Master 1 Mathématiques Calcul Haute Performance, Simulation

Travail Encadré de Recherche

Résolution de problèmes inverses en thermique

Cédric Marinel

Encadrante : Ana Matos

14 mai 2018

Remerciements

Je souhaite remercier Pr. Ana Matos pour son accompagnement tout au long de ce projet. Sa disponibilité ainsi que ses explications m'ont été d'une grande aide.

Je souhaite également remercier Messieurs M.Bentivegni et G.Druart pour leur accueil lors de la visite de l'unité de recherche d'Aulnoye-Aymeries. Cette rencontre m'a permis de mieux comprendre les enjeux de la recherche dans une grande entreprise.

Table des matières

Introduction

Dans ce travail encadré de recherche, nous nous intéresserons aux problèmes inverses et à leurs applications. Dans un premier temps nous donnerons la définition d'un problème inverse ainsi que divers exemples d'applications notamment pour des problèmes thermiques. Dans un second temps nous étudierons les principales difficultés de ce type de problème et les solutions pour y remédier. Ensuite nous verrons un algorithme à direction de descente permettant de résoudre le problème, puis nous appliquerons la théorie à un cas académique. À travers l'application, nous testerons l'influence de différents paramètres sur la solution.

Chapitre 1

Définition d'un problème inverse et exemples :

1.1 Définition d'un problème inverse

Un problème inverse consiste à déterminer une ou plusieurs causes de ce problème en ayant connaissance des effets. On peut opposer les problèmes inverses aux problèmes directs dans lesquels on cherche une solution à partir de paramètres connus. Les problèmes directs nous semblent plus naturels car les mêmes causes conduisent au même effets. En revanche, il semble moins évident qu'un effet soit la conséquence d'une unique cause. C'est cette caractéristique qui fait toute la difficulté des problèmes inverses.

Les problèmes inverses interviennent dans plusieurs domaines :

- l'imagerie médicale (échographie, scanners, etc)
- le radar
- la chimie
- le traitement d'image (restauration d'images floues)

On différencie les problèmes linéaires où l'on se ramène à la résolution d'équation intégrale de première espèce des problèmes non-linéaires qui interviennent dans les équations aux dérivées partielles.

On peut considérer plusieurs types de problèmes inverses non linéaires :

- la reconstitution de l'état passé d'un système à partir de son état actuel,
- l'*identification de paramètres* qui consiste à déterminer un ou plusieurs paramètres du système en connaissant une partie de son évolution.

Dans ce travail de recherche, nous nous intéresserons à l'identification de paramètres pour l'équation de la chaleur.

1.2 Différents problèmes inverses autour de l'équation de la chaleur

1.2.1 Reconstitution de l'état passé :

On s'intéresse ici à la répartition de la température T dans un matériau hétérogène occupant un domaine de $\Omega \in \mathbb{R}^3$.

$$\begin{cases} \rho c \frac{\partial T}{\partial t}(t,x,y,z) - div\left(K \nabla T(t,x,y,z)\right) = f & \text{dans } \Omega \\ \frac{\partial T}{\partial n} = g & \text{sur } \partial\Omega \\ T(0,x,y,z) = T_0(x,y,z) \end{cases}$$

On peut essayer de retrouver la condition initiale $T_0(x,y,z)$ à partir d'une mesure de T à un temps donné et en connaissant les paramètres ρ c, et K .

1.2.2 Identification de la conductivité :

$$\begin{cases} \rho c \frac{\partial T}{\partial t}(t,x,y,z) - div\left(K(x,y,z)\nabla T(t,x,y,z)\right) = f & \text{dans } \Omega \\ \frac{\partial T}{\partial n} = g & \text{sur } \partial\Omega \\ T(0,x,y,z) = T_0(x,y,z) \end{cases}$$

Pour l'identification du paramètre K, on peut par exemple prendre comme données connues : plusieurs mesures de la température à différents moments ainsi qu'en différents points en supposant également que ρ et c sont connus.

1.2.3 Identification des conditions aux bords :

Ce problème est celui rencontré par l'entreprise Vallourec. Pour donner une certaine structure à leurs tuyaux, il faut le refroidir à une certaine température à un moment donné. On pose donc le système suivant :

$$\begin{cases} \rho(x,y,z)c(x,y,z)\frac{\partial T}{\partial t}(t,x,y,z) - div\left(K(x,y,z)\nabla T(t,x,y,z)\right) = f & \text{dans } \Omega \\ \frac{\partial T}{\partial n} = g & \text{sur } \partial\Omega \\ T(0,x,y,z) = T_0(x,y,z) \end{cases}$$

Le but de ce problème inverse est de retrouver les conditions aux bords g qui sont des conditions de Neumann. L'entreprise connaît les différents paramètres de l'équation d'état : ρ c, et K, et elle mesure la température en neuf points du tube à différents instants.

1.3 Définition d'un problème mal posé

Selon Jacques Hadamard [1], un problème mal posé est un problème qui vérifie au moins une des trois propriétés suivantes :

- le problème n'admet pas de solution,
- le problème admet plusieurs solutions,
- la solution ne dépend pas continûment des données.

Le problème direct de l'équation de la chaleur est un problème bien posé. En revanche, certains problèmes inverses peuvent être mal posés. Nous verrons plus tard une méthode permettant de régulariser les problèmes mal posés.

Chapitre 2

Formulation d'un problème inverse non linéaire

Pour donner une formulation abstraite des problèmes inverses, nous allons considérer trois espaces de Hilbert :
- l'espace des *paramètres* M ;
- l'espace d'*état* U ;
- l'espace des *données* D ;

Nous allons également définir deux applications :

L'équation d'état relie de façon implicite l'état et les paramètres. Nous l'écrirons :

$$F(a,u) = 0, a \in M, u \in U, F(a,u) \in Z \tag{2.1}$$

où Z est un autre espace de Hilbert. On notera la solution de l'équation d'état :

$$u = S(a) = u_a \tag{2.2}$$

L'équation d'observation extrait de l'état la partie correspondant aux mesures. Elle s'écrit :

$$d = Hu \tag{2.3}$$

En injectant (2.2) dans (2.3), on a :

$$d = H(S(a)) = \Phi(a) \tag{2.4}$$

Le problème inverse consiste à résoudre l'équation étant donné une observation d_{obs} :

$$\Phi(a) = d_{obs} \tag{2.5}$$

L'application Φ est définie implicitement. De plus, Φ est non-linéaire et ce, même si l'équation d'état et l'équation d'observation sont linéaires. On peut donc penser que l'équation (2.4) n'a pas forcément de solution, et que même si elle admet une solution, son inverse n'est pas nécessairement continue.

Pour résoudre le problème $\Phi(a) = d_{obs}$, nous allons utiliser une formulation plus faible : la formulation par moindres carrés. Pour cela, nous allons résoudre :

$$\min_{\mathbf{a}} J(a) = \min_{\mathbf{a}} \frac{1}{2}||\Phi(\mathbf{a}) - d_{obs}||_D^2 \tag{2.6}$$

On appelle J la fonction coût.

Difficultés des problèmes inverses

On peut citer quatre difficultés pour les problèmes inverses :

Premièrement, la fonction coût peut être non convexe. Cela entraîne l'existence de minima locaux. La méthode d'optimisation peut donc converger vers n'importe lequel de ces minima.

La deuxième difficulté réside dans la quantité de données que l'on peut avoir. En effet, un manque de données peut entraîner un problème inverse *sous déterminé*. Il peut donc y avoir plusieurs solutions, c'est à dire plusieurs paramètres produisant les mêmes observations.

La qualité des données peut également être une difficulté. En effet, les données peuvent être bruitées et entraîner un manque de continuité qui produit une *instabilité*. On ne peut donc garantir de réussir à résoudre le problème pour des données fortement bruitées.

La dernière difficulté réside dans le coût de l'algorithme. La méthode de résolution va nous forcer à résoudre à chaque itération la fonction d'état, c'est à dire une ou plusieurs équations aux dérivées partielles.

On peut voir que les données jouent un rôle essentiel dans les problèmes inverses. Malheureusement, les contraintes réelles telles que le nombre d'instruments de mesure ou la précision des mesures entraînent généralement des difficultés dans la résolution du problème.

Chapitre 3

Régularisation du problème par la méthode de Tikhonov

Comme nous l'avons vu précédemment, la principale difficulté des problèmes inverses est que ces derniers sont souvent mal posés. La régularisation de Tikhonov permet de traiter un problème légèrement différent qui sera, lui, bien posé. En effet, on remplace $\underset{\mathbf{a}}{min}\ J(a) = \underset{\mathbf{a}}{min}\ \frac{1}{2}||\Phi(\mathbf{a}) - d_{obs}||^2$ par :

$$\underset{\mathbf{a}}{min}\ \widetilde{J}(\mathbf{a}) = \underset{\mathbf{a}}{min}\ \left(\frac{1}{2}||\Phi(\mathbf{a}) - d_{obs}||^2 + \frac{\epsilon^2}{2}||\widetilde{\mathbf{a}} - \mathbf{a}||^2\right) \tag{3.1}$$

$\widetilde{\mathbf{a}}$ est appelé estimation a priori de $\mathbf{a}$, ϵ est le coefficient de régularisation. Le choix du coefficient de régularisation est délicat. En effet, un ϵ trop petit rendra $\widetilde{J}$ trop proche de J et donc mal posé, alors qu'un ϵ trop grand forcera la solution à être proche de $\widetilde{\mathbf{a}}$.

Pour cette partie, on considérera que $\Phi(\mathbf{a})$ est linéaire, i.e. $\Phi(\mathbf{a}) = \mathbf{Ma}$.
On a donc $\widetilde{J}(\mathbf{a}) = \frac{1}{2}||\mathbf{Ma} - d_{obs}||^2 + \frac{\epsilon^2}{2}||\widetilde{\mathbf{a}} - \mathbf{a}||^2$.

Définition : On appelle adjoint de $A : E \to F$ l'unique opérateur $A^* : F \to E$ tel que :

$$\forall u \in E,\ \forall v F :\ (Au, v) = (u, A^* v)$$

Lemme : Posons : $\widetilde{\mathbf{M}} = \begin{pmatrix} \mathbf{M} \\ \epsilon\mathbf{I} \end{pmatrix}$ et $\widetilde{d_{obs}} = \begin{pmatrix} d_{obs} \\ \epsilon\ \widetilde{\mathbf{a}} \end{pmatrix}$

On a :

$$\underset{\mathbf{a}}{min}\ \widetilde{J}(\mathbf{a}) \ \text{ équivalent à } \ \underset{\mathbf{a}}{min}\ \frac{1}{2}||\widetilde{\mathbf{M}}\mathbf{a} - \widetilde{d_{obs}}||^2$$

Proposition 1 : Le problème (3.1) est équivalent à :

$$(\mathbf{M}^*\mathbf{M} + \epsilon^2 I)\mathbf{a} = \mathbf{M}^* d_{obs} + \epsilon^2 \widetilde{\mathbf{a}} \tag{3.2}$$

De plus, ce problème admet une unique solution qui dépend continûment de d_{obs}.

Avant de prouver la proposition, nous admettrons le théorème suivant :
Théorème de l'application ouverte : Soit A un opérateur linéaire de E dans F. L'image par A d'un ouvert de E est un ouvert de F.
En particulier l'inverse d'un opérateur linéaire continu et bijectif est continu.

Preuve de la proposition. En remarquant que $\widetilde{\mathbf{M}}^* = (\mathbf{M}^*, \epsilon I)$, on a que (3.2) est l'équation normale de (3.1). Étudions maintenant l'existence et l'unicité de la solution :

$$((\mathbf{M}^*\mathbf{M} + \epsilon^2 I)\mathbf{a}, \mathbf{a}) = ||\mathbf{M}\mathbf{a}||^2 + \epsilon^2||\mathbf{a}||^2 \geq \epsilon^2||\mathbf{a}||^2$$

On applique le lemme de Lax-Milgram. D'après le théorème de l'application ouverte, l'opérateur $M^*M + \epsilon^2 I$, continu et bijectif, a un inverse continu. En prenant le produit scalaire de l'équation (3.2), on obtient une estimation :

$$\begin{aligned} ||\mathbf{M}\mathbf{a}||^2 + \epsilon^2||\mathbf{a}||^2 &\leq ||\mathbf{M}*d_{obs}||\mathbf{a}|| + \epsilon^2||\widetilde{\mathbf{a}}||||\mathbf{a}|| \\ \epsilon^2||\mathbf{a}||^2 &\leq ||\mathbf{M}*d_{obs}||||a|| + \epsilon^2||\widetilde{\mathbf{a}}||||\mathbf{a}|| \\ ||\mathbf{a}|| &\leq \frac{1}{\epsilon^2}||\mathbf{M}*d_{obs}|| + ||\widetilde{\mathbf{a}}|| \end{aligned}$$

Nous allons maintenant nous intéresser au comportement de l'erreur. Nous allons montrer que la méthode de Tikhonov donne une estimation sur l'erreur commise. On suppose connue une mesure idéale : $\hat{d}_{obs} \in Im\ M$, ainsi qu'une suite d'observations bruitées $d_n \notin Im\ M, avec\ \delta_n = ||d_n - \hat{d}_{obs}|| \underset{n\to\infty}{\longrightarrow} 0$

On considère la suite de problèmes :

$$\min_{\mathbf{a}} \quad \frac{1}{2}||M\mathbf{a} - d_n||_F^2 + \epsilon^2||\mathbf{a} - \widetilde{\mathbf{a}}||_E^2 \tag{3.3}$$

D'après la proposition 1, (3.3) admet une unique solution que l'on notera : $\mathbf{a}_\epsilon^n$. On note également $\hat{\mathbf{a}}$ la solution de : $\min_{\mathbf{a}} \quad \frac{1}{2}||M\mathbf{a} - \hat{d}_{obs}||_F^2 + \epsilon^2||\mathbf{a} - \widetilde{\mathbf{a}}||_E^2$.

