

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

Point avancement n°01 du 14/10/2022

# 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.

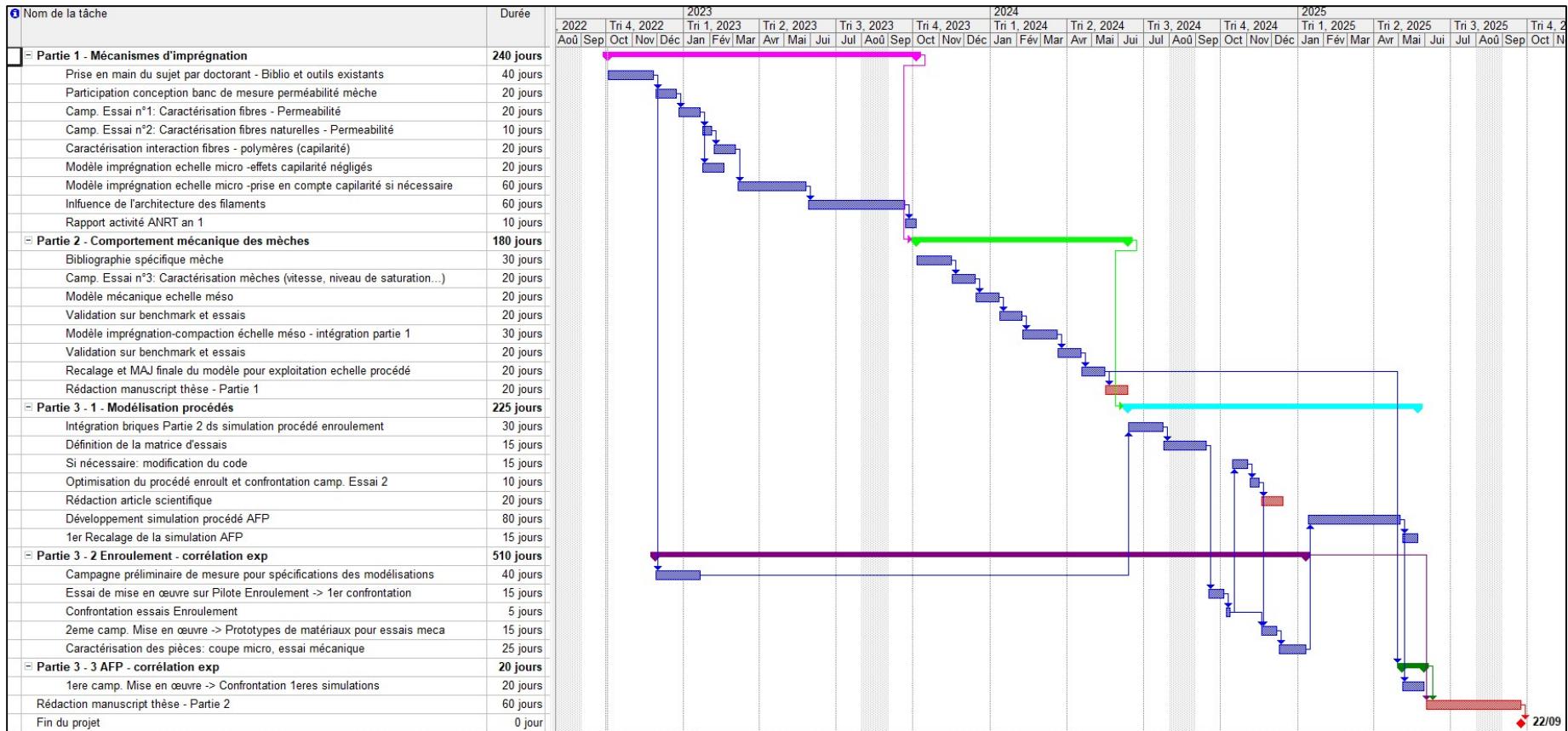
## Rappel du sujet

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.

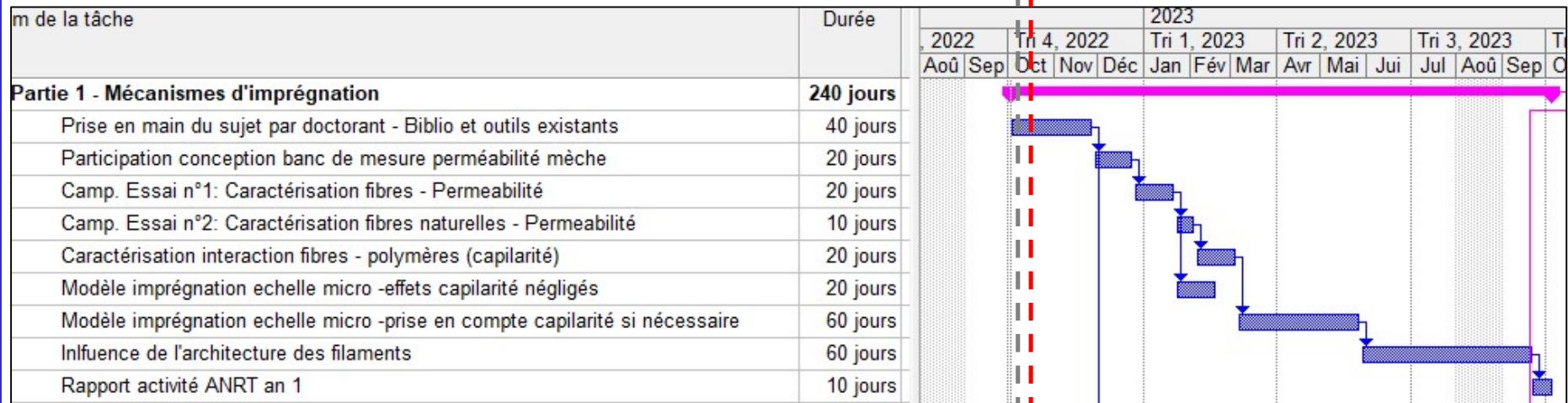
Le travail est décomposé en 3 parties:

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



Ajouter la prédiction du tenseur de perméabilité pour le benchmark

Pt 2  
Pt 1

# Bilan de la dernière réunion

VIDE.



## Ordre du jour

- A. Mathématiques
- B. Physiques
- C. Outils et méthodes informatiques
- D. Modélisation à l'échelle macroscopique
- E. Modélisation à l'échelle mésoscopique
- F. Modélisation à l'échelle microscopique
- G. Actions à venir
- H. Bibliographie

## A. Mathématiques

- Résolution de problèmes non linéaires par méthode de Newton-Raphson sur l'équation de Poisson linéaire, quasi-linéaire et non linéaire
- Ecriture du problème d'écrasement d'un milieu poreux saturé en coordonnées cylindriques

## Exemple de discrétisation par différences finies

- Problème de diffusion linéaire (équation de Poisson)

$$u''(x) = f(x) \quad \Omega$$

- Schéma à trois points de la dérivée seconde

$$u_i'' \sim \frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

# Elements d'analyse du schéma pour la dérivée seconde

- Erreur de troncature locale (notée tau)
  - u désigne la solution continue interpolée sur le maillage
  - uh désigne la solution du système linéaire
- Consistance
- Stabilité
- Un schéma est convergent s'il est stable et consistant.

$$\begin{aligned}\bar{\tau}_h &:= \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{\Delta x^2} - f(x_i) \\ \bar{\tau}_h &= \left[ u^{(2)}(x_i) + \frac{\Delta x^2}{12} u^{(4)}(x_i) + \Theta(\Delta x^4) \right] - f(x_i) \\ \bar{\tau}_i &= \frac{\Delta x^2}{12} u^{(4)}(x_i) + \Theta(\Delta x^4) \\ \Rightarrow \bar{\tau}_i &= \Theta(\Delta x^2) \\ \lim_{\Delta x \rightarrow 0} \bar{\tau}_i &= 0\end{aligned}$$

$$\lim_{\Delta x \rightarrow 0} \|\bar{\tau}_h\| = 0$$

$$\begin{aligned}E &:= u - u_h \\ A_h E_h &= -\bar{\tau}_h \Rightarrow E_h = -A_h^{-1} \bar{\tau}_h \\ \Rightarrow \|E_h\| &\leq \|A_h^{-1}\| \|\bar{\tau}_h\| \\ \Rightarrow \|E_h\| &\leq \|A_h^{-1}\| \|\bar{\tau}_h\| \\ \text{si } \exists C \in (0, +\infty) : \|A_h^{-1}\| \leq C \quad \forall \Delta x > 0 \\ \text{alors} \quad \lim_{\Delta x \rightarrow 0} \|E_h\| &= 0 \\ \text{et la décroissance de l'erreur globale } E &\text{ est dominée par celle de } \bar{\tau}_h.\end{aligned}$$

## Autres méthodes que celle de Newton

- Méthode linéarisée explicite
- Méthode linéarisée implicite
  
- C.f biblio 1 : page web de Marc Buffat

# Méthode de Newton-Raphson pour l'équation de Poisson non linéaire

- Soit à résoudre

$$\frac{\partial}{\partial x} (Q(u) \frac{\partial u}{\partial x}) = f \quad \Omega$$

où Q est une fonction non linéaire en u et avec des conditions aux limites de type Dirichlet ou mixtes Dirichlet / Neumann.

- On pose le problème sous forme de racine d'une fonctionnelle F

$$F(u) := \frac{\partial}{\partial x} (Q(u) \frac{\partial u}{\partial x}) - f$$

- On discrétise le résidu par méthode de différences finies

On représente  $\bar{\Omega}$  par  $m_x+2$  points  $\{x_i\}_{0 \leq i \leq m_x+1}$ .  $\Delta x := \frac{b-a}{m_x+1}$ ;  $\bar{\Omega} = [a, b]$

$$\forall i \in [1; m_x]: F_i \sim \frac{Q_{i+\frac{1}{2}}(u_{i+1} - u_i) - Q_{i-\frac{1}{2}}(u_i - u_{i-1})}{\Delta x^2}$$

donc  $F \in \mathbb{R}^{m_x+2}$ ,  $F_0$  et  $F_{m_x+1}$  dépendent des conditions aux limites.

