d'itération (ou jusqu'à un critère d'arrêt)
  - Assembler le résidu
  - Assembler la différentielle
  - Calcul de l'itérée suivante
  - Test d'arrêt

# A. Implémentation des données du problème

## Validation de l'algorithme de Newton sur cas linéaire

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 a = 0
4 b = 1
5 n = 63
6 h = (b-a)/n
7 x = np.linspace(a, b, n+1, endpoint=True)
8
9 def Q(u):
10     "Coefficient non linéaire."
11     return np.ones_like(u)
12
13 def dQ(u):
14     "Variation du coefficient non linéaire."
15     return np.zeros_like(u)
16
17 def f(u):
18     "Second membre."
19     return np.zeros_like(u)
20
21 def df(u):
22     "Variation du second membre."
23     return np.zeros_like(u)

```

## Validation de l'algorithme de Newton sur cas réellement non linéaire

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3
4 a = 0
5 b = 1
6 n = 63
7 h = (b-a)/n
8 x = np.linspace(a, b, n+1, endpoint=True)
9
10 def Q(u):
11     "Coefficient non linéaire."
12     return 1+np.power(u, 2)
13
14 def dQ(u):
15     "Variation du coefficient non linéaire."
16     return 2*u
17
18 def f(u):
19     "Second membre."
20     return 2*u
21
22 def df(u):
23     "Variation du second membre."
24     return 2*np.ones_like(u)

```

# A. Reprise du calcul pour les différences finies (réunions précédentes)

$$\frac{1}{h} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} Q(u) \partial_x u = \frac{1}{h} \left[ Q(u) \partial_x u \right]_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} = (Q_{i+\frac{1}{2}} u_{i+\frac{1}{2}} - Q_{i-\frac{1}{2}} u_{i-\frac{1}{2}}) \frac{1}{h}$$

$$= \frac{1}{2h^2} \left\{ \begin{array}{l} Q(u_{i+1}) + Q(u_i) \\ \hline (u_{i+1} - u_i) - (Q(u_i) + Q(u_{i+1})) (u_i - u_{i+1}) \end{array} \right.$$

$\forall i \in [1, m] : F_i = \frac{1}{2h^2} \left\{ \begin{array}{l} (Q(u_{i+1}) + Q(u_i)) u_{i+1} - (Q(u_{i+1}) + 2Q(u_i) + Q(u_{i-1})) u_i \\ \hline A_{i+1} \\ + (Q(u_i) + Q(u_{i-1})) u_{i-1} \end{array} \right.$

$F_0 = u_0 - 0$

$F_{m+1} = u_{m+1} - 1$

On peut calculer la jacobienne maintenant

$$\forall i \in [1, m] : \forall j \in [0, m+1] \quad J_{i,j} = \frac{\partial F_i}{\partial u_j}$$

On calcule les variations élémentaires

$$\frac{\partial A_{i,i-1}}{\partial u_{i-1}} = \frac{\partial (Q(u_i) + Q(u_{i-1}))}{\partial u_{i-1}} = Q'(u_{i-1})$$

$$\frac{\partial A_{i,i+1}}{\partial u_{i+1}} = Q'(u_i)$$

$$\frac{\partial A_{i,i}}{\partial u_{i-1}} = Q'(u_{i-1})$$

$$\frac{\partial A_{i,i}}{\partial u_i} = 2Q(u_i)$$

$$\frac{\partial A_{i,i}}{\partial u_{i+1}} = Q'(u_{i+1})$$

$$\frac{\partial A_{i,i+2}}{\partial u_{i+2}} = Q'(u_{i+2})$$

$$\frac{\partial A_{i,i+2}}{\partial u_{i+1}} = Q'(u_i)$$

$$\frac{\partial F_i}{\partial u_{i-1}} = \frac{1}{2h^2} \left\{ \begin{array}{l} \frac{\partial A_{i,i}}{\partial u_{i-1}} \times u_i + \frac{\partial A_{i,i-1}}{\partial u_{i-1}} u_{i-1} + A_{i,i-1} \\ \hline \end{array} \right\}$$

$$\frac{\partial F_i}{\partial u_i} = \frac{1}{2h^2} \left\{ \begin{array}{l} -Q'(u_{i-1}) u_i + Q'(u_{i+1}) u_{i+1} + (Q(u_i) + Q(u_{i+1})) \\ \hline \end{array} \right\}$$

$$\frac{\partial F_i}{\partial u_{i+1}} = \frac{1}{2h^2} \left\{ \begin{array}{l} \frac{\partial A_{i,i}}{\partial u_{i+1}} \times u_{i+1} + \frac{\partial A_{i,i+1}}{\partial u_{i+1}} u_{i+2} + A_{i,i+1} \\ \hline \end{array} \right\}$$

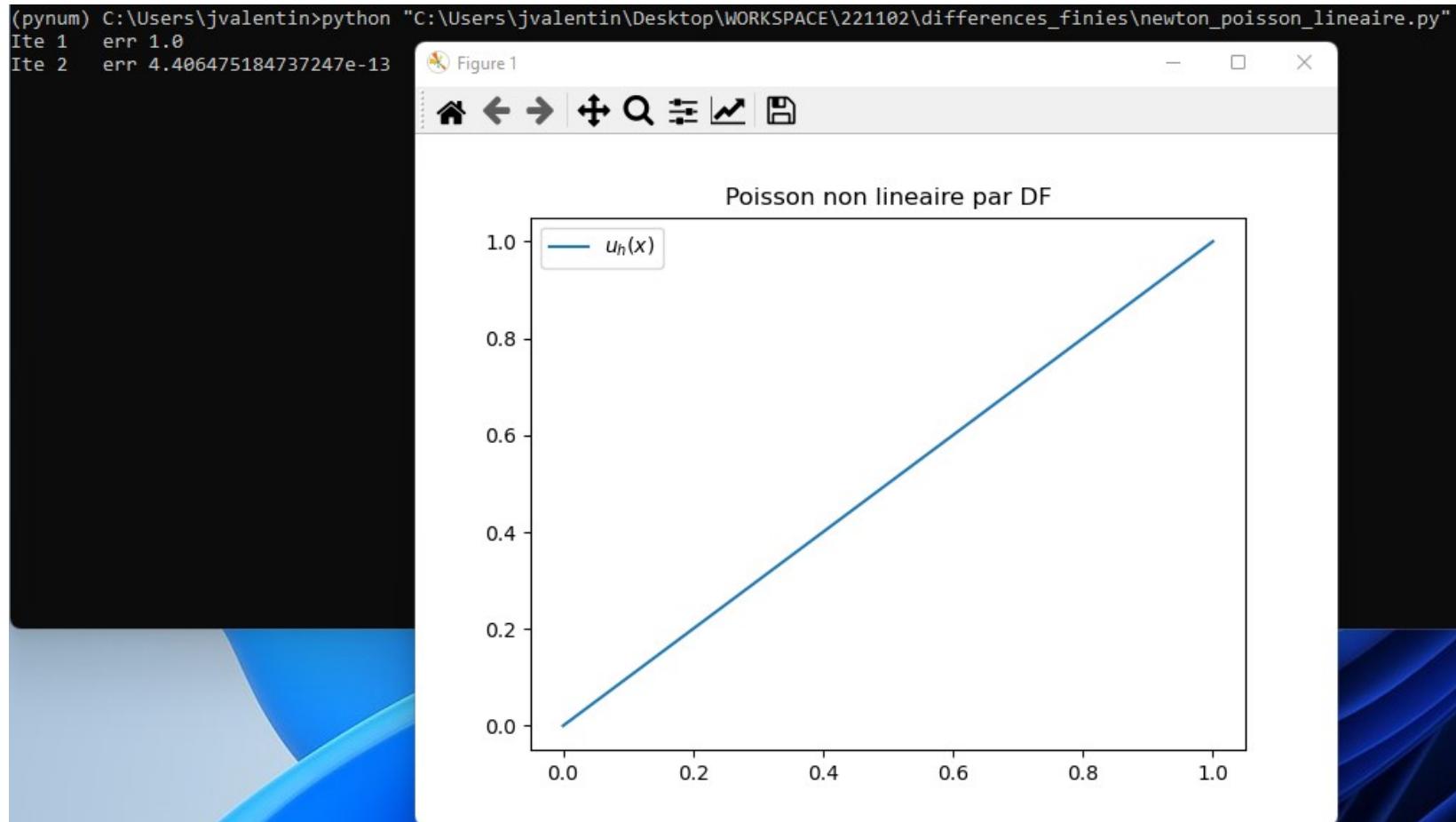
$$= \frac{1}{2h^2} \left\{ \begin{array}{l} -Q'(u_{i-1}) u_i + Q'(u_{i+1}) u_{i+1} + (Q(u_{i+1}) + Q(u_i)) \\ \hline \end{array} \right\}$$

```

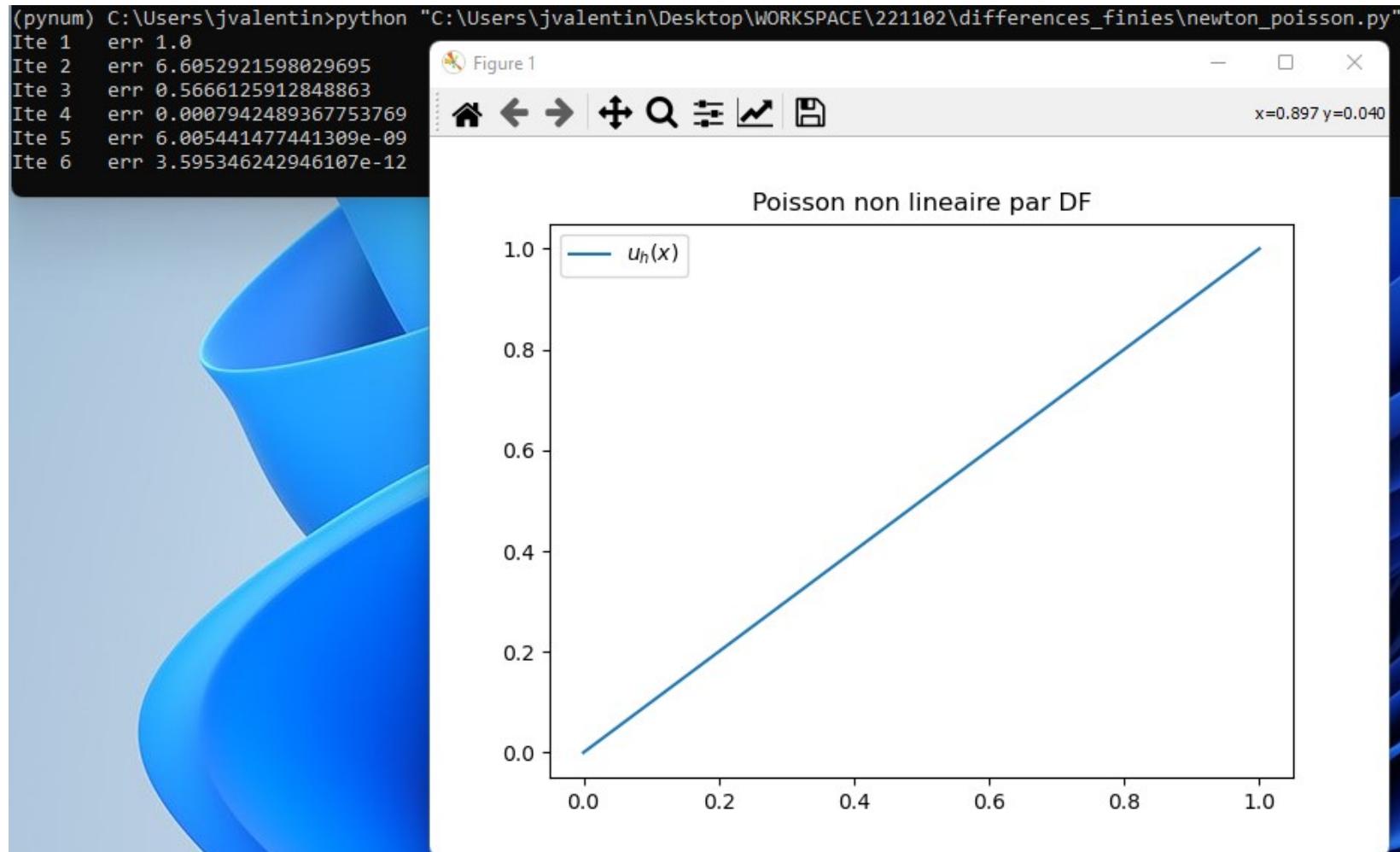
26 def F(u):
27     "Residu."
28     res = np.zeros_like(u)
29
30     for i in range(len(res)):
31         if i == 0:
32             res[i] = u[i] - 0
33         elif i == len(res)-1:
34             res[i] = u[i] - 1
35         else:
36             res[i] = 1/(2*h**2)*(
37                 + (Q(u[i]) + Q(u[i-1])) * u[i-1] \
38                 - (Q(u[i+1]) + 2*Q(u[i]) + Q(u[i-1])) * u[i] \
39                 + (Q(u[i+1]) + Q(u[i])) * u[i+1] \
40                 ) - f(u[i])
41
42     return res
43
44 def J(u):
45     "Variations du residu."
46     jac = np.zeros((len(u), len(u)))
47
48     for i in range(len(jac)):
49         if i == 0:
50             jac[i, i] = 1
51         elif i == len(jac)-1:
52             jac[i, i] = 1
53         else:
54             jac[i, i-1] = 1/(2*h**2) * (-dQ(u[i-1])*u[i] + dQ(u[i-1])*u[i-1] + (Q(u[i]) + Q(u[i-1])))
55             jac[i, i] = 1/(2*h**2) * (dQ(u[i])*(u[i+1]-2*u[i]+u[i-1]) - (Q(u[i+1]) + 2*Q(u[i]) + Q(u[i-1])))
56             jac[i, i+1] = 1/(2*h**2) * (-dQ(u[i+1])*u[i] + dQ(u[i+1])*u[i+1] + (Q(u[i]) + Q(u[i+1])))
57
58     return jac
59
60 ite = 10
61 tol = 1e-5
62 up = np.zeros_like(x)
63
64 for i in range(ite):
65     res = F(up)
66     jac = J(up)
67     u = up - np.linalg.inv(jac) @ res
68
69     eps = np.max(np.abs(res))
70     print(f"Ite {i+1} terr {eps}")
71     if eps < tol:
72         break
73
74     up = u
75
76 plt.plot(x, u, label="$u_h(x)$")
77 plt.legend()
78 plt.title("Poisson non linéaire par DF")
79 plt.show()