Proposition 2 : Supposons que $\hat{\mathbf{a}} \in Im\ M^*$, soit $\hat{\mathbf{a}} = M^*w$. Alors ,

$$\forall n,\ ||\mathbf{a}_\epsilon^n - \hat{\mathbf{a}}||_E \leq ||M^*||\frac{\delta_n}{\epsilon^2} + \frac{\epsilon}{\sqrt{2}}||w||_F$$

Preuve Introduisons la quantité $\mathbf{a}_\epsilon$ solution du problème :

$$M^*M\mathbf{a}_\epsilon + \epsilon^2\mathbf{a}_\epsilon = M^*\hat{d}_{obs} + \epsilon^2\widetilde{\mathbf{a}} \tag{3.4}$$

On a :

$$||\mathbf{a}_\epsilon^n - \widetilde{\mathbf{a}}|| \leq ||\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon|| + ||\mathbf{a}_\epsilon - \hat{\mathbf{a}}||$$

Nous allons majorer séparément les deux termes de droite. La première partie correspond à l'erreur sur les données, elle est amplifiée par le caractère mal posé du problème. La seconde partie est due à l'approximation de la solution exacte.

L'équation normale associée à (3.3) nous donne :

$$M^*M\mathbf{a}_\epsilon^n + \epsilon^2\mathbf{a}_\epsilon^n = M^*d_n + \epsilon^2\widetilde{\mathbf{a}} \tag{3.5}$$

En soustrayant (3.4) à (3.5), on a :

$$\begin{aligned} & M^*M(\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon) + \epsilon^2(\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon) = M^*(d_n - \hat{d}_{obs}) \\ \Rightarrow & ||M(\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon)||^2 + \epsilon^2||\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon||^2 \leq ||M^*||\ ||d_n - \hat{d}_{obs}||\ ||\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon|| \\ \Rightarrow & \epsilon^2||\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon||^2 \leq ||M^*||\delta_n \\ \Rightarrow & ||\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon||^2 \leq ||M^*||\frac{\delta_n}{\epsilon^2} \end{aligned}$$

On a donc une majoration de la première partie de l'inégalité triangulaire.

Pour la seconde partie :

$$\begin{aligned}
||M\mathbf{a}_\epsilon - M\hat{\mathbf{a}}||^2 &= (M\mathbf{a}_\epsilon - M\hat{\mathbf{a}}, M\mathbf{a}_\epsilon - M\hat{\mathbf{a}}) \\
&= (M^*M\mathbf{a}_\epsilon - M^*M\hat{\mathbf{a}}, \mathbf{a}_\epsilon - \hat{\mathbf{a}}) \\
&= (\epsilon^2\mathbf{a}_\epsilon - \epsilon^2\hat{\mathbf{a}} + \epsilon^2\hat{\mathbf{a}}, \mathbf{a}_\epsilon - \hat{\mathbf{a}}) \\
&= -\epsilon^2||\mathbf{a}_\epsilon - \hat{\mathbf{a}}||^2 + \epsilon^2(\hat{\mathbf{a}}, \mathbf{a}_\epsilon - \hat{\mathbf{a}}) \\
&= -\epsilon^2||\mathbf{a}_\epsilon - \hat{\mathbf{a}}||^2 + \epsilon^2\,(w, M(\mathbf{a}_\epsilon - \hat{\mathbf{a}})) \\
&\leq -\epsilon^2||\mathbf{a}_\epsilon - \hat{\mathbf{a}}||^2 + \epsilon^2||w||\;||M(\mathbf{a}_\epsilon - \hat{\mathbf{a}})|| \\
&\leq -\epsilon^2||\mathbf{a}_\epsilon - \hat{\mathbf{a}}||^2 + \frac{\epsilon^4}{2}||w||^2 + \frac{1}{2}||M(\mathbf{a}_\epsilon - \hat{\mathbf{a}})||^2 \quad \text{car : } bc \leq \frac{b^2+c^2}{2}
\end{aligned}$$

$$\begin{aligned}
\text{D'où : } \frac{1}{2}||M(\mathbf{a}_\epsilon - \hat{\mathbf{a}})||^2 + \epsilon^2||\mathbf{a}_\epsilon - \hat{\mathbf{a}}||^2 &\leq \frac{\epsilon^4}{2}||w||^2 \\
\Rightarrow ||\mathbf{a}_\epsilon - \hat{\mathbf{a}}|| &\leq \frac{\epsilon}{\sqrt{2}}||w||
\end{aligned}$$

On a donc :

$$\forall n,\ ||\mathbf{a}_\epsilon^n - \hat{\mathbf{a}}||_E \leq ||\mathbf{a}_\epsilon^n - \mathbf{a}_\epsilon|| + ||\mathbf{a}_\epsilon - \hat{\mathbf{a}}|| \leq ||M^*||\frac{\delta_n}{\epsilon^2} + \frac{\epsilon}{\sqrt{2}}||w||_F$$

□

L'erreur est donc composée de deux termes :
- l'un dû aux erreurs sur les données, qui tend vers l'infini quand $\epsilon \to 0$.
- l'autre dû à l'approximation de la solution exacte, qui tend vers 0 quand $\epsilon \to 0$.

Il faudra donc choisir le coefficient de régularisation en fonction des données. Pour cela, nous effectuerons plusieurs fois la méthode pour différents paramètres ϵ, puis nous choisirons celui qui minimise la fonction coût.

Chapitre 4

Application à l'équation de la chaleur 1D

On s'intéresse au problème suivant :

$$\begin{cases} \rho c \frac{\partial T}{\partial t}(t,x) - \frac{\partial}{\partial x}\left(K(x)\frac{\partial T}{\partial x}(t,x)\right) = 0 \quad \text{dans }]0,1[\\ T(t,0) = T(t,1) = 0 \\ T(0,x) = T_0(x) \end{cases}$$

L'objectif est de retrouver la conductivité K(x) à partir de cinq mesures en espace à quatre temps différents. Pour rappel, la conductivité est la grandeur physique qui caractérise l'aptitude d'un corps à conduire la chaleur. Plus la conductivité est élevée, plus le corps conduit la chaleur. Pour ce problème, on suppose que la conductivité ne dépend que de l'endroit où l'on se trouve dans le domaine.

On notera a le paramètre à retrouver (ici $K(x)$), d_{obs} les données observées, $\Phi(a)$ la partie de la solution du problème directe comparable avec les données.

Pour retrouver le paramètre a, on cherche à résoudre le problème de moindres carrés :

$$\min_{\mathbf{a}} J(a) = \min_{\mathbf{a}} \frac{1}{2}||\Phi(\mathbf{a}) - d_{obs}||_D^2$$

Pour résoudre ces problèmes de minimisation, nous utiliserons l'algorithme de Broyden-Fletcher-Goldfarb-Shanno que nous détaillerons plus tard. Cet algorithme est une méthode itérative semblable aux algorithmes vus au premier semestre [2] .

4.1 Résolution du problème direct

Pour résoudre l'équation d'état, nous utiliserons la méthode des différences finies. Puisque nous avons vu que l'une des difficultés dans la résolution était le coût de l'algorithme, nous allons prendre un schéma explicite en temps. Celui-ci nous évitera de résoudre un système linéaire à chaque itération de temps.

En utilisant les différences finies, on a :

$$\frac{\left[K(x+\frac{h}{2}) + K(x-\frac{h}{2})\right]\cdot T(t,x) - K(x+\frac{h}{2})\cdot T(t,x+h) - K(x-\frac{h}{2})\cdot T(t,x-h)}{h^2} = -\frac{\partial}{\partial x}\left(K(x)\cdot\frac{\partial T}{\partial x}(t,x)\right) + O(h^2)_{h\to 0}$$

Pour résoudre, le problème dans le sens direct, on utilisera donc le schéma explicite en temps suivant :

$$\rho c \frac{T_i^{n+1} - T_i^n}{\Delta t} + \frac{(K_{i+\frac{1}{2}} + K_{i+\frac{1}{2}})\cdot T_i^n - K_{i+\frac{1}{2}}\cdot T_{i+1}^n - K_{i-\frac{1}{2}}\cdot T_{i-1}^n}{h^2} = 0 \tag{4.1}$$

En utilisant les conditions au bord, on se ramène au système suivant à chaque pas de temps :

$$\begin{pmatrix} T_1^{n+1} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ T_{N-1}^{n+1} \end{pmatrix} = \begin{pmatrix} T_1^{n} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ T_{N-1}^{n} \end{pmatrix} - \frac{\Delta t}{\rho c h^2} A * \begin{pmatrix} T_1^{n} \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ T_{N-1}^{n} \end{pmatrix}$$

$$A = \begin{pmatrix} K_{1-\frac{1}{2}} + K_{1+\frac{1}{2}} & -K_{1+\frac{1}{2}} & 0 & \dots & \dots & 0 \\ -K_{2-\frac{1}{2}} & K_{2-\frac{1}{2}} + K_{2+\frac{1}{2}} & -K_{2+\frac{1}{2}} & 0 & & \vdots \\ 0 & \ddots & \ddots & 0 & & \vdots \\ \vdots & 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & 0 & -K_{N-2-\frac{1}{2}} & K_{N-2-\frac{1}{2}} + K_{N-2+\frac{1}{2}} & -K_{N-2+\frac{1}{2}} \\ 0 & \dots & \dots & 0 & -K_{N-1-\frac{1}{2}} & K_{N-1-\frac{1}{2}} + K_{N-1+\frac{1}{2}} \end{pmatrix}$$

Et on a $T_N^n = T_0^n = 0 \quad \forall n$

Consistance du schéma :

Théorème : Soient $T \in C^3(0,T;0,1)$ et $K \in C^3(0,1)$,
Le schéma (4.1) est consistant à l'ordre 1 en temps et 2 en espace

Preuve : Pour prouver l'ordre de consistance du schéma, on utilise les développements de Taylor suivants :

$$\begin{aligned} T(t+\Delta t,x) &= T(t,x) + \Delta t \frac{\partial T}{\partial t}(t,x) + O(\Delta t^2) \\ K(x+\frac{h}{2}) &= K(x) + \frac{h}{2}K'(x) + \frac{1}{2}\frac{h^2}{4}K''(x) + \frac{1}{6}\frac{h^2}{8}K^{(3)}(x) + O(h^4) \\ K(x-\frac{h}{2}) &= K(x) - \frac{h}{2}K'(x) + \frac{1}{2}\frac{h^2}{4}K''(x) - \frac{1}{6}\frac{h^2}{8}K^{(3)}(x) + O(h^4) \\ T(t,x+h) &= T(t,x) + h\frac{\partial T}{\partial x}(t,x) + \frac{h^2}{2}\frac{\partial^2 T}{\partial x^2}(t,x) + \frac{h^3}{6}\frac{\partial^3 T}{\partial x^3}(t,x) + O(h^4) \\ T(t,x-h) &= T(t,x) - h\frac{\partial T}{\partial x}(t,x) + \frac{h^2}{2}\frac{\partial^2 T}{\partial x^2}(t,x) - \frac{h^3}{6}\frac{\partial^3 T}{\partial x^3}(t,x) + O(h^4) \end{aligned}$$

Tout d'abord on a :

$$\frac{T(t+\Delta t,x) - T(t,x)}{\Delta t} = \frac{\partial T}{\partial t}(t,x) + O(\Delta t)$$

Ensuite, on a :

$$
\begin{aligned}
[K(x+\frac{h}{2})+K(x-\frac{h}{2})]\ T(t,x) &= [2K(x)+\frac{h^2}{4}K''(x)]\ T(t,x)+O(h^4)\\
-K(x+\frac{h}{2})\ T(t,x+h)-K(x-\frac{h}{2})\ T(t,x-h) &= -[K(x)\ T(t,x)+h\ K(x)\frac{\partial T}{\partial x}(t,x)+\frac{h^2}{2}K(x)\frac{\partial^2 T}{\partial x^2}(t,x)\\
&\quad+\frac{h^3}{6}K(x)\frac{\partial^3 T}{\partial x^3}(t,x)+\frac{h}{2}K'(x)T(t,x)+\frac{h^2}{2}K'(x)\frac{\partial T}{\partial x}(t,x)\\
&\quad+\frac{h^3}{4}K'(x)\frac{\partial^2 T}{\partial x^2}(t,x)+\frac{h^2}{8}K''(x)T(t,x)+\frac{h^3}{8}K''(x)\frac{\partial T}{\partial x}(t,x)\\
&\quad+\frac{h^3}{48}K^{(3)}T(t,x)+K(x)\ T(t,x)-h\ K(x)\frac{\partial T}{\partial x}(t,x)+\frac{h^2}{2}K(x)\frac{\partial^2 T}{\partial x^2}(t,x)\\
&\quad-\frac{h^3}{6}K(x)\frac{\partial^3 T}{\partial x^3}(t,x)-\frac{h}{2}K'(x)T(t,x)+\frac{h^2}{2}K'(x)\frac{\partial T}{\partial x}(t,x)\\
&\quad-\frac{h^3}{4}K'(x)\frac{\partial^2 T}{\partial x^2}(t,x)+\frac{h^2}{8}K''(x)T(t,x)-\frac{h^3}{8}K''(x)\frac{\partial T}{\partial x}(t,x)\\
&\quad-\frac{h^3}{48}K^{(3)}T(t,x)]+O(h^4)\\
&= -2K(x)T(t,x)-h^2K(x)\ \frac{\partial^2 T}{\partial x^2}(t,x)-h^2K'(x)\ \frac{\partial T}{\partial x}(t,x)-\frac{h^2}{4}K''(x)\ T(t,x)+O(h^4)
\end{aligned}
$$

En regroupant les deux termes, on a :

$$
\frac{[K(x+\frac{h}{2})+K(x-\frac{h}{2})]\ T(t,x)-K(x+\frac{h}{2})\ T(t,x+h)-K(x-\frac{h}{2})\ T(t,x-h)}{h^2}=-\frac{\partial}{\partial x}\left(K(x)\frac{\partial T}{\partial x}\right)+O(h^2)
$$

On a donc une consistance d'ordre 1 en temps et 2 en espace.

