INSTITUT NATIONAL DES SCIENCES APPLIQUÉES DE ROUEN
DÉPARTEMENT GÉNIE MATHÉMATIQUE

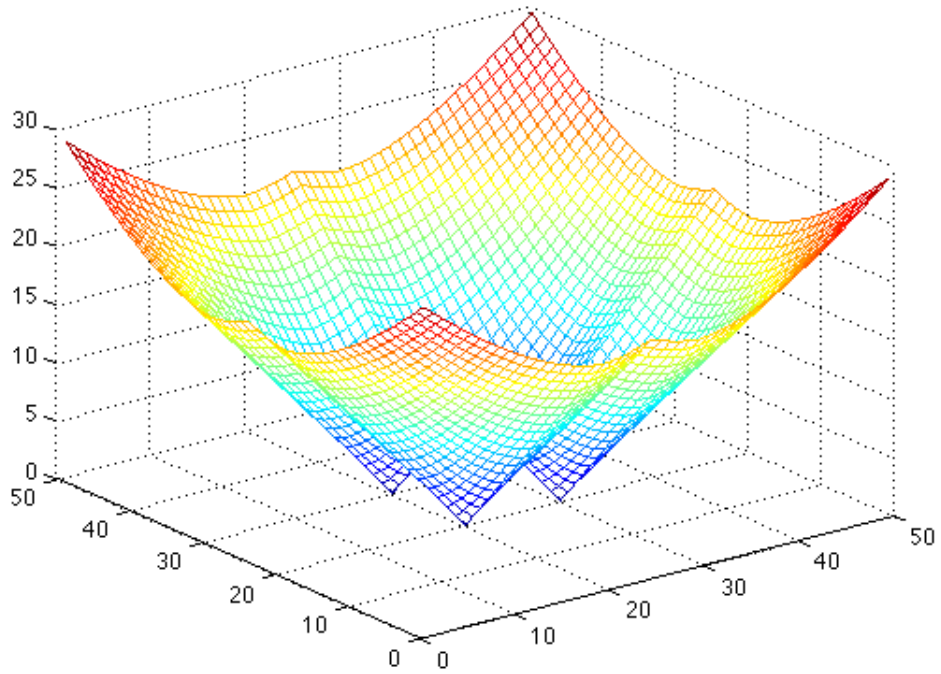
INSTITUT NATIONAL DES SCIENCES APPLIQUÉES ROUEN

27 novembre 2017

Méthode de *fast-marching* généralisée

Nathan ROUXELIN
Dhia STAMBOULI

Adam QUINQUENNEL
Timothée SCHMODERER



Semestre 9
Rapport de projet
Équations de Hamilton-Jacobi

TABLE DES MATIÈRES

INTRODUCTION

Notre projet pour l'EC *équations de Hamilton-Jacobi* porte sur la méthode *fast-marching* qui dans ce cas est appliquée à la segmentation des images (i.e. trouver les contours d'éléments présents dans une image). L'article étudié, publié le 29 février 2008, écrit par N. Forcadel, C. Le Guyader et C. Gout porte sur l'introduction d'une méthode numérique pour résoudre le problème de mouvement de front.
C'est à dire le problème de l'évolution d'un front (i.e. une courbe Γ_t fermée) dont la vitesse normale est donnée. Comme nous l'expliquerons dans le premier chapitre, une reformulation de ce problème en terme de ligne de niveau d'une fonction mène à une équation aux dérivées partielles, dont l'approximation numérique de sa soliution est au coeur de l'article.

Dans un premier, temps, la méthode dite de *fast-marching classique* est expliquée, cette méthode permet de répondre au problème posé dans le cas où la vitesse normale au front est de signe constant. Puis une généralisation de cette méthode permettra de considérer des vitesses normales quelconques.
Dans ce rapport, nous exposerons la méthode de *fast-marching* avec le plus de détails possibles, puis la méthode de *fast-marching* généralisée qui permet de répondre efficacement au problème de mouvement de front. Nous discuterons également de la complexité algorithmique de la méthode.

Dans un second, nous proposerons une implémentation de la méthode de *fast-marching* généralisée avec le logiciel *Matlab*, ainsi que des résultats numériques.

CHAPITRE 1

ÉTUDE THÉORIQUE

Dans un premier temps, nous nous intéressons à l'établissement de l'équation de Hamilton-Jacobi résolvant le problème de mouvement de front.

I Établissement de l'équation eikonale

Soit un domaine $\Omega \subset \mathbb{R}^2$, on s'intéresse au mouvement du front représenté par la courbe fermée Γ_t. On suppose que le mouvement a lieu dans la direction normale à la courbe, mathématiquement cela s'exprime ainsi

$$\frac{\partial \Gamma_t}{\partial t} = c(x) \cdot \overrightarrow{n_{x,t}} \tag{1.1}$$

où $\overrightarrow{n_{x,t}}$ désigne la normale extérieure à Γ_t au point x.
Résoudre cette équation est loin d'être trivial, une première idée est d'augmenter la dimension du problème pour représenter la courbe Γ_t comme la ligne de niveau 0 d'une fonction u dans l'espace $\mathbb{R}^3$, c'est la méthode *level-set*. C'est-à-dire :

$$\Gamma_t = \left\{ x \in \mathbb{R}^2 \middle| u(x,t) = 0 \right\}$$

Notons Ω_t le domaine de $\mathbb{R}^2$ délimité par Γ_t, on suppose que u vérifie les conditions suivantes :

$$\forall t \in \mathbb{R}_+,\ \forall x \in \mathbb{R}^2,\ \begin{cases} x \in \Omega_t \Rightarrow u(x,t) > 0 \\ x \in \Omega_t^c \Rightarrow u(x,t) < 0 \end{cases}$$