```

## A. Résultat de Newton sur le cas linéaire par D.F



## A. Résultat du cas non linéaire par D.F



## A. Calculs pour les éléments finis

Le cas  $P_1$  d'abord. La discrétisation pour les éléments  $P_1$  est donnée par

$$\Omega = (a, b) ; n \text{ éléments} \Rightarrow n+1 \text{ noeuds} ; h = \frac{b-a}{n}$$

On choisit les fonctions de forme  $P_1$ :

$$\forall i \in [0; n-1]: \varphi_i = \begin{cases} \frac{x - x_{i-1}}{h} & x \in [x_{i-1}, x_i] \\ \frac{x_{i+1} - x}{h} & x \in [x_i, x_{i+1}] \\ 0 & \text{sinon} \end{cases}$$

$$\varphi_0 = xc \mapsto \frac{x_1 - xc}{h} \quad xc \in [a, x_1] \quad 0 \text{ sinon}$$

$$\varphi_{n-1} = xc \mapsto \frac{xc - x_{n-1}}{h} \quad xc \in [x_{n-1}, x_n] \quad 0 \text{ sinon}.$$

$$(P): \left\{ \begin{array}{l} \int_{\Omega} (\partial_x(Q(u)) \partial_x u) v = 0 \\ u(0) = 0 \\ u(1) = 1 \end{array} \right.$$

$$\forall v \in H_0^1(\Omega; \mathbb{R}) \quad \int_{\Omega} \partial_x(Q(u)) \partial_x u v = 0$$

$$\Leftrightarrow \boxed{- \int_{\Omega} Q(u) \partial_x u \partial_x v = 0} \quad (P_v)$$

Soit  $i \in [1, n-1]$ :  $v = \varphi_i \in H_0^1(\Omega)$

$$(P_v): - \int_{\Omega} Q(u) \partial_x u \partial_x \varphi_i = 0 \Leftrightarrow - \int_{\Omega_{i-1}}^{x_{i+1}} Q(u) \partial_x u \partial_x \varphi_i = 0$$

Les seules fonctions non-nulles sur  $[x_{i-1}, x_{i+1}]$  sont  $\varphi_{i-1}, \varphi_i, \varphi_{i+1}$ .  
décomposée selon les supports

## A. Calculs pour les éléments finis

$$\begin{aligned}
-\int_{x_{i-1}}^{x_{i+2}} Q(u) \partial_x u \partial_x \varphi_i &= -\int_{x_{i-1}}^{x_{i+2}} Q(u) \left( \sum u_j \partial_x \varphi_j \right) \partial_x \varphi_i \\
&= -\int_{x_{i-1}}^{x_{i+2}} Q(u) \left\{ u_{i-2} \partial_x \varphi_{i-2} + u_i \partial_x \varphi_i + u_{i+2} \partial_x \varphi_{i+2} \right\} \partial_x \varphi_i \\
&= -\left\{ \int_{x_{i-1}}^{x_{i+2}} Q(u) \partial_x \varphi_{i-2} \partial_x \varphi_i \right\} u_{i-2} - \left\{ \int_{x_{i-1}}^{x_{i+2}} Q(u) \partial_x \varphi_i^2 \right\} u_i + \left\{ \int_{x_{i-1}}^{x_{i+2}} Q(u) \partial_x \varphi_{i+2} \partial_x \varphi_i \right\} u_{i+2} \\
&= \left\{ \int_{x_{i-1}}^{x_i} Q(u) \partial_x \varphi_{i-2} \partial_x \varphi_i \right\} u_{i-2} - \left\{ \int_{x_{i-1}}^{x_i} Q(u) \partial_x \varphi_i^2 \right\} u_i + \left\{ \int_{x_{i-1}}^{x_i} Q(u) \partial_x \varphi_{i+2} \partial_x \varphi_i \right\} u_{i+2}
\end{aligned}$$

On doit maintenant choisir ces quadratures. On peut tout de suite regarder l'impresion:

$$\frac{1}{h} \int_a^b \psi \, dx \approx \left( \frac{1}{6} \psi(a) + \frac{1}{6} \psi(b) + \frac{2}{3} \psi \left( \frac{a+b}{2} \right) \right)$$

$$\begin{aligned}
\textcircled{1} : \int_{x_{i-1}}^{x_i} Q(u) \left( -\frac{1}{h} \right) \left( \frac{1}{h} \right) &\approx -\frac{1}{h} \left( \frac{1}{6} Q(u(x_{i-1})) + \frac{1}{6} Q(u(x_i)) + \frac{2}{3} Q \left( \frac{u(x_{i-1}) + u(x_i)}{2} \right) \right) \\
\textcircled{2} : \int_{x_{i-1}}^{x_i} Q(u) \left( \frac{1}{h} \right)^2 &\approx \frac{1}{h} \left( \frac{1}{6} Q(u_{i-1}) + \frac{1}{6} Q(u_i) + \frac{2}{3} Q \left( \frac{u_{i-1} + u_i}{2} \right) \right) \\
\textcircled{3} : \int_{x_{i-1}}^{x_{i+1}} Q(u) \left( -\frac{1}{h} \right)^2 &\approx \frac{1}{h} \left( \frac{1}{6} Q(u_i) + \frac{1}{6} Q(u_{i+1}) + \frac{2}{3} Q \left( \frac{u_i + u_{i+1}}{2} \right) \right) \\
\textcircled{4} : \int_{x_{i-1}}^{x_{i+1}} Q(u) \left( \frac{1}{h} \right) \left( -\frac{1}{h} \right) &\approx -\frac{1}{h} \left( \frac{1}{6} Q(u_i) + \frac{1}{6} Q(u_{i+1}) + \frac{2}{3} Q \left( \frac{u_i + u_{i+1}}{2} \right) \right)
\end{aligned}$$