- On pose le schéma itératif suivant

$$\begin{cases} u^0 \\ u^{m+1} = u^m - \left(\frac{\partial F}{\partial u}\right)^{-1}(u^m)F(u^m) \end{cases} \quad \forall m \in \mathbb{N}$$

$$\frac{\partial F}{\partial u}(u^m) = J(u^m).$$

# Méthode de Newton-Raphson pour l'équation de Poisson non linéaire

- Reste à calculer les termes de la matrice jacobienne J

$$J \in \mathbb{R}^{(M_{xc}+2) \times (M_{xc}+2)}$$

$J_{00}$  et  $J_{(M_{xc}+1)(M_{xc}+1)}$  dépendent des conditions aux limites.

$\forall i \in [1; M_{xc}]$ :

$$\left\{ \begin{array}{l} J_{ij} = 0 \quad \text{si } j \notin \{i-1, i, i+1\} \\ J_{i(i-1)} = + \frac{\Omega_{i-1/2}}{\Delta x^2} + \frac{1}{2} \frac{\partial \Omega_{i-1}}{\partial u_{i-1}} \times (u_{i-1} - u_i) \\ J_{ii} = - \left( \frac{\Omega_{i+1/2} + \Omega_{i-1/2}}{4 \Delta x^2} + \frac{\partial \Omega_i}{\partial u_i} (2u_i - u_{i-1} - u_{i+1}) \right) \\ J_{i(i+1)} = \frac{\Omega_{i+1/2}}{\Delta x^2} + \frac{1}{2} \frac{\partial \Omega_{i+1}}{\partial u_{i+1}} (u_{i+1} - u_i) \end{array} \right.$$

## Jacobienne tronquée

- Le développement de la matrice jacobienne peut être tronquée à l'ordre 0 sans conséquence sur la solution mais seulement sur la vitesse de convergence vers la solution.
- Dans ce cas

$$\left\{ \begin{array}{l} J_{ij} = 0 \quad j \notin \{i-1, i, i+1\} \\ J_{i,i-1} \sim + \frac{Q_{i-1/2}}{\Delta x^2} \\ J_{ii} \sim - \frac{Q_{i+1/2} + Q_{i-1/2}}{\Delta x^2} \\ J_{i,i+1} \sim + \frac{Q_{i+1/2}}{\Delta x^2} \end{array} \right.$$

# Écriture d'un problème linéaire

$$\begin{cases} \Delta u = 0 & \Omega \\ u(0) = 0 \\ u(1) = 1 \end{cases}$$

$u: x \mapsto x$   
 $\Omega: u \mapsto 1$

# Écriture d'un problème quasi-linéaire

Problème du pendule

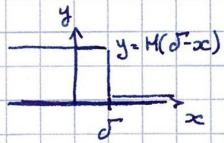
$$\begin{cases} \Delta u + \sin(u) = 0 \\ u(0) = 1 \\ u(1) = 1 \end{cases}$$

Ou autres C.L

# Écriture d'un problème non linéaire

$$\left\{ \begin{array}{l} -\partial_x(k(u)\partial_x u) + \sigma(u^4 - 1) = f \quad \Omega \\ \partial_x u(0) = 0 \\ u(1) = 1 \end{array} \right.$$

$k: u \mapsto k_0 u^2$  ;  $k_0 \in \mathbb{R}$   
 $f: x \mapsto \beta H(0,2-x)$  ;  $\beta \in \mathbb{R}$  ;



$$\begin{aligned} u &= 1 + h \quad (\text{i.e } u \approx 1) \\ u^4 &= (1 + h)^4 \\ u^4 &= 1 + 4h + 6h^2 + 4h^3 + h^4 \\ u^4 &= 1 + 4h + o(1h) \end{aligned}$$

$$\Rightarrow \sigma(u^4 - 1) \approx 4\sigma(u - 1) + o(1h)$$

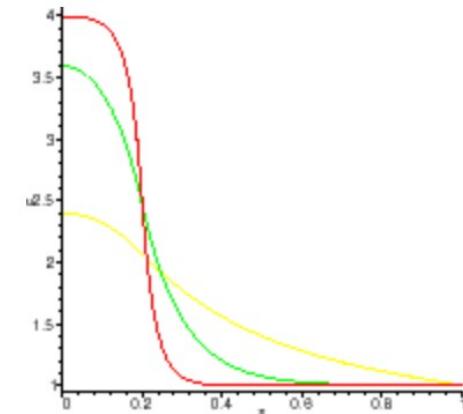
Problème linéarisé autour de  $u = 1$

$$u \approx 1 \Rightarrow \left\{ \begin{array}{l} k(u) \approx k_0 ; \\ \sigma(u^4 - 1) \approx 4\sigma(u - 1) \end{array} \right.$$

$$\left\{ \begin{array}{l} -\partial_x(k_0 \partial_x u) + 4\sigma(u - 1) = f \quad \Omega \\ \partial_x u(0) = 0 \\ u(1) = 1 \end{array} \right.$$

Solutions du problème linéaire pour différentes valeurs de  $\sigma$

- Rouge :  $\sigma = 2.5$  ( $4\sigma = 10$ )
- Vert :  $\sigma = 0.25$  ( $4\sigma = 1$ )
- Jaune :  $\sigma = 0.025$  ( $4\sigma = 0.1$ )



$$u = 1 + h \quad (\text{i.e } u \sim 1)$$

$$u^4 = (1 + h)^4$$

$$u^4 = 1 + 4h + 6h^2 + 4h^3 + h^4$$

$$u^4 = 1 + 4h + o(1h)$$

$$\Rightarrow G(u^4 - 1) \sim 4G(u - 1) + o(1h)$$

## Modèle eulérien vs modèle lagrangien

Un modèle est dit eulérien lorsqu'il se réfère à un repère fixe. L'état du système est alors décrit par des fonctions, des champs, définis sur le domaine en général. Dans ce cas, les particules individuelles ne sont pas identifiées, on préfère définir un volume de contrôle.

Un modèle est dit lagrangien lorsqu'on s'intéresse aux propriétés des constituants microscopiques du système. Le référentiel suit le flot des lois de conservation. Les propriétés de ces constituants élémentaires sont des fonctions du temps et solutions d'équations différentielles. Ce formalisme est directement adapté à l'application de lois physiques élémentaires comme les lois de Newton par exemple.

Les deux points de vue sont cependant équivalents.

Soit  $\underline{u}(t; x, y, z)$  le champ eulérien de vitesse ;  $\underline{u} \in \mathbb{R}^3$ . Alors la trajectoire lagrangienne d'une particule est donnée par  $M(t) \in \mathbb{R}^3$

$$\frac{dM}{dt}(t) = \underline{u}(M(t), t)$$

Réiproquement, le flot de l'équation différentielle ordinaire est

$$q_t : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \\ M(t_0) \mapsto M(t)$$

Plus généralement : pour toute grandeur scalaire  $f$  :

$$\frac{Df}{Dt} = \frac{df(t; M(t))}{dt}$$

$$\Leftrightarrow \frac{Df}{Dt} = \partial_t f + \underline{u} \cdot \nabla f$$

# Modèle eulérien d'écoulement dans un milieu poreux déformable

## Modèle de Farina

Le système à modéliser est caractérisé par les variables d'état

- vitesse solide  $\vec{v}_s$
- vitesse liquide  $\vec{v}_f$
- porosité  $\epsilon$  (fraction volumique liquide)
- pression fluide  $p$

Si le milieu est saturé en phase liquide alors:  $\epsilon = 1 - \nu_f$  où  $\nu_f$  est le taux de fibres.

On suppose ici que les masses volumiques  $\rho_s$  et  $\rho_f$  sont constantes.

## Lois d'évolution du système

On pose les hypothèses suivantes :

- (H1) les effets inertIELS sont négligeables devant le gradient de pression
- (H2) l'écoulement a lieu à faible nombre de Reynolds
- (H3) la force gravitationnelle est négligeable

Les lois d'évolution du système "éponge humide" sont :

$$\partial_t \vec{v}_f + \nabla_x \cdot (\vec{v}_f \vec{v}_x) = 0 \quad (M_s) \text{ conservation de la masse solide}$$

$$\partial_t \varepsilon + \nabla_x \cdot (\varepsilon \vec{v}_e) = 0 \quad (M_e) \text{ conservation de la masse liquide}$$

$$\nabla_x \cdot (-\vec{p} \overset{\Rightarrow}{Id} + \overset{\Rightarrow}{\sigma}) = 0 \quad (E) \text{ élasticité (cons. du moment)}$$

$$\vec{v}_e - \vec{v}_s + \frac{\vec{K}}{\varepsilon \mu} \vec{\nabla}_x p = 0 \quad (D) \text{ écoulement de Darcy}$$

$\mu$ : viscosité de la phase liquide

$K$ : perméabilité du milieu poreux.

- masses volumiques supposées constantes
- phase liquide incompressible

## Conservation de la masse composite

On introduit la *vitesse composite* du système

$$\vec{v}_c = v_f \vec{v}_s + \varepsilon \vec{v}_e$$

Alors (Ms) + (MI) est équivalente à

$$\begin{aligned} & \partial_t (1-\varepsilon) + \nabla_x \cdot ((1-\varepsilon) \vec{v}_s) + \partial_t \varepsilon + \nabla_x \cdot (\varepsilon \vec{v}_e) = 0 \\ \Leftrightarrow & \nabla_x \cdot \vec{v}_c = 0 \end{aligned}$$

## Lois d'équilibre du système

$$(E) : \nabla_{sc} \cdot (p \overset{\Rightarrow}{Id} + \overset{\Rightarrow}{Q}) = 0$$

$$(M) : \nabla_{sc} \cdot \vec{v}_c = 0$$

$$(D) : \vec{v}_c - \vec{v}_s + \frac{\overset{\Rightarrow}{K}}{\varepsilon_M} \nabla_x p = 0$$

## L'équation de la chaleur

- L'équation d'évolution de la température dans un milieu homogène pour lequel on néglige les effets de convection s'écrit en toute généralité

$$(C) : \rho C \frac{\partial T}{\partial t} - \nabla_{\!x} \cdot (\vec{k} \nabla_x T) = 0 \quad (\text{chaleur})$$

K n'est pas la perméabilité mais la conductivité

L'équation est juste mais le modèle est faux : le solide est traversé par un écoulement fluide, il n'est pas du tout évident qu'on puisse négliger la convection.