En supposant que u est assez régulière, on peut exprimer la normale à l'aide du gradient

$$\overrightarrow{n_{x,t}} = -\frac{\nabla_x u(x,t)}{|\nabla_x u(x,t)|}.$$

On sait que u est constante sur Γ_t, c'est-à-dire $u(\Gamma_t, t) \equiv 0$, d'où

$$\frac{\mathrm{d}u}{\mathrm{d}t}(x(t),t) = 0, \forall x \in \Gamma_t$$

En utilisant la règle de dérivation des fonctions composées, on a

$$\frac{\partial u}{\partial t}(x,t) + \frac{\partial \Gamma_t}{\partial t} \cdot \nabla_x u(x,t) = 0$$
$$\frac{\partial u}{\partial t}(x,t) + c(x)\overrightarrow{n_{x,t}} \cdot \nabla_x u(x,t) = 0$$

On obtient finalement l'équation eikonale

$$\frac{\partial u}{\partial t}(x,t) = c(x)\,|\nabla_x u(x,t)| \tag{Eik}$$

Cette équation, assez simple, a nécessité d'augmenter la dimension du problème de 1 en introduisant le temps comme une troisième dimension. Cette augmentation de la complexité du problème est numériquement non négligeable. C'est pourquoi une méthode comme la méthode de *fast-marching* ont été introduite.
Commençons par un résultat d'existence des solutions de l'équation (Eik) :

Théorème I.1 : EXISTENCE ET UNICITÉ DE LA SOLUTION POUR (Eik)

Supposons que la vitesse $c(x,t)$ soit lipschitzienne en espace. Alors le problème suivant admet une unique solution :

$$\begin{aligned} \frac{\partial u}{\partial t}(x,t) - c(x)\,|\nabla_x u(x,t)| &= 0 \qquad (x,t) \in \mathbb{R}^2 \times \mathbb{R} \\ u(x,0) &= u_0(x) \text{ bornée} \end{aligned}$$

Preuve :
Montrons qu'il existe une sous-solution barrière et une sur-solution barrière :
Posons $\tilde{u} = \|u\|_\infty$, c'est une sur solution barrière puisque :

$$\frac{\partial \tilde{u}}{\partial t} - c(x,t)\|\nabla \tilde{u}\| = 0 + c(x,t) * 0 = 0$$

Et, il est évident que $\tilde{u}(x,0) \geqslant u_0(x)$ De même, $\underset{\sim}{u} = -\|u\|_\infty$ est une sous-solution barrière.
Montrons que le problème satisfait aux hypothèses de monotonie et de stabilité :

Monotonie : Ici, $F(x,t,u,Du) = -c(x,t)\|Du\|$ il est alors évident que

$$\exists \delta = 0,\ \forall r \leqslant s,\ \forall p \qquad F(x,t,r,p) - F(x,t,s,p) \geqslant \delta(r-s)$$

Stabilité : Il s'agit de montrer que :

$$\forall R > 0\, \forall r \in [-R,R]\, \forall x,y \in \mathbb{R}^2\, \forall p \in \mathbb{R}^2 \qquad \|F(x,t,r,p) - F(y,t,r,p)\| \leqslant \omega_R(\|x-y\|(1+\|p\|))$$

Où ω_R est un module de continuité, et la majoration est uniforme en t.
On a :

$$\begin{aligned} \|F(x,t,r,p) - F(y,t,r,p)\| &= \|-c(x,t)\|p + c(y,t)\|p\|\| \\ &\leqslant \|c(y,t) - c(x,t)\|\ \|p\| \\ &\leqslant L_c\|x-y\|\ \|p\| \end{aligned}$$

Donc F a bien un module de continuité.
Alors par application du théorème du cours, nous avons existence et unicité de la solution pour le problème. ∎

II La méthode *fast-marching* classique

La résolution numérique de (Eik) peut être très coûteuse, nous allons donc présenter une méthode permettant de la résoudre efficacement. Cette méthode ne fonctionne que si la vitesse est positive.
Notons $T(x)$ le temps d'arrivée du front au point x. On cherche u sous la forme

$$u(x,t) = T(x) + t.$$

On peut alors réécrire (Eik) ainsi

$$1 = c(x)\,|\nabla T|$$

avec la condition $T \equiv 0$ sur Γ_0.

1 Schéma numérique pour l'équation en T

On commence par discrétiser notre domaine spatial . On note $\Delta x > 0$ le pas en espace. On utilise la grille

$$x_{ij} := (i\Delta x, j\Delta x) \qquad (i,j) \in \mathbb{Z}^2$$

On désigne par $T_{i,j}$ l'approximation de T au point x_{ij}. On utilise le schéma numérique suivant :

$$\max\left(T_{ij} - T_{i+1,j}, T_{ij} - T_{i-1,j}, 0\right)^2 + \max\left(T_{ij} - T_{i,j-1}, T_{i,j} - T_{i,j+1}, 0\right)^2 = \frac{\Delta x^2}{c_{ij}^2} \tag{1.2}$$

C'est à dire, de manière équivalente pour faire apparître les approximations du gradient sous forme de différences finies :

$$\max\left(\frac{T_{ij} - T_{i+1,j}}{\Delta x}, \frac{T_{ij} - T_{i-1,j}}{\Delta x}, 0\right)^2 + \max\left(\frac{T_{ij} - T_{i,j-1}}{\Delta x}, \frac{T_{i,j} - T_{i,j+1}}{\Delta x}, 0\right)^2 = \frac{1}{c_{ij}^2}$$

Ce schéma est adapté notre problème : on utilise les maxima pour calculer notre solution dans la direction du plus grand déplacement. Ils nous permettent également de ne pas prendre en compte les déplacements «en arrière» quand la dérivée est négative.

Nous allons montrer que ce schéma est convergent. Remarquons toutefois qu'il est valable même si la vitesse est négative. La contrainte sur le signe de la vitesse arrivera plus tard dans la méthode.