## A. Calculs pour les éléments finis

On cherche maintenant le jacobien du résidu. On différencie au sens de Fréchet :

$$\bar{u} \in V, \epsilon \in (0, 1)$$

$$F(u + \epsilon \bar{u}) = \int_{\Omega} Q(u + \epsilon \bar{u}) \partial_x \{ \epsilon u + \epsilon \bar{u} \} \partial_x v$$

$$J(u, v; \bar{u}) = \lim_{\epsilon \rightarrow 0} \frac{F(u + \epsilon \bar{u}) - F(u)}{\epsilon}$$

$$(1) = \int_{\Omega} Q(u + \epsilon \bar{u}) \partial_x \{ u + \epsilon \bar{u} \} \partial_x v - Q(u) \partial_x u \partial_x v$$

$$Q \in \mathcal{C}^1: Q(u + \epsilon \bar{u}) = Q(u) + \epsilon Q'(u) \bar{u} + o$$

$$\Rightarrow (1) = \int_{\Omega} (Q(u) + \epsilon Q'(u) \bar{u})(\partial_x u + \epsilon \partial_x \bar{u}) \partial_x v - Q(u) \partial_x u \partial_x v$$

$$= \int_{\Omega} Q(u) \partial_x u \partial_x v + \epsilon Q'(u) \bar{u} \partial_x u \partial_x v + \epsilon Q(u) \partial_x \bar{u} \partial_x v + \epsilon^2 Q''(u) \bar{u} \bar{u} \partial_x u \partial_x v - Q(u) \partial_x u \partial_x v$$

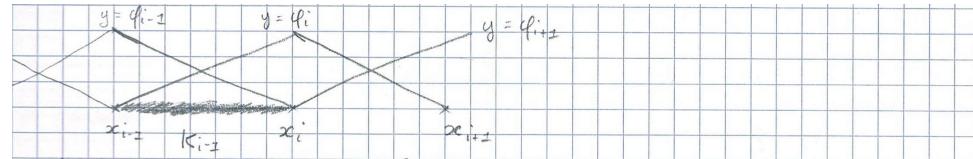
$$= \int_{\Omega} \epsilon Q'(u) \bar{u} \partial_x u \partial_x v + \epsilon Q(u) \partial_x \bar{u} \partial_x v$$

$$= \int_{\Omega} \epsilon \partial_x v (Q'(u) \partial_x u \bar{u} + Q(u) \partial_x \bar{u})$$

$$\Rightarrow J(u, v; \bar{u}) = \int_{\Omega} (Q'(u) \partial_x u \bar{u} + Q(u) \partial_x \bar{u}) \partial_x v$$

Et c'est une jacobienne bien calculée

## A. Calculs pour les éléments finis



Sur le segment  $[x_{i-1}; x_i]$ , seules  $\varphi_{i-1}$  et  $\varphi_i$  ont des variations non nulles.

$$\partial_x u(x) = u_{i-1} \partial_x \varphi_{i-1} + u_i \partial_x \varphi_i \quad \forall x \in [x_{i-1}; x_i]$$

$$\Leftrightarrow \partial_x u(x) = -\frac{u_{i-1}}{h_{i-1}} + \frac{u_i}{h_{i-1}}$$

On peut donc reprendre l'expression de  $J_{x_{i-1}x_i}^h$ :

$$\begin{aligned} J_{x_{i-1}x_i}^h &= -\frac{1}{h_{i-1}^2} \left\{ x_i \int_{x_{i-1}}^{x_i} Q(u) \frac{u_i - u_{i-1}}{h_{i-1}} - \int_{x_{i-1}}^{x_i} Q(u) \frac{u_i - u_{i-1}}{h_{i-1}} x \right\} u_{i-1} \\ &= -\frac{u_i - u_{i-1}}{h_{i-1}^3} \left\{ x_i \int_{x_{i-1}}^{x_i} Q(u) - \int_{x_{i-1}}^{x_i} Q(u) x \right\} u_{i-1} \end{aligned}$$

et maintenant on utilise la quadrature, ici de Simpson.

$$J_{x_{i-1}x_i}^h = -\frac{u_i - u_{i-1}}{h_{i-1}^3} \left\{ x_i \left[ \int_{x_{i-1}}^{x_i} Q(u) - \int_{x_{i-1}}^{x_i} Q(u) x \right] u_{i-1} \right\}$$

$$\begin{aligned} &\sim -\frac{u_i - u_{i-1}}{h_{i-1}^2} \left\{ x_i \left( \frac{1}{6} Q(u_{i-1}) + \frac{4}{6} Q(u_i) + \frac{2}{3} Q\left(\frac{u_i + u_{i-1}}{2}\right) \right) \right. \\ &\quad \left. - \left( \frac{1}{6} Q(u_{i-1}) x_{i-1} + \frac{4}{6} Q(u_i) x_i + \frac{2}{3} Q\left(\frac{u_i + u_{i-1}}{2}\right) \frac{x_i + x_{i-1}}{2} \right) \right\} \end{aligned}$$

Le même raisonnement conduit au terme  $J_{x_i x_{i+1}}^h$

$$\begin{aligned} J_{x_i x_{i+1}}^h &\sim -\frac{u_{i+1} - u_i}{h_i^2} \left\{ \left( \frac{1}{6} Q(u_i) x_i + \frac{4}{6} Q(u_{i+1}) x_{i+1} + \frac{2}{3} Q\left(\frac{u_i + u_{i+1}}{2}\right) \frac{x_i + x_{i+1}}{2} \right) \right. \\ &\quad \left. - x_i \left( \frac{1}{6} Q(u_i) + \frac{4}{6} Q(u_{i+1}) + \frac{2}{3} Q\left(\frac{u_i + u_{i+1}}{2}\right) \right) \right\} \end{aligned}$$

## A. Calculs pour les éléments finis

$$\begin{aligned}
J_{Rii} &= \int_{x_{i-1}}^{x_i} Q(u) \partial_x u \partial_x q_i \hat{u}_i q_i + \int_{x_i}^{x_{i+1}} Q'(u) \partial_x u \partial_x q_i \hat{u}_i q_i \\
&\quad \textcircled{a} \qquad \qquad \qquad \textcircled{b} \\
\textcircled{a} &= \int_{x_{i-1}}^{x_i} Q(u) \frac{u_i - u_{i-1}}{h_{i-1}} \left( \frac{1}{h_{i-1}} \right) \frac{x - x_{i-1}}{h_{i-1}} \hat{u}_i \\
&= \frac{1}{h_{i-1}^2} (u_i - u_{i-1}) \left( \left( \frac{1}{6} Q(u_{i-1}) x_{i-1} + \frac{1}{6} Q(u_i) x_i + \frac{2}{3} Q\left(\frac{u_i + u_{i-1}}{2}\right) \frac{x_0 + x_{i-1}}{2} \right) \right. \\
&\quad \left. - x_{i-1} \left( \frac{1}{6} Q'(u_{i-1}) + \frac{1}{6} Q'(u_i) + \frac{2}{3} Q'\left(\frac{u_i + u_{i-1}}{2}\right) \right) \right) \\
\textcircled{b} &= \int_{x_i}^{x_{i+1}} Q(u) \partial_x u \partial_x q_i \hat{u}_i q_i \\
&= \int_{x_i}^{x_{i+1}} Q(u) \frac{u_{i+1} - u_i}{h_i} \left( -\frac{1}{h_i} \right) \left( \frac{x_{i+1} - x}{h_i} \right) \hat{u}_i \\
&= -\frac{1}{h_i^3} \int_{x_i}^{x_{i+1}} Q'(u) (u_{i+1} - u_i) (x_{i+1} - x) \hat{u}_i \\
&= -\frac{1}{h_i^3} (u_{i+1} - u_i) \left\{ \int_{x_i}^{x_{i+1}} Q'(u) x_{i+1} - \int_{x_i}^{x_{i+1}} Q'(u) x \right\} \hat{u}_i \\
&\approx -\frac{1}{h_i^2} (u_{i+1} - u_i) \left\{ \left( \frac{1}{6} Q'(u_{i+1}) + \frac{1}{6} Q'(u_i) + \frac{2}{3} Q'\left(\frac{u_{i+1} + u_i}{2}\right) \right) x_{i+1} \right. \\
&\quad \left. - \left( \frac{1}{6} Q'(u_{i+1}) x_{i+1} + \frac{1}{6} Q'(u_i) x_i + \frac{2}{3} Q'\left(\frac{u_{i+1} + u_i}{2}\right) \frac{x_{i+1} + x_i}{2} \right) \right\} \hat{u}_i
\end{aligned}$$

## A. En résumé

Le résidu :

$$F_i = u_i - u_i^D$$

$$\forall i \in [1; m-1]$$

$$F_i = -\frac{1}{h_i} \left( Q(u_{i-2})/6 + Q(u_i)/6 + 2Q\left(\frac{u_i+u_{i-2}}{2}\right)/3 \right) \times u_{i-2}$$

$$+ \frac{1}{h_{i-1}} \left( Q(u_{i-1})/6 + Q(u_i)/6 + 2Q\left(\frac{u_i+u_{i-1}}{2}\right)/3 \right)$$

$$+ \frac{1}{h_i} \left( Q(u_i)/6 + Q(u_{i+2})/6 + 2Q\left(\frac{u_i+u_{i+2}}{2}\right)/3 \right) \boxed{u_i}$$

$$- \frac{1}{h_i} \left( Q(u_i)/6 + Q(u_{i+2})/6 + 2Q\left(\frac{u_i+u_{i+2}}{2}\right)/3 \right) \boxed{u_{i+2}}$$

$$F_m = u_m - u_m^D$$

Le jacobien :

$$i=0 \quad J_{0,0} = 1 \quad ; \quad \forall j > 0 : J_{ij} = 0$$

$$\forall i \in [1; m-1]$$

$$J_{i,i-2} = -\frac{1}{h_{i-1}} \left( Q(u_{i-2})/6 + Q(u_i)/6 + 2Q\left(\frac{u_i+u_{i-2}}{2}\right)/3 \right)$$

$$- \frac{u_i - u_{i-2}}{h_{i-1}^2} \left\{ x_i \left( Q(u_{i-2})/6 + Q(u_i)/6 + 2Q\left(\frac{u_i+u_{i-2}}{2}\right)/3 \right) \right.$$

$$\left. - \left( Q(u_{i-2})x_{i-2}/6 + Q(u_i)x_i/6 + 2Q\left(\frac{u_i+u_{i-2}}{2}\right)\frac{x_i+x_{i-2}}{2}/3 \right) \right\}$$

