Résolution de l'équation de la chaleur en 2D à l'aide de la méthode des éléments finis

Présentation du TP

Ce TP vous permet de créer un code modulaire en C++ pour les éléments finis. Le code propose la résolution de l'équation de la chaleur en 2D en éléments finis de type P1 mais il est possible relativement facilement de :

- changer l'équation du modèle,
- augmenter l'ordre (P2 ...),
- utiliser des éléments structurés (Q1 ...)
- passer en 3D,
- utiliser un *solver* parallèle.

Le livre suivant pourra être utilisé comme référence pour en savoir plus sur la méthode des éléments finis :
Alexandre Ern and Jean-Luc Guermond, **Eléments finis : théorie, applications, mise en oeuvre**, *Springler-Verlag Berlin Heidelberg, Mathématiques et Applications 36, 2002.*

Méthode des éléments finis

Cette partie présente la méthode des éléments finis. Pour bien comprendre la méthode, il est conseillé de prendre le temps de faire ce qui est proposé en rouge tout au long de cette section.

Modèle L'objectif est de résoudre le problème suivant : trouver u tel que

$$\left\{\begin{array}{rcll} \partial_t u - \sigma\Delta u & = & f, & \Omega, \\ u & = & g, & \Gamma_D, \\ \sigma\nabla u \cdot n & = & h, & \Gamma_N, \end{array}\right.$$

avec $\partial\Omega = \Gamma_D \cup \Gamma_N$ et $\Gamma_D \cap \Gamma_N = \emptyset$.

Mise sous forme variationnelle Si $g \in H^{1/2}(\partial\Omega)$, il existe une fonction $u_D \in H^1(\Omega)$ telle que $u_{D|\Gamma_D} = g$ (par relèvement). En considérant $u - u_D$ on peut donc se ramener quitte à changer f et h au cas où $g = 0$. On pose alors :

$$\mathcal{V} = \{u \in H^1(\Omega), u = 0 \text{ sur } \Gamma_D\}.$$

En définissant

$$a(u,v) = \int_\Omega \partial_t u\, v + \sigma \int_\Omega \nabla u \cdot \nabla v \quad \text{et} \quad l(v) = \int_\Omega fv + \int_{\Gamma_N} hv,$$

le problème s'écrit : trouver $u \in \mathcal{V}$ telle que

$$a(u,v) = l(v), \quad \forall v \in \mathcal{V}.$$

À faire : Retrouver la forme variationnelle ci-dessus.

Existence et unicité de la solution En utilisant le théorème de Lax Milgram, il existe une unique solution $u \in \mathcal{V}$ au problème avec $f \in L^2(\Omega)$, $h \in L^2(\Gamma_N)$. On peut ensuite revenir à la solution du problème avec des conditions de Dirichlet non homogènes si $g \in H^{1/2}(\Gamma_D)$.
À faire : Montrer que les hypothèses du théorème de Lax Milgram sont vérifiées.

Résolution par éléments finis

Méthode de Galerkin On considère $\mathcal{V}_h$ un sous espace de dimension finie de $\mathcal{V}$. Comme il est de dimension finie, il existe une base $(\phi_i)_{i=1,\cdots,N}$ où N est la dimension de $\mathcal{V}_h$. On cherche alors $u_h = \sum_{i=1}^{N} x_i\, \phi_i$ telle que

$$\sum_{i=1}^{N} x_i\, a(\phi_i, \phi_j) = l(\phi_j), \quad \forall j = 1, \cdots, N.$$

Avec $A = (a(\phi_i, \phi_j))_{i,j}$ $x = (x_i)_i$ et $b = (l(\phi_i))_i$, ce problème se réécrit sous la forme du système linéaire suivant :

$$Ax = b.$$

Méthode des éléments finis Une méthode d'éléments finis est un cas particulier de la méthode de Garlekin où $\mathcal{V}_h$ est défini sur un **maillage**. On suppose que ce maillage est un maillage triangulaire conforme d'une géométrie 2D. On note $\mathcal{T}_h$ notre triangulation et $\mathcal{L}_h^{N,a}$ les arêtes des triangles formant le bord Γ_N.