□

Stabilité du schéma :

Théorème : Soit $\lambda=\max\limits_{x\in[0,1]}K(x)$, si $\Delta t\leq\frac{\rho ch^2}{2\lambda}$, alors le schéma (4.1) est stable.

Preuve : Montrons par récurrence que $\max\limits_{0\leq n\leq N_T}||T^n||_{L^\infty}\leq C_1(T)||T^0||_{L^\infty}$. On a bien $0\leq T_0(x)\leq M_0,\ \forall x\in[0,1]$ avec $M_0=||T^0||_{L^\infty}$. Supposons que :$0\leq T_i^n(x)\leq M_0$

$$
\begin{aligned}
T_i^{n+1} &= T_i^n-\frac{\Delta t}{\rho ch^2}\left(-K_{i+\frac{1}{2}}T_{i+1}^n+(K_{i+\frac{1}{2}}+K_{i-\frac{1}{2}})T_i^n-K_{i-\frac{1}{2}}T_{i-1}^n\right)\\
&= T_i^n\left(1-(K_{i+\frac{1}{2}}+K_{i-\frac{1}{2}})\frac{\Delta t}{\rho ch^2}\right)+\frac{\Delta t}{\rho ch^2}K_{i+\frac{1}{2}}T_{i+1}^n+\frac{\Delta t}{\rho ch^2}K_{i-\frac{1}{2}}T_{i-1}^n
\end{aligned}
$$

En supposant que $1-(K_{i+\frac{1}{2}}+K_{i-\frac{1}{2}})\frac{\Delta t}{\rho ch^2}\geq 0,\ \forall i\in[0,N]$, on a bien :$0\leq T_i^{n+1}\leq M_0$.

Il faut donc que :

$$
\begin{aligned}
1-(K_{i+\frac{1}{2}}+K_{i-\frac{1}{2}})\frac{\Delta t}{\rho ch^2}\geq 0,\ \forall i\in[0,N] &\Leftrightarrow \Delta t\leq\frac{\rho ch^2}{K_{i+\frac{1}{2}}+K_{i-\frac{1}{2}}}\ \forall i\in[0,N]\\
&\Leftrightarrow \Delta t\leq\frac{\rho ch^2}{2\max\limits_{x\in[0,1]}K(x)}
\end{aligned}
$$

□

Corollaire : Soient $T \in C^3(0,T;0,1)$, $K \in C^3(0,1)$ et $\Delta t \leq \frac{\rho c h^2}{2\lambda}$, le schéma (4.1) est convergent.

Remarque : Pour la résolution on calculera $\lambda = \underset{x\in[0,1]}{max} K(x)$ puis on posera $\Delta t = \frac{\rho c h^2}{2\lambda}$. Si la conductivité est très grande, on peut donc se retrouver avec un grand nombre d'itération.

4.2 Résolution du problème à l'aide de l'algorithme BFGS

Pour résoudre ces problèmes de minimisation, nous utiliserons l'algorithme de Broyden-Fletcher-Goldfarb-Shanno. C'est un algorithme itératif basé sur la méthode de Quasi-Newton. Il est souvent utilisé pour la résolution de problèmes inverses [3].

Pour résoudre le problème $\min_{\mathbf{a}} \frac{1}{2}||\Phi(\mathbf{a}) - d_{obs}||^2$, on utilise l'algorithme BFGS.

- Initialisation : $a^0 =$ Voir articles d'origine
- Tant que $||\nabla J(\mathbf{a}^n)|| > tol, \quad n = 1, 2, \ldots$
  1. Calcul du gradient $\nabla J(\mathbf{a}^n)$ à l'aide des différences finies
  2. Calcul récursif d'une approximation de l'inverse de la matrice Hessienne $H^{-1}(\mathbf{a}^n)$.
  3. Calcul de la direction : $d^n = -H^{-1}(\mathbf{a}^n)\nabla J(\mathbf{a}^n)$
  4. Recherche d'un pas α^n vérifiant les conditions de Wolfe :

$$\begin{cases} J(\mathbf{a}^n + \alpha^n d^n) - J(\mathbf{a}^n) \leq \omega_1 \mathbf{a}^n(\nabla J(\mathbf{a}^n), d^n) \\ (\nabla J(\mathbf{a}^n + \alpha^n d^n), d^n) \geq \omega_2(\nabla J(\mathbf{a}^n), d^n) \end{cases}$$

  5. Calcul du nouvel itéré : $\mathbf{a}^{n+1} = \mathbf{a}^n + \alpha^n d^n$

Calcul de la fonction coût :

Avant de détailler les différentes étapes de l'algorithme BFGS, nous allons d'abord expliciter la manière dont on calcule la fonction coût $J(\mathbf{a})$:

— Tout d'abord, on utilise le schéma différences finies explicite en temps de la partie précédente pour stocker dans une matrice les valeurs de la solution de l'équation d'état à chaque pas de temps et pas d'espace.

— Ensuite, on extrait les valeurs de cette matrice aux mêmes temps et mêmes points que nos données, ce qui nous donne $\phi(\mathbf{a})$.

— Enfin, on calcule : $\frac{1}{2}||\Phi(\mathbf{a}) - d_{obs}||^2$.

Voici maintenant les détails des différentes étapes de l'algorithme BFGS :

4.2.1 Calcul du gradient :

Pour calculer le gradient, on peut utiliser différentes méthodes :

- la méthode des différences finies
- les fonctions des sensibilités
- la méthode de l'état adjoint

Ici nous nous intéresserons aux différences finies qui est une méthode déjà connue et facile à implémenter. Puis nous verrons à travers un exemple la méthode de l'état adjoint qui est basée sur la théorie des éléments finis.

Méthode des différences finies :

La méthode des différences finies consiste à approcher le gradient en utilisant les développements de Taylor. Si $a \in \mathbb{R}^m$, $\forall i \in 1, \ldots, m$:

$$J(a + h * e_i) = J(a) + h\frac{\partial J}{\partial a_i}(a) + h^2\frac{\partial^2 J}{\partial a_i^2}(a) + O(h^3)$$

$$J(a - h * e_i) = J(a) - h\frac{\partial J}{\partial a_i}(a) + h^2\frac{\partial^2 J}{\partial a_i^2}(a) + O(h^3)$$

Ce qui nous donne :

$$\frac{\partial J}{\partial a_i} = \frac{J(a_1, a_2, \ldots, a_i + h, \ldots, a_m) - J(a_1, a_2, \ldots, a_i - h, \ldots, a_m)}{2h} + O(h^2)_{h\to 0}$$

La méthode ci-dessus permet d'avoir une approximation du gradient d'ordre 2 mais elle impose deux calculs de la fonction coût pour chaque composante du gradient.

En fonction du niveau de bruit, on pourrait envisager, afin de limiter le nombre de calcul, de ne prendre qu'une approximation d'ordre 1. On aurait donc :

$$\frac{\partial J}{\partial a_i} = \frac{J(a_1, a_2, \ldots, a_i + h, \ldots, a_m) - J(a_1, a_2, \ldots, a_i, \ldots, a_m)}{2h} + O(h)_{h\to 0}$$

Cette deuxième méthode est donc moins précise mais divise le nombre de calcul par deux. Elle peut donc être utilisée pour réduire le coût de l'algorithme.

Méthode de l'état adjoint :

La méthode de l'état adjoint est une méthode permettant de calculer le gradient en utilisant la formulation variationnelle.

Tout d'abord, pour la méthode générale, on suppose que les variables a et u varient indépendamment, et on considère l'équation d'état comme une contrainte. Puis on introduit un lagrangien :

$$\mathbf{L}(a, u, p) = \frac{1}{2}||Hu - d_{obs}||^2 + (p, F(a, u)) \tag{4.2}$$

Si u vérifie l'équation d'état correspondant au paramètre a, on a l'identité :

$$\mathbf{L}(a, u(a), p) = J(a), \ \forall p \in Z$$

En dérivant cette relation, on a :

$$J'(a)\delta a = \partial_a \mathbf{L}(a, u)\delta a + \partial_u \mathbf{L}(a, u)\partial_a u(a)\delta a \tag{4.3}$$

$\partial_a u(a)$ étant difficile à calculer, on essayera de choisir $p \in Z$ pour que ce terme disparaisse. On considère que $\delta u = \partial_a u(a)\delta a$ est une quantité indépendante, on définit alors l'équation adjointe :

$$\partial_u \mathbf{L}(a, u(a))\delta u = 0, \quad \forall \delta u \in U \tag{4.4}$$

En explicitant l'expression de $\partial_u \mathbf{L}$, on a :

$$(H\delta u, Hu(a) - d_{obs}) + (p, \partial_u F(a, u(a))\delta u) = 0, \quad \forall \delta u \in U \tag{4.5}$$

On a donc :

$$\partial_u F(a, u)^* p = -H^*(Hu(a) - d_{obs}) \tag{4.6}$$

La différentielle de J se calcule par :

$$J'(a)\delta a = (p, \partial_a F(a, u(a))\delta a) \tag{4.7}$$

Théorème : Le calcul de $\nabla J(a)$ passe par trois étapes :

1. Définir le lagrangien par (4.2) ;
2. Résoudre l'équation adjointe (4.5) qui détermine l'état adjoint p ;
3. La différentielle de J est donnée par (4.7) et permet d'identifier le gradient de J.

Exemple pour notre problème inverse :
Étant donné $N \in \mathbb{N}$, étant notons $\Delta t = \frac{T}{N}$. On considère que a est déjà discret pour cet exemple. Notre problème peut s'écrire sous la forme suivante :

$$\begin{cases} \frac{y^n - y^{n-1}}{\Delta t} = f(y^{n-1}, a), \qquad n = 1, \dots, N, \\ y^0 = y_0 \end{cases} \tag{4.8}$$

Nous considérons, que les données sont connues en certains temps multiples de Δt que l'on notera $\tau_1, \dots, \tau_Q$. Nous noterons également $n_q = \frac{\tau_q}{\delta t}$. La fonction coût est définie par :

$$J(a) = \frac{1}{2} \sum_{q=1}^{Q} |y_a(\tau_q) - d_{obs}^q|^2 \tag{4.9}$$

où y_a est solution de (4.8).
Comme nous l'avons vu précédemment, nous allons passer par le lagrangien :

$$L_h(a, y, p) = \frac{1}{2} \sum_{q=1}^{Q} |y^{n_q} - d^q|^2 \Delta t + \sum_{n=1}^{N} (p^{n-1})^t \left(\frac{y^n - y^{n-1}}{\Delta t} + f(y^{n-1}, a) \right) \Delta t \tag{4.10}$$

Ensuite, nous formons l'équation adjointe $\forall \delta y = (\delta y^0, \dots, \delta y^N)$:

$$\frac{\partial L}{\partial y} \delta y = \sum_{q=0}^{N} (y^q - d^q)^t \delta y^q \Delta t + \sum_{n=1}^{N} (p^{n-1})^t \left(\frac{\delta y^n - \delta y^{n-1}}{\Delta t} - \partial_y f(y^{n-1}, a) \delta y^{n-1} \right) \Delta t = 0 \tag{4.11}$$

Nous cherchons à faire apparaître δy^n en facteur. Pour cela nous opérons un changement d'indice :

$$\sum_{n=1}^{N} \delta y^{n-1} p^{n-1} = \sum_{n=0}^{N-1} \delta y^n p^n$$

Nous pouvons donc réécrire l'équation adjointe de la manière suivante :

$$\sum_{q=1}^{Q} (y^{n_q} - d^q)^t \delta y^{n_q} \Delta t + \sum_{n=1}^{N} (p^{n-1})^t \delta y^n - \sum_{n=0}^{N-1} (p^n)^t \delta y^n - \sum_{n=0}^{N-1} (p^n)^t \partial_y f(y^n, a) \delta y^n \Delta t = 0 \tag{4.12}$$

