Deep Eikonal Solver

Rapport de stage

Yohan FRAISSINET-TACHET

Juin/Août 2020

Sommaire

1 Introduction

Ce document a pour but d'expliquer l'implémentation d'une méthode mélangeant Deep Learning et équation eikonale provenant d'un article d'une conférence, SSVM 2019. Article rédigé par Moshe Lichtenstein, Gautam Pai et Ron Kimmel [10]. Il existe plusieurs méthodes pour résoudre l'équation eikonale, la plus connue étant la méthode de fast marching qui a une compléxité en $Nlog(N)$ pour un espace discrétisé composé de N points. Différentes versions de l'algorithme de marche rapide ont été développées avec une précision en $O(h)$ [16],$O(h^2)$ [15] et $O(h^3)$ [1] dans le cas d'une grille cartésienne 2D avec h le pas de discrétisation. Celle retenue dans ce document se base sur un procédé de marche rapide avec un solveur local et une étape de mise à jour.

Le solveur local s'occupe d'approximer localement le gradient afin d'estimer la distance d'un point du front d'onde aux sources. La précision du schéma de marche rapide dépend fortement du solveur local qui pour être plus précis nécessite d'être plus complexe [18]. L'intérêt de ce document est de remplacer ce solveur numérique local par un réseau de neurones entraîné pour fournir des estimations précises des distances locales pour une variété de formes géométriques et de conditions d'échantillonnages différentes.

2 A propos de l'équation Eikonale

Pour pouvoir se plonger correctement dans le problème, il faut tout d'abord se concentrer sur sa base : l'équation eikonale. Le fait est qu'elle apparaît dans de nombreux domaines de l'informatique et de la physique comme expliqué dans cette partie. Il est d'autant plus pertinent de connaître et de surtout comprendre cette équation afin d'appréhender ce projet de la meilleure des façons. Par ailleurs sa compréhension et les diverses applications de l'équation eikonale sont, outre l'aspect technique, une des motivations de ce projet.

2.1 Qu'est-ce que l'équation Eikonale ?

L'équation Eikonale (du grec $\epsilon\iota\kappa\omega\nu$ qui signifie image) est une équation aux dérivées partielles non linéaires définie telle que pour Ω un ensemble ouvert inclus dans $\mathbb{R}^n$, on a :

$$\begin{aligned} |\nabla u(x)| &= \frac{1}{f(x)} \\ u(x) &= g(x) \quad pour\ x \in \partial\Omega \end{aligned}$$

Avec ∇ l'opérateur gradient, f une fonction à valeurs positives et g la fonction qui définit les conditions aux frontières. Cette équation possède diverses interprétations et applications.

2.2 Electromagnétisme/physique : origine de l'équation eikonale

A l'origine, cette équation apparaît dans le domaine de la physique notamment en optique géométrique. En approximant les équations de Maxwell selon les lois de l'optique géométrique et en considérant un champ électrique de la forme $E(r,t) = E_0(r,t)e^{i(\omega t - k.r)}$ avec ω la pulsation et k le vecteur d'onde moyens de l'onde électromagnétique, on aboutit au résultat suivant :

$$E(r,t) = E_0(r,t)e^{i(\omega t - k\phi(r))}$$

Où $\phi(r)$ correspond aux surfaces de phase constante. Cette fonction s'appelle l'eikonale et décrit la courbe du front d'onde. En réinjectant ce résultat dans les équations de maxwell, après calculs on obtient l'équation suivante :

$$||\nabla\phi||^2 = n^2$$

Cette équation est l'équation fondamentale de l'optique géométrique et s'appelle l'équation eikonale. Cette approximation nécessite une faible variation de l'indice de réfraction du milieu n. A partir de celle-ci, il est ainsi possible de prédire les trajectoires des faisceaux lumineux qui se propagent dans un milieu homogène ou faiblement hétérogène.

L'équation eikonale permet par ailleurs de démontrer d'autres lois, telles que les lois de Snell-Descartes ou encore le principe de Fermat.

2.3 Domaines d'applications

En dehors de l'optique géométrique, l'équation eikonale est utilisée dans bien d'autres domaines :

En l'occurrence le problème dans lequel nous utilisons cette équation est un problème de calcul de distance. Ainsi ici f correspond à la fonction qui à tout $x \in \Omega$ lui associe sa vitesse $f(x)$. Et $u(x)$ est la distance minimale nécessaire à parcourir pour aller de la frontière $\partial\Omega$ jusqu'à x dans l'espace Ω. Dans notre cas, u(x) est également le temps minimal car la distance et la vitesse sont égales à un facteur près. Par exemple pour calculer la distance entre deux points dans un plan, il est possible de résoudre l'équation eikonale localement en partant du point source en suivant la direction qui minimise la distance jusqu'à atteindre le point d'arrivée. Une fois atteint, il est possible de remonter l'algorithme pour obtenir le plus court chemin entre ces deux points. Cela permet entre autres d'obtenir le plus court chemin entre deux objets de l'espace à l'aide d'algorithme de plus court chemin.

L'équation eikonale est également utilisée pour modéliser la propagation d'un front d'onde. Ainsi elle apparaît dans des domaines tels que l'électromagnétisme, la sismologie [2] ou encore le son [11]. Dans chacun de ces domaines interviennent des ondes et des fronts d'ondes qui se propagent dans différents milieux. Par exemple,

l'approximation généralement utilisée en imagerie sismique pour prédire les temps de parcours dans des milieux latéralement hétérogènes est l'équation eikonale. Par ailleurs la résolution de l'équation eikonale peut permettre de résoudre des problèmes liés à la propagation de la la lumière dans différents milieux. Ici f correspond à la fonction qui à tout $x \in \Omega$ lui associe son indice de réfraction $f(x)$.

L'équation eikonale a beaucoup d'applications en analyse d'images. Une des applications qui a beaucoup évolué ces dernières années [13] [14] est le "Shape from Shading" (SFS). C'est le procédé qui consiste à reconstituer une forme en trois dimensions à partir d'une seule image en noir et blanc de la surface. Ce procédé de reconstruction diffère de procédé similaire tel que la photométrie stéréo par le nombre d'information nécessaire à son application. Le problème du Shape From Shading est connu pour être un problème mal posé car la solution obtenue s'avère être non-unique dans le cas général pour diverses raisons : plusieurs surfaces peuvent avoir la même image, ambiguïté concave/convexe... Tout cela est approfondi et analysé dans la littérature [6]. Quoiqu'il en soit sous certaines conditions simplificatrices, la reconstitution reste réalisable car la surface est obtenue grâce à la résolution d'une équation aux dérivées partielles non-linéaire qui peut être posée sous la forme d'une équation eikonale. Celle-ci peut être résolue à l'aide d'un algorithme de Fast Marching dont on verra la structure à la section suivante. En traitement d'images, l'équation eikonale sert majoritairement à calculer des distances géodésiques (plus court chemin) [3] dans le cadre de la géométrie riemannienne. Ces distances géodésiques permettent de résoudre des problèmes comme la segmentation à l'aide de diagramme de Voronoï, de faire des échantillonages de points régulier selon ces distances ou encore faire le maillage d'un espace avec la triangulation de Delauney. Elle peut apparaître lors de la modélisation d'un squelette (axe médian), par exemple lors du calcul de la carte des distances qui associe à chaque pixel de l'image la distance au point obstacle le plus proche. Ces points obstacles peuvent être les points du contour de formes dans une image binaire. L'intérêt de ces distances euclidiennes est de pouvoir suivre les courbures d'un espace. Chose qui est possible puisqu'elles se basent sur la géométrie riemanienne. Il existe bien d'autres utilisations de l'équation eikonale en vision par ordinateur et l'article [12] en développe un certain nombre avec beaucoup plus de détails et de précisions.

2.4 Algorithmes

De nombreux algorithmes ont été développés depuis les années 1990 pour résoudre l'équation eikonale. Beaucoup de ces algorithmes s'inspirent des algorithmes utilisés beaucoup plus tôt pour les problèmes de plus court chemin. Ils se servent de la causalité fournie par l'interprétation physique et discrétisent généralement le domaine à l'aide d'un maillage ou d'une grille régulière afin de calculer la solution à chaque point discrétisé. On va en voir quelque uns dont les principaux car il existe de nombreuses dérivées de ces algorithmes.