Élément de référence On définit le triangle $\hat{K}$ comme le triangle rectangle formé des points $(0,0)$, $(1,0)$ et $(0,1)$. Ce triangle est appelé élément de référence. Il existe une transformation affine $\mathcal{F}_K$ (c'est-à-dire $\mathcal{F}_K \in \mathbb{P}_1$) telle que $K = \mathcal{F}_K(\hat{K})$, $\forall K \in \mathcal{T}_h$. De même, pour tout $E \in \mathcal{L}_h^{N,a}$, on note $\mathcal{F}_E$ une transformation affine envoyant l'arête $(\hat{k}_1,\hat{k}_2)$ de $\hat{K}$ sur E.

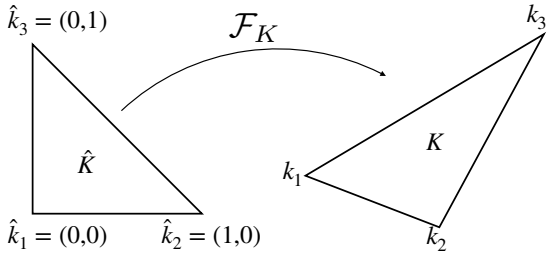

FIGURE 1 – Passage de l'élément de référence $\hat{K}$ au triangle K

Espace d'éléments finis On définit l'espace d'éléments finis $\mathcal{V}_h$ par

$$\mathcal{V}_h = \{v_h \in \mathcal{C}^0(\bar{\Omega}), \forall K \in \mathcal{T}_h,\ v_{h|_K} \in \mathbb{P}_p\},$$

c'est à dire les fonctions continues dont les restrictions à chaque triangle sont des polynômes de degré p. Dans ce TP, nous nous concentrerons sur le cas $p = 1$ mais il est possible d'étendre le code à un degré supérieur.

Fonctions de base Les fonctions de base sont construites sur $\hat{K}$ et sont ensuite transportées. Pour le cas $p = 1$ en 2D, il y a autant de fonctions de bases qu'il y a de noeuds (sommets des triangles) dans le maillage (donc N = nombre de noeuds). On définit les 3 fonctions suivantes :

$$\hat{\phi}_1(\hat{x}_1, \hat{x}_2) = 1 - \hat{x}_1 - \hat{x}_2,\ \hat{\phi}_2(\hat{x}_1, \hat{x}_2) = \hat{x}_1 \text{ et } \hat{\phi}_3(\hat{x}_1, \hat{x}_2) = \hat{x}_2.$$

Dans la suite on note $\hat{X} = (\hat{x}_1, \hat{x}_2)$. Pour tout $i = 1, \cdots, N$, on définit la fonction de base ϕ_i par

$$\phi_{i|_K} = \begin{cases} \hat{\phi}_{\hat{i}} \circ \mathcal{F}_K^{-1}, \text{ si } i \in K \\ 0, \text{ sinon.} \end{cases}$$

où $\hat{i}$ est la position de i sur l'élément de référence.
À faire : Vérifier que $(\phi_i)_{i=1,\cdots,N}$ forment bien une base de l'espace $\mathcal{V}_h$.

Résolution temporelle (schéma implicite) On discrétise l'intervalle $[t_0, T_{final}]$ par

$$(t_n)_{n=0,N_{it}} \text{ avec } t_n = t_0 + n\,\Delta t \text{ et } \Delta t = \frac{T_{final} - t_0}{N_{it}},$$

et on approche

$$u(t_n) \approx u^n = \sum_{i=1}^{N} x_i^n \phi_i.$$

Dans ce TP, nous allons considérer le schéma d'Euler implicite et nous allons résoudre :

$$(\mathcal{M} + \sigma \Delta t\, \mathcal{K}) X^{n+1} = \mathcal{M} X^n + \Delta t \mathcal{B}^n, \forall n \in [0, N_{it} - 1]$$