Définissons le vecteur $r^n, n = 0, \dots, N$ par

$$r^{n_q} = \begin{cases} d^q - y^{n_q}, & q = 1, \dots, Q, \\ 0 & sinon. \end{cases} \tag{4.13}$$

Puisque y_0 ne dépend pas de a, on doit avoir $\delta y^0 = 0$. Pour avoir toutes les sommes allant de 0 à N, nous introduisons un nouveau multiplicateur $p^N = 0$. Nous obtenons alors, $\forall \delta y = (\delta y^0, \dots, \delta y^N)$:

$$-\sum_{n=0}^{N} (r^n)^t \delta y^n \Delta t + \sum_{n=0}^{N} (p^{n-1})^t \delta y^n - \sum_{n=0}^{N} (p^n)^t \delta y^n - \sum_{n=0}^{N} (p^n)^t \partial_y f(y^n, a) \delta y^n \Delta t = 0 \tag{4.14}$$

Cette équation étant valable pour choix de $\delta y^0, \dots, y^N$. Nous pouvons donc prendre toutes les variations nulles sauf celles correspondant à un pas de temps à la fois. Après transposition, nous avons donc :

$$\frac{p^{n-1} - p^n}{\Delta t} + \partial_y f(y^n, a)^t p^n = r^n, \quad n = N, \dots, 1 \tag{4.15}$$

auquel nous devons ajouter la condition finale $p^N = 0$. L'équation adjointe apparaît ici comme un schéma aux différences finies en temps rétrograde. Nous pouvons donc maintenant calculer la différentielle :

$$J'(a)\delta a = \sum_{i=0}^{N} (p^n)^t \partial_a f(y^n, a)\delta a, \tag{4.16}$$

On en déduit donc le gradient de J :

$$\nabla J(a) = \sum_{n=0}^{N} \partial_a f(y^n, a)^t p^n \tag{4.17}$$

L'équation étant en temps rétrograde, elle nécessite de connaître simultanément $y^n = p^{n-1}$ pour calculer le gradient de J cela nécessite donc le stockage de l'ensemble des vecteurs y.

Remarque : Le calcul du gradient par la méthode de l'état adjoint peut également se faire par la méthode des éléments finis.

4.2.2 Calcul récursif des approximations des matrices Hessiennes inverses :

L'une des forces de l'algorithme BFGS réside dans le calcul de l'approximation de la matrice Hessienne. En effet, on peut en calculer une approximation de manière itérative. À l'étape $n+1$, on pose :

$$\mathbf{y}_n = \nabla J(\mathbf{a}^{n+1}) - \nabla J(\mathbf{a}^n)$$
$$\mathbf{s}_n = \mathbf{a}^{n+1} - \mathbf{a}^n$$

Puis on calcule la nouvelle matrice Hessienne de la manière suivante [4] :

$$\mathrm{H}_{n+1} = \mathrm{H}_n + \frac{\mathbf{y}_n \mathbf{y}_n^\top}{\mathbf{y}_n^\top \mathbf{s}_n} - \frac{\mathrm{H}_n \mathbf{s}_n \mathbf{s}_n^\top \mathrm{H}_n}{\mathbf{s}_n^\top \mathrm{H}_n \mathbf{s}_n}$$

Pour la première itération, on calcule la matrice Hessienne par la méthode des différences finies, puis on résout le système linéaire à l'aide d'une décomposition LU. Cette méthode nous permet d'éviter de résoudre k^2 problèmes directes pour trouver la matrice Hessienne (k= nombre de paramètres à retrouver).

4.2.3 Détail de l'algorithme Fletcher Lemaréchal pour la recherche du pas

Cet algorithme permet de trouver un pas α^n vérifiant les conditions de Wolfe [5] :

$$J(\mathbf{a}^n + \alpha^n d^n) - J(\mathbf{a}^n) \leq \omega_1 \mathbf{a}^n(\nabla J(\mathbf{a}^n), d^n) \quad (1)$$
$$(\nabla J(\mathbf{a}^n + \alpha^n d^n), d^n) \geq \omega_2(\nabla J(\mathbf{a}^n), d^n) \quad (2)$$

1. Soient $\alpha_1 = 0, \alpha_2 = +\infty$;
   On se donne un pas $\alpha > 0$;
2. Répéter :
   2.1. si (1) n'est pas vérifiée pour $\alpha^n = \alpha$
   2.2. alors $\alpha_2 = \alpha$ et on choisit un nouveau pas :
   $\alpha \in](1-\tau_1)\alpha_1 + \tau_1\alpha_2, \tau_1\alpha_1 + (1-\tau_1)\alpha_2[$; (I)
   2.3. Sinon
   2.3.1. si (2) est vérifiée avec $\alpha^n = \alpha$
   2.3.2. alors on sort avec $\alpha^n = \alpha$

2.3.3. sinon

2.3.3.1. $\alpha_1 = \alpha$

2.3.3.2. si $\alpha^n = +\infty$

2.3.3.3. alors on choisit un nouveau pas :
$\alpha \in [\tau_2\alpha_2, +\infty[$;

2.3.3.4. sinon on choisit un nouveau pas α comme en 2.2.

Cette algorithme n'impose aucune condition sur le signe des composantes du nouvel itéré. La difficulté supplémentaire pour la recherche du pas de ce problème est qu'aucune composante du nouvel itéré ne peut être négative ou nulle. En effet, la conductivité thermique est un paramètre positif. Avant de vérifier chacune des conditions, il faut tester si tous les éléments du vecteur $\mathbf{a}^n + \alpha^n d^n$ sont positifs, si ce n'est pas le cas, on divise systématiquement le pas α^n par 2 jusqu'à que cette condition préalable soit vérifiée.

Dans notre algorithme, la recherche du pas est l'un des éléments les plus coûteux. En effet, à chaque itération k de la recherche linéaire, on doit calculer la fonction coût $J(\mathbf{a}^n + \alpha^k d^n)$ et le gradient de celle-ci. Pour éviter que le calcul ne soit trop long, nous ajoutons une condition sur le nombre maximum d'itération pour la recherche du pas. Ainsi, au bout des 15 itérations, on renvoie le pas même si celui-ci ne respecte pas les conditions de Wolfe. Avant de renvoyer ce pas α, on vérifie tout de même que les composantes de $\mathbf{a}^n + \alpha^n d^n$ sont positives.

4.2.4 Critère d'arrêt :

Comme critère d'arrêt, nous avons choisi : $||\nabla J(\mathbf{a}^n)|| \leq 10^{-4}$. Afin que l'algorithme ne tourne pas indéfiniment, nous avons également ajouté un compteur d'itération. Ainsi, l'algorithme s'arrête au delà de 30 itérations.

Complications pour notre exemple :

Comme nous l'avons dit précédemment, la conductivité est un paramètre strictement positif. Malgré les garde-fous que nous avons ajouté dans la recherche linéaire du pas. Il peut arriver que l'un des paramètres soit fort proche de 0. Dans ce cas là, l'algorithme peut ne pas converger.

Éléments permettant d'améliorer notre méthode inverse :

Le but de ce travail étant de comprendre le fonctionnement d'une méthode inverse, j'ai préféré prendre une méthode de résolution directe afin de me concentrer sur le l'algorithme BFGS en lui même. On pourrait cependant améliorer la méthode de résolution directe en prenant un schéma implicite qui est inconditionnellement stable. Ceci éviterai donc d'avoir une restriction sur le choix du pas de temps. En revanche, cela forcerait à résoudre un système linéaire à chaque itération. Pour résoudre ce système linéaire, on pourrait utiliser le stockage profil pour la matrice A et effectuer sa décomposition LDL^T au début de la résolution. On aurait ainsi à effectuer uniquement une remontée et une descente à chaque pas de temps. [6]

Chapitre 5

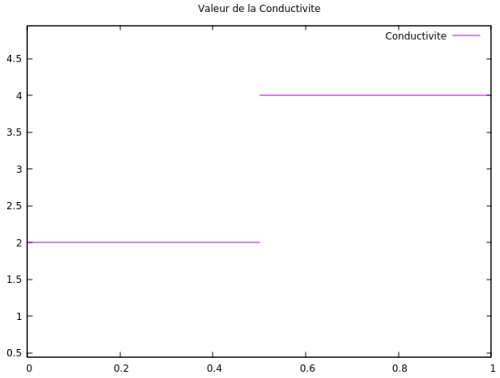
Exemples numériques et analyse des résultats

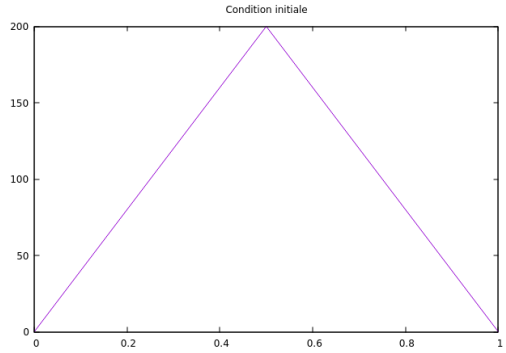
Afin de tester la méthode, nous allons fabriquer deux exemples complètement théoriques. Tout d'abord nous fixerons le paramètre à retrouver -ici la conductivité- pour fabriquer les données, puis nous effectuerons la méthode inverse pour tenter de retrouver une partie de ces données.

5.1 Exemple 1 :

5.1.1 Extraction des données

Pour notre premier exemple, nous considérons que la conductivité est constante par morceau. Nous prendrons une condition initiale comme ci-dessous à droite.





Pour avoir nos données, on fixe donc notre conductivité comme ci-dessus puis on résout le problème direct à l'aide du schéma explicite détaillé au paragraphe 4.1 . Enfin on extrait les données à certains instants et en certains points. On a donc pour données d_{obs} :

Temps \ Espace	0	0.2	0.4	0.6	0.8	1
0.1	0	6.35577	9.23807	7.91497	4.60187	0
0.3	0	0.0205189	0.029821	0.0255475	0.014853	0
0.5	0	6.62346e-05	9.62613e-05	8.24666e-05	4.7945e-05	0
0.7	0	2.1388e-07	3.10841e-07	2.66296e-07	1.54821e-07	0
0.9	0	6.90153e-10	1.00303e-09	8.59288e-10	4.99578e-10	0

5.1.2 Méthode inverse

Pour notre problème inverse, on cherchera à retrouver la conductivité en quatre points : 0.2, 0.4, 0.6, et 0.8. Puis nous considérerons que celle-ci est constante par morceau :

$$\mathbf{a}(x) = \begin{cases} \mathbf{a}(0.2) & x \in [0, 0.3] \\ \mathbf{a}(0.4) & x \in]0.3, 0.5] \\ \mathbf{a}(0.6) & x \in]0.5, 0.7] \\ \mathbf{a}(0.8) & x \in]0.7, 1] \end{cases}$$

Après l'implémentation de l'algorithme BFGS, et ses difficultés notamment au niveau du pas, les résultats n'étaient pas concluants car l'algorithme ne convergeait pas forcément vers un bon résultat. Cela laissait donc entendre que le problème était mal posé. Pour régler ce problème, il a donc fallu mettre un place une régularisation de Tikhonov.

Comme nous l'avons vu dans la partie 3, cela consiste à modifier légèrement le problème en passant de la fonction coût $J(\mathbf{a})$ à $\widetilde{J}(\mathbf{a}) = \frac{1}{2}||\mathbf{Ma} - d_{obs}||^2 + \frac{\epsilon^2}{2}||\mathbf{a} - \widetilde{\mathbf{a}}||^2$ où $\widetilde{\mathbf{a}}$ est l'estimation a priori et ϵ est le coefficient de régularisation.

Choix de l'estimation a priori :

Le choix de l'estimation a priori peut s'avérer déterminant pour la résolution du problème inverse. Dans cet exemple théorique, nous prendrons un estimation assez proche du résultat. Dans un cas concret, on peut imaginer que l'estimation soit basée sur des mesures connues.

Pour la première résolution, nous avons donc choisi :

Estimation a priori				
x	0.2	0.4	0.6	0.8
ã(x)	1.5	1.78	4.2	3.82

Choix du paramètre de départ :

Pour démarrer l'algorithme, nous prendrons un paramètre de départ proche de l'estimation a priori pour deux raisons :

— Tout d'abord, on s'attend à ce que la solution soit proche de l'estimation a priori, il semble donc plus judicieux de démarrer proche de celle-ci.

— Ensuite, le principal défaut de notre schéma explicite est que le pas de temps augmente linéairement avec maximum de la conductivité λ. Prendre une conductivité élevée augmenterait donc sensiblement les temps de calculs lors de la première itération.