2.4.1 Fast marching

Les méthodes de marche rapide (fast marching) sont les principaux schémas numériques pour résoudre l'équation Eikonale avec conditions aux limites sur des espaces euclidiens discrétisés ainsi que sur des surfaces courbes triangulées. Ces méthodes sont connues pour avoir une complexité de calcul en $O(N \log N)$ pour des espaces euclidiens discrétisés, avec N le nombre de points du maillage. Cependant plus récemment de nouvelles implémentations de la méthode de marche rapide voient le jour et atteignent une complexité en $O(N)$ à l'aide de systèmes de tas et de files d'attentes en désordre ingénieux. La méthode de marche rapide comprend un solveur numérique local et une étape de mise à jour du front et des points parcourus. Dans notre cas, on choisit d'utiliser un réseau de neurones pour résoudre et approximer au mieux l'équation localement. On se sert ainsi de la capacité d'apprentissage du réseau pour pouvoir réaliser ces calculs suivants le plus de conditions différentes.

Algorithm 1 Eikonal Estimation on Discretized Domain

Initialize:
$u(p) = 0$, Tag p as *visited*; $\forall p \in \mathfrak{s}$
$u(p) = \infty$, Tag p as *unvisited*; $\forall p \in \mathcal{S} \setminus \mathfrak{s}$
while there is a non-*visited* point **do**
    Denote the points adjacent to the newly *visited* points as $\mathcal{A}$
    for all $p \in \mathcal{A}$ **do**
        Estimate $u(p)$ based on *visited* points.
        Tag p as $wavefront$
    Tag the least distant $wavefront$ point as *visited*.
return u

FIGURE 1 – Algorithme marche rapide

L'algorithme quant à lui est similaire à l'algorithme de Dijkstra [5] et utilise le fait que l'information circule uniquement vers l'extérieur. Cela signifie que dès qu'un point du front est atteint, sa distance aux sources (qui est minimale par rapport aux points non atteints) ne se verra jamais modifiée. On valide ainsi les points de la distance la plus courte à la distance la plus grande. La distinction entre ces deux algorithmes est que l'algorithme de marche rapide remplace la mise à jour des distances du graphe de l'algorithme de Dijkstra par la résolution locale de l'équation eikonale.

2.4.2 Fast sweeping

L'idée principale de la méthode de balayage rapide (fast sweeping [19] [9]) est d'utiliser un schéma itératif utilisant des différences non linéaires en amont (upwind) et des itérations de la méthode de Gauss-Seidel avec ordre de balayage alterné. Contrairement à la méthode de marche rapide, la méthode de balayage rapide suit les courbes caractéristiques de manière parallèle ; c'est-à-dire que toutes les courbes caractéristiques sont divisées en un nombre fini de groupes selon leurs directions. Pour une équation aux dérivées partielles hyperboliques (ou pour un système d'équations aux dérivées partielles), les courbes caractéristiques (ou les

surfaces caractéristiques si le système a plus de deux dimensions) sont les courbes (ou les surfaces) sur lesquelles se propagent les perturbations. Pour être plus précis, pour une équation aux dérivées partielles du premier ordre (ce qui est le cas de l'équation eikonale), les caractéristiques sont des courbes le long desquelles l'équation aux dérivées partielles apparaît comme une équation différentielle ordinaire qui peut, dans les cas simples comme les équation linéaires, être résolue analytiquement. Ce qui simplifie le problème original. Chaque itération de Gauss-Seidel avec son ordre de balayage spécifique couvre simultanément un groupe de caractéristiques. La convergence est atteinte dès lors qu'aucune valeur n'est modifiée après un balayage. L'algorithme est optimal dans le sens où un nombre fini d'itérations est nécessaire. La complexité de l'algorithme est en $O(N)$ pour un total de N points de maillage. La précision est la même que toute autre méthode qui résout le même système d'équations discrétisées, a savoir en $O(h)$ avec h le pas de discrétisation.

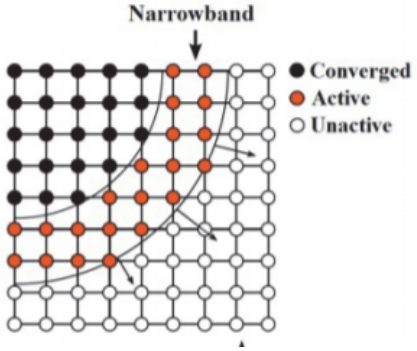

FIGURE 2 – Evolution du front d'onde avec Fast marching

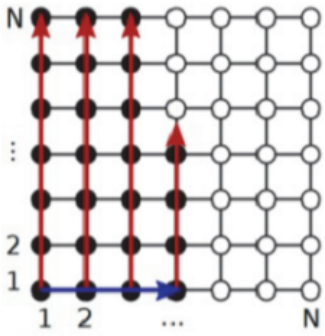

FIGURE 3 – Illustration du balayage avec Fast sweeping

La méthode de balayage rapide est plus précise et plus facile à mettre en œuvre que la méthode de marche rapide, mais le nombre réel de balayages requis pour la convergence dépend du problème à résoudre expérimentalement, il est observé que 2^D balayages sont requis pour un problème de dimension D.

2.4.3 Autres algorithmes

Il existe également d'autres procédés pour résoudre l'équation eikonale. Certains se basent sur d'autre algorithmes de plus court chemin tel que l'algorithme de Bellman–Ford qui est plus lent que l'algorithme de Dijkstra mais qui permet de travailler sur des poids négatifs. D'autres procédés utilisent des méthodes hybrides mettant en place à la fois la méthode de marche rapide et la méthode de balayage rapide. Elles sont basées sur la division du domaine en une collection de cellules rectangulaires disjointes à l'aide du fast marching suivie de l'application séquentielle de la méthode de balayage rapide sur les cellules. L'intérêt de cette décomposition est de diviser le problème en sous-problèmes.

Une alternative au schéma de fast sweeping classique avec la méthode de Gauss-Seidel est l'utilisation du schéma de Lax-Friedrichs [17]. Celui-ci permet

de résoudre numériquement des équations aux dérivées partielles de type hyperbolique, basée sur la méthode des différences finies. Cette technique repose sur l'utilisation de différence finie décentrée en temps et centrée en espace. On peut considérer le schéma de Lax–Friedrichs comme une alternative au schéma de Godunov mais nécessite d'ajouter de la viscosité artificielle à l'équation d'origine. Cette méthode permet de résoudre des équations d'Hamilton-Jacobi plus générales et donc l'équation eikonale qui est une équation d'Hamilton-Jacobi de la forme :

$$H(x, \nabla x) = |\nabla u(x)|^2 - \frac{1}{f^2(x)} = 0, \ \forall x \in \Omega$$

3 Résolution de l'équation eikonale : marche rapide

L'algorithme de marche rapide énoncé à la partie 2,4,1 est la base de tout ce qui va suivre. On commence d'abord par implémenter l'algorithme en utilisant deux solveurs numériques différents dont on verra leurs formalisations. Cela permet de comprendre et de faire une première manipulation de l'algorithme de marche rapide. Obtenir ces résultats sera toujours utile pour pouvoir étendre le problème en remplaçant le solveur local numérique par un réseau de neurones.

3.1 Avec solveur local numérique et carte de potentiel

Pour cette première implémentation, l'intégralité du code est réalisée sous Matlab. Comme expliqué précédemment, le plus important dans l'algorithme de marche rapide est d'avoir un solveur local efficace et précis. Ici on va en considérer deux différents : celui utilisant les 4 plus proches voisins et celui utilisant les 8 plus proches voisins. Ils sont également les plus simples. Mais malgré leur simplicité vis à vis des autres solveurs, ils permettent de réaliser des approximations de gradient à un point du front donné dans un espace pondéré (f variable) assez facilement. A l'inverse avec un solveur local crée par deep learning, il est plus facile d'intégrer un plus grand nombre de voisins pour une meilleure précision. Mais il est bien plus compliqué d'y intégrer des poids.