avec

$$X^n = (x_i^n)_i,\ \mathcal{M} = \left(m_{ij} = \int_\Omega \phi_i \phi_j\right)_{ij},\ \mathcal{K} = \left(k_{ij} = \int_\Omega \nabla\phi_i \cdot \nabla\phi_j\right)_{ij}$$
$$\text{et } \mathcal{B}^n = \left(b_i^n = \int_\Omega f(\cdot, t_n)\phi_i + \int_{\Gamma_N} g(\cdot, t_n)\phi_i\right)_i.$$

Les matrices $\mathcal{M}$ et $\mathcal{K}$ correspondent respectivement aux matrices de masse et de rigidité. Remarque : les conditions aux bords de type Dirichlet ne sont pour le moment pas prises en compte dans ce système ...

Formules de quadrature Les matrices de masse et de rigidité et le terme source peuvent être réécrits en utilisant la triangulation $\mathcal{T}_h$ et l'élément de référence :

$$\begin{aligned} m_{ij} &= \int_\Omega \phi_i(X)\,\phi_j(X) \mathrm{d}X \\ &= \sum_{K \in \mathcal{T}_h} \int_{\hat{K}} \left(\phi_i \circ \mathcal{F}_K(\hat{X})\right) \left(\phi_j \circ \mathcal{F}_K(\hat{X})\right) |\det \mathrm{J}\mathcal{F}_K(\hat{X})| \mathrm{d}\hat{X} \\ &= \sum_{K \in \mathcal{T}_h} \int_{\hat{K}} \hat{\phi}_{\hat{i}}(\hat{X})\ \hat{\phi}_{\hat{j}}(\hat{X}) |\det \mathrm{J}\mathcal{F}_K(\hat{X})|\, \mathrm{d}\hat{X}, \end{aligned}$$

$$\begin{aligned} k_{ij} &= \int_\Omega \nabla\phi_i(X)\, \nabla\phi_j(X) \mathrm{d}X \\ &= \sum_{K \in \mathcal{T}_h} \int_{\hat{K}} \left(\nabla\phi_i \circ \mathcal{F}_K(\hat{X})\right) \left(\nabla\phi_j \circ \mathcal{F}_K(\hat{X})\right) |\det \mathrm{J}\mathcal{F}_K(\hat{X})| \mathrm{d}\hat{X} \\ &= \sum_{K \in \mathcal{T}_h} \int_{\hat{K}} \left(\mathrm{J}\mathcal{F}_K^{-T} \cdot \hat{\nabla}\hat{\phi}_{\hat{i}}(\hat{X})\right) \left(\mathrm{J}\mathcal{F}_K^{-T} \cdot \hat{\nabla}\hat{\phi}_{\hat{j}}(\hat{X})\right) |\det \mathrm{J}\mathcal{F}_K(\hat{X})| \mathrm{d}\hat{X} \end{aligned}$$

et

$$
\begin{aligned}
b_i^n &= \int_\Omega f(X,t_n)\phi_i(X)\mathrm{d}X + \int_{\Gamma_N} h(X,t_n)\phi_i(X)\mathrm{d}\Gamma_X \\
&= \sum_{K\in\mathcal{T}_h} \int_{\hat{K}} \big(f\circ(\mathcal{F}_K(\hat{X}),t_n)\big)\ \big(\phi_i\circ\mathcal{F}_K(\hat{X})\big)|\det\ \mathrm{J}\mathcal{F}_K(\hat{X})|\mathrm{d}\hat{X} \\
&+ \sum_{E\in\mathcal{L}_h^{N,a}} \int_{\hat{E}} \big(h\circ(\mathcal{F}_E(\hat{X}),t_n)\big)\ \big(\phi_i\circ\mathcal{F}_E(\hat{X})\big)\sqrt{|\det\ \mathrm{J}\mathcal{F}_E(\hat{X})|}\mathrm{d}\Gamma_{\hat{X}} \\
&= \sum_{K\in\mathcal{T}_h} \int_{\hat{K}} \big(f\circ(\mathcal{F}_K(\hat{X}),t_n)\big)\ \hat{\phi}_{\hat{i}}(\hat{X})\ |\det\ \mathrm{J}\mathcal{F}_K(\hat{X})|\mathrm{d}\hat{X} \\
&+ \sum_{E\in\mathcal{L}_h^{N,a}} \int_{\hat{E}} \big(h\circ(\mathcal{F}_E(\hat{X}),t_n)\big)\ \big(\hat{\phi}_{\hat{i}}\circ\mathcal{F}_{K_E}\circ\mathcal{F}_E(\hat{X})\big)\sqrt{|\det\ \mathrm{J}\mathcal{F}_E(\hat{X})|}\mathrm{d}\Gamma_{\hat{X}},
\end{aligned}
$$