$$J_{ii} = \frac{1}{h_i} \left( \frac{1}{6}Q(u_i) + \frac{1}{6}Q(u_{i+2}) + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right) \right)$$

$$+ \frac{1}{h_i} \left( \frac{1}{6}Q(u_i) + \frac{1}{6}Q(u_{i+2}) + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right) \right)$$

$$- \frac{u_i - u_{i-2}}{h_{i-1}^2} \left( \left( Q(u_{i-2})x_{i-2}/6 + Q(u_i)x_i/6 + 2Q\left(\frac{u_i+u_{i-2}}{2}\right)\frac{x_i+x_{i-2}}{2}/3 \right) \right.$$

$$\left. - x_{i-2} \left( Q(u_{i-2})/6 + Q(u_i)/6 + 2Q\left(\frac{u_i+u_{i-2}}{2}\right)/3 \right) \right)$$

$$- \frac{u_{i+2} - u_i}{h_i^2} \left( \left( Q(u_{i+2})/6 + Q(u_i)/6 + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right) \right) x_{i+2} \right.$$

$$\left. - \left( \frac{1}{6}Q(u_{i+2})x_{i+2} + \frac{1}{6}Q(u_i)x_i + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right)\frac{x_i+x_{i+2}}{2}/3 \right) \right)$$

$$J_{i,i+2} = -\frac{1}{h_i} \left( \frac{1}{6}Q(u_i) + \frac{1}{6}Q(u_{i+2}) + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right) \right)$$

$$- \frac{u_{i+2} - u_i}{h_i^2} \left( \left( \frac{1}{6}Q(u_i)x_i + \frac{1}{6}Q(u_{i+2})x_{i+2} + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right)\frac{x_i+x_{i+2}}{2}/3 \right) \right.$$

$$\left. - x_i \left( \frac{1}{6}Q(u_i) + \frac{1}{6}Q(u_{i+2}) + \frac{2}{3}Q\left(\frac{u_i+u_{i+2}}{2}\right) \right) \right)$$

$i=m$  :  $J_{m,m} = 1$

Eléments P1, quadrature de Simpson, en 1.D

On implémente sur maillage uniforme  
(tous les nœuds sont également  
espacés entre eux)

## A. Tentatives d'assemblage du résidu et de la jacobienne : éléments P1. quadrature de Simpson

```

25 def F(u):
26     "Residu"
27     res = np.zeros_like(u)
28
29     for i in range(len(res)):
30         if i==0:
31             res[i] = u[i] - 0
32         elif i==(len(res)-1):
33             res[i] = u[i] - 1
34         else:
35             res[i] = 1/h * (1/6*Q(u[i-1]) + 1/6*Q(u[i]) + 2/3*Q((u[i-1]+u[i])/2)) * u[i-1] \
36             - 1/h * (1/6*Q(u[i-1])+1/6*Q(u[i])+2/3*Q((u[i-1]+u[i])/2) + 1/6*Q(u[i])+1/6*Q(u[i+1])+2/3*Q((u[i]+u[i+1])/2)) * u[i] \
37             + 1/h * (1/6*Q(u[i]) + 1/6*Q(u[i+1]) + 2/3*Q((u[i]+u[i+1])/2)) * u[i+1]
38
39     return res
40
41 def J(u):
42     "Variations du residu."
43     jac = np.zeros((len(u),len(u)))
44
45     for i in range(len(jac)):
46         if i==0:
47             jac[i, i] = 1
48         elif i==len(jac)-1:
49             jac[i, i] = 1
50         else:
51             jac[i, i-1] = -1/h*(1/6*Q(u[i-1])+1/6*Q(u[i])+2/3*Q((u[i]+u[i-1])/2)) \
52             - (u[i]-u[i-1])/h*(x[i]*(1/6*dQ(u[i-1])+1/6*dQ(u[i])+2/3*dQ((u[i]+u[i-1])/2)) \
53             - (1/6*dQ(u[i-1])*x[i-1]+1/6*dQ(u[i])*x[i]+2/3*dQ((u[i]+u[i-1])/2)*(x[i]+x[i-1])/2))
54
55             jac[i, i] = 1/h*((1/6*Q(u[i])+1/6*Q(u[i-1])+2/3*Q((u[i]+u[i-1])/2))+(1/6*Q(u[i])+1/6*Q(u[i+1])+2/3*Q((u[i+1]+u[i])/2)) \
56             - (u[i]-u[i-1])/h*(1/6*dQ(u[i-1])*x[i-1]+1/6*dQ(u[i])*x[i]+2/3*dQ((u[i]+u[i-1])/2)*(x[i]+x[i-1])/2 \
57             - x[i-1]*(1/6*dQ(u[i])+1/6*dQ(u[i-1])+2/3*dQ((u[i]+u[i-1])/2))) \
58             - (u[i+1]-u[i])/h*(x[i+1]*(1/6*dQ(u[i+1])+1/6*dQ(u[i])+2/3*dQ((u[i+1]+u[i])/2)) \
59             - (1/6*dQ(u[i+1])*x[i+1]+1/6*dQ(u[i])*x[i]+2/3*dQ((u[i+1]+u[i])/2)*(x[i+1]+x[i])/2)))
60
61
62             jac[i, i+1] = -1/h*(1/6*Q(u[i])+1/6*Q(u[i+1])+2/3*Q((u[i]+u[i+1])/2)) \
63             - (u[i+1]-u[i])/h*((1/6*dQ(u[i])*x[i]+1/6*dQ(u[i+1])*x[i+1]+2/3*dQ((u[i]+u[i+1])/2)*(x[i+1]+x[i])/2)) \
64             - x[i]*(1/6*dQ(u[i])+1/6*dQ(u[i+1])+2/3*dQ((u[i+1]+u[i])/2)))
65
66     return jac

```

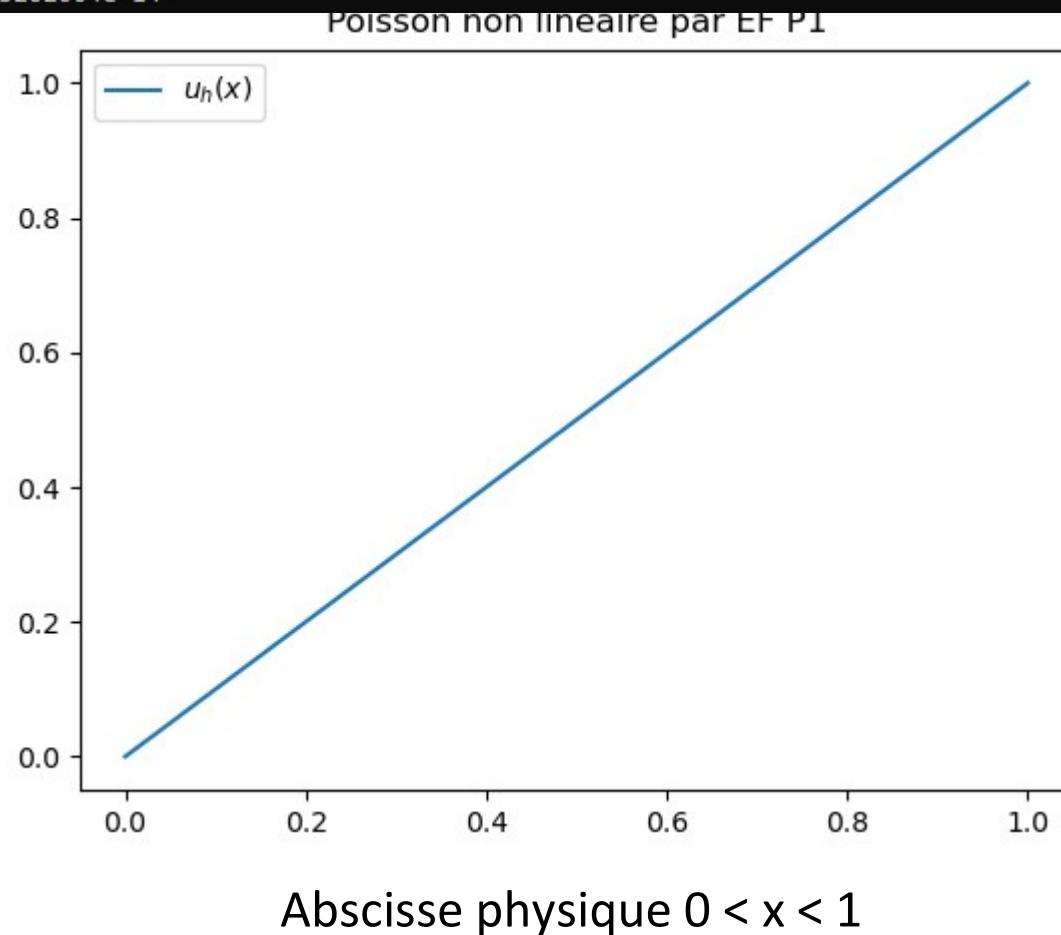
Impossible de ne pas se tromper. Décomposer en routines élémentaires, c.f la structure du code exemple pour l'élasticité linéaire homogène isotrope slide 27.