Consistance

On commence par poser

$$S_h((x,y),T,\varphi) :=$$
$$\max\left(\frac{T - \varphi(x-h,y)}{h}, \frac{T - \varphi(x+h,y)}{h}, 0\right)^2 + \max\left(\frac{T - \varphi(x,y-h)}{h}, \frac{T - \varphi(x,y+h)}{h}, 0\right)^2$$

Comme le schéma est régulier, on a

$$\lim_{\substack{(v,w)\to(x,y)\\ \xi\to 0\\ h\to 0}} S_h((v,w),T+\xi,\varphi+\xi) = \lim_{h\to 0} S_h((x,y),T,\varphi).$$

Il suffit donc de montrer que notre schéma converge bien vers la bonne équation. On peut alors réécrire (1.2) sous la forme suivante

$$\max\left(\frac{T_{ij}-T_{i+1,j}}{\Delta x},\frac{T_{ij}-T_{i-1,j}}{\Delta x},0\right)^2+\max\left(\frac{T_{ij}-T_{i,j-1}}{\Delta x},\frac{T_{i,j}-T_{i,j+1}}{\Delta x},0\right)^2=\frac{1}{c_{ij}^2} \quad (1.3)$$

En faisant un développement de Taylor, on obtient

$$\begin{aligned} T\left(x_{i+1,j}\right) &= T\left(x_{ij}\right)+\Delta x\frac{\partial T}{\partial x}\left(x_{ij}\right)+o\left(\Delta x^2\right) \\ \frac{T\left(x_{i+1,j}\right)-T\left(x_{i,j}\right)}{\Delta x} &= \frac{\partial T}{\partial x}\left(x_{ij}\right)+o\left(\Delta x^2\right) \end{aligned}$$

On peut faire des calculs analogues pour les autres quotients dans (1.3). Ainsi si les maxima ne sont pas nuls, quand $\Delta x \to 0$, notre schéma tend vers

$$\left(\frac{\partial T}{\partial x}\right)^2+\left(\frac{\partial T}{\partial y}\right)^2=\frac{1}{c^2} \iff |\nabla T|=\frac{1}{c}$$

Il faut maintenant répondre à la question suivante : que se passe-t-il si un des maxima vaut zéro ?
Supposons, par exemple, que le premier maximum est nul. On a

$$T\left(x_{ij}\right)\leqslant T\left(x_{i+1,j}\right) \quad \text{et} \quad T\left(x_{ij}\right)\leqslant T\left(x_{i-1,j}\right)$$

Cela signifie que dans la première direction spatiale, T admet un minimum local en x_{ij}. On a alors

$$\frac{\partial T}{\partial x}\left(x_{ij}\right)=0$$

ce qui correspond bien à la valeur donnée par notre schéma.
Le schéma proposé est donc consistant.

Monotonie

On veut montrer que

$$\begin{gathered} \forall (x,y)\in\overline{G},\forall u\in\mathbb{R},\forall u_1,u_2:G\to\mathbb{R},\ \text{tq}\ u_1\geqslant u_2, \\ S_h\left((x,y),u,u_1\right)\leqslant S_h\left((x,y),u,u_2\right). \end{gathered}$$

C'est quasiment immédiat. En effet, on a

$$u_1(x-h,y)\geqslant u_2(x-h,y) \quad \text{et} \quad u_1(x+h,y)\geqslant u_2(x+h-y).$$

D'où

$$\frac{u-u_1(x-h,y)}{h}\leqslant\frac{u-u_2(x-h,y)}{h} \quad \text{et} \quad \frac{u-u_1(x+h,y)}{h}\leqslant\frac{u-u_2(x+h,y)}{h}.$$

Ces inégalités sont valables sur toute la grille. En particulier, elles sont vraies pour les maxima.
On en déduit immédiatement le résultat.

Stabilité

Ce schéma est sable : à chaque itération, soit on obtient 0 lors des calculs (et donc il ne se passe rien), soit on obtient un temps élevé qui diminue par décroissance du schéma. Il ne peut donc pas y avoir d'explosion numérique.

2 Algorithme

Pour résoudre efficacement ce problème, nous n'allons pas résoudre (1.2) sur tout le domaine. Nous allons séparer notre domaine en trois régions :

Les *frozen points* : ce sont les points déjà visités par le front et pour lesquels on connaît la valeur de T. À l'instant t_n, on note Fr^n l'ensemble de ces points.

La *narrow band* : il s'agit du voisinnage des *frozen points*, c'est-à-dire les points non visités mais qui ont un voisin déhà visité. À l'instant t_n, on la note NB^n.

Les points éloignés : ce sont tous les autres points du domaine.

Si on connaît le front à l'instant t_n, les points accessibles à l'instant t_{n+1} sont les points de la *narrow band* NB^n. Il suffira donc de résoudre (1.2) pour ces points, ce qui permettra de réduire le nombre de calculs et la complexité de l'algorithme.
Cette approche a toutefois une limite : le front ne peut pas reculer. La vitesse c doit donc rester positive sur tout le domaine.

			Méthode *fast-marching*	
Initialisation				
	Matrice de passage	:	$T_{ij} = 0$	$\forall (i,j)$ tq $(i\Delta x, j\Delta x) \in \Omega_0$
	Temps	:	$n = 0$	
Boucle				
1.	Calculer T_{ij}	:	Résoudre (1.2)	$\forall (i,j) \in NB^n$
2.	Mettre à jour Fr^{n+1}	:	Accepter les (i,j)	tq $T_{i,j} = \min_{(a,b)} T_{a,b}$
3.	Mettre à jour NB^{n+1}	:	Frontière de Fr^{n+1}	
4.	Mettre à jour n	:	$n = n+1$	

III Méthode généralisée

Nous souhaitons maintenant à généraliser cette première méthode au cas ou la vitesse n'est pas de signe constant. La notion de *frozen points* présenté précédemment est alors inadaptée pour répondre au problème, car un point peut être traversé plusieurs fois par le front. Dans un premier temps, suivant la méthode utilisée dans l'article, nous reformulons la méthode précédente en introduisant un champ θ dans le domaine. Cet ajout permettra, en modifiant l'algorithme légèrement, de considérer des vitesses de signe variable.