Nous prendrons donc le paramètre suivant :

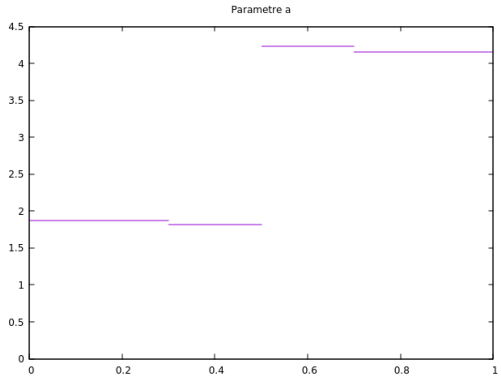
x	0.2	0.4	0.6	0.8
a(x)	1	5	3	5

Premiers résultats :

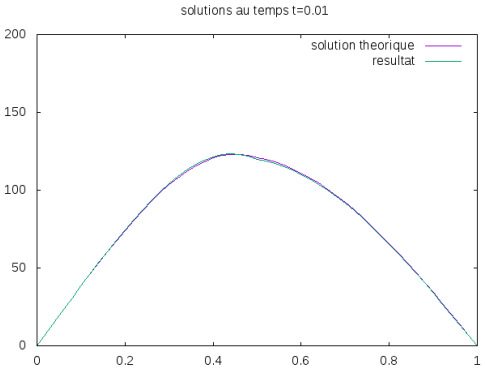
Pour un premier essai, nous prendrons le coefficient de régularisation $epsilon = 2$.
Nous obtenons les paramètres suivants :

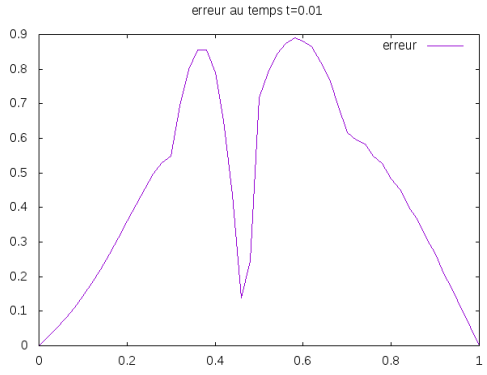
Résultat pour ε=2.0				
x	0.2	0.4	0.6	0.8
a(x)	1.86966	1.81355	4.23414	4.15905

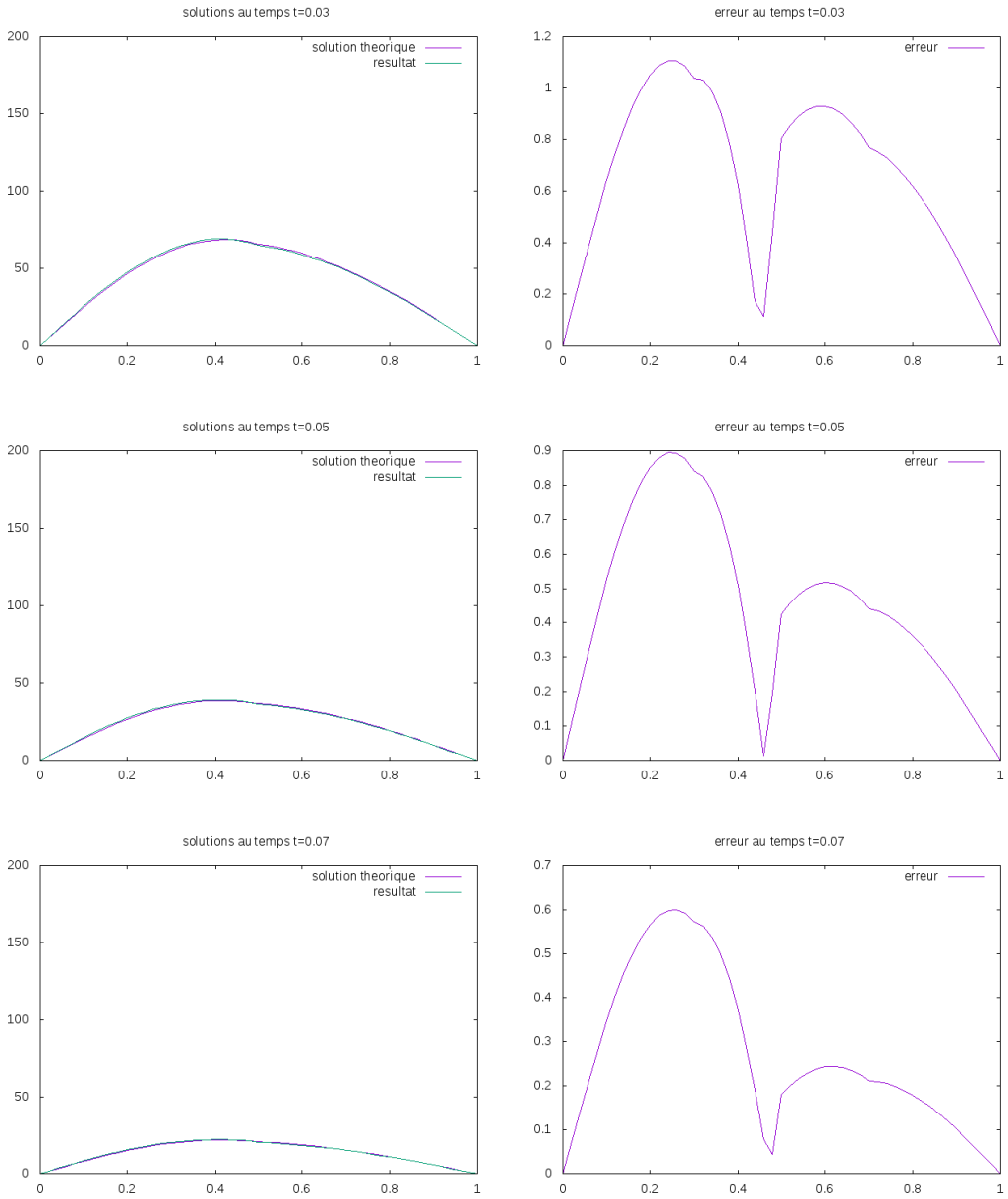
Nous allons donc utiliser la conductivité ci-dessous pour comparer notre résultat avec la solution réelle.



Voici le comparatif du comportement des solutions avec la solution avec la conductivité de départ et la solution avec la méthode inverse. Les graphiques de droite représentent l'erreur à l'instant t.





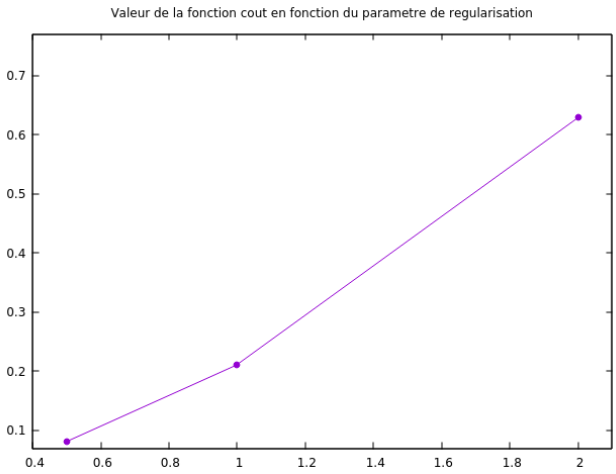


On peut voir que la solution trouvée grâce à la méthode inverse est proche de la solution réelle. En effet, on a une erreur d'un degré au maximum, ce qui est satisfaisant.

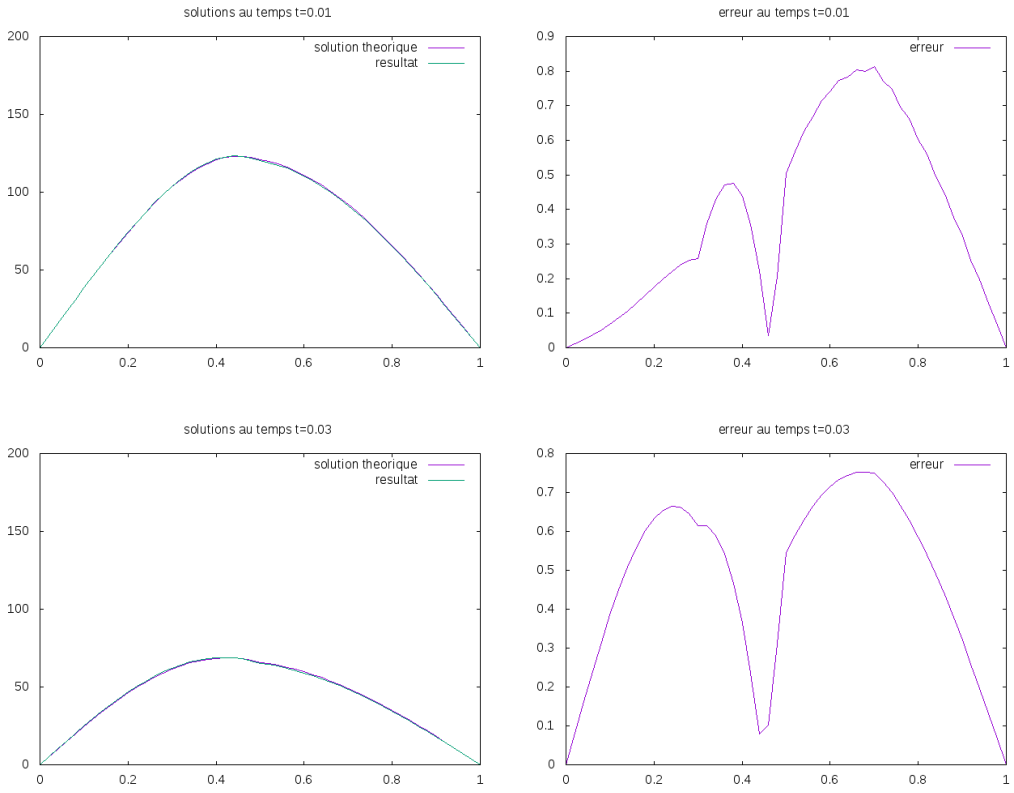
Choix du coefficient de régularisation

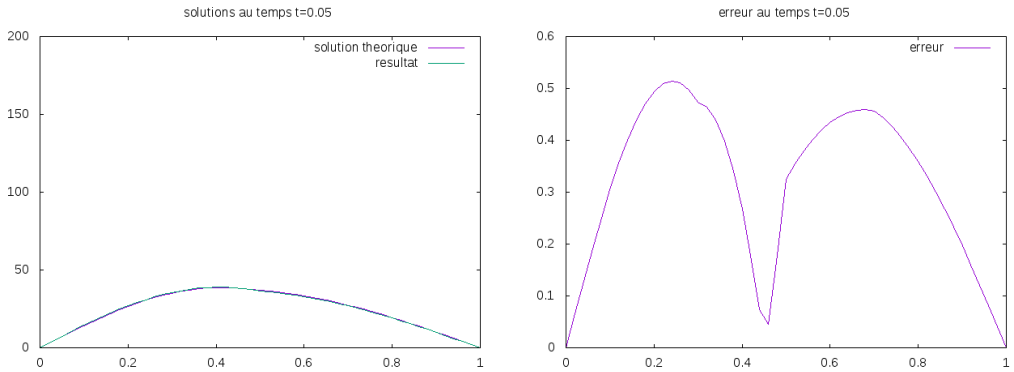
Comme nous l'avons vu dans la partie sur la régularisation de Tikhonov, l'erreur dépend du paramètre de régularisation. Il faut réussir à trouver un ϵ qui régularise le problème sans être trop élevé. Pour cela, nous allons refaire la méthode inverse pour les mêmes estimations a priori et pour le même paramètre de départ mais en baissant progressivement le coefficient de régularisation ϵ.

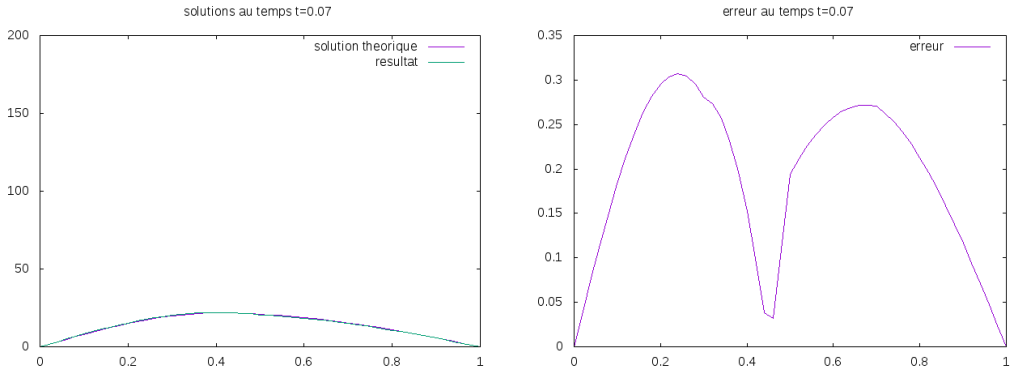
Après des premiers essais, on peut voir que le problème est toujours régularisé avec $\epsilon = 0.5$, et que la valeur de la fonction coût diminue :



Nous allons donc refaire la méthode inverse avec le coefficient de régularisation $\epsilon = 0.5$. Voici une comparaison entre la solution théorique et la solution obtenue par méthode inverse avec $\epsilon = 0.5$r. Les graphiques de droite représentent l'erreur à l'instant t.







On peut voir que l'erreur avec $\epsilon = 0.5$ est moins élevée qu'avec $\epsilon = 2$, $\epsilon = 0.5$ est donc un meilleur coefficient de régularisation.