→ Ré-écriture à partir de la loi de conservation de l'énergie

# Conservation de l'énergie

$$\frac{\partial}{\partial t} \left[ \rho \left( e + \frac{1}{2} v^2 \right) \right] + \nabla \cdot \left[ \rho \mathbf{v} \left( e + \frac{1}{2} v^2 \right) \right] = -\nabla \cdot \mathbf{q} + \nabla \cdot (\boldsymbol{\sigma} \cdot \mathbf{v}) + \rho \mathbf{v} \cdot \mathbf{F}$$

rate of increase      convection of      net      work of      work of  
 of energy per      energy into a      heat      surface      body  
 unit volume      point by flow      flux      forces      forces

In this equation:

- $\rho$  is the density
- $e$  is the internal energy per unit mass
- $\mathbf{v}$  is the velocity vector
- $v^2$  is the square of the velocity magnitude
- $\mathbf{q}$  is the conductive heat flux vector
- $\boldsymbol{\sigma}$  is the total stress tensor
- $\mathbf{F}$  is the body force per unit mass; i.e., the volume force

## L'équation de la chaleur (2)

- On introduit le tenseur des contraintes visqueuses tau  $\sigma = -p\mathbf{I} + \tau$
- On obtient la forme de la conservation d'énergie avec terme source

$$\frac{\partial}{\partial t}(\rho e) + \nabla \cdot (\rho \mathbf{v} e) = -\nabla \cdot \mathbf{q} - p \nabla \cdot \mathbf{v} + \tau : \nabla \mathbf{v} + \mathcal{Q}$$

- On pose la définition de l'enthalpie  $e = h - \frac{p}{\rho}$

- On obtient la conservation de l'enthalpie

$$\frac{\partial}{\partial t}(\rho h) + \nabla \cdot (\rho \mathbf{v} h) = -\nabla \cdot \mathbf{q} + \frac{\partial p}{\partial t} + \mathbf{v} \cdot \nabla p + \tau : \nabla \mathbf{v} + \mathcal{Q}$$

- La loi reliant l'enthalpie et la pression à la température est

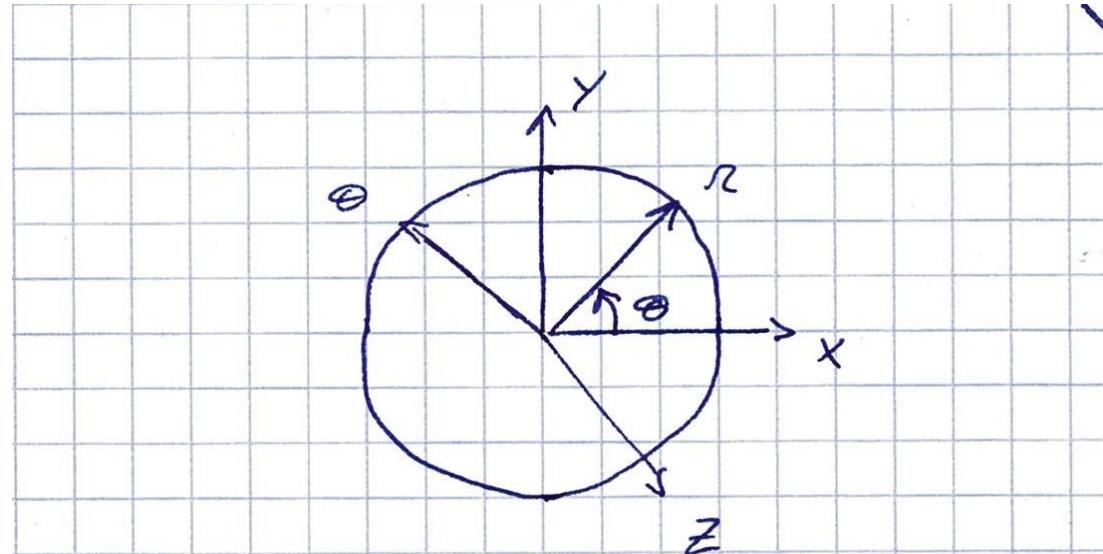
$$dh = C_p dT + \frac{1}{\rho} \left[ 1 + \frac{T}{\rho} \frac{\partial \rho}{\partial T} \right]_p dp = C_p dT + \frac{1}{\rho} [1 - T\beta] dp$$

- D'où l'équation de la chaleur

$$\rho C_p \frac{\partial T}{\partial t} + \rho C_p \mathbf{v} \cdot \nabla T = \nabla \cdot (k \nabla T) + \beta T \left( \frac{\partial p}{\partial t} + \mathbf{v} \cdot \nabla p \right) + \tau : \nabla \mathbf{v} + \mathcal{Q}$$

# Conclusion sur l'équation de la chaleur

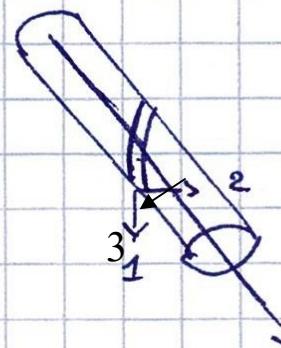
## Les trois repères d'intérêt



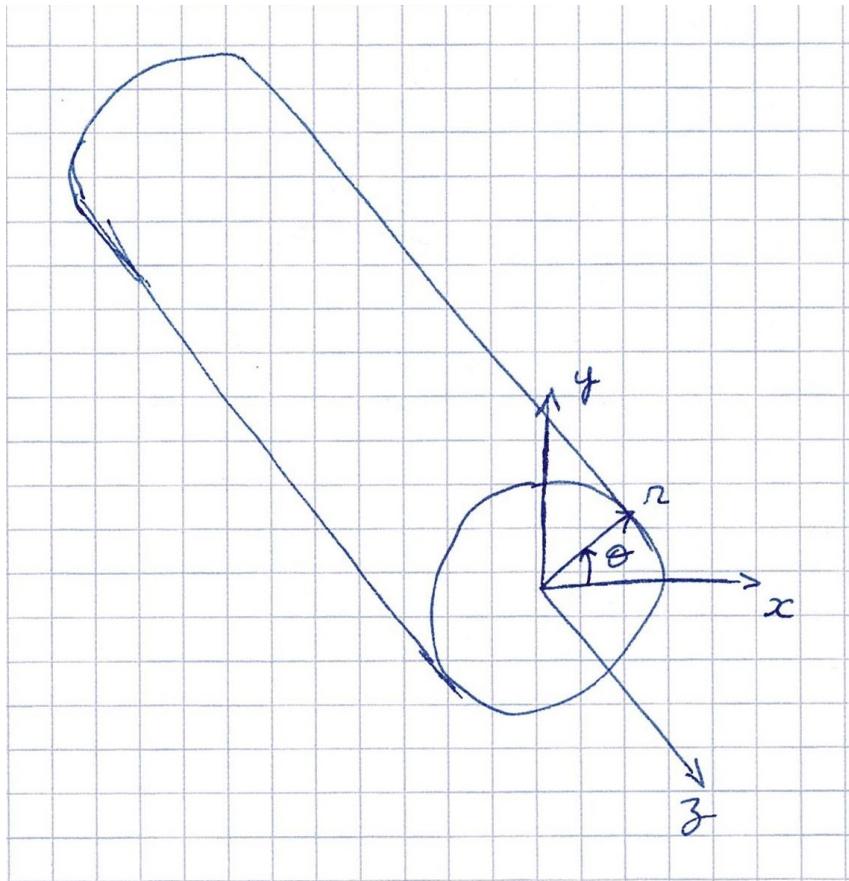
Repère global:  $(x, y, z)$

Repère matériau:  $(1, 2, 3)$

Repère cylindrique:  $(r, \theta, z)$



## Sur le référentiel cylindrique



$$\begin{cases} x = r \cos(\theta) \\ y = r \sin(\theta) \\ z = z \end{cases}$$

$$\begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \arctan(y/x) \\ z = z \end{cases}$$

$$\begin{cases} dr = \cos(\theta) dx - r \sin(\theta) d\theta \\ dy = \sin(\theta) dx + r \cos(\theta) d\theta \\ dz = dz \end{cases}$$

$$\begin{cases} dr = \frac{2x}{\sqrt{x^2+y^2}} dx + \frac{2y}{\sqrt{x^2+y^2}} dy \\ d\theta = \frac{-y}{x^2+y^2} dx + \frac{x}{x^2+y^2} dy \\ dz = dz \end{cases}$$

# Écriture des opérateurs différentiels dans les référentiels cartésien et cylindrique

Gradient :

$$\nabla f = \begin{bmatrix} \partial_x f \\ \partial_y f \\ \partial_z f \end{bmatrix}$$

$$\nabla f = \begin{bmatrix} \partial_r f \\ \frac{1}{r} \partial_\theta f \\ \partial_z f \end{bmatrix}$$

Divergence :

$$\nabla \cdot \vec{v} = \partial_x v^x + \partial_y v^y + \partial_z v^z$$

$$\nabla \cdot \vec{v} = \frac{1}{r} \partial_r (r v^r) + \frac{1}{r} \partial_\theta v^\theta + \partial_z v^z$$

Laplacien :

$$\Delta u = \partial_{xx}^2 u + \partial_{yy}^2 u + \partial_{zz}^2 u \quad \text{cartésien}$$

$$\Delta u = \nabla \cdot (\vec{\nabla} u) \quad \text{"général"}$$

$$\Delta u = \frac{1}{r} \partial_r (r \partial_r u) + \frac{1}{r^2} \partial_\theta^2 u + \partial_{zz}^2 u \quad \text{cylindrique..}$$