1 Re-formulation de la méthode classique

Pour pallier au fait que les points puissent être atteint plusieurs fois par le front, nous introduisons un champ θ dans notre domaine tel que

$$\begin{cases} \theta = 1 & \text{dans } \Omega_t \\ \theta = -1 & \text{dans } \Omega_t^c \end{cases}$$

et le front est alors représenté par la discontinuité de θ.

Définissons certaines notations avant de réécrire l'algorithme.
On définit le voisinage d'un point $I \in \mathbb{Z}^2$ par $V(I) = \left\{ J \in \mathbb{Z}^2, |I - J| \leqslant 1 \right\} \setminus \{I\}$.
On note

$$Fr^n = \{I, \theta_I^n = 1\}$$

la *région gelée* à la n-ième itération. Remarquons que l'expression *gelée* n'est peut être pas la plus adaptée ici, puisque cette région sera redéfinie à chaque itération, si le front revient en arrière, un point qui était dans la région gelée pourrait en sortir.
Nous avons alors la *narrow band* définie à l'étape n définie comme le voisinage des *frozen points*, c'est-à-dire

$$NB^n = \left\{ I \in Fr^n,\ \exists J \in V(I), \theta_J^n = -\theta_I^n = 1 \right\}.$$

On définit enfin l'ensemble des points utiles à un point I à l'étape n par

$$\mathcal{U}^n(I) = \left\{ J \in V(I), \theta_J^n = 1 \right\}.$$

et l'ensemble des points utiles à l'étape n par

$$\mathcal{U}^n = \bigcup_{I \in NB^n} \mathcal{U}^n(I).$$

C'est à dire, comme dans cette reformulation la vitesse est de signe constant, l'ensemble des points dans le voisinage du front qui vont être atteint :
L'algorithme se réécrit de la manière suivante :

Méthode *fast-marching* II

Initialisation

	Temps	:	$n = 1, t_0 = 0$
	θ_0	:	$\theta^0 = \begin{cases} 1 & \text{si } x_I \in \Omega_0 \\ -1 & \text{sinon} \end{cases}$
	u_0	:	$u_I^0 = \begin{cases} 0 & \text{si } I \in \mathcal{U}^0 \\ +\infty & \text{sinon} \end{cases}$

Boucle

1.	Calculer $\tilde{u}_I^{n-1}$	:	Résoudre (1.4) $\forall I \in NB^{n-1}$
2.	Calcul du temps	:	$t_n = \inf_{I \in NB^{n-1}} \tilde{u}_I^{n-1}$
3.	Mettre à jour NA^n	:	$NA^n = \left\{ I, \tilde{u}_I^{n-1} = t_n \right\}$
4.	Mettre à jour θ^n	:	$\theta_I^n = \begin{cases} 1 & \text{si } I \in NA^n \\ \theta_I^{n-1} & \text{sinon} \end{cases}$
5.	Calcul de u_I^n	:	$u_I^n = \begin{cases} t_n & \text{si } I \in NA^n \text{ et } I \in \mathcal{U}^n \\ u_I^{n-1} & \text{si } I \in \mathcal{U}^{n-1} \setminus NA^n \text{ et } I \in \mathcal{U}^n \\ +\infty & \text{sinon} \end{cases}$
6.	Mettre à jour n	:	$n = n + 1$

Cet algorithme repose sur la résolution de l'équation suivante dans l'étape 1. :

$$\max\left(0, \tilde{u}_I^{n-1} - u_{i_1+1,i_2}^{n-1}, \tilde{u}_I^{n-1} - u_{i_1-1,i_2}^{n-1}\right)^2 + \max\left(0, \tilde{u}_I^{n-1} - u_{i_1,i_2+1}^{n-1}, \tilde{u}_I^{n-1} - u_{i_1,i_2-1}^{n-1}\right)^2 = \frac{(\Delta x)^2}{|c_I^{n-1}|^2} \tag{1.4}$$

Remarque : Ici, nous optons pour le calcul de u mais il est en fait équivalent et plus intuitif de se dire que nous calculons le temps de passage, comme précédemment.

2 La méthode *fast-marching* généralisée

A présent, nous allons modifier l'algorithme pour que les vitesses quelconques puissent être prises en compte. Pour garder une vitesse lipschitzienne nous ajoutons des zéro la pou la vitesse change de signe. Soit $c_I^n \equiv c(x_I, t_n)$, nous posons alors :

$$\hat{c}_I^n \equiv \begin{cases} 0 & \text{si il existe } J \in V(I) \text{ tel que } c_I^n c_J^n < 0 \text{ et } |c_I^n| < |c_J^n| \\ c_I^n & \text{sinon} \end{cases}$$

Nous pouvons maintenant passer à la description de l'algorithme de *fast-marching* généralisé pour c non signé.
Nous définissons, comme pour la méthode du *fast-marching* classique, la *Narrow Band* comme l'ensemble le des points directement atteignables par le front, c'est-à-dire

$$NB^n = \left\{ I \in \mathbb{Z}^2,\ \exists J \in V(I),\ \theta_I^n = -\theta_J^n \text{ et } \theta_I^n \hat{c}_I^n < 0 \right\}$$

Nous redéfinissons, l'ensemble des points utiles a I :

$$\mathcal{U}^n(I) = \left\{ J \in V(I),\ \theta_I^n = -\theta_J^n \right\}$$

Remarque : Comme annoncé, l'introduction du champ θ nous permet d'adapter l'algorithme très facilement. Par exemple, l'ensemble de la narrow band est définie grâce au champ θ comme les points pouvant être atteint par le front immédiatement (suivant le signe de c.

Dans le calcul des points utiles, certains n'ont pas d'intérêt, notamment si J et I sont du même côté du front, la connaissance de J ne donne pas plus d'information sur le temps d'arrivée du front en I. Ainsi, nous n'utiliserons pas les points points $J \in V(I) \cap \mathcal{U}^n$ si $\theta_J^n = \theta_I^n$. Suivant la même idée, introduison une dernière notation :

$$u_{J\to I}^n = \begin{cases} u_J^n & \text{si } J \in \mathcal{U}^n(I) \\ +\infty & \text{sinon} \end{cases}$$

Cette notation, permet de ne se servir que des points utiles dans la résolution de (1.3).