Essai avec un autre paramètre au début de l'algorithme

Afin de voir si le paramètre de départ joue sur la convergence de l'algorithme, nous fixons à 7 les 4 composantes du vecteur paramètre au début de l'algorithme. En gardant le coefficient de régularisation à 0.5, nous n'obtenons pas de convergence de l'algorithme.

En revanche, en prenant $\epsilon = 2.5$, nous retombons sur un résultat tout à fait acceptable :

x	0.2	0.4	0.6	0.8
a(x)	1.87646	1.81126	4.23714	4.12533

Nous proposons donc comme méthode générale de commencer par un coefficient de régularisation très élevé pour forcer la régularisation du problème, puis de descendre progressivement ce coefficient.

Essai avec une autre estimation a priori

Maintenant que nous avons vu l'impact du coefficient de régularisation et du paramètre de départ sur la convergence de la solution, nous allons nous intéresser à l'estimation a priori. Pour cela, nous reprenons le paramètre de départ $(1, 5, 3, 5)$, nous fixons $\epsilon = 1$ et $\widetilde{\mathbf{a}} = (2.5, 3, 3, 3.5)$. En gardant les données de départ nous obtenons :

x	0.2	0.4	0.6	0.8
a(x)	2.1282	3.04865	2.92021	3.97713

Ces résultats semblent donc plus proche de l'estimation que du résultat théorique. On peut se demander si le coefficient de régularisation n'est pas trop élevé. Malheureusement, le résultat obtenu avec $\epsilon = 0.5$ est :

x	0.2	0.4	0.6	0.8
a(x)	2.00423	3.07062	2.9851	4.20381

On peut donc penser que l'estimation a priori semble donc trop mauvaise. On peut également supposer que nos données ne sont pas assez précises et ne permettent donc pas d'obtenir un résultat plus satisfaisant avec cette estimation.

5.2 Exemple 2

Dans l'exemple précédent, nous avions choisi une fonction constante par morceau. Or, l'une des conditions pour la convergence du schéma était que la fonction de conductivité K soit $C^3(]0,1[)$ ce qui n'était pas le cas. Nous allons donc faire un exemple avec un polynôme de degré 2 : $K(x) = 1 - 4x(x-1)$. Cette fonction est $C^\infty(]0,1[)$.

Pour cet exemple, nous allons chercher la conductivité en trois points : $0.1, 0.5$ et 0.9. Puis nous utiliserons l'interpolation de Lagrange pour retrouver $K(x)$.

Comme précédemment, nous résolvons le problème direct pour extraire les données. Nous gardons également la même condition initiale.

Nous choisissons comme paramètre de départ :

FIGURE 5.1 – Paramètres de départ

x	0.1	0.5	0.9
a(x)	1	1	1

Nous prenons comme estimation a priori :

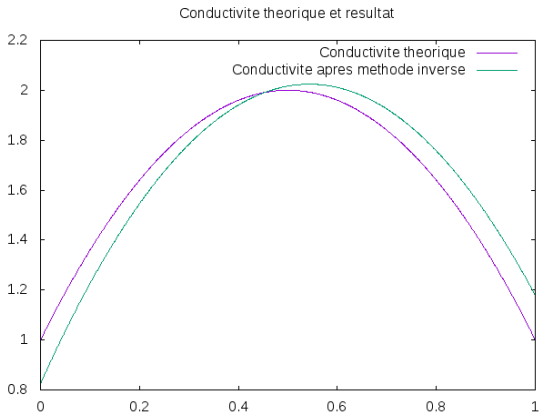
FIGURE 5.2 – Estimation a priori

x	0.1	0.5	0.9
a(x)	1.1	2.2	0.9

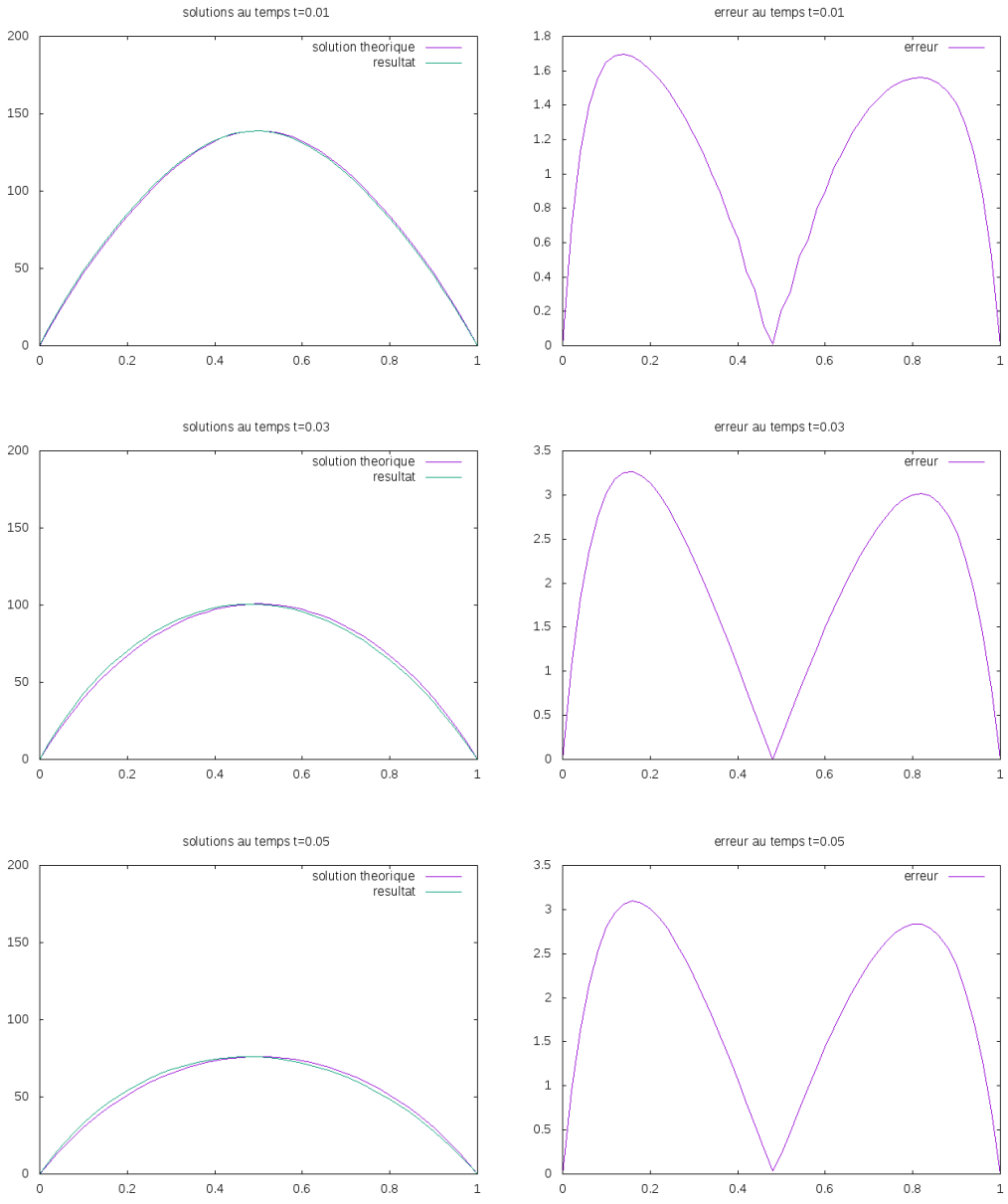
En prenant $\epsilon = 0.5$, la méthode inverse converge vers :

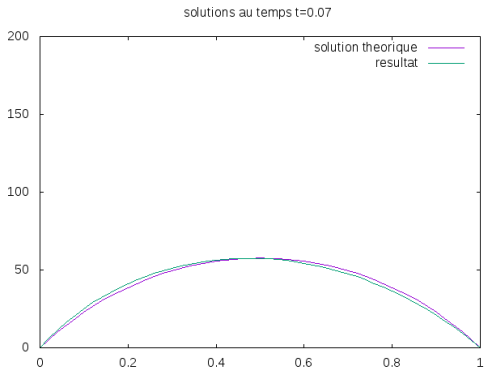
x	0.1	0.5	0.9
a(x)	1.22633	2.01682	1.50824

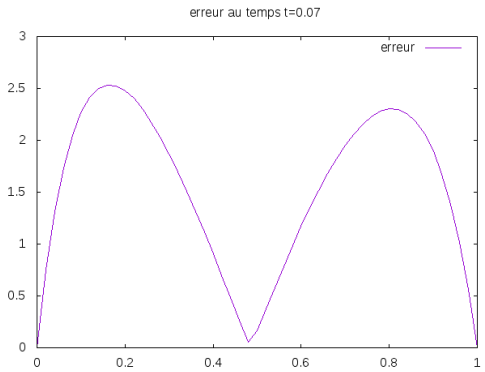
Après avoir retrouvé les trois paramètres, nous pouvons utiliser l'interpolation de Lagrange pour retrouvé la fonction conductivité. Nous traçons maintenant la conductivité théorique et notre résultat :



On peut voir que le résultat est assez proche. Nous pouvons maintenant comparer les équations d'état avec la conductivité théorique et la conductivité obtenue avec la méthode inverse :





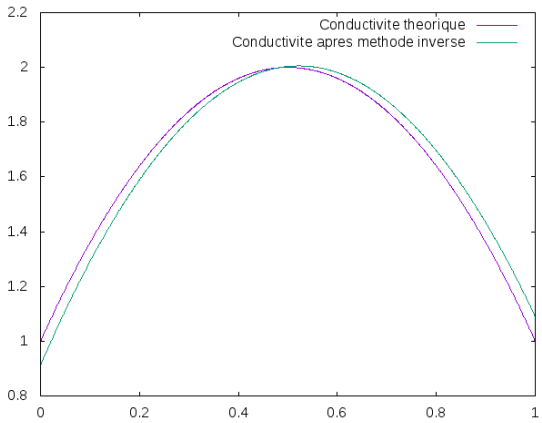


Pour cet exemple, nous avons continué d'appliquer la méthode de l'exemple 1 en baissant progressivement le coefficient de régularisation. Pour ce problème, également convergence pour $\epsilon = 0$, c'est à dire sans régularisation de Thikonov. Le problème semble donc bien posé au voisinage de la solution. En effet, la méthode inverse converge encore vers le même résultat que celui obtenu précédement. Nous obtenons une erreur de l'ordre de 3 degré. Ceci est plus élevé que pour le premier exemple mais cela reste plutôt raisonnable. Les erreurs peuvent provenir des données, qui ne sont pas exactes, ou des calculs dans la méthode inverse.

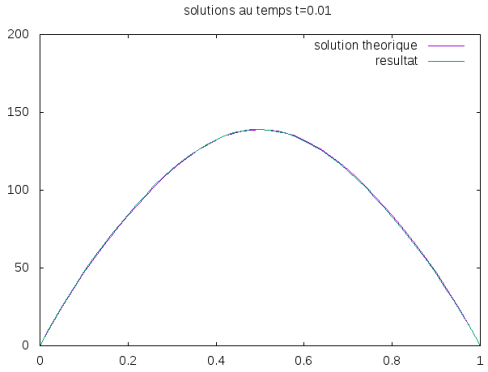
Afin de vérifier si les erreurs provenaient de la méthode inverse, nous l'avons refaite avec un pas d'espace divisé par 2. Les résultats obtenus sont bien meilleurs, en effet, nous obtenons :

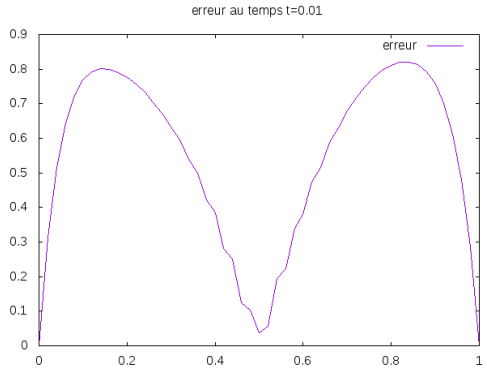
x	0.1	0.5	0.9
a(x)	1.29096	2.0036	1.43277

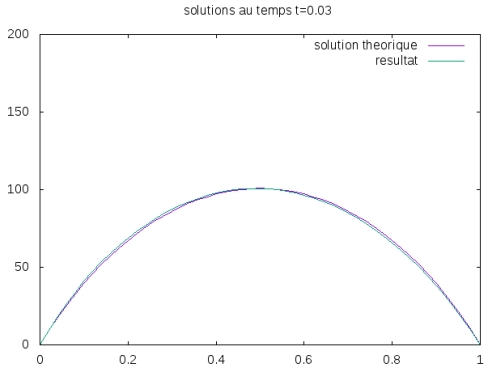
Ce qui nous donne après interpolation :