## A. Algorithme de Newton (commun)

```
68  ite = 10
69  tol = 1e-10
70  up = np.ones_like(x)
71
72  for i in range(ite):
73      res = F(up)
74      jac = J(up)
75      u = up - np.linalg.inv(jac) @ res
76
77      eps = np.max(np.abs(res))
78      print(f"Ite {i+1}\terr {eps}")
79      if(eps < tol):
80          break
81
82  up = u
83
84 plt.plot(x, u, label="$u_h(x)$")
85 plt.legend()
86 plt.title("Poisson non linéaire par EF P1")
87 plt.show()
```

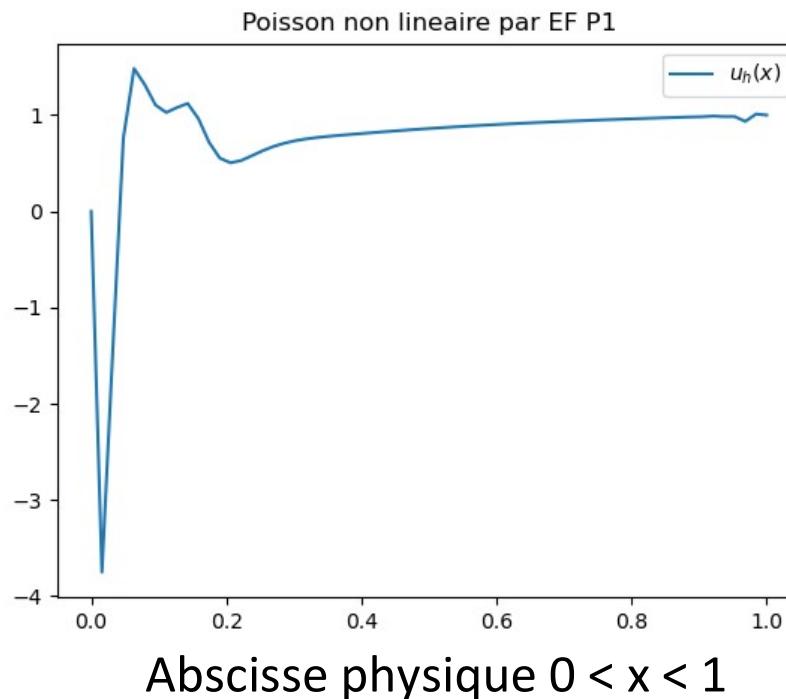
## A. Résultats de Newton sur le cas linéaire

```
(pynum) C:\Users\jvalentin>python C:\Users\jvalentin\Desktop\WORKSPACE\221103\elements_finis\newton_poisson_lineaire.py
Ite 1    err 1.0
Ite 2    err 1.4210854715202004e-14
```



## A. Résultats de l'algorithme sur le cas réellement non linéaire

```
(pynum) C:\Users\jvalentin>python C:\Users\jvalentin\Desktop\WORKSPACE\221103\elements_finis\newton_poisson.py
Ite 1 err 1.9310489627949945
Ite 2 err 1.2778755059525144
Ite 3 err 1.4067287228536838
Ite 4 err 2.5919010394994952
Ite 5 err 4.890724574908283
Ite 6 err 9.451670894957807
Ite 7 err 18.985044267991512
Ite 8 err 42.59142082027987
Ite 9 err 168.93883002609192
Ite 10 err 822.4208406985633
```



« Plutôt éloignée »  
de ce qu'on attend.

Erreur dans  
l'implémentation de  
la jacobienne

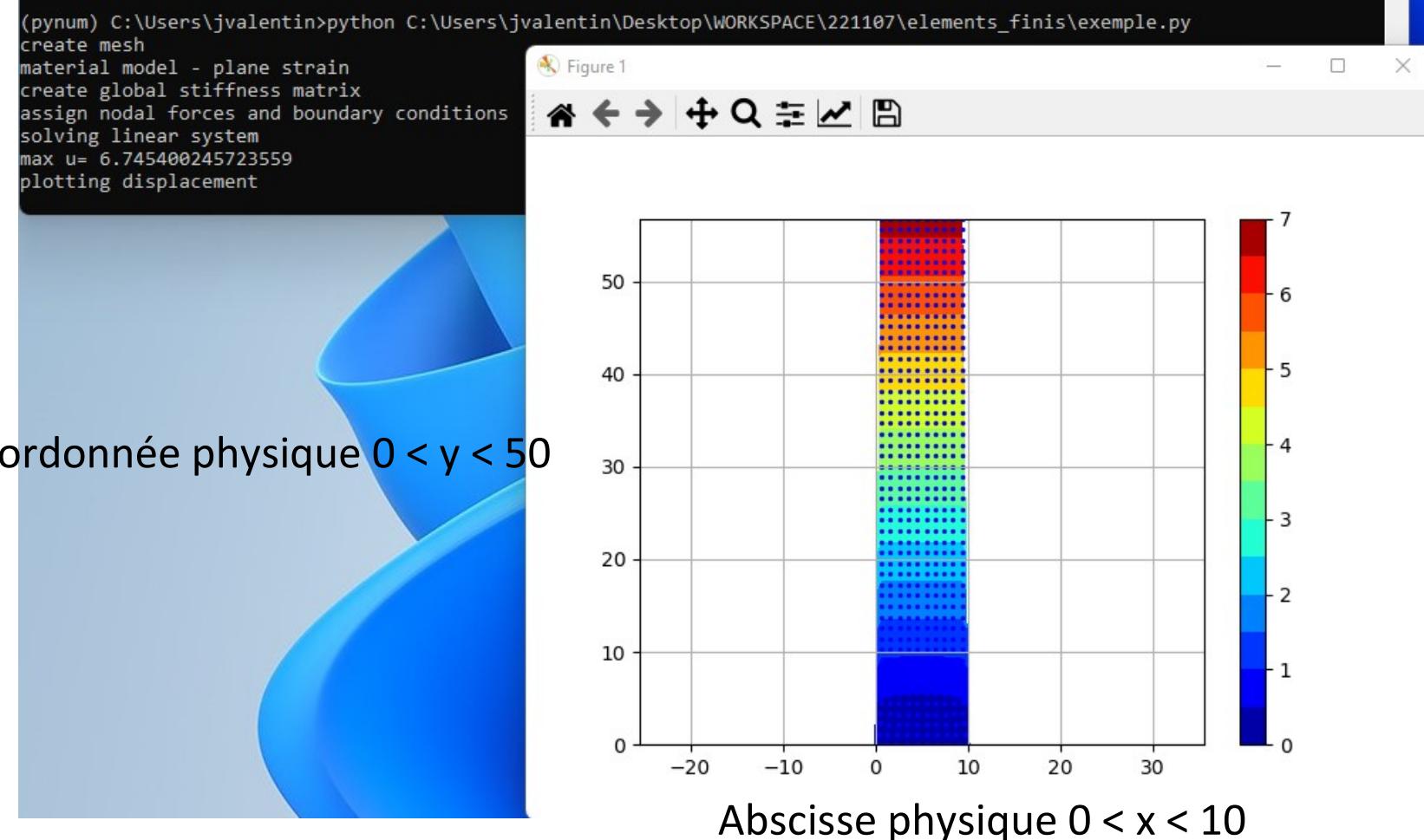
# A. Programme d'exemple pour l'élasticité homogène linéaire en 2.D

```

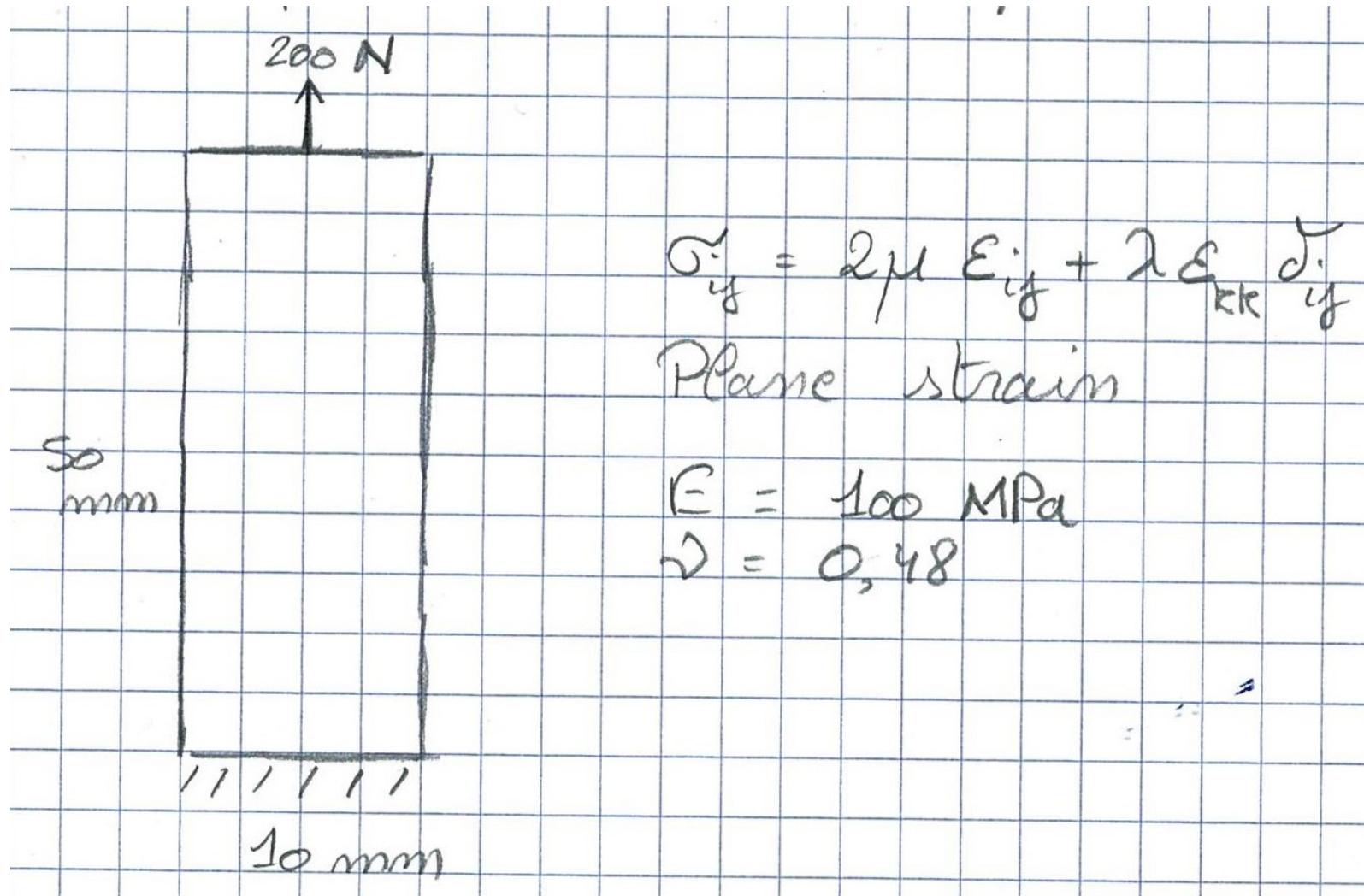
1 import numpy as np
2 import math
3 from matplotlib import pyplot as plt
4 def shape(xi):
5     x,y = tuple(xi)
6     N = [(1.0-x)*(1.0-y), (1.0+x)*(1.0-y), (1.0+x)*(1.0+y), (1.0-x)*(1.0+y)]
7     return 0.25*np.array(N)
8 def gradshape(xi):
9     x,y = tuple(xi)
10    dN = [(-1.0-y), (1.0-y), (1.0+y), -(1.0+y)],
11        [-1.0-x], -(1.0+x), (1.0+x), (1.0-x)]
12    return 0.25*np.array(dN)
13 #####
14 print('create mesh')
15 # input
16 mesh_ex = 9
17 mesh_ey = 49
18 mesh_lx = 10.0
19 mesh_ly = 50.0
20 # derived
21 mesh_nx = mesh_ex + 1
22 mesh_ny = mesh_ey + 1
23 num_nodes = mesh_nx * mesh_ny
24 num_elements = mesh_ex * mesh_ey
25 mesh_hx = mesh_lx / mesh_ex
26 mesh_hy = mesh_ly / mesh_ey
27 nodes = []
28 for y in np.linspace(0.0, mesh_ly, mesh_ny):
29     for x in np.linspace(0.0, mesh_lx, mesh_nx):
30         nodes.append([x,y])
31 nodes = np.array(nodes)
32 conn = []
33 for j in range(mesh_ey):
34     for i in range(mesh_ex):
35         n0 = i + j*mesh_nx
36         conn.append([n0, n0 + 1, n0 + 1 + mesh_nx, n0 + mesh_nx])
37 #####
38 print ('material model - plane strain')
39 E = 100.0
40 v = 0.48
41 C = E/(1.0+v)/(1.0-2.0*v) * np.array([[1.0-v, v, 0.0],
42                                         [v, 1.0-v, 0.0],
43                                         [0.0, 0.0, 0.5-v]])
44 #####
45 print('create global stiffness matrix')
46 K = np.zeros((2*num_nodes, 2*num_nodes))
47 q4 = [[x/math.sqrt(3.0),y/math.sqrt(3.0)] for y in [-1.0,1.0] for x in [-1.0,1.0]]
48 B = np.zeros((3,8))
49 for c in conn:
50     xIe = nodes[c,:]
51     Ke = np.zeros((8,8))
52     for q in q4:
53         dN = gradshape(q)
54         J = np.dot(dN, xIe).T
55         dN = np.dot(np.linalg.inv(J), dN)
56         B[0,0::2] = dN[0,:]
57         B[1,1::2] = dN[1,:]
58         B[2,0::2] = dN[1,:]
59         B[2,1::2] = dN[0,:]
60         Ke += np.dot(np.dot(B.T,C),B) * np.linalg.det(J)
61     for i,I in enumerate(c):
62         for j,J in enumerate(c):
63             K[2*I,2*j] += Ke[2*i,2*j]
64             K[2*I+1,2*j] += Ke[2*i+1,2*j]
65             K[2*I+1,2*j+1] += Ke[2*i+1,2*j+1]
66             K[2*I,2*j+1] += Ke[2*i,2*j+1]
67 #####
68 print('assign nodal forces and boundary conditions')
69 f = np.zeros((2*num_nodes))
70 for i in range(num_nodes):
71     if nodes[i,1] == 0.0:
72         K[2*i,:] = 0.0
73         K[2*i+1,:] = 0.0
74         K[2*i,2*i] = 1.0
75         K[2*i+1,2*i+1] = 1.0
76     if nodes[i,1] == mesh_ly:
77         x = nodes[i,0]
78         f[2*i+1] = 20.0
79         if x == 0.0 or x == mesh_lx:
80             f[2*i+1] *= 0.5
81 #####
82 print('solving linear system')
83 u = np.linalg.solve(K, f)
84 print('max u= ', max(u))
85 #####
86 print('plotting displacement')
87 ux = np.reshape(u[0::2], (mesh_ny,mesh_nx))
88 uy = np.reshape(u[1::2], (mesh_ny,mesh_nx))
89 xvec = []
90 yvec = []
91 res = []
92 for i in range(mesh_nx):
93     for j in range(mesh_ny):
94         xvec.append(i*mesh_hx + ux[j,i])
95         yvec.append(j*mesh_hy + uy[j,i])
96         res.append(uy[j,i])
97 t = plt.tricontourf(xvec, yvec, res, levels=14, cmap=plt.cm.jet)
98 plt.scatter(xvec, yvec, marker='o', c='b', s=2)
99 plt.grid()
100 plt.colorbar(t)
101 plt.axis('equal')
102 plt.show()
103