On peut alors écrire l'algorithme comme suit

Méthode *fast-marching* généralisée

Initialisation

	Temps	:	$n = 1, t_0 = 0$
	θ_0	:	$\theta^0 = \begin{cases} 1 & \text{si } x_I \in \Omega_0 \\ -1 & \text{sinon} \end{cases}$
	u_0	:	$u_I^0 = \begin{cases} 0 & \text{si } I \in \mathcal{U}^0 \\ +\infty & \text{sinon} \end{cases}$

Boucle

1.	Calculer $\tilde{u}_I^{n-1}$	:	Résoudre (1.5) $\forall I \in NB^{n-1}$
2.	Calcul du temps	:	$\tilde{t}_n = \inf_{I \in NB^{n-1}} \tilde{u}_I^{n-1}$
	Tronquer $\tilde{t}_n$	:	$t_n = \max\left(t_{n-1}, \min\left(\tilde{t}_n, t_{n-1} + \Delta t\right)\right)$
	Si $\begin{cases} t_n = t_{n-1} + \Delta t \\ t_n < \tilde{t}_n \end{cases}$	:	Revenir à 1 avec $\begin{cases} n = n+1 \\ \theta^n = \theta^{n-1} \\ u^n = u^{n-1} \end{cases}$
3.	Mettre à jour NA^n	:	$NA^n = \left\{ I, \tilde{u}_I^{n-1} = \tilde{t}_n \right\}$
4.	Mettre à jour θ^n	:	$\theta_I^n = \begin{cases} -\theta_I^{n-1} & \text{si } I \in NA^n \\ \theta_I^{n-1} & \text{sinon} \end{cases}$
5.	Calcul de u_I^n	:	$u_I^n = \begin{cases} t_n & \text{si } I \in NA^n \text{ et } I \in \mathcal{U}^n \\ u_I^{n-1} & \text{si } I \in \mathcal{U}^{n-1} \backslash NA^n \text{ et } I \in \mathcal{U}^n \\ +\infty & \text{sinon} \end{cases}$
6.	Mettre à jour n	:	$n = n + 1$

Cette algorithme repose sur la résolution de l'équation suivante dans l'étape 1.

$$\begin{aligned} &\max\left(0, \tilde{u}_I^{n-1} - u^{n-1}_{(i_1+1,i_2)\to I}, \tilde{u}_I^{n-1} - u^{n-1}_{(i_1-1,i_2)\to I}\right)^2 \\ +&\max\left(0, \tilde{u}_I^{n-1} - u^{n-1}_{(i_1,i_2+1)\to I}, \tilde{u}_I^{n-1} - u^{n-1}_{(i_1,i_2-1)\to I}\right)^2 \\ =&\frac{(\Delta x)^2}{|\hat{c}_I^{n-1}|^2} \end{aligned} \tag{1.5}$$

CHAPITRE 2

IMPLÉMENTATION ET RÉSULTATS NUMÉRIQUES

L'implémentation a été réalisée en Matlab, pour des raisons de simplicité.

I Méthode de Fast-marching

Dans l'esprit de Matlab [1], les données sont représentés par des matrices. Dans un premier temps, le domaine Ω (le carré $[0,1] \times [0,1]$) est discrétisé en $N+1$ points dans les deux directions. Dans nos tests, la vitesse normale est fixée à 1 dans tout le domaine.

```
% donne la vitesse sur la grille
function c = velocity(N)
 c = ones(N+3);
end
```

Listing 2.1 – Code donnant la vitesse en tous les points du domaine

Partant d'une matrice T_0 donnant les temps initiaux : $T_{ij} = 0$ si $ij \in \Omega_0$ et $+\infty$ sinon.

```
N = 50; % Nb de pts de discretisations dans le sens des x et des y
dx = 1;

% matrice des temps de passage
% on padd T par des bordures (pour gerer les bords facilements)
init = 100;
T = 10^5*ones(N+3);
T(2:end-1,2:end-1) = init*ones(N+1); % les temps effectifs de
    passage

% Initialisation :
% On cree des masques AP, NB

% t = 0 les poinst acceptes sont
AP = zeros(N+3);
AP(2,2:end-1) = 1; AP(16,15) = 1;
AP = logical(AP);
```

1. Les codes sont disponibles à l'dresse suivante : https://github.com/tschmoderer/fmm-prj

```
T(AP) = 0; % les temps de passages a l'instant 0
```

Listing 2.2 – Initialisation

Cette matrice est calculée à partir de la matrice *AP* (pour *accepted point*) qui agit comme un masque sur les éléments de *T*. Le calcul de la narrow band se fait facilement grâce au padding appliqué aux données :

```
% Etant donn  un champ de points accept s
% Renvoi la narrow bandwidth
% en entr e on a deja une grille padd e

function NB = narrow(AP);
 s = size(AP);
 nb = zeros(s);

 for i = 2:s(1)-1
  for j = 2: s(2)-1
    if (AP(i,j) == 1)
      if (AP(i+1,j) ~= 1) nb(i+1,j) = 1; end;
      if (AP(i,j+1) ~= 1) nb(i,j+1) = 1; end;
      if (AP(i-1,j) ~= 1) nb(i-1,j) = 1; end;
      if (AP(i,j-1) ~= 1) nb(i,j-1) = 1; end;
   end
  end
 end
 % on a juste a remettre des zeros dans le sbordures de NB
 NB = zeros(s);
 NB(2:end-1,2:end-1) = nb(2:end-1,2:end-1);
end
```