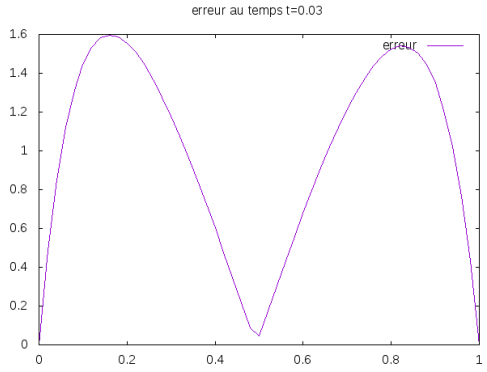


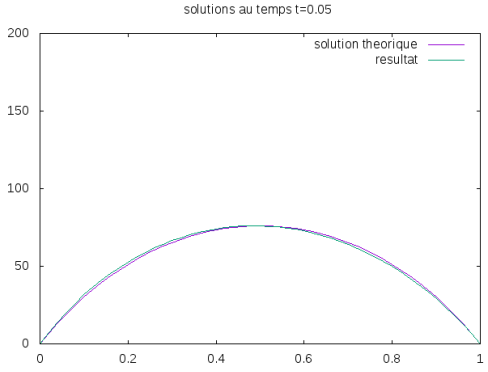
Nos résultats sont donc bien plus précis. En effet, nous avons maintenant une erreur maximale de 1.6 degré, comme le montrent les graphiques suivants :

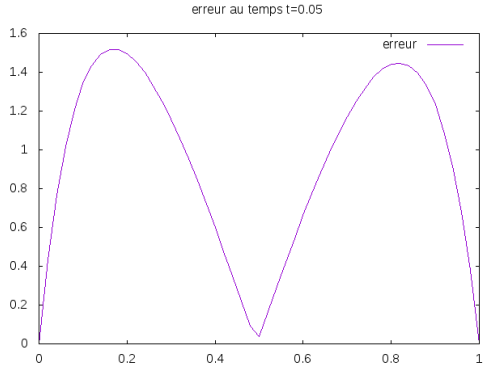

solutions au temps t=0.01
solution theorique
resultat
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1

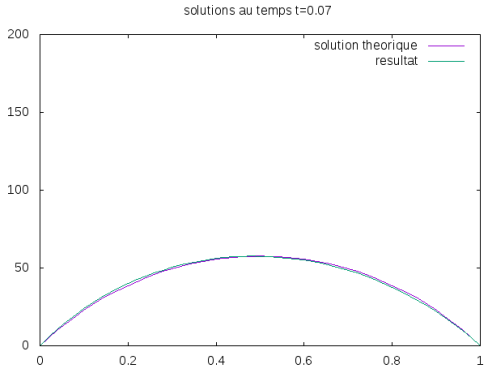

erreur au temps t=0.01
erreur
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
0.2
0.4
0.6
0.8
1

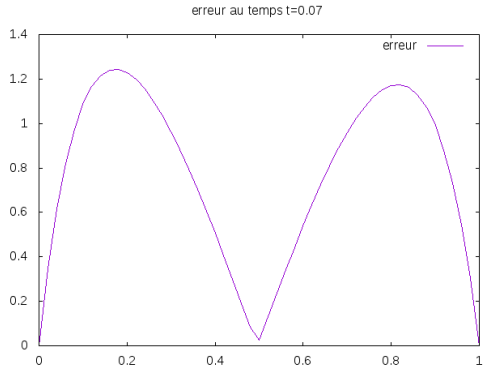

solutions au temps t=0.03
solution theorique
resultat
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1


erreur au temps t=0.03
erreur
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1


solutions au temps t=0.05
solution theorique
resultat
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1


erreur au temps t=0.05
erreur
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1


solutions au temps t=0.07
solution theorique
resultat
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1


erreur au temps t=0.07
erreur
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1

Conclusion

Les problèmes inverses sont compliqués à résoudre car ce sont des problèmes généralement mal posés. La régularisation de Tikhonov permet de régulariser le problème mais cela entraîne une dépendance des résultats vis-à-vis de l'estimation a priori et du coefficient de régularisation. Le choix de l'estimation a priori nécessite une très bonne connaissance du problème direct, ce qui n'est pas forcément le cas pour un problème concret.

La résolution de problèmes inverses est très coûteuse en opérations. En effet, elle requiert la résolution du problème direct à chaque itération de l'algorithme BFGS. La résolution du problème direct doit donc être optimisée pour réduire au maximum le coût de l'algorithme. De plus, celle-ci doit être suffisamment précise. En effet, comme nous venons de le voir lors du dernier exemple, affiner un maillage nous permet d'obtenir un résultat bien plus précis.

L'algorithme de Broyden-Fletcher-Goldfarb-Shanno est un algorithme itératif appartenant aux méthodes de Quasi-Newton. Cette algorithme nécessite le calcul du gradient de la fonction à minimiser. Pour cela nous avons vu deux méthodes différentes :

1. la méthode des différences finies
2. la méthode de l'état adjoint, basée sur la théorie des éléments finis

Dans des cas concrets, la résolution de problème inverse peut ne pas aboutir par manque de données ou à cause de données trop bruitées.

Piste future :

Pour ce travail, nous avons choisi d'utiliser la méthode des différences finies. Nous aurions pu également utiliser les éléments finis. En effet on pourrait ainsi utiliser FreeFem++ pour résoudre le problème direct et calculer le gradient par la méthode de l'état adjoint.

Les éléments finis permettent de ne pas avoir un pas de temps uniforme, ce qui est utile pour les problèmes thermiques. En effet, la solution de l'équation de la chaleur est de plus en plus régulière avec le temps, on pourrait donc imaginer un petit pas de temps au début de la résolution et un plus grand pas de temps vers la fin.

Bibliographie

[1] Jacques Hadamard. Sur les problèmes aux dérivées partielles et leur signification physique. *Princeton University Bulletin*, pages 49–52, 1902.

[2] Ana Matos. *Cours d'optimisation convexe*.

[3] Michel Kern. Problèmes inverses : aspects numériques. *HAL archives : cel-00168393*, 2002.

[4] J. Greenstadt. Variations on variable metric methods. *Math. Comp.*, pages 1–22, 1970.

[5] C. Lemaréchal. A view of line searches. *Optimization and optimal control*, pages 59–78, 1981.

[6] Bernhard Beckermann. *Cours de traitement informatique de l'analyse numérique*.

Annexes

Fichier main

```
#include <iostream>
#include <fstream>
#include <assert.h>
#include <cmath>
#include <cstdlib>
#include "edp.h"


using namespace std;

int main(){
     int N;
     double L=1;
     double T=1;
     N=50;
    Vecteur A(N+1);

    //Condition initiale :
    for(int j=0; j<N+1; j++){
     if(j*L/N< 0.5){
     A(j)=j*L/N;
     }
     else{
     A(j)=1.0-j*L/N;
     }
    }
    A=400*A;

    //parametre de depart :
    Matrice cond(2,4);
    cond[0](0)=0.2;
    cond[0](1)=0.4;
    cond[0](2)=0.6;
    cond[0](3)=0.8;
    cond[1](0)=1.0;
    cond[1](1)=5.0;
    cond[1](2)=3.0;
    cond[1](3)=5.0;
```

```
    // Estimation a priori :
    Matrice estimation(cond);
    estimation[1](0)=1.5;
    estimation[1](1)=1.5;
    estimation[1](2)=3;
    estimation[1](3)=3.5;

    // Coefficient de regularisation
    double epsilon = 0.5;


    Matrice donnees(read_matr("donnees2.txt"));
    ofstream file1("fonction_cout.txt");
    ofstream file2("param_a_epsilon.txt");

        cout << "Les donnees sont : " << endl << donnees;

        Matrice cond2(2,4);
     cout << endl << "Epsilon = " << epsilon << endl;

     //Methode inverse :
        cond2=BFGS(cond,N,T,L,A, donnees, epsilon, estimation);
     cout << "-------------------------------------------------------------------------" << endl;
     cout << "-------------------------------------------------------------------------" << endl;
     cout << "-------------------------------------------------------------------------" << endl;
        cout << "Le \"a\" qui minimise la fonction cout est : " << endl << cond2 << endl;

        file1 << epsilon << " " << fonction_cout(cond2, N, T, L, A, donnees, epsilon, estimation) << end

     file2 << epsilon << " " << cond2(1,0) << " " << cond2(1,1) << " " << cond2(1,2)
     << " " << cond2(1,3) << endl;

    //system("gnuplot plot \"fonction_cout.txt\" with linespoints pointtype 7 pointsize 0.5");
     Matrice M=(equation_etat(cond2, N, T, L, A));
     save_matr("matrice_sol.txt",M);
     save_matr("matrice_cond.txt",cond2);

return 0;
}
```