```

## A. Résultat de l'exemple



## A. Analyse du programme d'exemple : setup



## A. Analyse du programme d'exemple

- Problème d'élasticité linéaire, homogène et isotrope en 2.D
- Maillage quadrangulaire structuré du rectangle  $[0, 10] \times [0, 50]$
- Pilotage de la discrétisation en nombre d'éléments, longueur et largeur
- Interpolation linéaire 2.D sur les nœuds
- Matériau :
  - Module d'Young  $E = 100$
  - Coefficient de Poisson  $\nu = 0.48$
- Approximation des intégrales par points de Gauss  $-1/\sqrt{3}$ ,  $1/\sqrt{3}$  sur l'élément de référence  $[-1; 1] \times [-1; 1]$  (exacte pour les polynômes de degré  $2n-1$  où  $n$  est le nombre de points, ici sur les polynômes de degré  $2*2-1 = 3$ )
- <https://polymerfem.com/full-finite-element-solver-in-100-lines-of-python/>
- <https://polymerfem.com/full-finite-element-solver-in-200-lines-of-python/>

## A. Adaptation en 1D à venir

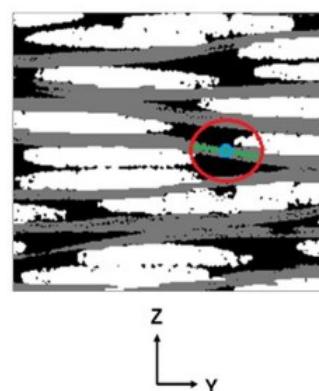
## B. Maillage volumique

- Le but est d'extraire d'un empilement d'images segmenté au format TIFF un maillage 3.D de volumes.
- Le volume est distingué en trois valeurs (niveaux de gris)

The following data is provided:	
File names:	Segmented data
	VPB_meso_8microns_700x680x345_seg.tif
	VPB_meso_8microns_700x680x345_seg_theta.raw
	VPB_meso_8microns_1923x1553x345_seg.tif
	VPB_meso_8microns_1923x1553x345_seg_theta.raw
Raw data of X-ray-scans	
	VPB_meso_8microns_700x680x345.tif
	VPB_meso_8microns_1923x1553x345.tif

RAW : format d'image brut, non compressé.

TIFF : format d'images labellisées



Global mean value for microscale permeability:

$$\mathbf{K} = \begin{bmatrix} K_{xx} & 0 & 0 \\ 0 & K_{yy} & 0 \\ 0 & 0 & K_{zz} \end{bmatrix}$$

Input for white voxels with  $\theta=0$ :

$$= \begin{bmatrix} 9.4E-13 & 0 & 0 \\ 0 & 3.2E-14 & 0 \\ 0 & 0 & 3.2E-14 \end{bmatrix}$$

Input for grey voxels with  $\theta=0$ :

$$= \begin{bmatrix} 3.2E-14 & 0 & 0 \\ 0 & 9.4E-13 & 0 \\ 0 & 0 & 3.2E-14 \end{bmatrix}$$

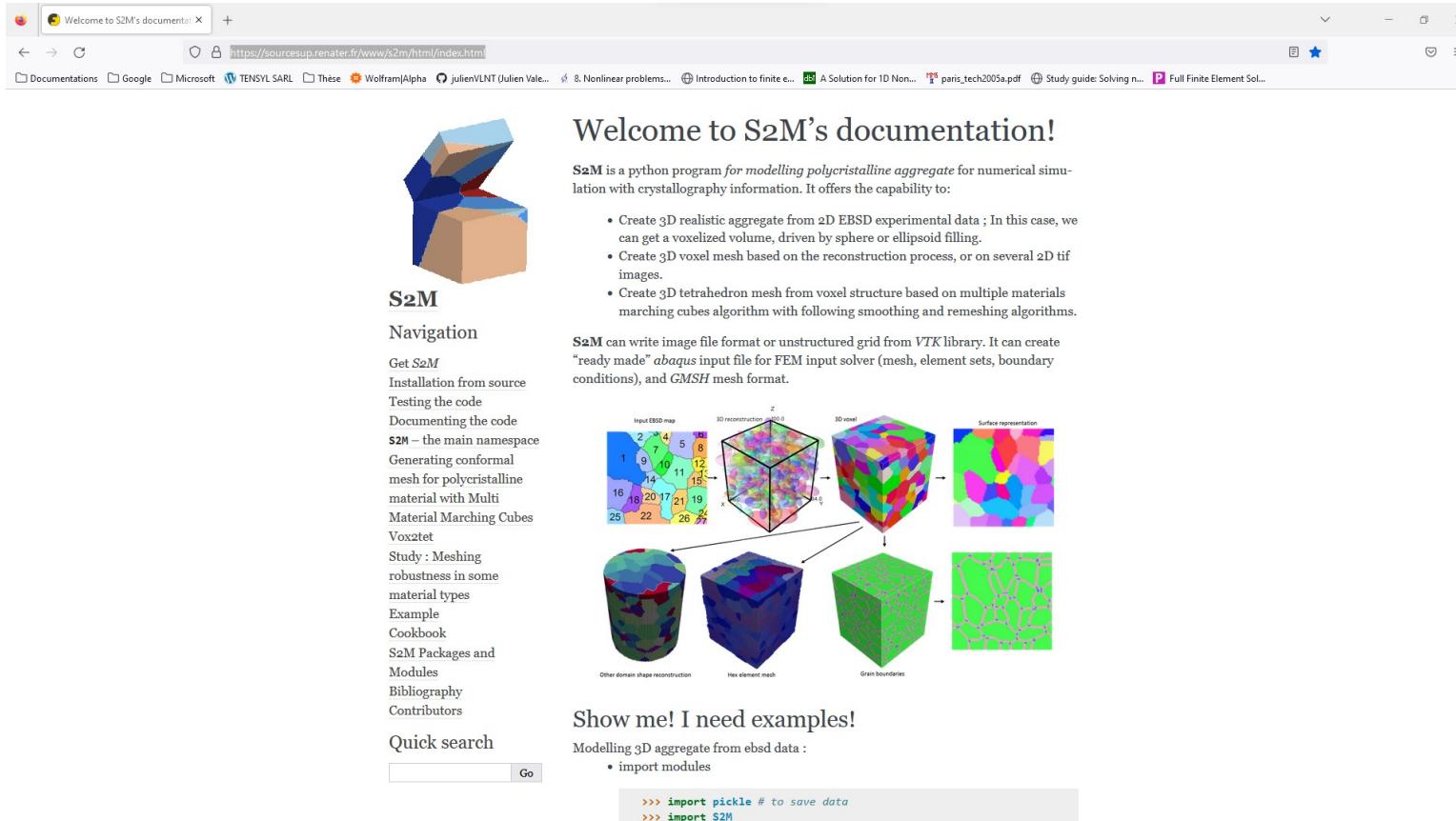
You can adjust permeability tensor if  $\theta \neq 0$

## B. Librairie iso2mesh (MATLAB/Octave)

- <https://iso2mesh.sourceforge.net/cgi-bin/index.cgi?Home>
- => Impossible de faire fonctionner ne serait-ce que les exemples livrés avec la librairie. Tests effectués sur MATLAB (nécessite la librairie de traitement d'images, payante) et sur Octave (équivalent open-source de MATLAB), qui dispose pourtant de la même librairie de traitement d'images.