où $\mathrm{J}\mathcal{F}_K$ (resp. $\mathrm{J}\mathcal{F}_E$) correspond à la Jacobienne de $\mathcal{F}_K$ (resp. $\mathcal{F}_E$). Nous aurons donc besoin de calculer des termes de la forme $\int_{\hat{K}} f(\hat{X})\mathrm{d}\hat{X}$ ou de la forme $\int_{\hat{E}} f(\hat{X})\mathrm{d}\Gamma_{\hat{X}}$. Nous approcherons ces intégrales par des formules de quadrature c'est-à-dire :

$$\int_{\hat{K}} f(\hat{X})\mathrm{d}\hat{X} \approx \sum_{q=0}^{Q} w_q f(\hat{X}_q),$$

où les $\hat{X}_q$ sont des points de $\hat{K}$ (appelés points de quadrature) et les w_q des coefficients de pondération. On choisit en général ces formules de telle sorte qu'elles soient exactes pour les polynômes jusqu'à un certain degré. On note $\hat{k}_1, \hat{k}_2, \hat{k}_3$ les sommets, $\hat{k}_{12}, \hat{k}_{13}, \hat{k}_{23}$ les milieux des arêtes, $\hat{k}_0$ le centre et $|\hat{K}|$ la surface du triangle $\hat{K}$. On peut citer les 3 formules suivantes :

- Centrée : $\int_{\hat{K}} f(\hat{X})\mathrm{d}\hat{X} \approx |\hat{K}| f(\hat{k}_0)$ (exacte pour polynômes de degré inférieur ou égal à 1).
- Sommets : $\int_{\hat{K}} f(\hat{X})\mathrm{d}\hat{X} \approx \frac{|\hat{K}|}{3}\big(f(\hat{k}_1)+f(\hat{k}_2)+f(\hat{k}_3)\big)$ (degré inférieur ou égal à 1).
- Milieux : $\int_{\hat{K}} f(\hat{X})\mathrm{d}\hat{X} \approx \frac{|\hat{K}|}{3}\big(f(\hat{k}_{12})+f(\hat{k}_{13})+f(\hat{k}_{23})\big)$ (degré inférieur ou égal à 2).

Nous utiliserons pour les matrices $\mathcal{M}$ et $\mathcal{K}$ la formule sur les milieux.
Pour l'intégrale sur la frontière, nous utiliserons une formule de quadrature 1D, par exemple celle de Simpson (exacte pour les polynômes de degré 2) :

$$\int_{\hat{E}} f(\hat{E})\mathrm{d}\Gamma_{\hat{X}} \approx \frac{|\hat{E}|}{6}\big(f(\hat{k}_1)+4f(\hat{k}_{12})+f(\hat{k}_2)\big),$$

si $\hat{E}$ est l'arête formée par les sommets $\hat{k}_1$ et $\hat{k}_2$.
À faire : Vérifier que la méthode des milieux (resp. de Simpson) est une méthode de quadrature 2D (resp. 1D) d'ordre 2.