3.1.1 Solveur utilisant les quatre plus proches voisins

Pour obtenir le solveur local, il faut repartir de l'équation eikonale :

$$\begin{aligned} |\nabla u(x)| &= \frac{1}{f(x)} \\ u(x) &= g(x) \quad \textit{pour } x \in \partial\Omega \end{aligned}$$

On cherche à approximer numériquement la fonction $u(x)$ de l'équation eikonale. On pose $u_{ij} = u(ih, jh)$ avec h le pas de la discrétisation, qui est le même

selon l'axe des x et l'axe des y. L'équation peut se réécrire sous cette forme :

$$u_x^2 + u_y^2 = \frac{1}{f(x,y)^2} = |\nabla u(x,y)|^2$$

Et ainsi on peut approximer le résultat à l'aide de la méthode des différences finies du premier ordre en amont (up-wind).

$$|\nabla u_{i,j}|^2 \approx max(D_{i,j}^{-x}u, -D_{i,j}^{+x}u, 0)^2 + max(D_{i,j}^{-y}u, -D_{i,j}^{+y}u, 0)^2$$

Avec $D_{i,j}^{-x}, D_{i,j}^{+x}, D_{i,j}^{-y}, D_{i,j}^{+y}$ les opérateurs de différences finies avant et arrière au point (i,j) selon l'axe x ou l'axe y tels que :

$$D_{i,j}^{-x}u = \frac{u_{i,j} - u_{i-1,j}}{h} \quad et \quad D_{i,j}^{+x}u = \frac{u_{i+1,j} - u_{i,j}}{h}$$
$$D_{i,j}^{-y}u = \frac{u_{i,j} - u_{i,j-1}}{h} \quad et \quad D_{i,j}^{+y}u = \frac{u_{i,j+1} - u_{i,j}}{h}$$

Ce qui revient à résoudre l'équation du second degré suivante :

$$max(D_{i,j}^{-x}u, -D_{i,j}^{+x}u, 0)^2 + max(D_{i,j}^{-y}u, -D_{i,j}^{+y}u, 0)^2 = \frac{1}{f_{i,j}^2}$$

Avec $f_{i,j} = f(ih, jh)$ connu et à valeurs positives. L'algorithme permet d'approximer la distance minimale d'un point de la grille à l'ensemble des points initiaux. La grille est représentée par une carte de potentiel contenant les poids de chaque point. Ces poids correspondent aux valeurs prises par la fonction f.

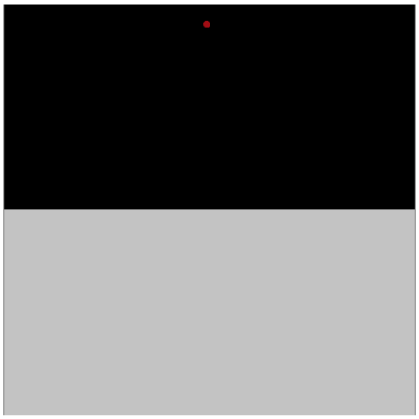

FIGURE 4 – Carte de potentiel initiale (binaire), le point rouge étant le point initial. la partie sombre correspond aux poids forts alors que la partie grise correspond aux poids faibles

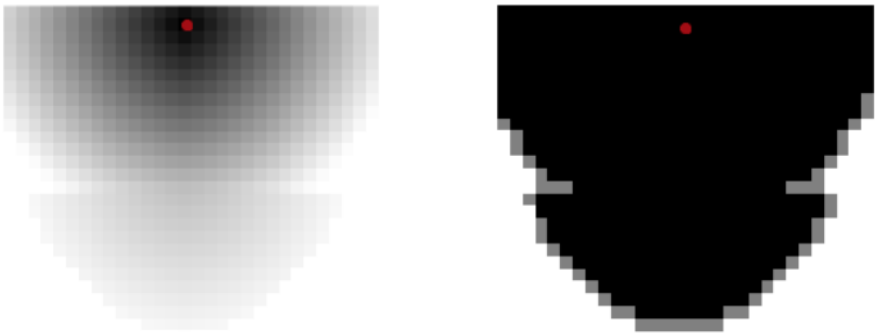

FIGURE 5 – A gauche : évolution de la carte des distances. A droite : évolution du front, en noir les points visités, en gris le front d'onde et en blanc les points non visités

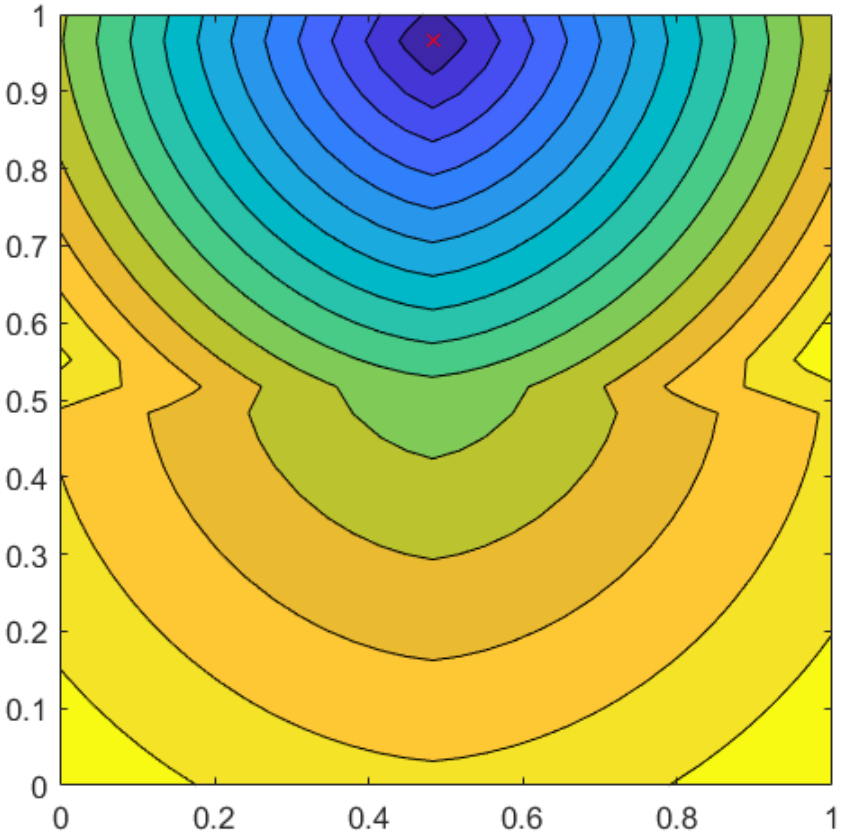

FIGURE 6 – Carte des distances avec isocontours. Le point initial en rouge

Ici le point de départ est dans une zone où les poids sont importants (zone noire sur la carte de potentiel). Ainsi la vitesse est la même partout dans cette zone, et les distances sont régulières et proportionnelles au pas de discrétisation. A l'inverse la zone grise correspond à des poids faibles, la vitesse est plus importante dans cette zone, c'est pourquoi les isocontours sont plus espacés dans la partie

inférieure de l'image.

L'algorithme fonctionne également avec plusieurs points de départs, on peut donc utiliser une droite comme ensemble de départ. Pour mieux voir le résultat, la carte de potentiel choisie est une carte uniforme. Ainsi les seules distances intervenant dans cet exemple sont celles entre les points de la grille, ce qui va permettre d'avoir des isocontours plus réguliers.

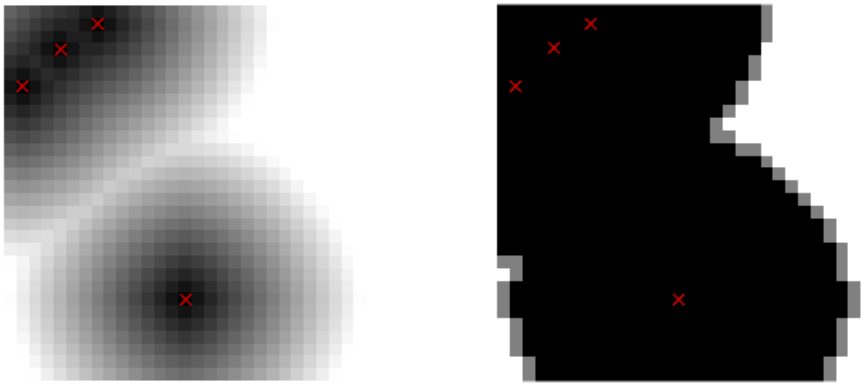

FIGURE 7 – A gauche : évolution de la carte des distances. A droite : évolution du front, en noir les points visités, en gris le front d'onde et en blanc les points non visités