## B. Librairie s2m

- <https://sourcesup.renater.fr/www/s2m/html/index.html>



The screenshot shows the "Welcome to S2M's documentation!" page. On the left, there is a navigation sidebar with links to various S2M features and documentation. The main content area displays a flow diagram illustrating the process of generating different types of meshes from an input EBSD map. The diagram shows the progression from an "Input EBSD map" (with numbered regions) through "3D reconstruction" and "3D voxel" stages to a "Surface representation". Below this, arrows point to "Other domain shape reconstruction", "Hex element mesh", and "Grain boundaries". A code snippet at the bottom shows how to import S2M modules.

Welcome to S2M's documentation!

**S2M** is a python program for modelling polycrystalline aggregate for numerical simulation with crystallography information. It offers the capability to:

- Create 3D realistic aggregate from 2D EBSD experimental data ; In this case, we can get a voxelized volume, driven by sphere or ellipsoid filling.
- Create 3D voxel mesh based on the reconstruction process, or on several 2D tif images.
- Create 3D tetrahedron mesh from voxel structure based on multiple materials marching cubes algorithm with following smoothing and remeshing algorithms.

S2M can write image file format or unstructured grid from VTK library. It can create "ready made" abaqus input file for FEM input solver (mesh, element sets, boundary conditions), and GMSH mesh format.

Input EBSD map  
2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

3D reconstruction  
x z

3D voxel

Surface representation

Other domain shape reconstruction

Hex element mesh

Grain boundaries

Show me! I need examples!

Modelling 3D aggregate from ebsd data :

- import modules

```
>>> import pickle # to save data
>>> import S2M
```

## B. Usage de s2m : prétraitement, mise en place d'un offset sur les niveaux de gris

```
1  #! -*- coding: utf-8 -*-
2  # env: s2m
3
4  # IMPORTS
5  # - numpy    : manipulations numériques
6  # - S2M      : mailleur 3D
7  # - skimage : manipulation d'image (offsets)
8  import numpy as np
9  import S2M.vox2tet as v2t
10 from skimage import io as skio
11
12 # Lecture de l'image
13 img = skio.imread("./data/VPB_meso_8microns_700x680x345_seg.tif", plugin="tifffile")
14
15 # Création de l'offset pour s2m
16 img = np.where(img==0, 100, img)
17 img = np.where(img==255, 200, img)
18
19 # Troncature : on ne retient qu'un volume de 50x50x50 pour les tests
20 img = img[:100, :100, :100]
21
22 # Sauvegarde de l'image modifiée
23 skio.imsave("./data/img.tif", img, plugin="tifffile")
24
25 # PREPARATION DU MAILLEUR VOLUMETRIQUE
26 repertoire_donnees = "./data/"
27 nom_image = "img"
28 extension = ".tif"
29 input_img_file = repertoire_donnees + nom_image + extension
30
31 mesher = v2t.get_settings()                      # VALEURS PAR DEFAUT
```

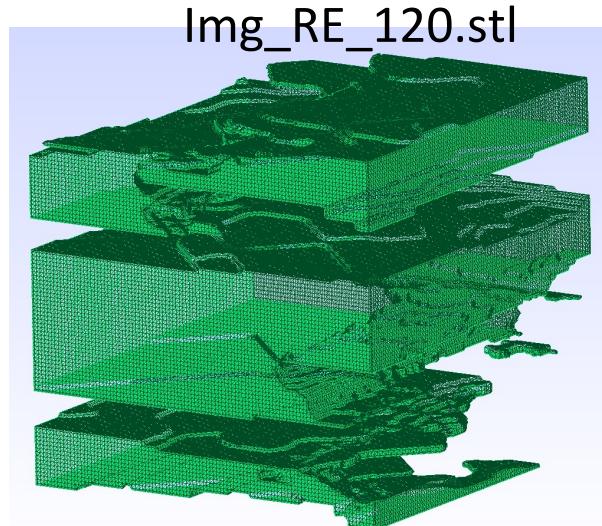
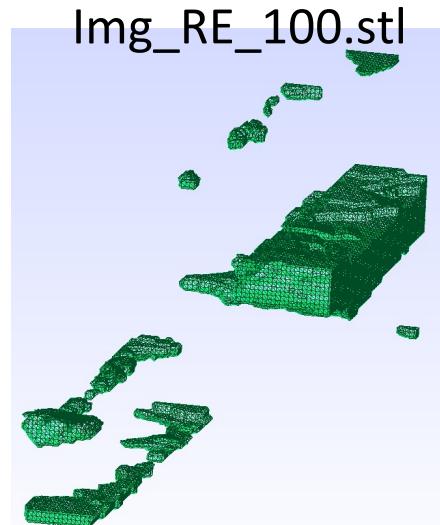
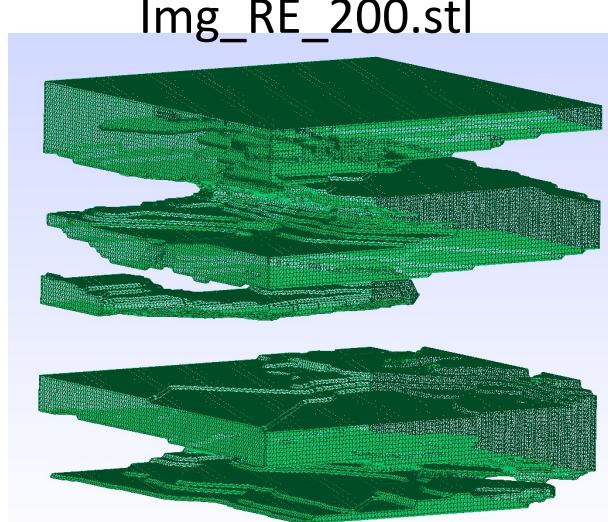
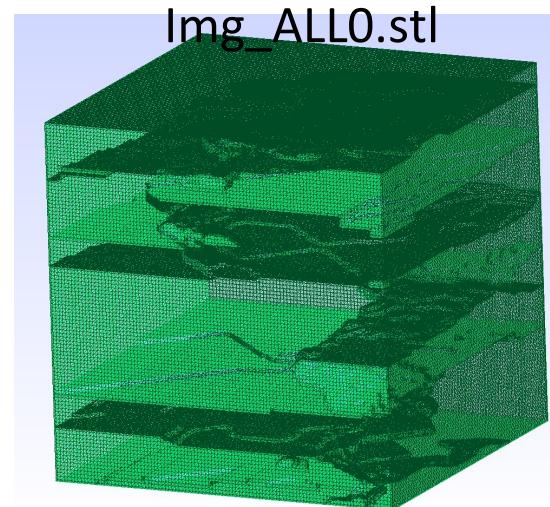
## B. Liste exhaustive (selon les exemples) des paramètres, enregistrés dans un fichier .JSON

```

25 # PREPARATION DU MAILLAGE VOLUMETRIQUE
26 repertoire_donnees = "./data/"
27 nom_image = "img"
28 extension = ".tif"
29 input_img_file = repertoire_donnees + nom_image + extension
30
31 mesher = v2t.get_settings()                                # VALEURS PAR DEFAUT
32
33 mesher.brep_degree = 3                                    # 3
34 mesher.brep_smoothness = 7                               # 7
35 mesher.connectivity = 3                                 # 3
36 mesher.do_2x2patterns = True                            # True
37 mesher.do_abaqus_verification = True                   # True
38 mesher.do_brep2iges = False                            # False
39 mesher.do_brep2iges_split = False                      # False
40 mesher.do_diagonal_connectivity = True                # True
41 mesher.do_init_surf_load = False                       # False
42 mesher.do_initial_surface = True                      # True
43 mesher.do_preremesh_load = False                     # False
44 mesher.do_preremesh_steps = False                    # True
45 mesher.do_print_redirect = False                     # False
46 mesher.do_remeshing = False                           # True
47 mesher.do_remove_small = False                        # True
48 mesher.do_save_init_grains_inp = True                # False
49 mesher.do_save_init_grains_stl = True                # False
50 mesher.do_save_init_interfaces = True                # False
51 mesher.do_save_remesh_grains_inp = True              # True
52 mesher.do_save_remesh_grains_stl = True              # True
53 mesher.do_save_remesh_interfaces = True              # False
54 mesher.do_save_smooth_grains_inp = True              # False
55 mesher.do_save_smooth_grains_stl = True              # False
56 mesher.do_save_smooth_interfaces = True              # False
57 mesher.do_save_vtk = True                            # True
58 mesher.do_show_hist = False                          # False
59 mesher.do_tetgen_meshing = True                     # True
60 mesher.do_x_rotation = True                         # True
61 mesher.Lmax = 40.0                                  # 40.0
62 mesher.input_img_file = input_img_file             # "JMA_30.tif"
63 mesher.max_D2self = 0.8                            # 0.8
64 mesher.max_remove_size = 5                         # 64
65 mesher.min_D2other = 0.3                           # 0.3
66 mesher.min_dangle_boundary = 20                  # 20
67 mesher.min_dangle_internal = 40                 # 40
68 mesher.ncpus=1                                     # 1
69 mesher.n_remesh_itr = 0                           # 7
70 mesher.n_smooth_steps = 0                         # 7
71 mesher.out_path_base = "./tmp/img"               # "out/out_retest_JMA"
72 mesher.smooth_alpha = 0.5                         # 0.5
73
74 # Creation du maillage volumetrique
75 v2t.generate(mesher)
76

```

## B. Exploration des fichiers intermédiaires



## B. Obtention d'un maillage 3.D

- A l'aide de GMSH
- Créer une nouvelle surface physique, sélectionner la surface définie par le fichier STL ouvert
- Dans l'onglet *mesh*, sélectionner 3D puis exporter dans votre format préféré (.msh en ce qui nous concerne).