Calculs à faire avant l'assemblage Nous listons ici les calculs qui sont à effectuer avant de pouvoir assembler la matrice :

- Calculer les 3 fonctions de base géométriques :

$$\hat{\phi}_1(\hat{X}) = 1-\hat{x}_1-\hat{x}_2,\ \hat{\phi}_2(\hat{X}) = \hat{x}_1 \text{ et } \hat{\phi}_3(\hat{X}) = \hat{x}_2.$$

- Calculer une transformation $\mathcal{F}_K$ et sa jacobienne $\mathrm{J}\mathcal{F}_K$, par exemple :

$$\mathcal{F}_K(\hat{X}) = \sum_{i=1}^{3} k_i \, \hat{\phi}_i(\hat{X}), \text{ où } k_i \text{ est le i-ème sommet de } K \text{ et}$$

$$\left(\mathrm{J}\mathcal{F}_K\right)_{mn} = \sum_{i=1}^{3} (k_i)_m \, \partial_{\hat{x}_n} \hat{\phi}_i(\hat{X}), \text{ avec } (k_i)_m \text{ la composante numéro } m.$$

- Calculer une transformation $\mathcal{F}_E$ et sa jacobienne $\mathrm{J}\mathcal{F}_E$, par exemple :

$$\mathcal{F}_E(\hat{X} = (\hat{X}_1, \hat{X}_2)) = \begin{pmatrix} (k_{2,1} - k_{1,1})\hat{X}_1 + (k_{1,2} - k_{2,2})\hat{X}_2 + k_{1,1} \\ (k_{2,2} - k_{1,2})\hat{X}_1 + (k_{2,1} - k_{1,1})\hat{X}_2 + k_{1,2} \end{pmatrix},$$

où $k_i = (k_{i,1}, k_{i,2})$ est le i-ème sommet de E.
À faire : Vérifier les formules proposées pour les transformations ainsi que celles pour les jacobiennes.

- Calculer le gradient des fonctions de base d'approximation (pour la matrice de rigidité) :

$$\nabla \phi_i \circ \mathcal{F}_K(\hat{X}) = \mathrm{J}\mathcal{F}_K^{-T} \cdot \hat{\nabla} \hat{\phi}_{\hat{i}}(\hat{X}).$$

À faire : Vérifier cette formule.

Assemblage Une étape très importante consiste à assembler les matrices $\mathcal{M}$ et $\mathcal{K}$ en utilisant l'algorithme suivant :

```
// Boucle sur les éléments K dans T_h
for (int K = 0; K < num_tri ; K++)
{
  // Boucle sur les points de quadrature w_q, hatX_q
  for (int q = 0 ; q < num_quad_pts ; q++)
  {
    // Boucle sur les noeuds de l'élément de référence géométrique
    for (int hati = 0 ; hati < 3 ; hati++)
    {
      // Boucle sur les noeuds de l'élément de référence géométrique
      for (int hatj = 0 ; hatj < 3 ; hatj++)
      {
        // i = numéro global du noeud hati de K
        // j = numéro global du noeud hatj de K
        // M(i,j) = M(i,j) + contribution de la masse (hati,hatj) en hatX_q
        // K(i,j) = K(i,j) + contribution de la rigidité (hati,hatj) en hatX_q
      }
    }
  }
}
```

Il faut bien sûr ne pas oublier de construire (à chaque pas de temps si f et/ou h évoluent au cours du temps) le terme $\mathcal{B}^n$.

Conditions au bords Avant d'aborder les condition aux bords, une petite remarque s'impose. Cette remarque est très importante pour expliquer comment reconstruire la solution pour la sauvegarder et l'afficher. En effet, en P1, on a :

$$x_i^n = u_h(s_i, t_n), \text{ où } s_i \text{ correspond au i-ème sommet du maillage.}$$