## Développement des équations dans le repère cylindrique

$$(M_s) \quad \partial_r v_r + \frac{1}{r} \partial_n (r v_r v_\theta^2) + \frac{1}{r} \partial_\theta (v_r v_\theta^2) + \partial_z (v_r v_z^2) = 0$$

$$(M_e) \quad \partial_r \epsilon + \frac{1}{r} \partial_n (\epsilon v_r^2) + \frac{1}{r} \partial_\theta (\epsilon v_\theta^2) + \partial_z (\epsilon v_z^2) = 0$$

$$(E) \quad \begin{cases} \frac{1}{r} \partial_n (r(p + \sigma^{rr})) + \frac{1}{r} \partial_\theta (\sigma^{r\theta}) + \partial_z (\sigma^{rz}) \\ \frac{1}{r} \partial_n (r \sigma^{r\theta}) + \frac{1}{r} \partial_\theta (p + \sigma^{\theta\theta}) + \partial_z (\sigma^{\theta z}) \\ \frac{1}{r} \partial_n (r \sigma^{rz}) + \frac{1}{r} \partial_\theta (\sigma^{z\theta}) + \partial_z (p + \sigma^{zz}) \end{cases} = 0$$

$$(D) \quad \begin{cases} v_r^r - v_\theta^\theta + \frac{\vec{K}_n \cdot \vec{\nabla}_x p}{\epsilon \mu} = 0 \\ v_\theta^\theta - v_z^z + \frac{\vec{K}_\theta \cdot \vec{\nabla}_x p}{\epsilon \mu} = 0 \\ v_z^z - v_r^r + \frac{\vec{K}_z \cdot \vec{\nabla}_x p}{\epsilon \mu} = 0 \end{cases}$$

~~$$(C) \quad \rho C \partial_r T = \left( \frac{1}{r} \partial_n (r \vec{\lambda}_r \cdot \vec{\nabla}_x T) + \frac{1}{r^2} \partial_\theta (\vec{\lambda}_\theta \cdot \vec{\nabla}_x T) + \partial_z (\vec{\lambda}_z \cdot \vec{\nabla}_x T) \right) = 0$$~~

Lambda est la conductivité du matériau

## Expression du taux de fibres

- Taux de fibres :
  - Le taux de fibres est fonction des déformations (Simacek, 1996).  
En compression (b pour bulk, dans un isotrope transverse)

$$V_f = V_{f_0} (1 - \varepsilon_b)^2$$

- Plus généralement  $\tilde{V}_f = \tilde{V}_{f_0} (1 - \varepsilon_{xx})(1 - \varepsilon_{yy})(1 - \varepsilon_{zz})$

- Or

$$\varepsilon = \frac{1}{2} (\nabla \vec{u} + \nabla \vec{u}^\top)$$

$$\varepsilon_{rr} = \partial_r u^r$$

$$\varepsilon_{\theta\theta} = \frac{1}{r} \partial_\theta u^\theta$$

$$\varepsilon_{zz} = \partial_z u^z$$

- Donc

$$\tilde{V}_f = \tilde{V}_{f_0} (1 - \partial_r u^r)(1 - \frac{1}{r} \partial_\theta u^\theta)(1 - \partial_z u^z)$$

## Hypothèses supplémentaires

- On suppose le problème invariant par translation selon l'axe z : les variations des grandeurs dans cette direction sont nulles
- On suppose le problème invariant par rotation autour de l'axe z (selon l'axe  $\theta$ ), les variations par rapport à  $\theta$  sont donc nulles.
- Question : quel avenir pour l'hypothèse de cylindre infini ? De fait, dans l'algorithme d'enroulement, nous serons vraisemblablement amenés à considérer le temps de parcours du cylindre selon cet axe...

## Système simplifié

$$(M_s) : \partial_t v_f + \frac{1}{\rho} \partial_r (r v_f v_s^r) = 0$$

$$(M_e) : \partial_t \varepsilon + \frac{1}{\rho} \partial_r (r \varepsilon v_e^r) = 0$$

$$(E) : \begin{cases} \frac{1}{\rho} \partial_r (\rho + \sigma^m) \\ \frac{1}{\rho} \partial_r (\sigma^{er}) \end{cases} = 0$$

$$\begin{cases} \frac{1}{\rho} \partial_r (\sigma^{er}) \\ \frac{1}{\rho} \partial_r (\sigma^{sr}) \end{cases} = 0$$

$$(D) \begin{cases} v_e^r - v_s^r + \frac{k_{er}}{\varepsilon \mu} \partial_r p \\ v_e^s - v_s^s + \frac{k_{or}}{\varepsilon \mu} \partial_r p \end{cases} = 0$$

$$\begin{cases} v_e^s - v_s^s + \frac{k_{sr}}{\varepsilon \mu} \partial_r p \end{cases} = 0$$

$$(C) \rho C \partial_t T - \frac{1}{\rho} \partial_r (r \lambda_{sr} \partial_r T) = 0$$

## Système 1.D proposé par Farina

$$(M_e): \partial_t v_p - \partial_x (\epsilon v_e) = 0$$

$$(M_c): \partial_x v_c = 0$$

$$(D): v_e - v_s = - \frac{\kappa}{\epsilon \mu} \partial_x p$$

$$(E): \partial_x \sigma + \partial_x p = 0$$

Dans un référentiel cartésien.

# Conditions initiales et conditions aux limites du système

- Le système est un système d'équations d'évolutions :
  - On fixe la température initiale,
  - Le taux de fibres initial
  - En milieu saturé : la porosité initiale est connue, égale à 1 moins le taux de fibres, sinon elle est à fixer
- Les conditions aux limites peuvent être :
  - Dirichlet,
  - Neumann (uniquement pour les problèmes d'évolution)
  - Mixtes Dirichlet / Neumann
  - Robin : une combinaison linéaire de la grandeur et de son flux est connue

$$\lambda u + \mu \partial_m u = u^R \quad ; \quad \lambda, \mu \in \mathbb{R} \times \mathbb{R}$$

## Lois constitutives des matériaux

- Il s'agit de fixer, à l'aide d'hypothèses sur l'homogénéité et l'isotropie ou pas, les formes des matrices représentant les tenseurs.
- Les lois constitutives sont
  - La relation entre les contraintes et les déformations pour le système élastique
  - La loi de perméabilité du milieu poreux
  - La loi de conduction thermique dans le poreux

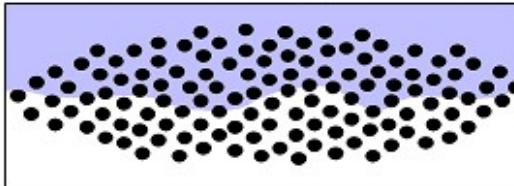
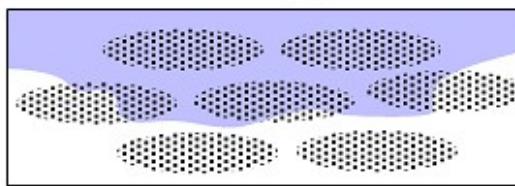
## Stratégies d'analyse numérique

- Discrétisation temporelle par différences finies
  - Euler explicite,
  - Euler implicite,
  - Runge-Kutta,
  - Etc...
- Discrétisation spatiale à choisir parmi les trois familles de schéma :
  - Différences finies,
  - Éléments finis,
  - Volumes finis
- Une fois le schéma discréteisé temporellement et spatialement, on peut décider de l'assemblage d'un ou plusieurs systèmes linéaires à résoudre.
- Choix des algorithmes de résolution
- Implémentation
- Validation par exemple sur les essais de Banerjee ou de Springer

## B. Physiques

- Non traité sur la période.

## 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			

## C. Outils et méthodes informatique

- Comprendre les soucis de convergence de la méthode `scipy.optimize.root(method="krylov") (rien fait)`
- Implémenter à la main une méthode de type Newton pour une équation différentielle non linéaire

## *scipy.optimize.root(method="krylov")*

- Temporairement laissée de côté pour se concentrer sur la méthode de Newton.

# Implémentation de la méthode de Newton

Script [\\TSL-DATA\Public\JulienValentin\5-PROJETS\2-Analyse numérique\Python3.x\newton\\_raphson\\_v3.py](\\TSL-DATA\Public\JulienValentin\5-PROJETS\2-Analyse numérique\Python3.x\newton_raphson_v3.py)

```

7  # Discrétisation spatiale
8  f_x0 = 0
9  f_x1 = 1
10 i_nx = 1022
11 f_dx = (f_x1 - f_x0) / (i_nx+1)
12 t_x = numpy.linspace(f_x0, f_x1, i_nx+2, endpoint=True)
13
14 # Setup du problème (e_ pour "expression", le f de "fonct"
15 def e_u(x):
16     "Expression analytique de la solution u."
17     return x
18
19 def e_Q(u):
20     "Exression analytique du comportement Q."
21     return numpy.ones_like(u)
22
23 def e_f(x):
24     "Expression du second membre."
25     return numpy.zeros_like(x)

54  # Initialisation et paramétrage de la méthode de Newton
55  f_tol      = 1e-8
56  i_max_iter = 1000
57  t_suite_u  = numpy.empty((i_max_iter, i_nx+2))
58  t_suite_du = numpy.empty((i_max_iter, i_nx+2))
59
60  # t_suite_u[0] = numpy.zeros(i_nx+2)    # lève une exception
61  t_suite_u[0] = numpy.ones(i_nx+2)
62  t_suite_u[0, 0] = e_u(f_x0)      # Dirichlet
63  t_suite_u[0, -1] = e_u(f_x1)      # Dirichlet
64
65  t_suite_du[0] = numpy.zeros((i_nx+2))