## B. Import du maillage en FreeFEM++ (A. GUILLOUX)

```

Public > JulienValentin > 5-PROJETS > 3-benchmark_permeabilite_numerique > 2022 > maillage_volumetrique > freefem++ > 221107_LectureMaillage3D_main.edp
 1 //Lecture du fichier 3D généré dans GMSH avec un export *.msh Version 2 format ASCII
 2 load "medit"
 3 load "msh3"
 4 load "gmsh"
 5 load "iovtc"
 6 int[int] fforder=[1];//pour export vtk
 7
 8 //import du maillage
 9 mesh3 Th = gmshload3("FormMSH_V2.msh"); // GMSH version 2 ascii.
10 //mesh Th=square(10,10);
11 //Affiche le maillage
12 plot(Th,cmm="Th maillage 3D");
13
14
15
16
17 //Creation d'un champ pour la visualisation
18 fespace Vh(Th,P1);
19 Vh u1;
20
21 u1=y;
22
23 //infos maillage
24 int NbTriangles = Th.nt;
25
26 cout<< "Nombre de Tetraedre ="<<NbTriangles<<endl;
27 //Détermination des dimensions du maillage
28 real[int] bb[6];
29 boundingbox(Th,bb);
30 // bb[0] = xmin, bb[1] = xmax, bb[2] = ymin, bb[3] =ymax
31 real ymin = bb[2];
32 real ymax = bb[3];
33
34 cout << "boundingbox:" << endl;
35 cout <<"xmin = " << bb[0]
36 <<, xmax = " << bb[1]
37 <<, ymin = " << bb[2]
38 <<, ymax = " << bb[3]
39 <<, zmin = " << bb[4]
40 <<, zmax = " << bb[5] << endl;
41
42
43 real eps=1e-6;
44
45 Th = change(Th,flabel = ((y<ymin+eps) ? 10 : label) );
46 Th = change(Th,flabel = ((y>ymax-eps) ? 20 : label) );
47
48
49
50
51 //Export le maillage (visible dans paraview)
52 savevtk("maillage.vtu",Th,u1,order=fforder,dataname="Champ");
53
54 //Prochaine étapes:
55 //-- conserver uniquement un seul volume fermé (supprimer dans GMSH)
56 //-- mettre des labels sur les surfaces extérieures du volume maillé

```

## B. Import du maillage en FreeFEM++ (A. GUILLOUX)



The image shows a Windows desktop with two windows open:

- Left Window (Command Prompt):**

```

38 :           <<" , ymax = " << bb[3]
39 :           <<" , zmin = " << bb[4]
40 :           <<" , zmax = " << bb[5] <<endl;
41 :
42 :
43 : real eps=1e-6;
44 :
45 : Th = change(Th,flabel = ((y<ymin+eps) ? 10 : label) );
46 : Th = change(Th,flabel = ((y>ymax-eps) ? 20 : label) );
47 :
48 :
49 :
50 :
51 : //Export le maillage (visible dans paraview)
52 : savevtk("maillage.vtu",Th,u1,order=fforder,dataname="Champ");
53 :
54 : //Prochaine |@tapes:
55 : //-- conserver uniquement un seul volume ferm|@ (supprimer dans GMSH)
56 : //-- mettre des labels sur les surfaces ext|@rieures du volume maill|@ sizestack + 1024 =1496 ( 472 )
100443 tetrahedrons
36732 triangles
137175 numElements
Plot:: Sorry no ps version for this type of plot 5
-- FESpace: Nb of Nodes 26148 Nb of DoF 26148
Nombre de Tetraedre =100443
boundingbox:
xmin = 31, xmax = 100, ymin = 0, ymax = 100, zmin = 7, zmax = 100
times: compile 1.533s, execution 2.08s, mpirank:0

```
- Right Window (FreeFEM++):**

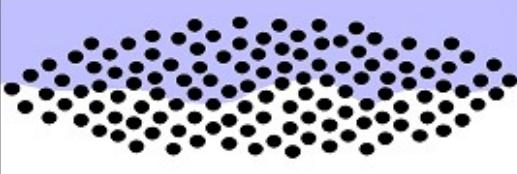
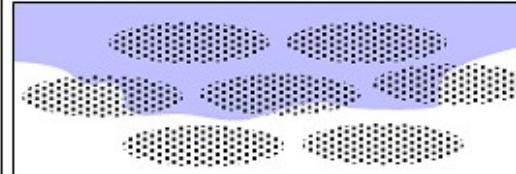
FreeFem++ / Program ended; enter ESC to exit  
**Th maillage 3D**



# Résumé de la chaîne de calcul – V1

- Data in Benchmark
  - Fichier .raw ou .tiff
  - Image stack (3D, 3 niveaux de couleur)
- Python bibliothèque S2M
  - Import fichiers .tiff
  - maillage des surfaces à l'intersection isocouleur
  - export fichier .stl
- GMSH
  - Import fichier .stl
  - Création du maillage 3D
  - Export fichier .msh (ASCII GMSH V2)
- Freefem
  - Import maillage 3D (.msh)
  - Change label (critère géométrique, « flabel »)
  - Equation Stocks
  - $Qv \rightarrow K$
  - (option/obligatoire? Calcul MPI)

## B. Caractérisations des écoulements

Echelle	Microscopique	Mésoscopique	Macroscopique
Schéma			
Caractéristiques principales	Milieu triphasique : solide (fibres), liquide (résine), gaz (air)	2 milieux : mèche et espace intermèche 4 zones : liquide, air, mèche saturée, mèche insaturée	Milieu poreux homogénéisé
Grandeurs caractéristiques	Diamètre d'une fibre : ~1-10 um Espace intra-mèche (= inter-fibre) : ~1-10 um	Largeur mèche : ~ 1-10 mm Epaisseur mèche : ~ 0,1 mm Espace inter mèches : ?	Tenseur de perméabilité
Domaine d'étude	Quelques dizaines de fibres	Quelques mèches	Quelques cm à quelques mètres
Nombre caractéristique			
Reynolds			
Conséquences sur l'écoulement	Effets visqueux >> effets inertIELS		
Modèles d'écoulement / équations utilisées	Stokes	Stokes + Darcy (selon zone) ou : Brinkman (approche monolithique)	Milieu poreux : Darcy
Prise en compte capillarité	Terme de force à l'interface résine/air, dépend de la tension de surface, de la courbure, ... Possible prise en compte d'un angle statique à l'interface solide/liquide/gaz	Dans les mèches : pression capillaire (saut de pression à l'interface saturé/insaturé ?) A priori, canaux inter mèches trop grands pour qu'il y ait de la capillarité	Pression capillaire (saut de pression à l'interface saturé/insaturé ?)
Travaux dans biblio			

## D. Modélisation à l'échelle macroscopique

Non traité sur la période.

## D. Caractérisation des enroulements par les durées d'écoulement de la résine dans un pli (1/2)

### Durée d'écoulement de la résine à travers une mèche :

$$t_f = t^* \frac{4\kappa_z \mu h_0^2}{A_s r_d^2}$$

Diagram illustrating the components of the equation:

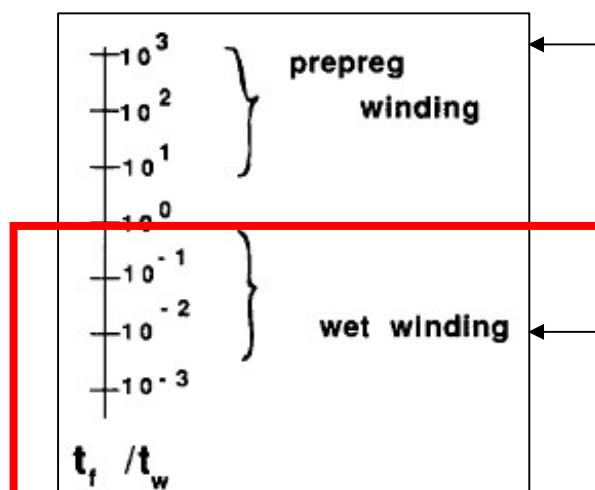
- Constante de Kozeny
- Viscosité de la résine
- Épaisseur de la mèche
- Rayon d'une fibre
- « Fiber bed spring constant »
- Temps adimensionnel, chute significative de la pression  
 $t^* = 0,5$

(Cai et al., 1992)

## D. Caractérisation des enroulements par les durées d'écoulement de la résine dans un pli (2/2)

Mèche		AS4 3k	T700 12K
Epaisseur d'une couche	m	1.52E-04	1.20E-04
Viscosité	Pa.s	10	6.20E-01 620 mPa.s à 25°C selon FT
Nombre de mèches		65	
Temps d'enroulement	s	90	120 Durée 1 couche / 2
Rayon de fibre	m	3.50E-06	3.50E-06
Constante de Kozeny		0.2	0.2
Constante "Fiber spring"	Pa	4.14E+02	4.14E+02 Déterminée expérimentalement
Constante de temps d'écoulement	sec	36.7	1.4
Temps adimensionnel		0.204	0.006

REF: Données tvx 2016



Très peu d'écoulement.  
Matériau : approximation linéaire élastique  
Rigidité des fibres + pression de résine

Temps d'écoulement très faible devant temps d'enroulement.  
L'état de déformation final est principalement lié aux fibres.  
Pas de pression de résine ?

## Bibliographie

- Cf slides 30, 33, 34 pour respectivement les codes éléments finis en exemple, la documentation de iso2mesh (MATLAB) et s2m (Python)