Les conditions aux bords de type Neumann sont dites naturelles car elles sont naturellement apparues dans la formulation variationnelle à travers le terme $l(v) = \int_\Omega fv + \int_{\Gamma_N} hv$, donc elles sont prises en compte dans notre équation dans la deuxième partie de $\mathcal{B}^n$. Cependant, il nous manque les conditions de Dirichlet qui sont dites essentielles car elles permettent l'unicité de la solution. D'un point de vue numérique, nous allons traiter de la même manière les conditions de Dirichlet homogènes et non homogènes. On définit l'ensemble des sommets $\{s_\gamma, \gamma \in I^{d,s}\}$ formant le bord Γ_D et nous allons résoudre $\forall n \in [0, N_{it} - 1]$:

$$\begin{cases} (\mathcal{M} + \sigma \Delta t\, \mathcal{K}) X^{n+1} &= \mathcal{M} X^n + \Delta t \mathcal{B}^n, \\ x_\gamma^{n+1} &= g(s_\gamma, t_{n+1}),\ \gamma \in I^{d,s}. \end{cases}$$

Soient $\mathcal{A} = (\mathcal{M} + \sigma \Delta t\, \mathcal{K})$ et $\mathcal{C}^n = \mathcal{M} X^n + \Delta t \mathcal{B}^n, \forall n \in [0, N_{it} - 1]$, nous allons imposer fortement les conditions aux bords $x_\gamma^{n+1} = g(s_\gamma, t_{n+1})$ en modifiant la matrice $\mathcal{A}$ et le terme source $\mathcal{C}^n$ de la manière suivante :

- Modification de certaines lignes de $\mathcal{A}$: pour tout $\gamma \in I^{d,s}$, pour tout $j = 1, \cdots, N$

$$\mathcal{A}_{\gamma j} = \begin{cases} 1, & \text{si } j = \gamma \\ 0, & \text{sinon.} \end{cases}$$

- Modification de certaines lignes de $\mathcal{C}^n$: pour tout $\gamma \in I^{d,s}$,

$$\mathcal{C}_\gamma^n = g(s_\gamma, t_{n+1}).$$

À faire : Vérifier que les modifications de $\mathcal{A}$ et $\mathcal{C}^n$ proposées ci-dessous permettent d'imposer vos conditions aux bords. Quelle propriété intéressante pour la résolution numérique la matrice $\mathcal{A}$ a t-elle perdue dans cette manipulation ?

Présentation du code

Le code est constitué de 6 classes. Plusieurs versions du code peuvent être récupérées en suivant ce lien : lien.

- La Version 1 correspond à celle dans laquelle les calculs géométriques et l'assemblage doivent être implémentés.
- La Version 2 correspond à celle dans laquelle l'assemblage doit être implémenté.
- La version finale du code est aussi donnée.

La classe DataFile

Elle permet de gérer le fichier de données contenant tous vos paramètres. Le fichier *data.txt* vous montre tous les choix que votre code devra proposer à l'utilisateur. Le fichier de données est automatiquement copié dans le dossier de résultats. Pour voir tous les paramètres accessibles, il faut regarder dans le fichier *DataFile.h* toutes les fonctions *Get* disponibles :

```
// Renvoient le temps initial, le temps final, le pas de temps,
// le nombre d'itérations et la période de sauvegarde des solutions
const double & gett0() const {return _t0;};
const double & gettfinal() const {return _tfinal;};
const double & getdt() const {return _dt;};
const int & getNumberOfIterations() const {return _nIterations;};
const double & getPeriodToSaveSol() const {return _periodToSaveSol;};
// Renvoie le paramètre de diffusion σ
const double & getSigma() const {return _sigma;};
// Renvoient les noms du maillage et du dossier de résultat
const std::string & getMeshName() const {return _meshName;};
const std::string & getResultsFolder() const {return _resultsFolder;};
// Renvoient les références associées aux conditions Dirichlet et de Neumann
const std::vector<int> & getDirichletReferences() const {return _dirichlet;};
const std::vector<int> & getNeumannReferences() const {return _neumann;};
// Renvoie un booléen pour dire s'il y a une solution exacte connue
// (très utile pour valider le code !)
const bool & getIsExactSol() const {return _isExactSol;};
```