Listing 2.3 – Code calculant la narrow band

Avec cette représentation des données, il est facile de calculer le temps de passage du front dans la narrow band, il suffit de traiter tous les cas possibles :

```
while sum(AP(:)) < (N+1)^2
 NB = narrow(AP);
 NB = logical(NB);

 c = velocity(N);

 % calcul du temps de passage sur la narrow band
 % On va construire TG, TD, TH, TB

 NG = zeros(size(NB)); NG(2:end-1,1:end-2) = NB(2:end-1,2:end-1); %
     les noeuds a gauche;
 ND = zeros(size(NB)); ND(2:end-1,3:end) = NB(2:end-1,2:end-1); %
    les noeuds a droite;
 NH = zeros(size(NB)); NH(1:end-2,2:end-1) = NB(2:end-1,2:end-1); %
     les noeuds a en haut;
 Nb = zeros(size(NB)); Nb(3:end,2:end-1) = NB(2:end-1,2:end-1); %
    les noeuds a bas;

 NG = logical(NG); ND = logical(ND); NH = logical(NH); Nb = logical
    (Nb);

 TG = T(NG); TD = T(ND); TH = T(NH); Tb = T(Nb);
```

```
Ti = T(NB);
ci = c(NB);

for j = 1:length(Ti)
 if (Ti(j) < TG(j) && Ti(j) < TD(j) && Ti(j) < TH(j) && Ti(j) < Tb
    (j))
  error('erreur resolution de l equation');
 elseif (max(max(Ti(j) - TG(j),Ti(j) - TD(j)),0) == 0)
  if (Ti(j) - TH(j) <= Ti(j) - Tb(j))
   % (Ti-Tb)^2 = ..
   Ti(j) = Tb(j) + (dx/ci(j));
  else
   % (Ti-TH)^2 = ..
   Ti(j) = TH(j) + (dx/ci(j));
  end
 elseif (max(max(Ti(j) - TH(j),Ti(j) - Tb(j)),0) == 0)
  if (Ti(j) - TG(j) <= Ti(j) - TD(j))
   % (Ti-TD)^2 = ..
   Ti(j) = TD(j) + (dx/ci(j));
  else
   % (Ti-TD)^2 = ..
   Ti(j) = TG(j) + (dx/ci(j));
  end
 elseif (Ti(j) - TG(j) <= Ti(j) - TD(j) && Ti(j) - TH(j) <= Ti(j)
    - Tb(j))
  % (Ti-TD)^2 + (Ti-Tb)^2=..
  delta = (2*TD(j)+2*Tb(j))^2-4*2*(TD(j)^2+Tb(j)^2-(dx/ci(j))^2);
  Ti(j) = 0.5*(TD(j) + Tb(j)) + 0.25*sqrt(delta);
 elseif (Ti(j) - TG(j) <= Ti(j) - TD(j) && Ti(j) - TH(j) >= Ti(j)
    - Tb(j))
  % (Ti-TD)^2 + (Ti-TH)^2=..
  delta = (2*TD(j)+2*TH(j))^2-4*2*(TD(j)^2+TH(j)^2-(dx/ci(j))^2);
  Ti(j) = 0.5*(TD(j) + TH(j)) + 0.25*sqrt(delta);
 elseif (Ti(j) - TG(j) >= Ti(j) - TD(j) && Ti(j) - TH(j) <= Ti(j)
    - Tb(j))
  % (Ti-TG)^2 + (Ti-Tb)^2=..
  delta = (2*TG(j)+2*Tb(j))^2-4*2*(TG(j)^2+Tb(j)^2-(dx/ci(j))^2);
  Ti(j) = 0.5*(TG(j) + Tb(j)) + 0.25*sqrt(delta);
 elseif (Ti(j) - TG(j) >= Ti(j) - TD(j) && Ti(j) - TH(j) >= Ti(j)
    - Tb(j))
  % (Ti-TG)^2 + (Ti-TH)^2=..
  delta = (2*TG(j)+2*TH(j))^2-4*2*(TG(j)^2+TH(j)^2-(dx/ci(j))^2);
  Ti(j) = 0.5*(TG(j) + TH(j)) + 0.25*sqrt(delta);
 end
end
```

Listing 2.4 – Calcul des temps de passage

Enfin, la recherche du temps de passage de minimum est réalisé grâce à la fonction *min* de matlab. Puis tout les point de la *narrow band* réalisant ce temps sont mis à jour alors que les autres sont remis à un temps égal à $+\infty$.

```
% trouver le min de Ti
[m I] = min(Ti);
 I = find(Ti == m);
```

```
 Ti(~ismember(I,[1:length(Ti)])) = init;
T(NB) = Ti;

% mise a jour des ts AP :
accept = zeros(length(T(NB)),1); accept(I) = 1;
AP(NB) = accept;
AP = logical(AP);

t = t+ m;
```

Listing 2.5 – Mise à jour des points acceptés

A partir de là, il suffit d'itérer le processus jusque tout les ponts soient acceptées. Voici les résultats que nous obtenons pour le dommaine discrétisé en $N = 30$ points, ceux-ci sont conforment à l'intuition dan ce cas très simple où la vitesse est constante positive :