Module avec algorithme BFGS (fichier "edp.h")

```
#include <iostream>
#include <fstream>
#include <assert.h>
#include <cmath>
#include <cstdlib>
#ifndef MATRICE_H
#include "matrice.h"
#endif

using namespace std;
```

```
  ////////////////////////////////////////
 // Conductivite constante par morceaux //
////////////////////////////////////////

double K(const Matrice& cond, const double& x, const double& L=1){

    double sortie=1000;
    int i=1;

        if(0<=x && x<((cond[0](0)+cond[0](1))/2.0)){
            sortie = cond[1](0);
        }
        else {
            while((((cond[0](i)+cond[0](i-1))/2.0)>x || x>=((cond[0](i)+cond[0](i+1))/2.0)) && i<(cond.d
                i=i+1;
            }
            if(((cond[0](i)+cond[0](i-1))/2.0)<=x && x<((cond[0](i)+cond[0](i+1))/2.0)){
                sortie= cond[1](i);
            } else {
                sortie = cond[1](cond.dim2()-1);
            }
        }
    return(sortie);
}


//-----------------------------------------------------------------------------------------------------


  ////////////////////////////////
 // Regularisation de Tikhonov //
////////////////////////////////

double tikhonov(const double& epsilon, const Matrice& parametre, const Matrice& estimation){
return(pow(epsilon,2)/2.0 *pow(((parametre[1]-estimation[1]).norme_2()),2));
}




//-----------------------------------------------------------------------------------------------------
  ////////////////////////////////////////
 // Resolution de l equation d etat  //
////////////////////////////////////////

Matrice equation_etat(const Matrice& cond ,const int& N,const double& T,const double& L,const Vecteur& U
     double delta_X= L/N;

     double k_max=0;
     for(int i=0; i<=2*N; i++){
     if(k_max<K(cond,i*delta_X/(2.0)) ){
```

```
k_max=K(cond,i*delta_X/(2.0));
}
}
if(k_max <4){
k_max=4;
}

double delta_T=pow(delta_X,2)/(k_max*2.);

Matrice M(floor(T/delta_T)+1,N+1);
ofstream file1("solution.txt");
double ymin=1e10;
   double ymax=-1e10;




M[0]=U_init;
for(int i=0; i <=N; i++){  // Sauvegarde de la condition initiale
file1 << i*delta_X << " " << M[0](i) << endl;
}
file1 << endl << endl;

for(int n=1; n<=floor(T/delta_T);n++){ // Calcul la solution a l'instant (n)*delta_T

file1 << 0 << " " << M[n](0) << endl;
M[n](1)=M[n-1](1) - delta_T /(pow(delta_X,2)) * ((K(cond,(1+1/2.)*delta_X)+K(cond,(1-1/2.)*delta_X)
file1 << delta_X << " " << M[n](1) << endl;

for(int i=2; i<(N-1); i++){
M[n](i)= M[n-1](i) - delta_T /(pow(delta_X,2)) * ((K(cond,(i+1/2.0)*delta_X)+K(cond,(i-1/2.0)*delta
file1 << i*delta_X << " " << M[n](i) << endl;
}

M[n](N-1)=M[n-1](N-1) - delta_T /(pow(delta_X,2)) * ((K(cond,(N-1+1/2.)*delta_X)+K(cond,(N-1-1/2.)*
file1 << (N-1)*delta_X << " " << M[n](N-1) << endl;
file1 << N*delta_X << " " << M[n](N) << endl;
file1 << endl << endl;

}

file1.close();
// trace de l'evolution de la temperature
ofstream file2("film.gnuplot");
   file2 << "set terminal x11" << endl;
   file2 << "set xrange[" << 0 << ":" << L << "]" << endl;
   file2 << "set yrange[" << 0 << ":" << 100 << "]" << endl;
   file2 << "imax=" << floor(T/delta_T) << endl;
   file2 << "i=0" << endl;
   file2 << "load \"script2.gnu\" ";
   file2.close();
```

```
        ofstream file3("script2.gnu");
        file3 << "plot \"solution.txt\" index i " << "with linespoints pointtype 7
        pointsize 0.5" << endl;
        file3 << "pause" << " " << delta_T <<  endl; //delta_T
        file3 << "i=i+1"<< endl;
        file3 << "if (i<imax) reread" << endl;

        //system("gnuplot film.gnuplot");
     system("rm solution.txt");

     return M;
};

//-----------------------------------------------------------------------------------------------------------
  ////////////////////////////////////////////////////
 // Resolution de l equation d etat, sortie formatee //
////////////////////////////////////////////////////

    Matrice equation_etat2(const Matrice& cond ,const int& N,const double& T,const double& L,const Vecte
     double delta_X= L/N;


     double k_max=0;
     for(int i=0; i<=2*N; i++){
         if(k_max<K(cond,i*delta_X/(2.0))){
             k_max=K(cond,i*delta_X/(2.0));
         }
     }
     if(k_max <4){
         k_max=4;
     }

     double delta_T=pow(delta_X,2)/(k_max*2.);

     Matrice M(floor(T/delta_T)+1,N+1);
     Matrice F(dobs.dim1(),dobs.dim2());

     M[0]=U_init;
     F[0]=dobs[0];


     for(int n=1; n<=floor(T/delta_T);n++){ // Calcul la solution a l'instant (n+1)*delta_T

         M[n](1)=M[n-1](1) - delta_T /(pow(delta_X,2)) * ((K(cond,(1+1/2.)*delta_X)
             +K(cond,(1-1/2.)*delta_X))*M[n-1](1) -K(cond,(1+1/2.)*delta_X)*M[n-1](2));

         for(int i=2; i<(N-1); i++){
             M[n](i)= M[n-1](i) - delta_T /(pow(delta_X,2)) *
             ((K(cond,(i+1/2.0)*delta_X)+K(cond,(i-1/2.0)*delta_X))*M[n-1](i)
                 -K(cond,(i+1/2.0)*delta_X)*M[n-1](i+1)-K(cond,(i-1/2.0)*delta_X)*M[n-1](i-1));
         }
```

```
         M[n](N-1)=M[n-1](N-1) - delta_T /(pow(delta_X,2)) *
         ((K(cond,(N-1+1/2.)*delta_X)+K(cond,(N-1-1/2.)*delta_X))*M[n-1](N-1)
                 -K(cond,(N-1-1/2.)*delta_X)*M[n-1](N-2));
     }
     for(int i=1; i< dobs.dim2()-1; i++){
         F(i,0)=dobs(i,0);
     }


int b=0; //pointeur colonne
for(int n=0; n<floor(T/delta_T)-1;n++){
    for(int a=1; a<F.dim1(); a++){
        if((n*delta_T<=F(a,0))&&(F(a,0)<=(n+1)*delta_T)){
            b=1;
            for(int i=0; i<N-2; i++){
            if((i*delta_X<=F(0,b))&&(F(0,b)<=(i+1)*delta_X)){
                F(a,b)=(M(n,i));//+M(n,i+1)+M(n+1,i)+M(n+1,i+1))/4.0;
                    b+=1;
                }
            }
        }
    }
}


//cout << F << endl;
return(F);

};

//-------------------------------------------------------------------------------------------------------
  ////////////////////////////
 // Calcul fonction cout //
////////////////////////////

double fonction_cout(const Matrice& cond, const int& N,const double& T,const double& L,
                     const Vecteur& U_init, const Matrice& dobs, const double& epsilon,
                     const Matrice& estimation){
    double sortie=0;
    Matrice etat(dobs.dim1(),dobs.dim2());
    etat=equation_etat2(cond,N,T,L,U_init,dobs);
    etat=etat-dobs;
    sortie= etat.norme_m_2();
    sortie=pow(sortie,2)/2.0+tikhonov(epsilon,cond,estimation);
    //cout << "fonction cout : " << sortie << endl;

return(sortie);
}
//-------------------------------------------------------------------------------------------------------
```

```
  ////////////////////////////////////////
 // Calcul du gradient de la fonction cout //
////////////////////////////////////////

Vecteur gradient(Matrice& cond, const int& N,const double& T,const double& L,const Vecteur& U_init, cons
                const double& h,const double& epsilon, const Matrice& estimation){
    Vecteur sortie(cond.dim2());
    double fonc_cout=fonction_cout(cond,N,T,L,U_init,dobs,epsilon, estimation);
    for(int i=0; i<sortie.dim(); i++){
        cond(1,i)=cond(1,i)+h;
        sortie(i)=(fonction_cout(cond,N,T,L,U_init,dobs,epsilon, estimation)-fonc_cout)/(1.0*h);
        cond(1,i)=cond(1,i)-h;
    }

    return(sortie);
}

//-----------------------------------------------------------------------------------------------------

  ////////////////////////////////////////////////////////////////////////////
 // Calcul de la matrice Hessienne de la fonction cout pour la premiere iteration //
////////////////////////////////////////////////////////////////////////////

Matrice hessienne(Matrice& cond, const int& N,const double& T,const double& L,const Vecteur& U_init, con
                  const double& h,const double& epsilon, const Matrice& estimation){
    Matrice sortie(cond.dim2(),cond.dim2());
    Vecteur grad(cond.dim2());
    grad=gradient(cond,N,T,L,U_init,dobs,h,epsilon,estimation);
    for(int i=0; i<sortie.dim2();i++){
        cond(1,i)=cond(1,i)+h;
        sortie[i]=(gradient(cond,N,T,L,U_init,dobs,h,epsilon,estimation)-grad)/(1.0*h);
        cond(1,i)=cond(1,i)-h;
    }

    return(sortie);
}
//-----------------------------------------------------------------------------------------------------

  ///////////////////////////////////
 // Comparaison parametre avec 0//
///////////////////////////////////

bool comparaison(const Matrice& cond, const double& pas, const Vecteur& dir){
    bool sortie=false;
    int i=-1;
    while(sortie== false && i < cond.dim2()-1){
        i++;
        if((cond[1]+pas*dir)(i)<0) sortie=true;
    }
    return(sortie);
}
```

```
//-----------------------------------------------------------------------------------------------------------

  //////////////
 // Pas test//
//////////////

double pas_test(const Matrice& cond, const Vecteur& dir){
    double sortie=1.0;
    while(comparaison(cond,sortie,dir)==true){
        sortie=sortie/2.0;
    }
    return sortie;
}
//-----------------------------------------------------------------------------------------------------------

  ////////////////////////////////////////////////////
 // Recherche d'un pas respectant les criteres de Wolfe //
////////////////////////////////////////////////////

double fletcher_marechal(const Matrice& cond, const int& N,const double& T,
                         const double& L,const Vecteur& U_init, const Matrice& dobs,
                         const Vecteur& dir, const Vecteur& grad, const double& h,
                         const double& epsilon,const Matrice& estimation){

    double pas=1.0;
    double alpha1 = 0.001;
    double alpha2= pow(10,10);
    double Ti = 0.25;
    double w1 = pow(10,-4); // pow(10,-4)
    double w2 = 0.99; // 0.99
    double Te = 4.;
    Matrice cond2(cond);
    while(comparaison(cond,pas,dir)==true){
        pas=pas/2.0;
    }
    cond2[1]=cond2[1]+pas*dir;
    double f_c = fonction_cout(cond,N,T,L,U_init,dobs,epsilon,estimation);
    double condition1=fonction_cout(cond2,N,T,L,U_init,dobs,epsilon,estimation);
    double condition2=gradient(cond2,N,T,L,U_init,dobs,h,epsilon,estimation)*dir;
    int nbit =0;


    while(((condition1>f_c+w1*pas*(grad*dir))|| (condition2<w2*(grad*dir))||(comparaison(cond,pas,dir)==
        nbit++;
        //cout << pas << endl;
        if((condition1>f_c+w1*pas*(grad*dir))||(comparaison(cond,pas,dir)==true)){
            if(comparaison(cond,pas,dir)==true) cout << "inf 0" << endl;
            if(condition1>f_c+w1*pas*(grad*dir)) cout << "condition 1" << endl;
            alpha2=pas;
            pas=(alpha1+alpha2)/2.0;
     }
```

```
     else{
            cout << "condition 2 " << endl;
            alpha1=pas;
            if(alpha2=pow(10,10)){
                cout << " cond 2.1 " << endl;
                pas=(Te+1)*alpha2;
            }
            else {
                cout << " cond 2.2 " << endl;
                pas=(alpha1+alpha2)/2.0;
            }
     }
        while(comparaison(cond,pas,dir)==true){
            pas=pas/2.0;
        }
        cond2[1]=cond[1]+pas*dir;
        condition1=fonction_cout(cond2,N,T,L,U_init,dobs,epsilon,estimation);
        condition2=gradient(cond2,N,T,L,U_init,dobs,h,epsilon,estimation)*dir;
        //cout << pas << endl;
    }

    return(pas);
}


//-------------------------------------------------------------------------------------------------------
  ////////////////////////////////////////////
 // Resolution du probleme inverse par BFGS //
////////////////////////////////////////////

Matrice BFGS( const Matrice& cond,const int& N,const double& T,const double& L,
                const Vecteur& U_init, const Matrice& dobs,
                const double& epsilon, const Matrice& estimation){
    Matrice sortie(cond); // Ce qu'on retourne a la fin
    Vecteur dir(cond.dim2()); // La direction pour passer de a^n -> a^(n+1)
    Matrice preced(cond.dim1(),cond.dim2()); // Permet de stocker a^(n-1)
    double pas=0.01; // Le pas pour passer de a^n -> a^(n+1)
    Vecteur s(cond.dim2()); // Vecteur permettant de calculer recursivement la matrice Hessienne
    Vecteur y(cond.dim2()); // Vecteur permettant de calculer recursivement la matrice Hessienne
    Vecteur grad(cond.dim2()); // Gradient de la fonction
    Vecteur grad_preced(cond.dim2()); // Gradient de l'iteration precedente
    Matrice hess(cond.dim2(),cond.dim2()); // Matrice Hessienne de la fonction
    Vecteur pivot(cond.dim2()); // Vecteur pivot pour la décomposition LU qui permet
                                    de calculer la direction
    Matrice dec_lu_hess(cond.dim2(),cond.dim2());
    double h=0.1;
    int nb_iter=0;

    cout << "-------------------------------------------------------------------------" << endl;
    cout << "Parametre de depart : "<< sortie[1] << endl;
    cout << "-------------------------------------------------------------------------" << endl;
```

```
grad=gradient(sortie,N,T,L,U_init,dobs,h,epsilon,estimation);
cout << " gradient : " << grad;
hess=hessienne(sortie, N, T, L, U_init, dobs, h,epsilon,estimation);
cout << " hessienne : " << hess;
dec_lu_hess=hess.lu(pivot);
dir=-dec_lu_hess.solvelu(grad,pivot);
//dir=hess*grad;
cout << "direction : " << dir << endl;
pas=fletcher_marechal(sortie, N, T, L, U_init, dobs, dir, grad, h,epsilon,estimation);
cout << " pas : " << pas << endl;
preced=sortie;
sortie[1]=preced[1]+pas*dir;
cout << "------------------------------------------------------------------------------" << endl;
cout << "Nouveau parametre : " << sortie[1] << endl;
cout << "------------------------------------------------------------------------------" << endl;

while(((grad).norme_2() > 0.0001)&&(nb_iter<30)){


    cout << "------------------------------------------------------------------------------" << endl;

    // Calcul du gradient de la Fonction
    grad_preced=grad;
    grad=gradient(sortie,N,T,L,U_init,dobs,h,epsilon,estimation);

    cout << " gradient : " << grad;

    // Calcul de l'approximation de la matrice Hessienne

    s=sortie[1]-preced[1];
    y=grad-grad_preced;

    hess=hess+produit(y,y)/(y*s)-hess*produit(s,s)*hess/(s*(hess*s));

    cout << " hessienne: " << endl << hess;

    // Calcul de la direction :

    dec_lu_hess=hess.lu(pivot);
    dir=-dec_lu_hess.solvelu(grad,pivot);
    cout << "direction : " << dir << endl;

    // Recherche du pas vérifiant la condition de Wolfe :

    pas=fletcher_marechal(sortie, N, T, L, U_init, dobs, dir, grad, h,epsilon,estimation);

    cout << " pas : " << pas << endl;

    // Calcul du nouvel itéré

    preced=sortie;
    sortie[1]=preced[1]+pas*dir;
```

```
        cout << "-------------------------------------------------------------------------" << endl;
        cout << "-------------------------------------------------------------------------" << endl;
        cout << "Nouveau parametre : " << sortie[1] << endl;
    }
    if(nb_iter==30) cout << "ATTENTION, MAX D'ITERATION ATTEINT !" << endl;


    return(sortie);
}
```

Programme permettant de tracer la solution à certains temps :

```
#include <iostream>
#include <fstream>
#include <assert.h>
#include <cmath>
#include <cstdlib>
#include "edp.h"


using namespace std;
double max(const Matrice& M){
    double retour=0;
    for(int i=0; i<M.dim2(); i++){
     if(retour<M(1,i)) retour=M(1,i);
    }
    return retour;
}

int main(){
    int N;
    double L=1;
    double T=1;
    N=50;
    double delta_X=1.0/50.0;
    Matrice M(read_matr("matrice_sol.txt"));
    Matrice cond(read_matr("matrice_cond.txt"));


    double delta_T=pow(delta_X,2)/(max(cond)*2.);
    cout << delta_T << endl;
    Vecteur temps(8);
    for(int i=0;i<8;i++){
     temps(i)=i*0.01;
    }
    cout << temps;

    int k=0;
```

```
    for(int i=0;i<M.dim1();i++){
        if(k<=7){
            if((i*delta_T)>=temps(k)){
                ofstream file("trace.txt");
                for(int j=0;j<M.dim2();j++){
                 file << j*1.0/50.0 << " " << M(i,j) << endl;
                }
                file << endl << endl ;
                ofstream file1("trace1.gnuplot");
                file1 << "reset" << endl;
                file1 << "set terminal png" << endl;
                file1 << "set output\"epsilon_0_5_t=" << k << ".png\""<<endl;
                file1 << "set yrange[0:200]" << endl;
                file1 << "set title \"t=" << temps(k) << "\""<<endl;
                file1 << "plot 'trace.txt' with lines notitle" << endl;
                //file1 << "pause -1" << endl;
                system("gnuplot trace1.gnuplot");
                k++;
            }
        }
    }
return 0;
}
```