```

# Implémentation de la méthode de Newton

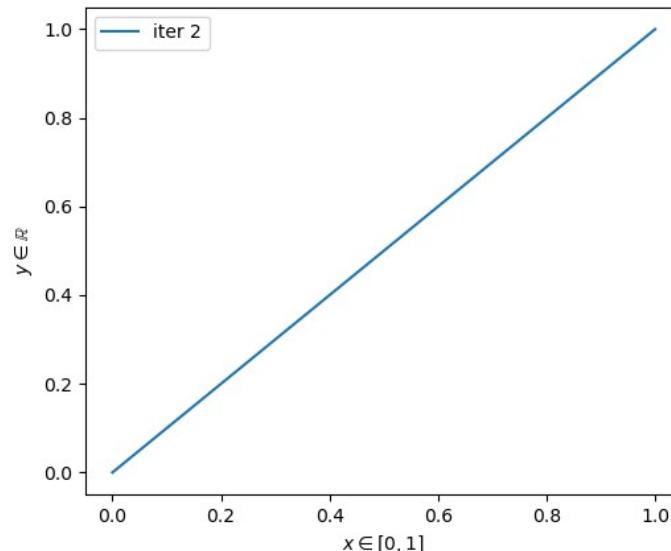
```

67 # Itérations de la méthode de Newton
68 for i in range(i_max_iter-1):
69     # demi-somme des coefficients non linéaires
70     t_Q = 1/2*(e_Q(t_suite_u[i, 1:])+e_Q(t_suite_u[i, :-1]))
71
72     # assemblage du résidu
73     t_res = numpy.zeros(i_nx+2)
74
75     t_res[0] = 0      # Dirichlet au bord
76     t_res[-1] = 0    # Dirichlet au bord
77
78     for j in range(1, i_nx+1):
79         t_res[j] = 1/f_dx**2 * (t_Q[j]*(t_suite_u[i, j+1]-t_suite_u[i, j]) - t_Q[j-1]*(t_suite_u[i, j]-t_suite_u[i, j-1])) + e_f(t_x[j])
80
81     # Assemblage de la jacobienne
82     t_jac = numpy.zeros((i_nx+2, i_nx+2))
83
84     for j in range(i_nx+2):
85
86         if j == 0:      # Dirichlet
87             t_jac[j, j] = 1
88             t_jac[j, j+1] = 0
89
90         elif j == i_nx+1:    # Dirichlet
91             t_jac[j, j-1] = 0
92             t_jac[j, j] = 1
93
94         else:
95             t_jac[j, j-1] = - 1/f_dx**2 * t_Q[j-1]
96             t_jac[j, j] = 1/f_dx**2 * (t_Q[j] + t_Q[j-1])
97             t_jac[j, j+1] = - 1/f_dx**2 * t_Q[j]
98
99     t_suite_du[i+1] = numpy.linalg.inv(t_jac) @ t_res
100    t_suite_u[i+1] = t_suite_u[i] + t_suite_du[i+1]
101
102    f_err = numpy.sqrt(f_dx) * numpy.linalg.norm(t_res, 2)
103
104    print(f"Itération {i+1}/{i_max_iter} : erreur {f_err}")

```

# Résultats

```
(pynum) C:\Users\jvalentin\Documents\ProcedeEnroulementMacroscopique\rapports\codesRapport>python newton_raphson_v3.py
Itération 1/1000 : erreur 32720.011720658047
Itération 2/1000 : erreur 9.290810203789129e-11
```



# Autre problème : quasi-linéaire

Script \\TSL-DATA\\Public\\JulienValentin\\5-PROJETS\\2-Analyse\_numérique\\Python3.x\\newton\_quasi\_lineaire.py

```

6   # Discrétisation spatiale
7   x0 = 0
8   x1 = 2*np.pi
9   nx = 126
10  dx = (x1 - x0) / (nx+1)
11  xx = np.linspace(x0, x1, nx+2, endpoint=True)
12
13  # Setup du problème : second membre et ses variations
14  def f(u):
15      "Expression du second membre."
16      return np.sin(u)
17
18  def df(u):
19      "Expression des variations du second membre."
20      return np.cos(u)

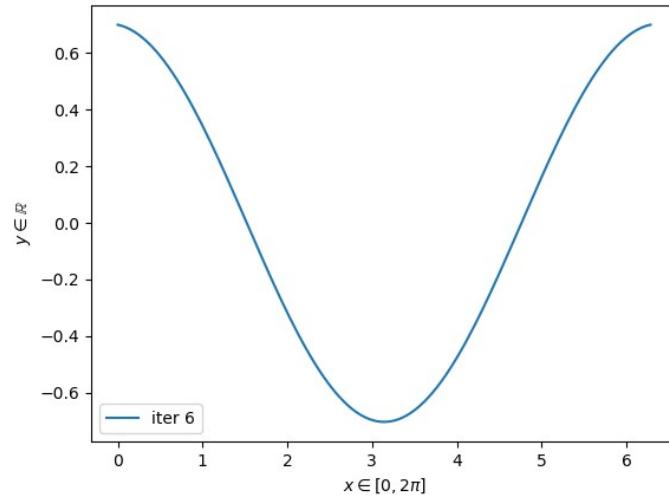
22  # Définition du résidu et de la jacobienne
23  def res(u):
24      "Residu de l'équation du pendule."
25      vec = np.zeros((nx+2))
26
27      vec[0] = 0
28      vec[-1] = 0
29
30      for i in range(1, len(vec)-1):
31          vec[i] = (-2*u[i] + u[i+1] + u[i-1])/dx**2 + f(u[i])
32
33      return vec
34
35  def jac(u):
36      "Jacobienne du résidu"
37      mat = np.zeros((nx+2, nx+2))
38
39      mat[0, 0] = 1
40      mat[-1, -1] = 1
41
42      for i in range(1, nx+1):
43          mat[i, i-1] = 1/dx**2
44          mat[i, i] = -2/dx**2 + df(u[i])
45          mat[i, i+1] = 1/dx**2
46
47      return mat

```

# Autre problème : quasi-linéaire

```
57     for i in range(max_iter-1):
58
59         b = res(suite_u[i, :])
60         A = -jac(suite_u[i, :])
61         du = np.linalg.inv(A) @ b
62         suite_u[i+1] = suite_u[i] + du
63
64         err = np.sqrt(dx) * np.linalg.norm(b, 2)
65
66         print(f"Itération {i+1}/{max_iter} : erreur {err}")
67
68         if err < tol:
69             break
```

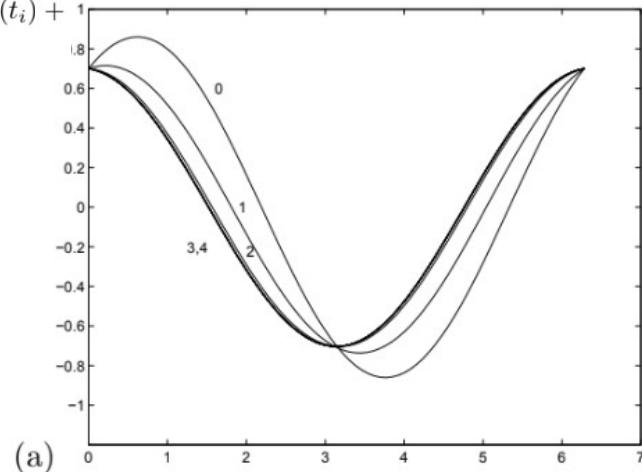
# Résultats



```
(pynum) C:\Users\jvalentin\Documents\ProcedeEnr
py
Itération 1/150 : erreur 0.14310101368383557
Itération 2/150 : erreur 0.030966293843114633
Itération 3/150 : erreur 0.0070005019243238745
Itération 4/150 : erreur 0.00017315713714649718
Itération 5/150 : erreur 7.96532266557587e-09
Itération 6/150 : erreur 7.354442507573413e-14
```

Figure 2.4: Convergence of Newton iterates towards a solution of the pendulum problem. The iterates  $\theta^{[k]}$  for  $k = 1, 2, \dots$  are denoted by the number  $k$  in the plots. (a) Starting from  $\theta_i^{[0]} = 0.7 \cos(t_i) + 0.5 \sin(t_i)$  (b) Starting from  $\theta_i^{[0]} = 0.7$ .

Biblio 2 : R. J. LeVeque, p. 28



(a)

# Autre problème : non linéaire

```
7 # Discrétisation spatiale
8 f_x0 = 0
9 f_x1 = 1
10 i_nx = 254
11 f_dx = (f_x1 - f_x0) / (i_nx+1)
12 t_x = numpy.linspace(f_x0, f_x1, i_nx+2, endpoint=True)
13
14 # Setup du problème (e_ pour "expression", le f de "fonction")
15 k0 = 0.01
16 sigma = 1
17 beta = 300
18 delta = 0.2
19
20 def e_K(t):
21     return numpy.power(t, 2)
22
23 def e_Q(t):
24     return int(t<delta)*beta
26 # Initialisation et paramétrage de la méthode de Newton
27 f_tol      = 1e-10
28 i_max_iter = 10000
29 t_suite_du = numpy.empty((i_max_iter, i_nx+2))
30 t_suite_u  = numpy.empty((i_max_iter, i_nx+2))
31
32 t_suite_u[0] = numpy.ones(i_nx+2)
33 t_suite_du[0] = numpy.zeros(i_nx+2)
```

# Autre problème : non linéaire

```

35 # Itérations de la méthode de Newton
36 for i in range(i_max_iter-1):
37     # demi-somme des coefficients non linéaires
38     t_k = 1/2*(e_K(t_suite_u[i, 1:])+e_K(t_suite_u[i, :-1]))
39
40     # assemblage du résidu
41     t_res = numpy.zeros(i_nx+2)
42
43     t_res[0] = k0/f_dx**2 * (t_k[0]*(t_suite_u[i, 1]-t_suite_u[i, 0]) + t_k[0]*(t_suite_u[i, 1]-t_suite_u[i, 0])) - sigma*(t_suite_u[i, 0]**4-1) + e_Q(t_x[0])
44     t_res[-1] = 0    # Dirichlet au bord
45
46     for j in range(1, i_nx+1):
47         t_res[j] = k0/f_dx**2 * (t_k[j]*(t_suite_u[i, j+1]-t_suite_u[i, j]) - t_k[j-1]*(t_suite_u[i, j]-t_suite_u[i, j-1])) - sigma*(t_suite_u[i, j-1]**4 + e_Q(t_x[j]))
48
49     # Assemblage de la jacobienne
50     t_jac = numpy.zeros((i_nx+2, i_nx+2))
51
52     for j in range(i_nx+2):
53
54         if j == 0:    # Neumann
55             t_jac[j, j] = k0/f_dx**2 * 2*t_k[0] + 4*sigma*t_suite_u[i, 0]**3
56             t_jac[j, j+1] = - k0/f_dx**2 * 2*t_k[0]
57
58         elif j == i_nx+1:    # Dirichlet
59             t_jac[j, j-1] = 0
60             t_jac[j, j] = 1
61
62         else:
63             t_jac[j, j-1] = - k0/f_dx**2 * t_k[j-1]
64             t_jac[j, j] = k0/f_dx**2 * (t_k[j] + t_k[j-1]) + 4*sigma*t_suite_u[i, j]**3
65             t_jac[j, j+1] = - k0/f_dx**2 * t_k[j]
66
67     t_suite_du[i+1] = numpy.linalg.inv(t_jac) @ t_res
68     t_suite_u[i+1] = t_suite_u[i] + t_suite_du[i+1]
69
70     f_err = numpy.sqrt(f_dx)*numpy.linalg.norm(t_res, 2)
71
72     print(f"Itération {i}/{i_max_iter} : erreur {f_err}")
73
74     if f_err < f_tol:
75         break