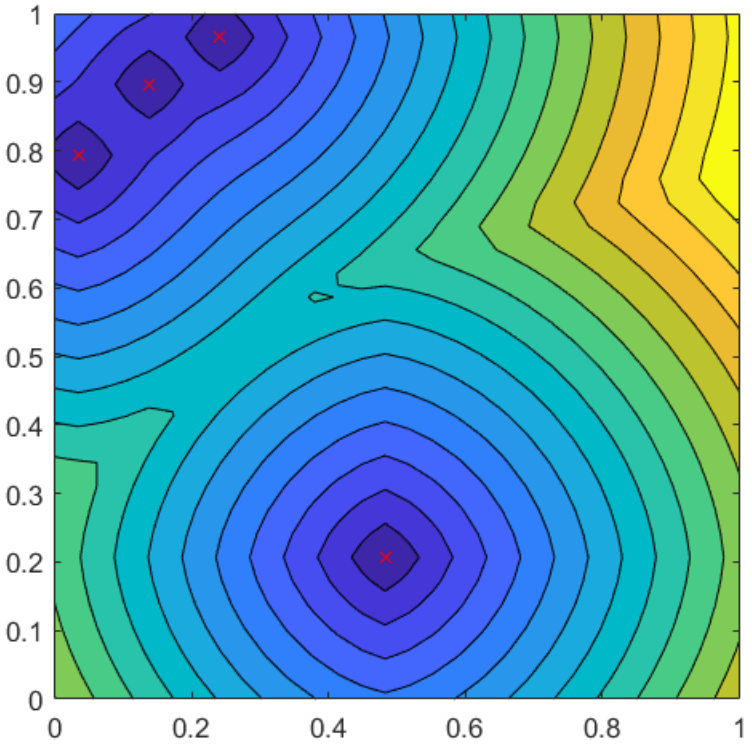

FIGURE 8 – Carte des distances avec isocontours. Les points initiaux en rouge

L'objectif est d'avoir des isocontours complètement lisses. Or d'après la figure 6 et la figure 8 on peut voir que le résultat est impacté par notre solveur qui ne calcule qu'en fonction des quatre plus proches voisins. On peut en effet voir la forme quadratique des lignes de niveaux. Il est possible d'utiliser un solveur plus performant mais cela impliquerait d'avoir des calculs plus complexes.

3.1.2 Solveur utilisant les huit plus proches voisins

Le but de ce projet n'est pas d'obtenir un solveur local (numérique) performant, cependant il reste intéressant de voir comment évolue le résultat en fonction de la qualité du solveur et il sera utile pour comparer nos résultats par la suite. Pour l'améliorer il est possible d'utiliser un ordre de discrétisation supérieur ou d'augmenter le nombre de voisins tout en continuant à propager l'information vers l'extérieur pour ne pas se servir d'informations fausses ou obsolètes.

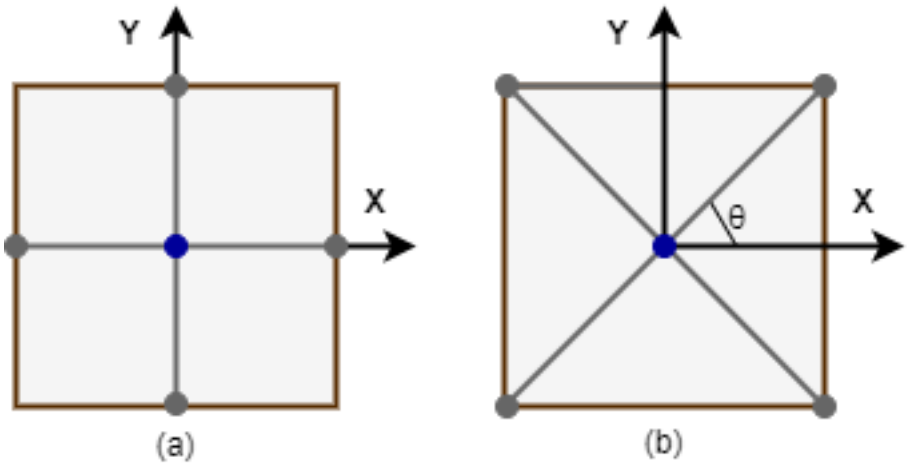

FIGURE 9 – Rotation de repère de (a) vers (b)

Pour un voisinage de 8 points [8] on considère deux repères. Le premier est celui des quatre plus proches voisins le long des axes x et y. Le deuxième étant celui qui passe par les diagonales, ce repère est le même que le premier après une rotation d'angle θ ($= \frac{\pi}{4}$ si même discrétisation sur X et Y). Ainsi les coordonnées d'un point (x_D, y_D) du nouveau système de coordonnées s'écrivent

$$x_D = x\cos\theta + y\sin\theta$$
$$y_D = -x\sin\theta + y\cos\theta$$

D'après la règle de la chaîne, les dérivées partielles de u s'écrivent

$$U_x = \frac{\partial u}{\partial x_D}\cos\theta - \frac{\partial u}{\partial y_D}\sin\theta$$
$$U_y = \frac{\partial u}{\partial x_D}\sin\theta + \frac{\partial u}{\partial y_D}\cos\theta$$

En mettant les deux équations au carré, on retrouve l'équation eikonale

$$U_{x_D}^2 + U_{y_D}^2 = |\nabla u(x_D, y_D)|^2 = \frac{1}{f(x_D, y_D)^2}$$

Ainsi pour résoudre l'équation eikonale selon les deux systèmes de coordonnées, il faut les résoudre séparément. Dans notre cas la discrétisation est régulière et égale à h sur X et Y, mais il faut prendre en compte la modification de la distance entre le centre du repère et le point du nouveau repère. Les calculs restent les mêmes mais pour le repère tourné de $\frac{\pi}{2}$, il faut utiliser un pas $h_D = \sqrt{2} * h$. On obtient donc deux solutions différentes pour la distance en ce point. Il suffit par la suite de conserver la plus petite valeur des deux. Cependant cette méthode ne fonctionne qu'avec une grille régulière et isotrope.

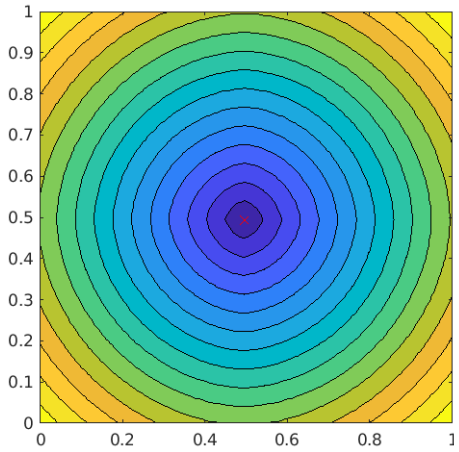

FIGURE 10 – Avec solveur à 4 voisins

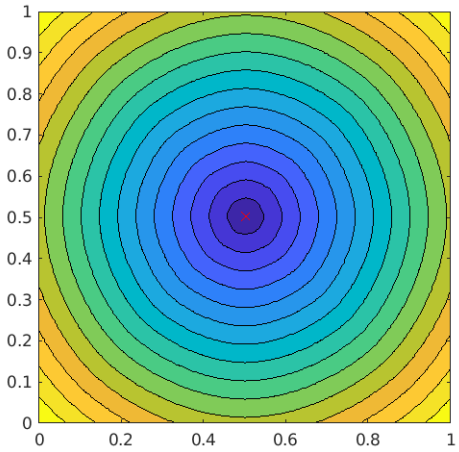

FIGURE 11 – Avec solveur à 8 voisins

3.2 Solveur local par Deep learning

Le but de cette section est d'améliorer la précision et l'efficacité du solveur. On a vu précédemment qu'augmenter le nombre de voisins lors de l'approximation de l'équation permet d'obtenir de meilleurs résultats et de lisser les isocontours mais en complexifiant les calculs. C'est ici que le réseau de neurones prend tout son sens puisqu'il va apprendre de lui même à résoudre l'équation sans devoir traiter tous les cas possibles manuellement.

3.2.1 Mais qu'est-ce que le deep learning ?

Par définition, le deep learning [7] signifie apprentissage profond, qui est une branche du machine learning qui désigne l'apprentissage automatique, qui est lui-même une branche de l'intelligence artificielle (IA). Le principe est de rendre possible pour une machine l'apprentissage par elle-même contrairement à la programmation plus classique où tout est déterministe.

Le deep learning se base sur des réseaux de neurones artificiels en s'inspirant du fonctionnement du cerveau humain. Un neurone artificiel n'est rien d'autre qu'une opération mathématique relativement simple. La complexité repose avant tout dans l'interconnexion de plusieurs neurones. L'histoire du deep learning remonte jusqu'en 1943, lorsque Walter Pitts et Warren McCulloch ont crée un modèle informatique basé sur les réseaux neuronaux du cerveau humain. Ils ont utilisé une combinaison d'algorithmes et de mathématiques qu'ils ont nommé "treshold logic" (logique par seuil) pour imiter le processus de pensée. Peu de temps après, en 1956, Frank Rosenblatt a mis au point le perceptron qui a suscité un grand engouement. Les scientifiques misaient alors énormément sur les réseaux de neurones mais les résultats ont fini par décevoir. En effet il a fallu attendre ces dernières années pour disposer d'une puissance de calcul et de capacités de stockage de données suffisamment évoluées pour permettre au deep learning d'engendrer de nouvelles technologies prometteuses. Les réseaux de neurones sont composés de plusieurs couches afin de décomposer l'information brute fournie à l'entrée du réseau. Par exemple en traitement d'image l'entrée peut être une matrice de pixels, les couches inférieures peuvent servir à identifier les bords ou les lignes tandis que les couches supérieures peuvent identifier des formes plus pertinentes pour un humain telles que les chiffres, les lettres ou les yeux d'un visage. Un réseau de neurones peut apprendre de lui même à régler ses neurones pour optimiser les résultats. Bien sûr cela n'empêche pas le besoin de réglages manuels. En effet faire varier le nombre de couches ou même la taille des couches permet d'aboutir à différents degrés d'abstraction et donc des performances différentes.