La classe Mesh

Elle permet de gérer tout ce qui concerne votre maillage. Vous pouvez utiliser n'importe quel maillage 2D triangulaire non structuré construit par exemple avec `Gmsh` à condition que les arêtes du bord de votre maillage soient incluses. Des exemples de fichier *.geo* sont proposés dans le dossier *Meshes* : il faut les sauvegarder avec *Physical Entity* pour avoir les bonnes références d'arête. Voilà la liste de ce qui est disponible à partir de la classe *Mesh* :

```
// Renvoie une matrice dont chaque ligne correspond à un point du maillage :
// la colonne i correspond à la i-ème coordonnée du point
const Eigen::Matrix<double, Eigen::Dynamic, 2> & getVertices()
                              const {return _vertices;};
// Renvoie une matrice dont chaque ligne correspond à un triangle du maillage :
// la colonne i correspond au i-ème point du triangle
const Eigen::Matrix<int, Eigen::Dynamic, 4> & getTriangles()
                              const {return _triangles;};
// Renvoie une matrice dont chaque ligne correspond à une arête Neumann du
// maillage pour laquelle :
// la colonne 1 (resp. 2) correspond à la 1ère (resp. 2ème) coord. de l'arête ;
// la colonne 3 correspond à la référence
// la colonne 4 au triangle auquel appartient l'arête
// la colonne 5 au numéro de l'arête du triangle (1, 2 ou 3)
const Eigen::Matrix<int, Eigen::Dynamic, 5> & getNeumannEdges()
                              const {return _edgesNeumann;};
// Renvoie un vecteur de pairs correspondant aux points pour lesquels il faut
// imposer les conditions de Dirichlet avec les références associées
const std::vector<std::pair<int,int>> & getRefVerticesDirichlet()
                              const {return _refVerticesDirichletWithRef;};
```

Ce qui est peut être fait dans cette classe : Pour passer en 3D ou avec des maillages contenant des quadrangles, cette classe doit être modifiée.

La classe TimeScheme

Elle permet de gérer le schéma en temps. Le seul schéma proposé pour le moment est *Euler*. Les 2 fonctions *Initialize* et *Advance* correspondent à la résolution en temps proposée ci-dessus.
Ce qui est peut être fait dans cette classe : D'autres schémas en temps peuvent être implémentés par héritage.

La classe Solver

Elle permet de gérer le solveur utilisé. Pour le moment, seulement le solveur LU pour les matrices creuses d'`Eigen` est implémenté.
Ce qui est peut être fait dans cette classe : D'autres solveurs (parallèles par exemple ...) peuvent être ajoutés par héritage.

La classe Geometry

Elle permet de gérer tout ce qui concerne les calculs géométriques qui sont nécessaires à l'assemblage des matrices et du terme source :

```
// Construction des 3 fonctions de base (phihat0, phihat1, phihat2)
double phihat(int hati, Eigen::Vector2d hatX);
// Construction des 3 gradients des fonctions de base
// (gradphihat0, gradphihat1, gradphihat2) (indépendant du vecteur hatX)
Eigen::Vector2d gradphihat(int hati);
// Construction de la fonction de transformation du triangle de
// référence hatK en K
Eigen::Vector2d FK(int refTriangleK, Eigen::Vector2d hatX);
// Construction de la fonction de transformation
// de l'arête de référence hatE en E
Eigen::Vector2d FE(int refEdgeE, Eigen::Vector2d hatX);
// Construction de la jacobienne de la fonction de transformation
// du triangle de référence hatK en K (indépendant du vecteur hatX)
Eigen::Matrix2d JFK(int refTriangleK);
// Construction de la jacobienne de la fonction de transformation
// de l'arête de référence hatE en E (indépendant du vecteur hatX)
Eigen::Matrix2d JFE(int refEdgeE);
// Construction de la mesure associée à la transformation FK pour
// le changement de variable de l'intégrale d X = measK d hatX
double measK(int refTriangleK);
// Construction de la mesure associée à la transformation FE pour
// le changement de variable de l'intégrale d Gamma_X = measK d Gamma_hatX
double measE(int refEdgeE);
// Construction du gradient (gradphi_i) avec i qui est le hati-eme sommet de K
// appliqué en F(hatX)
Eigen::Vector2d gradphi(int hati, int refTriangleK);
// Points et poids de quadrature de la formule du milieu (intégration 2D)
void quadraturePointsAndWeightsMidpointFormula(Eigen::VectorXd& weights,
```

```
                Eigen::Matrix<double, Eigen::Dynamic, 2>& points);
// Points et poids de quadrature de la formule de Simpson (intégration 1D
// sur une arête) en fonction du numéro de l'arête du triangle de référence
void quadraturePointsAndWeightsSimpsonFormula (Eigen::VectorXd& weights,
                Eigen::Matrix<double, Eigen::Dynamic, 2>& points, int numedge);
```