```

# Résultats

## Détail du début du calcul

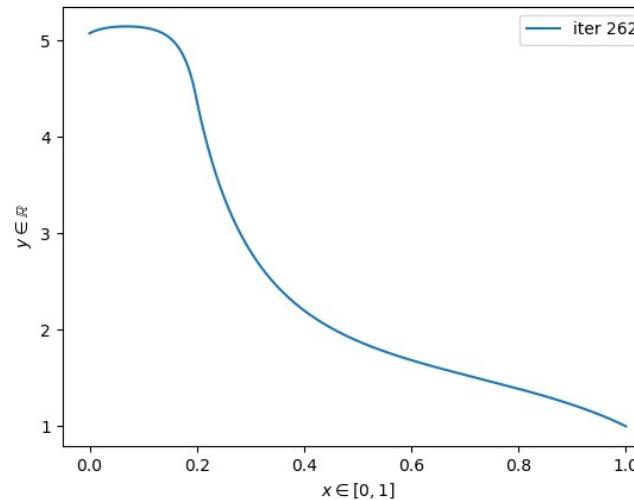
```
(pnum) C:\Users\jvalentin\Documents\ProcedeEnroulementMacroscopique\rapports\codesRapport>python newton_raphson_v1.py
Itération 0/10000 : erreur 134.16407864998737
Itération 1/10000 : erreur 10016331.596583929
Itération 2/10000 : erreur 3334828.1594646913
Itération 3/10000 : erreur 1128933.381936934
Itération 4/10000 : erreur 390001.9187132375
Itération 5/10000 : erreur 138151.98994897527
Itération 6/10000 : erreur 58455.85836059761
Itération 7/10000 : erreur 19106.91989206093
Itération 8/10000 : erreur 7541.175221456839
Itération 9/10000 : erreur 3113.33810054695
Itération 10/10000 : erreur 1345.422190867553
Itération 11/10000 : erreur 606.4692209473905
Itération 12/10000 : erreur 282.8212558865264
Itération 13/10000 : erreur 134.91488460187392
Itération 14/10000 : erreur 65.18827873247607
Itération 15/10000 : erreur 31.791875360380352
Itération 16/10000 : erreur 15.756392606760226
Itération 17/10000 : erreur 8.053112111948895
Itération 18/10000 : erreur 4.3293560273564555
Itération 19/10000 : erreur 2.4936088859800347
Itération 20/10000 : erreur 1.5510115282867474
Itération 21/10000 : erreur 1.0349728116924621
Itération 22/10000 : erreur 0.7299085963680974
Itération 23/10000 : erreur 0.5360469015852588
Itération 24/10000 : erreur 0.4055353428499295
Itération 25/10000 : erreur 0.313833638821345
Itération 26/10000 : erreur 0.24732993064277645
Itération 27/10000 : erreur 0.1979164743229056
```

} Explosion (<6)  
} Décroissance (>6)

## Dernières itérations

```
Itération 250/10000 : erreur 2.7766773334338515e-10
Itération 251/10000 : erreur 2.5464959892807114e-10
Itération 252/10000 : erreur 2.3353705757798956e-10
Itération 253/10000 : erreur 2.1417729163315612e-10
Itération 254/10000 : erreur 1.9643124568625515e-10
Itération 255/10000 : erreur 1.801434799640588e-10
Itération 256/10000 : erreur 1.6522137984813664e-10
Itération 257/10000 : erreur 1.515677316652477e-10
Itération 258/10000 : erreur 1.390035412060674e-10
Itération 259/10000 : erreur 1.2745304444268197e-10
Itération 260/10000 : erreur 1.1690329815212576e-10
Itération 261/10000 : erreur 1.0722860761520568e-10
Itération 262/10000 : erreur 9.843225519903176e-11
```

## Graphe de la dernière itérée calculée



# Reprise du problème non linéaire

```

7 # Discrétisation spatiale
8 f_x0 = 0
9 f_x1 = 2
10 i_nx = 510
11 f_dx = (f_x1 - f_x0) / (i_nx+1)
12 t_x = numpy.linspace(f_x0, f_x1, i_nx+2, endpoint=True)

```

## Neumann dans le résidu

```

42     # Neumann à gauche par schéma décentré aval (trois points pour la dérivée première)
43     t_res[0] = -(3*t_suite_u[i, 0]-4*t_suite_u[i, 1]+t_suite_u[i, 2])/(2*f_dx)

```

## Neumann dans la jacobienne

```

55     if j == 0:    # Neumann (dérivée du schéma décentré aval trois points) (/!\ au signe)
56         t_jac[j, j] = 3/(2*f_dx)
57         t_jac[j, j+1] = -4/(2*f_dx)
58         t_jac[j, j+2] = 1/(2*f_dx)

```

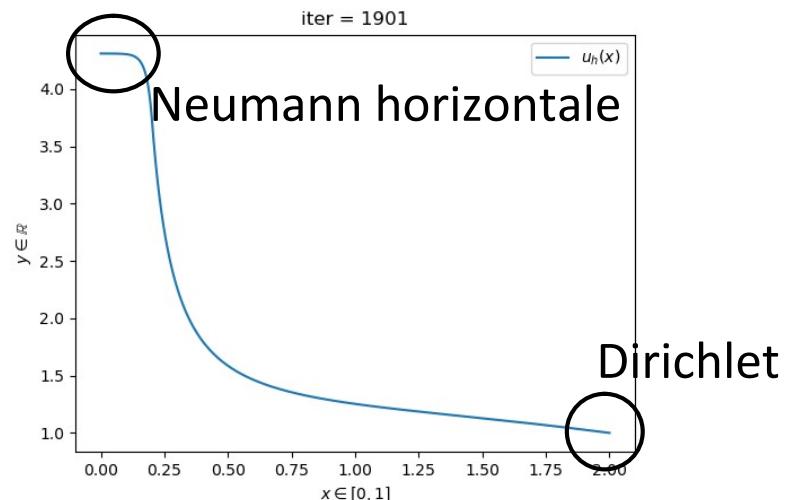
## 20 dernières itérations

```

Itération 1881/5000 : erreur 1.1710927573974926e-08
Itération 1882/5000 : erreur 1.1618741307346997e-08
Itération 1883/5000 : erreur 1.1527280677779305e-08
Itération 1884/5000 : erreur 1.1436540084301866e-08
Itération 1885/5000 : erreur 1.1346513928692314e-08
Itération 1886/5000 : erreur 1.1257196238382905e-08
Itération 1887/5000 : erreur 1.1168581566721643e-08
Itération 1888/5000 : erreur 1.1080664735113761e-08
Itération 1889/5000 : erreur 1.0993439841680125e-08
Itération 1890/5000 : erreur 1.090690151809853e-08
Itération 1891/5000 : erreur 1.0821044586764739e-08
Itération 1892/5000 : erreur 1.073586328250606e-08
Itération 1893/5000 : erreur 1.065135275095224e-08
Itération 1894/5000 : erreur 1.0567507263376652e-08
Itération 1895/5000 : erreur 1.0484321921030753e-08
Itération 1896/5000 : erreur 1.0401791357482965e-08
Itération 1897/5000 : erreur 1.0319910372436696e-08
Itération 1898/5000 : erreur 1.023867396779194e-08
Itération 1899/5000 : erreur 1.0158077182757899e-08
Itération 1900/5000 : erreur 1.007811479225873e-08
Itération 1901/5000 : erreur 9.998781748677935e-09

```

## Graphe de la dernière itération



# Benchmark ; problème inverse : simulation de perméabilité

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

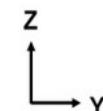
Raw data of X-ray-scans

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:

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

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  $\theta=0$ :

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

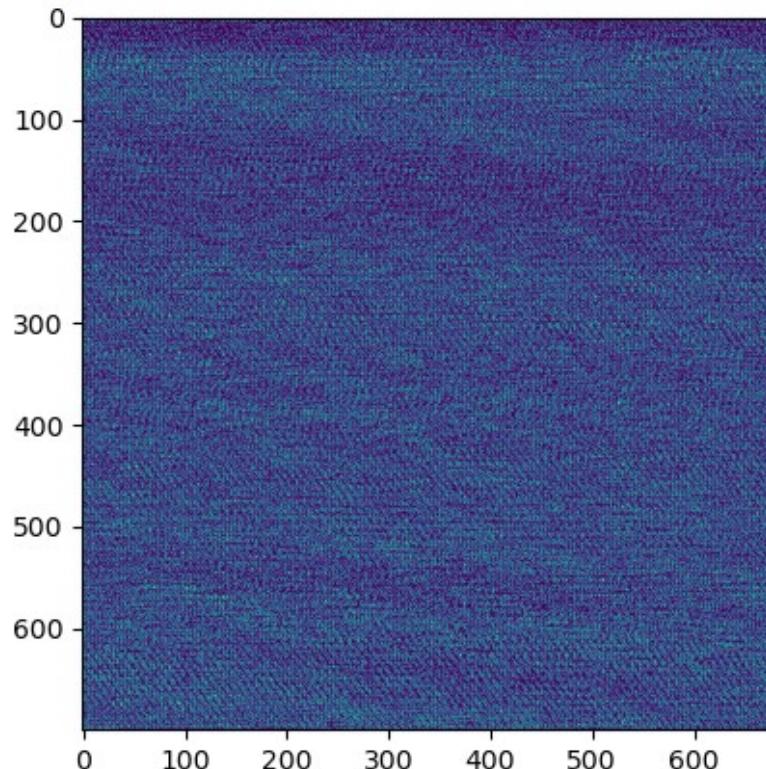
You can adjust permeability tensor if  $\theta \neq 0$

## Benchmark : tâches à effectuer

- Écoulement à une seule échelle:
  - Prédiction du tenseur de perméabilité à partir de l'analyse de l'image segmentée 700 x 680 x 345 en négligeant l'imprégnation des bandes
- Écoulement à deux échelles :
  - Prédiction du tenseur de perméabilité à partir de l'analyse de l'image segmentée 700 x 680 x 345 en prenant en compte l'imprégnation des bandes, en négligeant ou pas les angles locaux et en utilisant le modèle de perméabilité microscopique fourni.
- Prédiction du tenseur de perméabilité à partir du modèle complet 1923x1553x345
- Prédiction du tenseur de perméabilité à partir de l'image segmentée par une méthode *different*e
- Prédiction du tenseur de perméabilité à partir d'algorithmes de segmentation privés
- Prédiction du tenseur de perméabilité à partir de la considération des deux échelles en considérant un tenseur de perméabilité micro adapté.