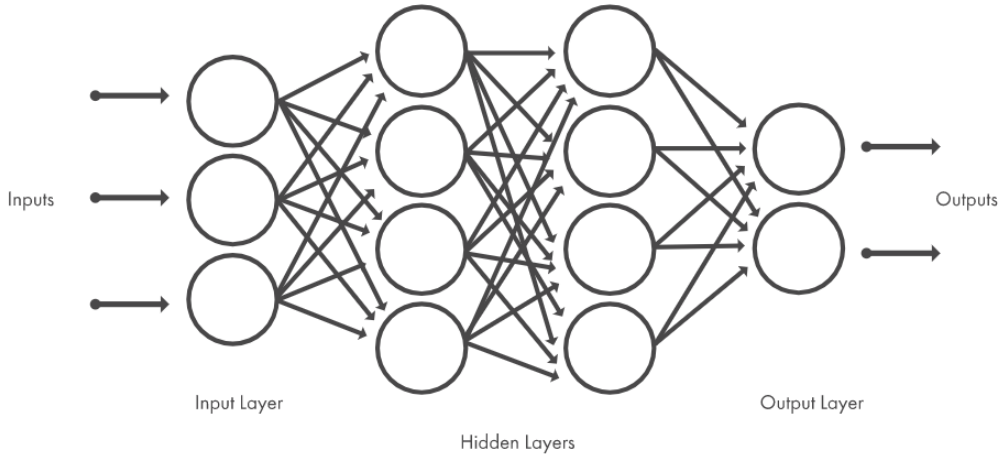

FIGURE 12 – Structure d'un réseau de neurones

Il existe différents types d'apprentissages. Le plus courant est l'apprentissage supervisé. Dans ce cas là l'entraînement des modèles s'effectue à l'aide de vastes ensembles de données labellisées et d'architectures de réseaux de neurones qui apprennent directement depuis les données, sans avoir à effectuer une extraction manuelle. Le réseau prend en entrée les données annotées pour ensuite comparer cette réponse aux bonnes réponses indiquées par l'annotation. Si les réponses correspondent, le réseau garde cette réussite en mémoire et s'en servira plus tard. Dans le cas contraire, le réseau prend note de son erreur et ajuste le poids placé sur les différents neurones pour corriger son erreur. Il est donc très important d'avoir une base de donnée représentant le plus de cas possible afin d'avoir un réseau performant.

A l'inverse l'apprentissage non supervisé prend en entrée des données non étiquetées. Les réseaux de neurones doivent reconnaître des patterns au sein des ensembles de données pour apprendre par eux-mêmes quels biais activer. En général, des systèmes d'apprentissage non supervisé permettent d'exécuter des tâches plus complexes que les systèmes d'apprentissage supervisé, mais ils peuvent aussi être plus imprévisibles. Même si un système d'IA d'apprentissage non supervisé parvient tout seul, par exemple, à faire le tri entre des chats et des chiens, il peut aussi ajouter des catégories inattendues et non désirées, et classer des races inhabituelles, introduisant plus de bruit que d'ordre. A la frontière entre l'apprentissage supervisé et l'apprentissage non supervisé réside l'apprentissage semi-supervisé. C'est une méthode d'apprentissage qui utilise un ensemble de données étiquetées et non étiquetées. Il a été démontré que l'utilisation de données non étiquetées, en combinaison avec des données étiquetées, permet d'améliorer significativement la qualité de l'apprentissage. Un exemple parlant d'apprentissage semi-supervisé est celui du professeur qui montre à ses élèves comment résoudre un exercice afin qu'ils puissent par la suite résoudre les autres exercices portant sur le même thème. Un des intérêts de l'apprentissage semi-supervisé est de n'avoir besoin que de peu de données annotées. L'annotation nécessite souvent l'intervention d'experts ce qui implique un coût important et également un temps d'étiquetage d'autant plus grand que la base de donnée est grande (nécessaire pour l'apprentissage supervisé).

Il existe d'autres méthodes d'apprentissage mais la troisième méthode la plus utilisée est la méthode d'apprentissage par renforcement. Avec l'apprentissage par renforcement, l'objectif est de pondérer le réseau (concevoir une politique) pour qu'il exécute des actions qui minimisent le coût à long terme. A chaque instant, l'agent (robot, IA d'un jeu vidéo etc...) effectue une action et l'environnement génère une observation et un coût instantané, selon certaines règles (généralement inconnues). Les règles et le coût à long terme ne peuvent généralement être estimés. À tout moment, l'agent décide d'explorer de nouvelles actions pour découvrir leurs coûts ou d'exploiter les apprentissages antérieurs pour procéder plus rapidement. Le but étant de maximiser la somme des récompenses au cours du temps.

Dans ce projet on suit la méthode de l'apprentissage supervisé puisque la génération de données labellisées est naïve et peu coûteuse. De plus l'apprentissage supervisé permet de minimiser l'erreur entre la sortie du réseau et le résultat désiré, ce qui s'inscrit parfaitement dans le cadre de la résolution de l'équation eikonale. Il faut par ailleurs noter qu'un réseau peut posséder des couches linéaires comme des couches non linéaires. La fonction d'activation a pour but d'introduire la non-

linéarité dans le réseau (même s'il en existe des linéaires). Cela permet de modéliser une variable de réponse (variable cible, étiquette de classe ou de score) qui varie de manière non linéaire. D'autant plus que sans fonction d'activation non linéaire, le réseau se comporterait quelques soit son nombre de couches comme un perceptron monocouche par linéarité.

3.2.2 Entraînement et structure du réseau

Le réseau de neurones utilisé est un perceptron multicouche assez simple composé de quatre couches linéaires qui sont chacune suivie d'une fonction Relu. Il prend en entrée tous les points situés à au plus $2h$ de distance du point courant. De plus puisque la grille est uniformément espacée, il n'est pas utile de passer les coordonnées des points dans le réseau. C'est le pas h qui englobe l'information spatiale à la place.

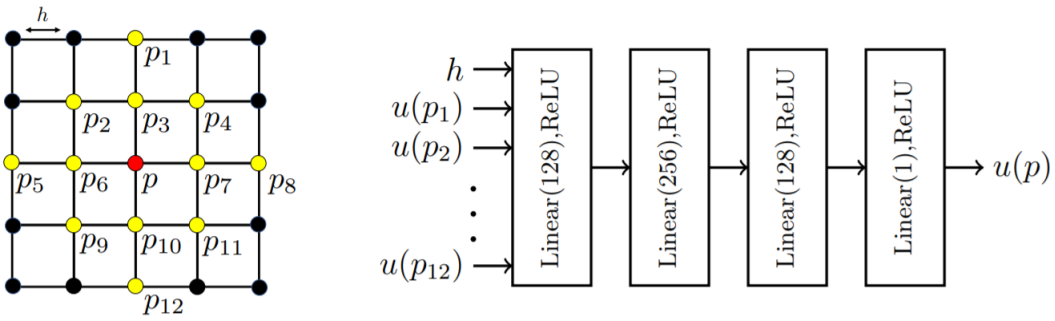

FIGURE 13 – Architecture du réseau

Pour l'entraînement on donne au réseau 10000 exemples correspondants à diverses configurations de sources et diverses configurations de patchs. Il faut utiliser des formes différentes comme des cercles, des lignes, des quadrilatères, des courbes etc... car le réseau a besoin de s'adapter à chaque situation. Il faut également fournir la valeur que doit retourner le réseau pour chaque patch afin de la comparer avec la valeur en sortie du réseau. Cela est fait à l'aide d'une fonction de perte au sens des moindres carrées. On cherche ainsi à minimiser la moyenne des erreurs quadratiques entre la sortie et l'étiquette. On utilise la distance de vérité terrain pour étiqueter les patch et ici la distance de vérité terrain u_{gt} entre deux points est égale à la distance euclidienne entre ces deux points. Ensuite pour simuler la propagation du front d'onde dans divers scénarios, nous utilisons la stratégie suivante. Pour un point q dans le patch et différent du point central :

$$if \quad u_{gt}(q) < u_{gt}(p) \quad then \quad u(q) = u_{gt}(q)$$

Cela signifie que si la distance de vérité terrain du point q est plus faible que celle du point p, sachant que l'algorithme de Fast Marching propage l'information vers l'extérieur, le point q est déjà atteint et donc connu. Ainsi sa valeur devrait correspondre à celle de vérité terrain. A l'inverse les points qui ont une distance de vérité terrain supérieure à celle du point p sont considérés comme non visités. On fixe ainsi leur valeur à $u_{gt}(p) + 2h$

$$if \quad u_{gt}(q) >= u_{gt}(p) \quad then \quad u(q) = u_{gt}(p) + 2h$$

La grande variété d'information est indigeste pour le réseau de neurones, il faut donc simplifier les entrées. On dimensionne donc nos données vers $[0; 1]$ en soustrayant ces données par la distance minimale du patch et en divisant ensuite par la distance maximale moins la distance minimale. Et avant de procéder à la phase de calcul de perte, on redimensionne nos données.