Ce qui est doit être fait dans cette classe : Si vous partez de la version 1, les fonctions de cette classe doivent être implémentées. Chercher les *TODO* pour voir ce qui doit être fait. Des indications sont parfois données. Sinon pour passer en 3D, en P2 etc ..., de nombreuses modifications doivent être faites.

La classe Model

Elle permet de gérer tout ce qui concerne l'assemblage des matrices et du terme source ainsi que les conditions aux bords.

```
// Assemblage des matrices de masse et de rigidité
void assembleMassAndStiffnessMatrices();
// Assemblage du terme source et des conditions de Neumann
void assembleSourceAndNeumann(double t);
// Pour appliquer les conditions de Dirichlet à la matrice du système
void applyBCToSystemMatrix(Eigen::SparseMatrix<double,Eigen::RowMajor>
                                                    & systemMatrix);
// Pour appliquer les conditions de Dirichlet au membre de droite
void applyBCToRHS(Eigen::SparseVector<double> & RHS, double t);
```

Ce qui est doit être fait dans cette classe : Si vous partez des versions 1 et 2, les fonctions de cette classe doivent être implémentées. Chercher les *TODO* pour voir ce qui doit être fait. Des indications sont parfois données.

Validation du code

Le code utilise la librairie `Eigen`, se compile en utilisant :

```
make
```

et s'execute avec :

```
./run DataExactSol/data1.txt
```

si on souhaite utiliser le fichier de données *data1.txt* contenu dans le dossier *DataExactSol*. Les informations concernant la condition initiale, le terme source et les conditions aux bords (ainsi que la solution exacte si elle est connue) sont à mettre dans le fichier *InitialConditionSourceTermFile.cpp*.

Solution exacte Dans le dossier *DataExactSol* il y a tout ce qu'il faut pour valider sur une solution exacte le code. La fonction u est donnée par :

$$u(x,y,t) = \frac{1}{t+1}\cos(4\pi x y^2)\sin(5\pi x^2 y)$$

sur le carré $[0,1]\times[0,1]$. Avec le fichier *DataExactSol/data1.txt* correspondant au maillage *square1.mesh*, vous devez obtenir les erreurs suivantes :

```
Solve system at time 0.1
Error: 0.000231856
...
Solve system at time 1
Error: 0.000128479
```

Avec le fichier *DataExactSol/data2.txt* correspondant au maillage *square2.mesh* qui est deux fois plus raffiné, vous devez obtenir les erreurs suivantes :

```
Solve system at time 0.1
Error: 3.62397e-05
...
Solve system at time 1
Error: 1.96141e-05
```

Autres exemples Pour jouer un peu avec le code, deux autres exemples sont proposés :

- Un cas plus réaliste correspond au cas 2D proposé à la fin du TP 6. Tout est donné dans le dossier *DataRealisticSituation* (il faut changer le fichier *InitialConditionSourceTermFile*).
- Le cas d'un maillage avec un trou dans le dossier *DataHole*. Ne pas oublier non plus de changer le fichier *InitialConditionSourceTermFile*.