# Benchmark : lire un fichier RAW en Python

```
1  #! -*- coding : utf-8 -*-
2
3  import matplotlib.pyplot as plt
4  import numpy as np
5
6  chemin = "VPB_meso_8microns_700x680x345_seg_theta.raw"
7
8  rows = 700
9  cols = 680
10 dept = 345
11
12 # Ouverture du flux : fichier
13 fichier = open(chemin, "r")
14
15 # Lecture et stockage dans un vecteur numpy.ndarray
16 image = np.fromfile(fichier, dtype=np.uint8, count=rows*cols*dept)
17
18 # Conversion en tenseur 700 x 680 x 345
19 image = image.reshape(rows, cols, dept)
20
21 # Affichage de l'image de profondeur 0
22 plt.imshow(image[:, :, 0])
23 plt.show()
24
```



# Benchmark : lire un fichier TIF en Python

Image rendue par *matplotlib.pyplot.imshow*

```
1  #! -*- coding: utf-8 -*-
2
3  from skimage import io
4  import matplotlib.pyplot as plt
5
6  # Lecture des données
7  img = io.imread('VPB_meso_8microns_700x680x345_seg.tif')
8
9  # Affichage de l'image
10 plt.imshow(img[:, :, 0] ,cmap='gray')
11 plt.axis('off')
12
13 plt.show()
14
```



# Benchmark : visualiser l'image 3D avec *napari*

- Impossible sur ma machine. L'installation se termine mais le logiciel ne se lance pas, et l'exécution du script non plus.
- L'exécution de

```
1  #! -*- coding: utf-8 -*-
2
3  import napari
4  from skimage import io
5
6  # Lecture des données
7  img = io.imread('VPB_meso_8microns_700x680x345_seg.tif')
8
9  # Affichage de l'image
10 viewer = napari.view_image(img, contrast_limits=[0, 1])
11
```

Lève l'exception

```
AttributeError: module 'time' has no attribute 'clock'
```

Et l'exécution hors d'un script :

```
(benchmark) C:\Users\jvalentin>napari
Traceback (most recent call last):
  File "C:\Users\jvalentin\.conda\envs\benchmark\Scripts\napari-script.py", line 9, in <module>
    sys.exit(main())
  File "C:\Users\jvalentin\.conda\envs\benchmark\lib\site-packages\napari\__main__.py", line 431, in main
    _run()
  File "C:\Users\jvalentin\.conda\envs\benchmark\lib\site-packages\napari\__main__.py", line 230, in _run
    from napari import run, view_path
ImportError: cannot import name 'view_path' from 'napari' (C:\Users\jvalentin\.conda\envs\benchmark\lib\site-packages\napari\__init__.py)
```

# Benchmark : de l'image à la géométrie avec iso2Mesh

<https://groups.google.com/g/iso2mesh-users/c/91XBC5ViYh8?pli=1>

need a help to mesh tiff images from X-ray CT 150 vues

S'abonner



jaona.rand...@gmail.com  
à iso2mes...@googlegroups.com

23 nov. 2014, 13:31:20



Dear Prof. Fang,

I am a new user of Iso2mesh and before of all, I want to say "thank you a lot for making this code available to us."

Although I read carefully the user guide and have take a look on various samples, I do not arrive to mesh my images as i expected.

So let me introduce you my problem, I have a 3D tif image from X-ray CT after using ImagJ for binarization (you can download the source file of the image at <http://esuptools.univ-reims.fr/esup-flex/getfile.php?hash=689b8fa936b6708b4254408bfd973e21>) and I want to mesh both black and white colors of the image with tetrahedral elements and i want to separate them into 2 regions/domains (region 1 for the white color and region 2 for the black corlo) and export the mesh into abaqus format. I tried the following script:

```
%  
for i=1:256  
    test_f3(:, :, i)=imread('test_f3.tif', i);  
end  
  
[node, elem, face]=v2m(test_f3, 0.5, 5, 100);  
  
saveabaqus(node, face, elem, 'test_f3.inp')  
  
plotmesh(node, [2 1 3]), face, facealpha, 0.7;  
%
```

This script works but it meshes only the white region. So, could you give me some advises how to mesh both white and black regions of the images and separate them into 2 element sets in abaqus file?

Thank you by advance.

Best regards,

Jaona Randrianalisoa  
Assist. Prof. University of Reims, France

# Benchmark : de l'image à la géométrie avec Iso2mesh

Réponse sur le même URL



Qianqian Fang

à iso2mes...@googlegroups.com

\*\*\*

hi Jaona

to generate a mesh for your input, you need to use the 'cgalmesh' option in v2m. Here is my sample code:

```
[node,elem,face]=v2m(test_f3+1,[],5,10,'cgalmesh');  
plotmesh(node(:,1:3),elem,'z<150');  
hold on;  
plotmesh(node(:,1:3),elem(elem(:,end)==1,:));
```

the output mesh screenshot is attached.

there was another workflow: binsurface->meshresample->surf2mesh,  
however, the appearance of self-intersecting elements  
after the meshresample often makes the last step impossible.  
When it works, you can get something like these in the  
mesh gallery:

[http://iso2mesh.sourceforge.net/upload/vessel\\_mesh\\_surface.png](http://iso2mesh.sourceforge.net/upload/vessel_mesh_surface.png)  
[http://iso2mesh.sourceforge.net/upload/vessel\\_mesh\\_cut.png](http://iso2mesh.sourceforge.net/upload/vessel_mesh_cut.png)

Qianqian

# Exemples d'utilisation de vol2mesh

<https://iso2mesh.sourceforge.net/cgi-bin/index.cgi?Doc/Examples>

## 2. Demonstration 1 for vol2mesh

The first vol2mesh demonstration is to create an FEM mesh from a CT scan of a rat-head. In this example, the volumetric data is a thresholded CT image of a rat. In a vol2mesh session, we first wrap the volume with a layer of zeros, in order to make the non-zero region close. Then we extract the boundary of the non-zero regions by calling Matlab's `isosurface()`. The created iso-surface, in the resolution of voxel sizes, will be re-sampled to specified mesh density. Before and after the simplification, we perform a mesh validation and repair process to avoid the appearance of "non-manifold" nodes. Once the surface mesh is completed, we produce volumetric mesh within the region bounded by the re-sampled surface mesh.

The voxelated rat head binary image looks like	<a href="#">upload:mouse_vox.png</a>
The re-sampled rat surface mesh is shown below	<a href="#">upload:mouse_mesh.png</a>
The cross-section of the 3D volumetric mesh looks like	<a href="#">upload:mouse_head.png</a>

## 3. Demonstration 2 for vol2mesh

In this example, we try to make a little bit fun using iso2mesh while showing off the extreme flexibility and simplicity of making a complex mesh using this toolbox. We will make a mesh on a slab-shaped object with "ISO2Mesh" etched on one of the surface, just like the header image of this document showing.

To make this "object", we first create a black-white image, "iso2mesh\_bar.tif", with an image editor. This image is then read as a 2D array into matlab, we then invert the 0-1 values, and stack it to 30 layers to make a etched pattern, then, append another 30 layers of pure 1 image at the end of the stack as the base. Now we have this binary array with etched letter patterns.

The original 2D binary pattern is	<a href="#">upload:iso2mesh_bar_2d.png</a>
and the 3D object surface look like	<a href="#">upload:iso2mesh_bar_3d_vox_small.png</a>

We pass this array to vol2mesh, and set the surface simplification ratio to 10%, and maximum volume of 100 (in voxel unit), after going through similar processes as in demo1, we will get a volumetric mesh just like what you see from the header bar image.

The generated surface mesh is	<a href="#">upload:iso2mesh_bar_1.png</a>
-------------------------------	---

## 4. Demonstration 3 for vol2mesh

Demonstration 3 is another challenging example showing you how to apply iso2mesh in complex geometries such as head and brain imaging. The 3D head and brain images were extracted from a real head scan from MRI. To save space, we convert them into a binary format as head.tif and brain.tif, respectively, where the brain is simple a segmented subregions from the head scan.

The raw binary head image is shown as	<a href="#">upload:head_orig.png</a>
---------------------------------------	--------------------------------------

Reading the head binary image into matlab and plot a slice, you will find there are lots of gaps and openings in the volume. As the dimensions of these gaps are close to the size of a few voxel, it will give iso2mesh a really bad time trying to mesh with these gaps and holes. Therefore, we first perform a binary volume repairing process: `deisland3d`, to fill the holes and remove the isolated regions from the image. The cleaned up version of the head image can be found at here. The brain volume is clean by itself, so, we do not need to do hole-filling and island-removing.

The head image after hole-filling/island-removing is	<a href="#">upload:head_fill.png</a>
--	--------------------------------------

In this case, we will make a 2-level surface meshes: we will make 3D FEM mesh to fill both the regions between the head surface and brain surface, as well as the volume within the brain surface, and we want to make sure that the surface representation of the brain appears exactly the resulting volumetric mesh (i.e., there is a conformal surface layer in the FEM mesh to match the brain surface). To do this, we first add the brain binary image on top of the head image, to create a two level image: voxels with value of 0 represent air, voxels of 1 represent the head region outside the brain, and voxels of value 2 represent the brain region. We pass this 3-valued array, fulfilling, to vol2mesh, vol2mesh will first extract the iso-surface the interface between 0 and 1, performing mesh repairing and simplification; and then the interfaces between 1 and 2. After both surfaces complete, we will produce volumetric mesh for this mesh cluster. The resulting mesh will have a exterior surface conforms to the head contour, and an internal surface to conform to the brain surface.