3.2.3 Résultats et comparaison

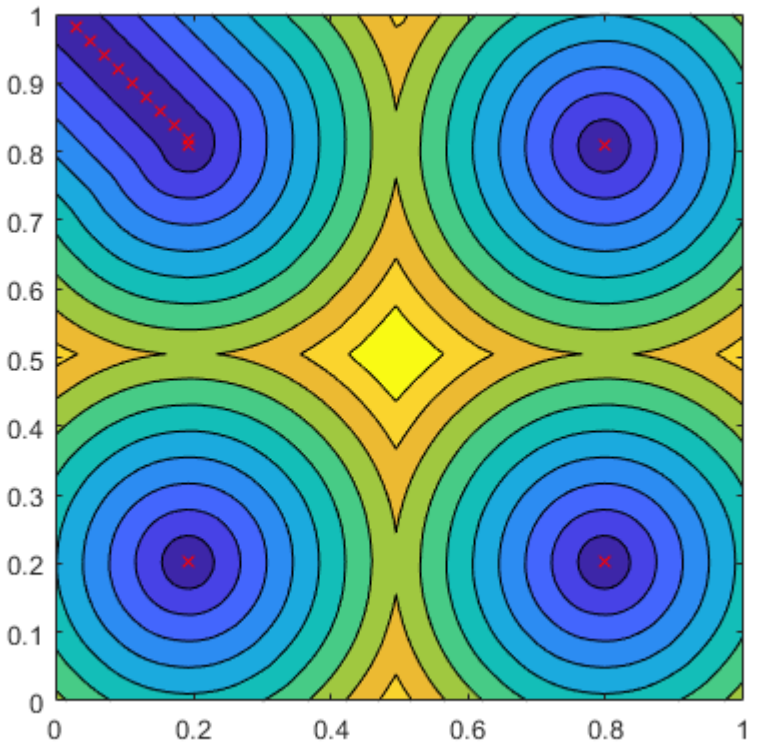

FIGURE 14 – Résultat du Deep Eikonal Solver

Maintenant que nous avons tout le nécessaire, il faut tester le programme en utilisant diverses formes. Les résultats suivants sont obtenus pour une grille de taille 100×100 sauf la figure 19 qui est obtenue pour une grille de taille 200×200. Ce choix vient du fait que le but était d'avoir un cercle le plus régulier possible ce qui est difficile à réaliser quand tout est discrétisé. C'est pourquoi le cercle, malgré le fait qu'il soit le plus régulier possible, aura toujours quelques irrégularités qui seront alors moins visibles sur un domaine plus large afin de mieux voir les isocontours dans leur ensemble. Néanmoins il est quand même possible de voir ces irrégularités sur les isocontours les plus proches. C'est également ce qui est mis en avant figure 15 et figure 16. Les lignes de niveaux sont impactés par l'irrégularité de la disposition des sources. On peut de nouveau remarquer que plus l'on s'éloigne

des sources, plus les courbes sont lisses. A l'inverse lorsque que les sources sont disposées de manière régulière figure 14, figure 17 et figure 18, les lignes de niveaux sont aussitôt bien plus lisses.

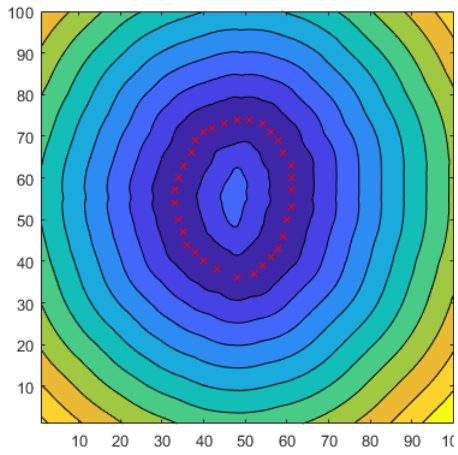

FIGURE 15 – Isocontours cercle irrégulier

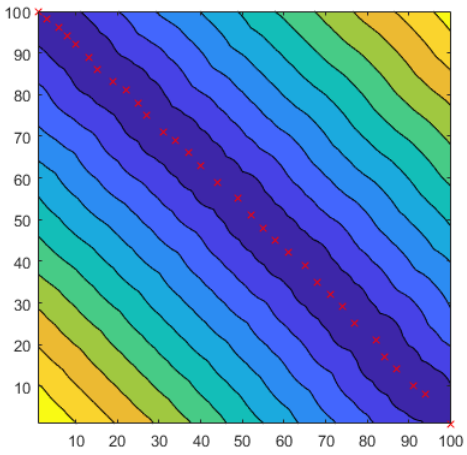

FIGURE 16 – Isocontours diagonale irrégulière

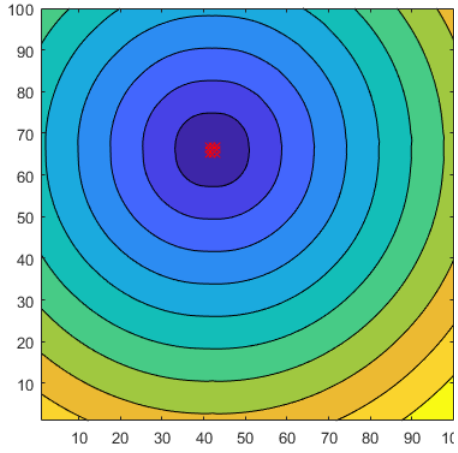

FIGURE 17 – Isocontours petit carré régulier

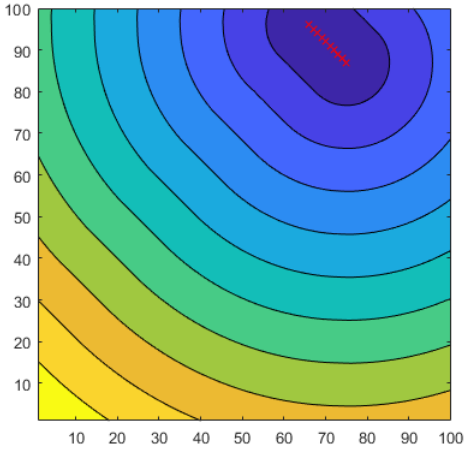

FIGURE 18 – Isocontours petit trait régulier

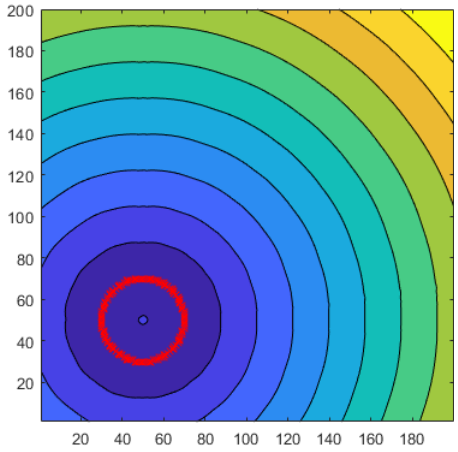

FIGURE 19 – Isocontours cercle (presque) régulier

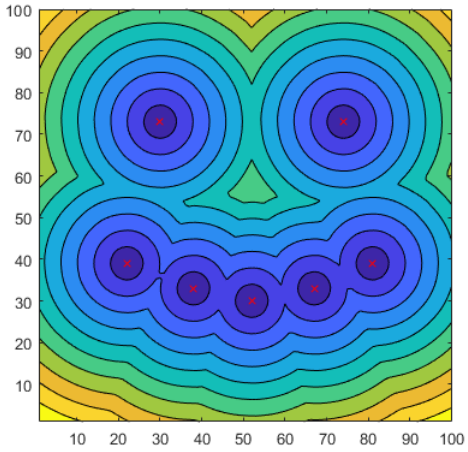

FIGURE 20 – Isocontours sourire étrange

En comparant les résultats précédents avec ceux de cette section et avec la distance de vérité terrain, on obtient les résultats figure 21, figure 22 et figure 23. Il est visuellement évident que les résultats découlant du deep eikonal solver sont bien plus proche de la distance de vérité terrain que ne le sont ceux découlant des solveurs numériques vus aux sections précédentes notamment sur les isocontours circulaires où les courbes de vérité terrain et celle du deep eikonal solver se chevauchent.