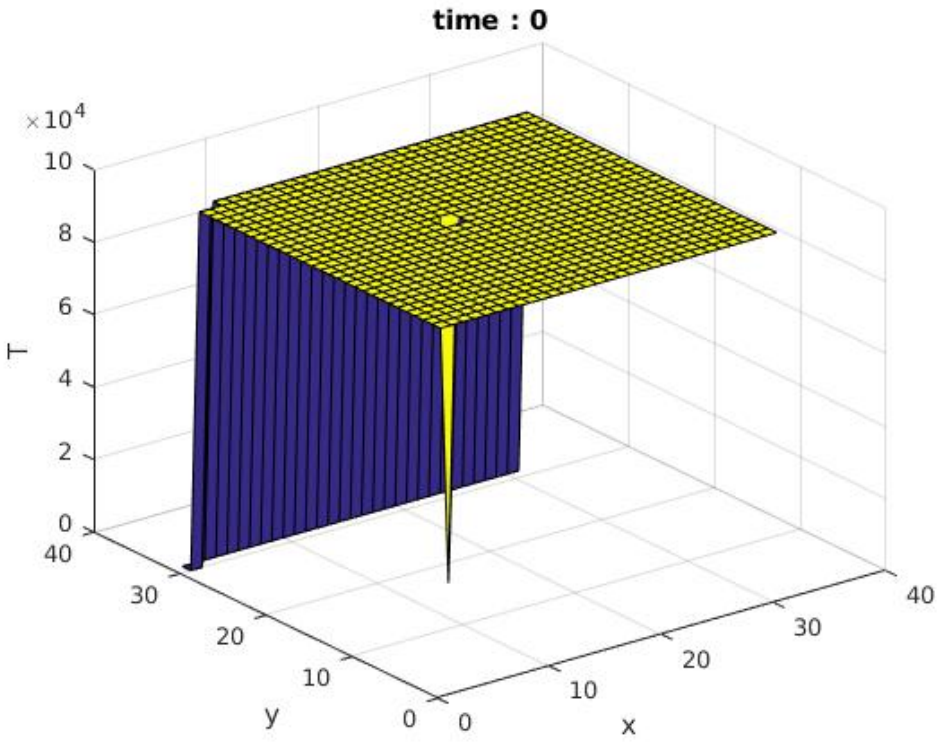
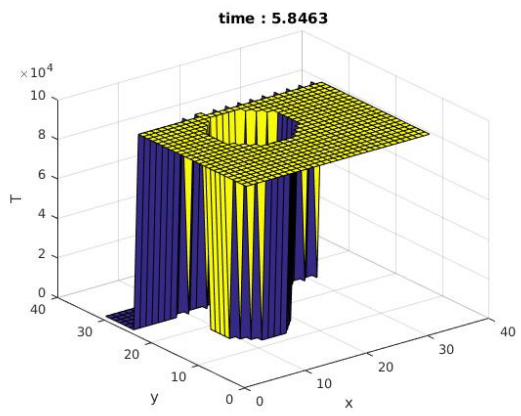
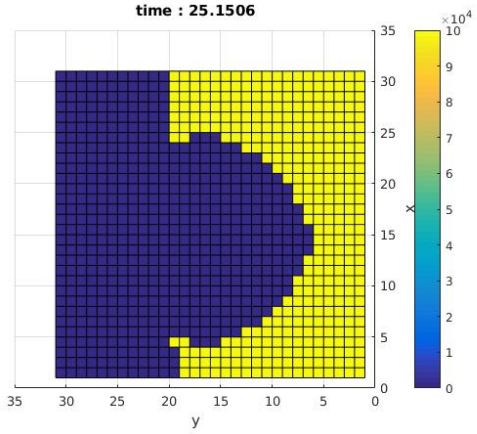


FIGURE 2.1 – Front à l'instant initial

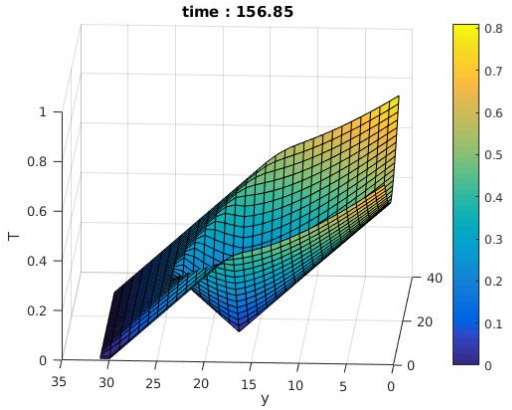
Remarque : Notre programme manque un peu d'efficacité car à chaque itération, tout les temps sur la narrow band sont recalculés alors qu'en vrai il suffirait de recalculer autour des points nouvellement acceptés.
Les modifications serait mineures mais un peu lourdes à mettre en place, par manque de temps, nous avons choisis de garder cette version (qui s'exécute assez rapidement sur des grilles raisonnable).



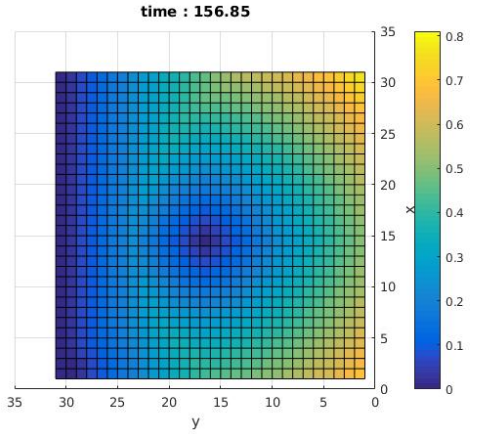
(a) Progression du front



(b) Progression vu du dessus



(a) Front final



(b) Front final vu du dessus

FIGURE 2.3

II Méthode généralisée

Dans un premier temps, nous avons modifié notre implémentation classique pour introduire θ, ce qui ne pose pas de problème majeur. Puis nous avons implémenté la méthode avec vitesse de signe variable. Comme présenté avec l'algorithme, l'implémentation de la méthode généralisée demande peu de modifications [2]

1 Application à la segmentation d'images

Enfin, nous avons pu appliquer la méthode de *fast-marching* généralisée à la segmentation d'images. Nous allons travailler sur le modèle simplifié de cerveau suivant

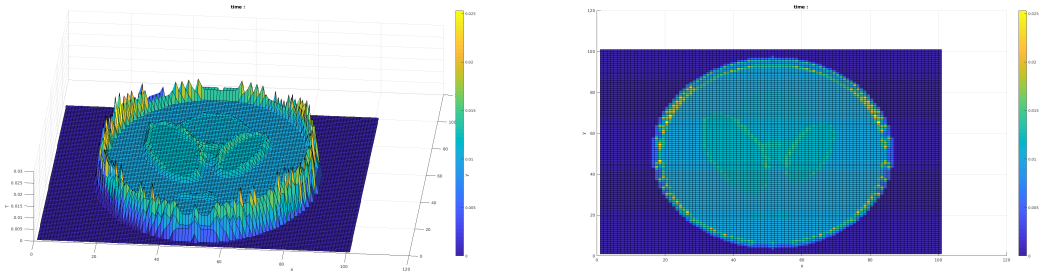
2. Le code est disponible au lien suivant : https://github.com/tschmoderer/fmm-prj/tree/master/GFMM