The 3-level head and segmented brain image	<a href="#">upload:head_seg.png</a>
The brain surface mesh is	<a href="#">upload:head_brain_meshed.png</a>
The head surface mesh is	<a href="#">upload:head_surface.png</a>
The produced volumetric mesh cross-cut view 1	<a href="#">upload:head_mesh_cut1.png</a>
The produced volumetric mesh cross-cut view 2	<a href="#">upload:head_mesh_cut2.png</a>

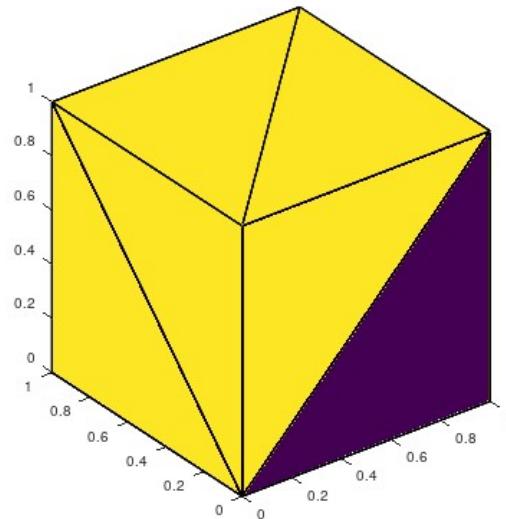
# Installation de GNU Octave, iso2mesh

- Installation du logiciel GNU Octave (C:/TENSYL/Programmes/Octave\*)
- Téléchargement du plug in iso2mesh  
(<https://sourceforge.net/projects/iso2mesh/files/iso2mesh/>)
- Décompression dans C:/TENSYL/Programmes/iso2mesh
- Lancer Octave
- Saisir la commande *addpath("C:/TENSYL/Programmes/iso2mesh")*
- Manipuler...

# Premiers pas avec Octave/iso2mesh

- Maillage 3.D d'une boîte

```
>> addpath("C:/TENSYL/Programmes/iso2mesh")
>> [node,face,elem]=meshabox([0,0,0],[1,1,1],1,1);
generating tetrahedral mesh from closed surfaces ...
creating volumetric mesh from a surface mesh ...
volume mesh generation is complete
>> plotmesh(node,face,elem)
>>
```



# Première tentative de maillage

```

1 clear *;
2 close *;
3
4 % Import de la bibliothèque d'images
5 printf("Import du package Image...\n");
6 pkg load image
7
8 % Renseignement du chemin de la bibliothèque iso2mesh
9 printf("Import de la librairie iso2mesh...\n");
10 addpath("C:/TENSYL/Programmes/iso2mesh");
11
12 % Lecture de l'image .tif
13 % On lit la pile d'images en profondeur
14 % Pour la démonstration, on ne retient que les 100 premières couches.
15 printf("Chargement de la pile d'images 3D... ");
16
17 reverseStr = '';
18 nbimg = 345;
19 for i=1:nbimg
20     img(:,:,i)=imread('VPB_meso_8microns_700x680x345_seg.tif',i);
21
22     message = sprintf('%d / %d images lues.',i,nbimg);
23     fprintf([reverseStr, message]);
24     reverseStr = repmat(sprintf('\b'), 1, length(message));
25 end
26 printf("\n");
27
28 % Affichage des valeurs uniques
29 printf("Valeurs des pixels :");
30 unique(img)
31
32 % Renseignement des options pour la fonction vol2mesh
33 % vol2mesh(img,ix,iy,iz,opt,maxvol,dofix,method,isovalues)
34 ix = 1:680;
35 iy = 1:700;
36 iz = 1:nbimg;
37 opt(1).keepratio = 0.1;
38 opt(1).radbound = 10;
39 opt(2).keepratio = 0.1;
40 opt(2).radbound = 10;
41 opt(3).keepratio = 0.1;
42 opt(3).radbound = 10;
43 maxvol = 0.1;
44 dofix = 1;
45 method = "cgalsurf";
46 isovalues = [0,120,255];
47
48 % Création du maillage 3D
49 [node,elem,face,regions] = vol2mesh(img,ix,iy,iz,opt,maxvol,dofix,method,isovalues);
50
51 % Affichage du maillage
52 plotmesh(node,face,'facealpha',0.7);
53

```

```

>> maillage_tif_3D
Import du package Image...
Import de la librairie iso2mesh...
Chargement de la pile d'images 3D... 345 / 345 images lues.
Valeurs des pixels :ans =
0
120
255

extracting surfaces from a volume ...
region 1 centroid : 341.000000 1.666667 1.333333

region 2 centroid : 290.500000 1.666667 1.333333

region 3 centroid : 630.500000 1.666667 1.333333

processing threshold level 1...
Surface Mesh Extraction Utility (Based on CGAL 5.0)
(modified for iso2mesh by Qianqian Fang)
http://iso2mesh.sf.net

RNG seed 1648335518
set initial mesh size to 50
Final number of points: 20273
processing threshold level 2...
Surface Mesh Extraction Utility (Based on CGAL 5.0)
(modified for iso2mesh by Qianqian Fang)
http://iso2mesh.sf.net

RNG seed 1648335518
set initial mesh size to 50
Final number of points: 25701
error: elem(_,_3): out of bound 2 (dimensions are 49818x2)
error: called from
    readoff at line 54 column 16
    vol2restrictedtri at line 54 column 12
    vol2surf at line 172 column 18
    vol2mesh at line 63 column 23
    maillage_tif_3D at line 49 column 26

```

# Deuxième tentative de maillage

```

1 %clear *;
2 %close *;
3
4 % Import de la bibliothèque d'images
5 %printf("Import du package Image...\n");
6 %pkg load image
7
8 % Renseignement du chemin de la bibliothèque iso2mesh
9 %printf("Import de la librairie iso2mesh...\n");
10 %addpath("C:/TENSYL/Programmes/iso2mesh");
11
12 % Lecture de l'image .tif
13 % On lit la pile d'images en profondeur
>> maillage_tif_3D
creating surface and tetrahedral mesh from a multi-domain volume ...
Volume/Surface Mesh Generation Utility (Based on CGAL 5.0)
(modified for iso2mesh by Qianqian Fang and Peter Varga)
http://iso2mesh.sf.net

RNG seed=164833518
Mesh sizes are (label=size) (l=0.1)
terminate called after throwing an instance of 'CGAL::Assertion_exception'
what(): CGAL ERROR: assertion violation!
Expr: !r_tr_.is_infinite(*cell_it)
File: C:/maya64/mingw64/include/CGAL/Mesh_3/Refine_cells_3.h
Line: 891
error: output file was not found, failure was encountered when running command: \n"C:\TENSYL\Programmes\iso2mesh\bin\cgalmesh_x86-64.exe" "C:\Users\JVALEN-1\AppData\Local\Temp\iso2mesh-jvalentin\pre_cgalmesh.inr" "C:\Users\JVALEN-1\AppData\Local\Temp\iso2mesh-jvalentin\post_cgalmesh.mesh" 30.000000 6.000000 0.500000 3.000000 0.100000 164833518
error: called from
    cgalmesh at line 87 column 5
    vol2mesh at line 54 column 20
    maillage_tif_3D at line 49 column 26
>> |
    %unique(img)
30
31
32 % Renseignement des options pour la fonction vol2mesh
33 % vol2mesh(img,ix,iy,iz,opt,maxvol,dofix,method,isovalues)
34 ix = 1:680;
35 iy = 1:700;
36 iz = 1:nbimg;
37 opt(1).keepratio = 0.1;
38 opt(1).radbound = 10;
39 opt(2).keepratio = 0.1;
40 opt(2).radbound = 10;
41 opt(3).keepratio = 0.1;
42 opt(3).radbound = 10;
43 maxvol = 0.1;
44 dofix = 1;
45 method = "cgalmesh";
46 isovalues = [0,120,255];
47
48 % Cr ation du maillage 3D
49 [node,elem,face,regions] = vol2mesh(img,ix,iy,iz,opt,maxvol,dofix,method,isovalues);
50
51 % Affichage du maillage
52 plotmesh(node,face,'facealpha',0.7);
53

```

Ex ecution du script jusqu' a l'erreur : 20 min

## 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 :

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

Constante de Kozeny

Viscosité de la résine

Épaisseur de la mèche

Temps adimensionnel, chute significative de la pression  
 $t^* = 0,5$

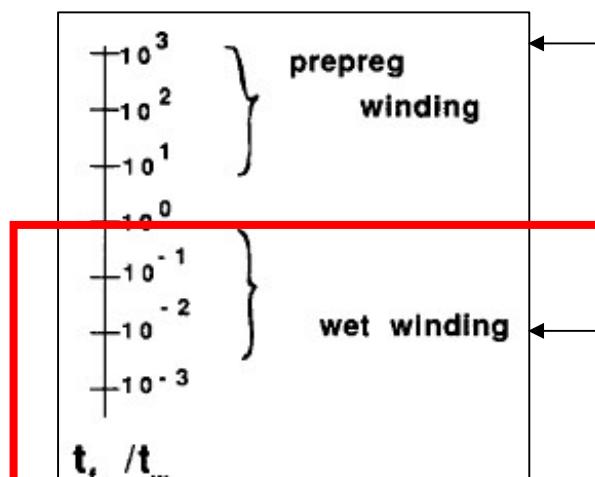
Rayon d'une fibre  
« Fiber bed spring constant »

(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 ?

## D. Modélisation à l'échelle mésoscopique

- Non traité sur la période.

## E. Modélisation à l'échelle microscopique

- Non traité sur la période.

## F. Actions à venir

## G. Bibliographie de la présentation

- [https://perso.univ-lyon1.fr/marc.buffat/COURS/COURSDF\\_HTML/node29.html](https://perso.univ-lyon1.fr/marc.buffat/COURS/COURSDF_HTML/node29.html)
- <https://www.comsol.com/multiphysics/heat-transfer-conservation-of-energy>
- R. J. LeVeque, *Finite Difference Methods for Differential Equation*, Université de Washington, 2005
- A. Farina, P. Cocito, G. Boretto - Flow in deformable porous media, modelling and simulations of compression moulding processes (M. C. M., 1997)
- Packages pour l'analyse numérique en Python
  - Éléments finis
    - <https://sfepy.org/doc-devel/index.html>
    - <https://docu.ngsolve.org/latest/>
    - <https://github.com/kinnala/scikit-fem>
    - <https://getfem-examples.readthedocs.io/en/latest/>
  - Différences finies
    - <https://scikit-fdiff.readthedocs.io/en/latest/#>
  - Volumes finis
    - <https://pypi.org/project/pyfvm/>
    - <https://www.ctcms.nist.gov/fipy/>
    - <https://www.clawpack.org/pyclaw/>
    - <https://pypde.readthedocs.io/en/latest/>
- Logiciel de script pour les éléments finis Python / C++ : <https://fenicsproject.org/documentation/>