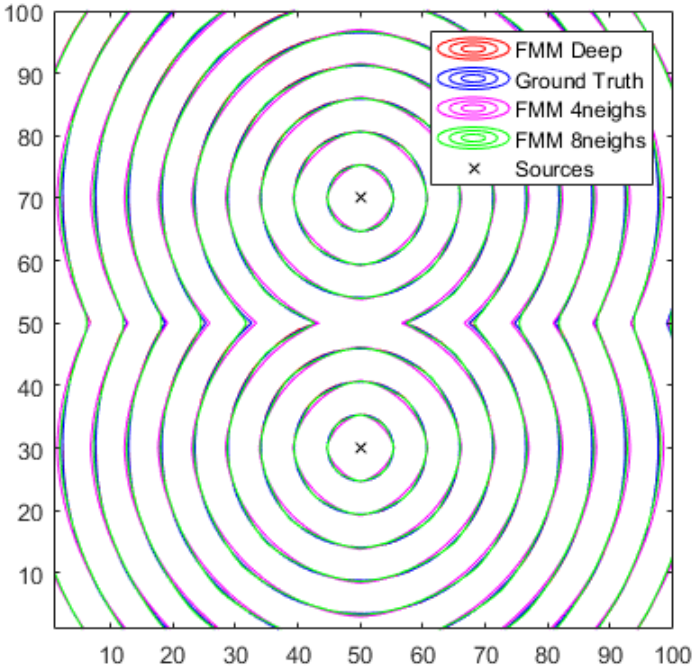

FIGURE 21 – Résultats superposés

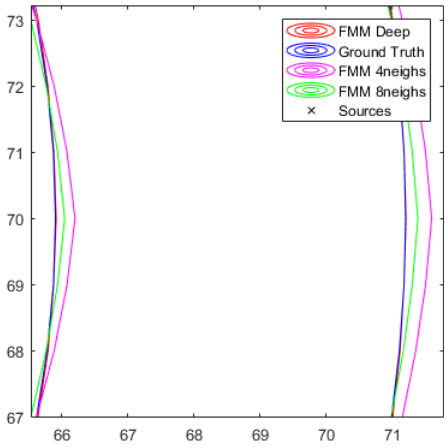

FIGURE 22 – Comparaison zoom iso-contour (cercle)

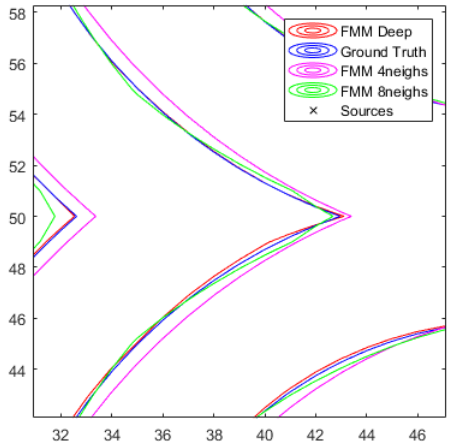

FIGURE 23 – Comparaison zoom iso-contour (pointe)

Pour vérifier cela, il est pertinent de comparer l'erreur moyenne entre les différents solveurs (4 voisins, 8 voisins et deep solver) et la distance de vérité terrain en fonction du pas de discrétisation h. Les résultats présentés sur les figures précédentes ont toutes été réalisées pour $h = 1$. En effet deux deep solver différents ont été générés : un qui dispose d'une base de donnée construite uniquement avec $h = 1$ et un autre qui dispose d'une base de données construite pour h variant de 1 à 2. Les base de données sont de même taille, cela implique donc que le premier deep eikonal solveur est légèrement plus précis car il considère moins de cas. On test les modèles en fonction de plusieurs pas de discrétisation h dans les mêmes conditions que la figure 21.

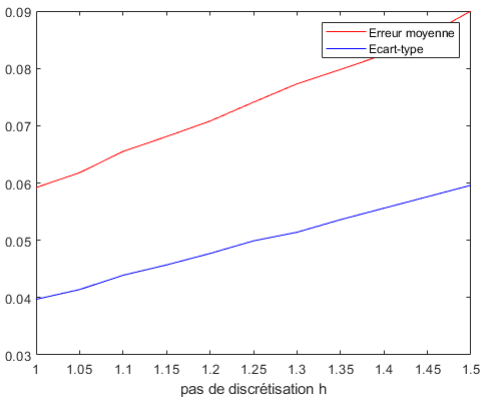

FIGURE 24 – Erreur moyenne et écart-type deep eikonal solver

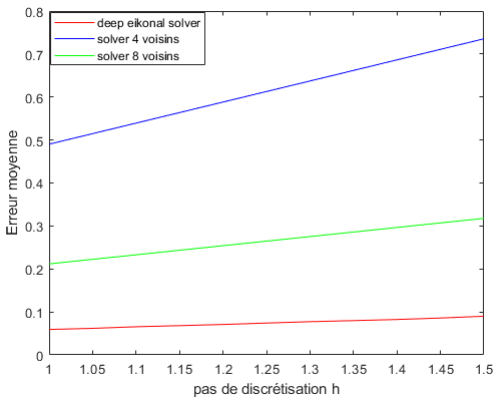

FIGURE 25 – Comparaison erreurs moyennes solveurs

A l'aide de la figure 24, on peut voir que la précision du deep eikonal solver est d'environ 7.10^{-2} et l'écart-type est du même ordre mais légèrement plus petit avec ces conditions. La précision du réseau diminue plus les points dont on calcule la distance sont loin des sources car l'erreur (aussi petite soit-elle) se propage : le

réseau calcule des distances en fonction de patch dont les valeurs ont été calculées auparavant. La figure 25 quant à elle montre que le deep eikonal solver est plus précis à une puissance de 10 près que le solveur numérique à 4 voisins et le solveur numérique à 8 voisins.

4 Intégration de librairies

Une des motivations de ce projet est d'aboutir à une application précise mais également performante. Beaucoup d'éléments variés interviennent dans le programme et c'est pourquoi utiliser seulement Matlab n'est pas la bonne solution. En effet si le problème est à grande échelle, il y aura énormément de données à gérer notamment celles contenues dans le front. Front dans lequel on doit pouvoir récupérer le point de distance minimale, insérer des points et mettre à jour des distances. Par ailleurs même si Matlab possède quelques bibliothèques permettant de faire du deep learning, il existe des solutions bien plus efficaces et plus optimisées que celles fournies par Matlab. L'enjeu ici est donc de garder l'interface Matlab et de s'en servir pour appeler du code écrit dans un langage de bas-niveau bien plus pertinent vis à vis du problème donné.

4.1 Intérêts

A chaque itération il faut calculer la solution à chaque point au plus 4 fois et il faut chercher le point du front d'onde de distance minimale aux points initiaux. Ceci peut être réalisé par bien des méthodes différentes et c'est là, outre le solveur, le point le plus important de l'algorithme car c'est celui qui garantit l'efficacité et la rapidité d'exécution du code. La méthode la plus répandue lors de l'utilisation d'un algorithme de marche rapide est l'utilisation d'un tas binaire minimal (la plus petite valeur se situe à la racine). La propriété principale que doit respecter la structure est que la valeur de n'importe quel noeud doit être inférieure à celle de ses fils. Dans les faits on représente le tas binaire séquentiellement dans un tableau où l'on stocke chaque noeud à une position k et ses deux fils aux positions $2k$ et $2k+1$. Ainsi pour un noeud en position k son père est en position $\lfloor \frac{k}{2} \rfloor$. L'implémentation du tas binaire et des opérations nécessaires pour le fast marching suivent les algorithmes de la publication [4].