FIGURE 2.4 – Modèle de cerveau

Nous utiliserons un détecteur de contours extrêmement simple. La vitesse au point (i, j) sera donnée par

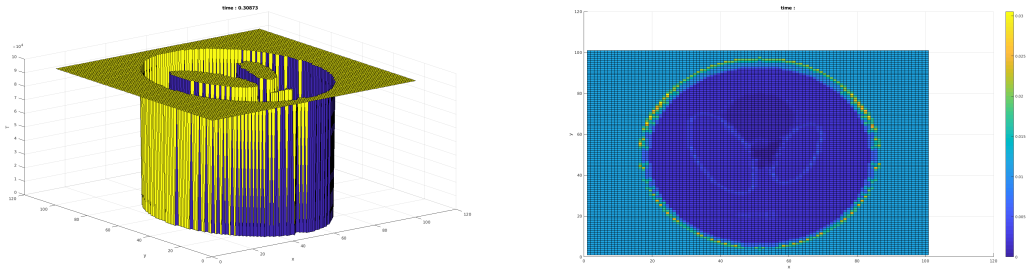
$$c(i, j) = \frac{1}{|\nabla w(i, j)|}$$

où w désigne l'image originale.
Nous avons obtenu des résultats assez spectaculaires :



(a) Front final – vue de côté (b) Front final – vue du dessus

FIGURE 2.5 – Résultats obtenus – Front initial à l'extérieur du cerveau



(a) Front final – vue de côté (b) Front final – vue du dessus

FIGURE 2.6 – Résultats obtenus – Front initial à l'intérieur du cerveau

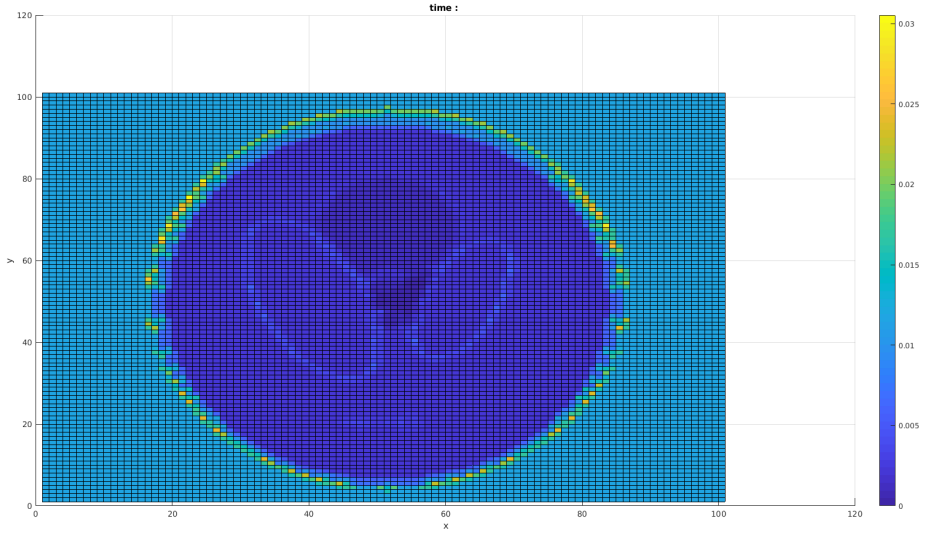
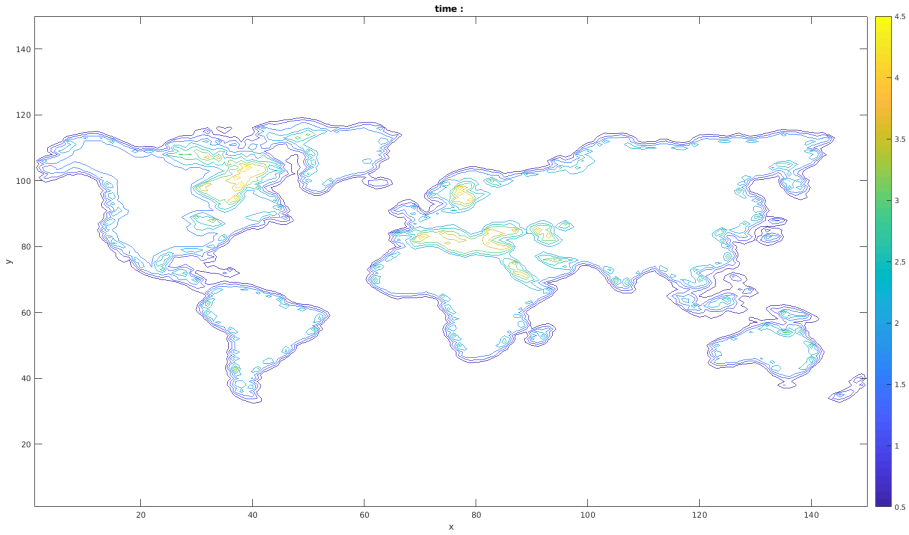


FIGURE 2.7 – Vue rapprochée

2 Segmentation d'un planisphère

Nous avons également obtenu des résultats satisfaisant lors de la segmentation d'un planisphère.

(a) Image originale



(b) Planisphère segmenté

FIGURE 2.8 – Segmentation d'un planisphère

CONCLUSION

Ce projet, nous a permis de nous familiariser avec la lecture d'articles scientifiques et de nous introduire auprès de la méthode de fast marching. Malheureusement, par manque de temps, nous n'avons pas pu faire toutes les expérimentations que que nous souhaitions.

Malgré l'utilisation d'un détecteur de contours rudimentaire, nous avons pu obtenir des résultats satisfaisants lors de nos essais numériques de segmentation sur différents types d'images.