Dans le cas de l'algorithme de fast marching, il est essentiel de savoir à quel point de la grille correspond chaque distance stockée dans le tas. Ainsi on stocke à la fois les coordonnées et la distance minimale. L'algorithme nécessite de pouvoir faire trois opérations différentes sur le tas. 1) **Insert** : ajouter un nouveau point au front. 2) **Update** : mettre à jour un point déjà présent dans le front avec sa nouvelle distance. 3) **Min** : retire l'élément de distance minimale du tas. Chacune de ces opérations a une complexité dans le pire des cas en $O(ln(N))$ en notant N la taille du front. Ainsi si on note n le nombre de points du maillage alors le coût total de l'algorithme serait en $O(nln(n))$. Cette structure est déjà utilisée pour la partie réalisée uniquement en Matlab, néanmoins Matlab n'est absolument pas optimisé pour la gestion de ce type de données comparé à d'autres langages de programmations plus classiques.

4.2 Installation et utilisation

Dans la version finale du projet, le codage de l'algorithme est réalisé en C++ pour les raisons vues précédemment, et est intégré dans du code Matlab à l'aide de la librairie MEX . Pour se faire il faut écrire une fonction interface MEX en C++ qui permet de faire tourner l'algorithme de marche rapide. C'est cette fonction qui génère le fichier MEX appelable dans matlab. Par ailleurs on utilise le réseau créé avec PyTorch dans le code C++ à l'aide de la librairie LibTorch. Celle-ci permet de manipuler des tenseurs Torch propre à Python afin de les passer dans un réseau de neurones enregistré en amont dans notre cas. Enfin pour compiler et générer le tout, il est recommandé d'utiliser CMake même s'il est possible d'utiliser d'autres systèmes de build tels que Visual Studio, QMake...etc. La subtilité réside dans le fait qu'on utilise libtorch dans l'interface mex, il faut donc bien intégrer ces deux librairies au projet. CMake facilite le tout car il permet également de générer automatiquement le fichier MEX sans passer par matlab à l'aide de la fonction "matlab_add_mex".

Mode d'emploi pour utiliser l'application :

— 1) Commencer par télécharger et unzip libtorch sans cuda (version release pour windows) à partir de pytorch.org. Il faut bien se souvenir du chemin pour accéder au dossier car il sera nécessaire pour compiler

— 2) Modifier la ligne 9 du CMakeLists.txt avec le chemin de la racine du dossier matlab. Si besoin, taper "matlabroot" dans matlab permet de l'obtenir.

— 3) Dans la racine du projet faire :

mkdir build
cd build
cmake $-DCMAKE\_PREFIX\_PATH = [PATHTOLIBTORCH]$ ..
make (sur unix)
cmake –build . -config Release (sur windows, et il faut bien utiliser la version release car c'est en mode debug par défaut)

— 4) Ouvrir le dossier du projet avec matlab et s'assurer que tout est bien dans le path. Sous windows, le fichier MEX est dans le dossier build/Release.

— 5) Tout est bon il ne reste plus qu'à appeler la fonction (il y a un exemple d'utilisation dans le projet)

5 Conclusion

L'approche utilisée fait le lien entre plusieurs domaines de l'informatique. Elle se sert de la puissance d'approximation des réseaux de neurones pour atteindre de meilleures performances et une meilleure compréhension d'algorithmes de calculs.

L'implémentation du deep eikonal solver présentée dans ce document est une première approche à la résolution de problèmes à plus grande échelles et dans des conditions différentes. En effet la résolution de l'équation eikonale permet de simuler la propagation d'un front d'onde, il est donc possible que le milieu de propagation soit hétérogène. Il est également possible d'étendre le problème à un domaine 3D comme présenté par Lichtenstein, Pai et Kimmel [10]. La méthodologie reste la même, à savoir, entraîner le réseau de neurones à partir d'une base de données variée et grande. Et programmer l'algorithme à l'aide d'un langage approprié pour enfin se servir de différentes librairies pour appeler le réseau de neurones. L'apprentissage du réseau reste la partie qui va le plus différer entre ce projet et d'autres approfondissements du sujet.

L'étape suivante de ce projet serait d'approfondir le travail réalisé et d'implémenter le deep eikonal solver pour f variable, ce qui correspond au cas du milieu hétérogène. Bien que cette nouvelle étape ressemble à ce qui a été vu auparavant, elle diffère de cette dernière car la distance de vérité terrain est impactée par les poids f associés à chaque point de la grille. La nouvelle distance de vérité terrain devient par le biais de ces poids, une distance de vérité terrain locale. Par ailleurs lorsque f est non variable, la distance employée est une distance euclidienne classique où chaque point du maillage possède la même métrique. L'interpolation de ces métriques se fait donc automatiquement. A l'inverse pour f variable, il est nécessaire d'interpoler f pour calculer les géodésiques ce qui est assez délicat car le choix de l'interpolation va dépendre du problème à résoudre. Mais faute de temps, la suite de ce projet n'a pas pu être réalisée.

Références

[1] S. Ahmed et al. "A Third Order Accurate Fast Marching Method for the Eikonal Equation in Two Dimensions". In : *SIAM J. Sci. Comput.* 33 (2011), p. 2402-2420.

[2] Nidhal Belayouni. "New efficient 2D and 3D modeling algorithms to compute travel times, take-off angles and amplitudes". Theses. Ecole Nationale Supérieure des Mines de Paris, avr. 2013. URL : `https://pastel.archives-ouvertes.fr/pastel-00871200`.

[3] D Chen et Laurent Cohen. "From Active Contours to Minimal Geodesic Paths : New Solutions to Active Contours Problems by Eikonal Equations". In : sept. 2019. ISBN : 9780444641403. DOI : `10.1016/bs.hna.2019.07.009`.

[4] T. H. Cormen et al. "Introduction to Algorithms, Second Edition". In : 2001.

[5] E. Dijkstra. “A note on two problems in connexion with graphs”. In : *Numerische Mathematik* 1 (1959), p. 269-271.

[6] Jean-Denis Durou, Maurizio Falcone et Manuela Sagona. “Numerical methods for shape-from-shading : A new survey with benchmarks”. In : *Computer Vision and Image Understanding* 109.1 (2008), p. 22-43. issn : 1077-3142. doi : https://doi.org/10.1016/j.cviu.2007.09.003. url : http://www.sciencedirect.com/science/article/pii/S1077314207001361.

[7] I. Goodfellow, Yoshua Bengio et Aaron C. Courville. “Deep Learning”. In : *Nature* 521 (2015), p. 436-444.

[8] M. Hassouna et A. Farag. “MultiStencils Fast Marching Methods : A Highly Accurate Solution to the Eikonal Equation on Cartesian Domains”. In : *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29 (2007).

[9] F. Li et al. “A second order discontinuous Galerkin fast sweeping method for Eikonal equations”. In : *J. Comput. Phys.* 227 (2008), p. 8191-8208.

[10] Moshe Lichtenstein, Gautam Pai et Ron Kimmel. “Deep Eikonal Solvers”. In : *SSVM*. 2019.

[11] Marcus Noack et Stuart Clark. “Acoustic wave and eikonal equations in a transformed metric space for various types of anisotropy”. In : *Heliyon* 3 (mar. 2017), e00260. doi : 10.1016/j.heliyon.2017.e00260.

[12] Gabriel Peyré et al. “Geodesic Methods in Computer Vision and Graphics”. In : *Foundations and Trends in Computer Graphics and Vision* 5.3-4 (2010), p. 197-397. url : https://hal.archives-ouvertes.fr/hal-00528999.

[13] Emmanuel Prados et Olivier Faugeras. “Rôle clé de la Modélisation en Shape From Shading”. In : (jan. 2006).

[14] Emmanuel Prados et Olivier Faugeras. “Shape From Shading”. In : *Handbook of Mathematical Models in Computer Vision*. Sous la dir. d’Y. Chen N. Paragios et O. Faugeras. Springer, 2006, p. 375-388. url : https://hal.inria.fr/inria-00377417.

[15] J. Sethian. “Level Set Methods and Fast Marching Methods : Evolving Interfaces in Computational Geometry, Fluid”. In : 1999.

[16] J. A. Sethian. “A fast marching level set method for monotonically advancing fronts.” In : *Proceedings of the National Academy of Sciences of the United States of America* 93 4 (1996), p. 1591-5.

[17] Eran Treister et E. Haber. “A fast marching algorithm for the factored eikonal equation”. In : *ArXiv* abs/1607.00973 (2016).

[18] J. Tsitsiklis. “Efficient algorithms for globally optimal trajectories”. In : *Proceedings of 1994 33rd IEEE Conference on Decision and Control* 2 (1994), 1368-1373 vol.2.

[19] H. Zhao. “A fast sweeping method for Eikonal equations”. In : *Math. Comput.* 74 (2004), p. 603-627.