Deep Eikonal Solver

Rapport de stage

Yohan FRAISSINET-TACHET

Juin/Août 2020

Sommaire

1 Introduction

TODO

2 A propos de l'équation Eikonale

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 $\partial\Omega$ 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. 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 ou encore le son. 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.

L'équation eikonale a plusieurs applications en analyse d'images, la reconstruction de surface à partir de nuages de points et la segmentation d'images en sont quelques-unes. Elle peut apparaître lors de la modélisation d'un squelette (axe médian), par exemple lors du calcul de la carte de 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. Elle est donc souvent utilisée en vision par ordinateur car sa résolution permet de calculer des distances entre objets.

2.4 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. 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. L'algorithme quant à lui est similaire à l'algorithme de Dijkstra et utilise le fait que l'information circule uniquement vers l'extérieur.

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

2.4.2 Fast sweeping

L'idée principale de la méthode de balayage rapide (fast sweeping) 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. 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. Donc la complexité de l'algorithme est en $0(N)$ pour un total de N points de grille. 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.

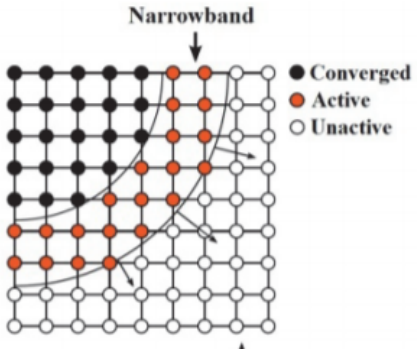

FIGURE 2 – Evolution du front d'onde avec Fast marching

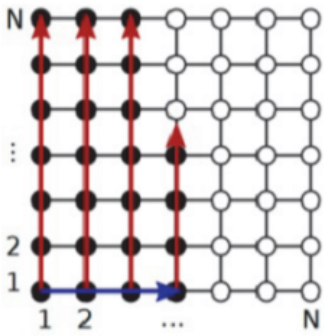

FIGURE 3 – Illustration du balayage avec Fast sweeping

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. Par ailleurs, 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.

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 existent. 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.

3 Approche avec paramétrisation régulière

Pour cette première partie, l'intégralité du code est réalisé sous Matlab. Celui-ci suit l'algorithme de marche rapide énoncé à la partie 2,5. On commence d'abord par implémenter l'algorithme en utilisant un solveur local numérique dont on verra la formalisation. Et on étend le problème en remplaçant le solveur local numérique par un réseau de neurones.

3.1 Avec solveur local numérique et carte de potentiel

3.1.1 Solveur utilisant les quatre plus proches voisins

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

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

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 à ce stade du projet permet de calculer 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.

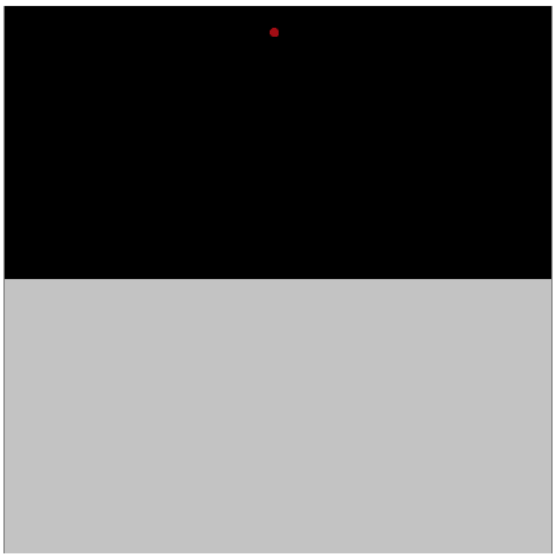

FIGURE 4 – Carte de potentiel initiale (binaire), le point rouge étant le point initial. la partie sombre correspond aux poids forts alors que la partie grise correspond aux poids faibles

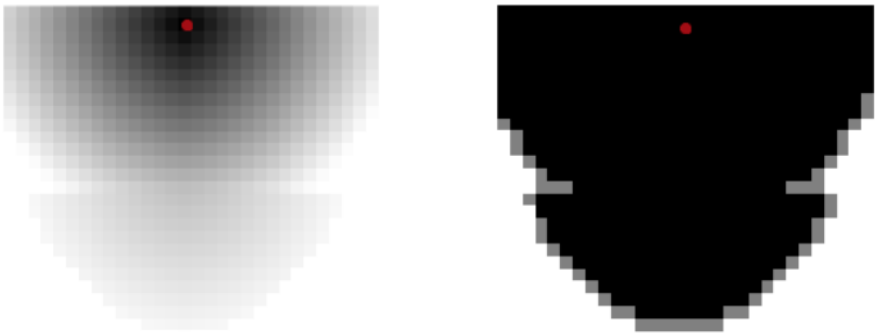

FIGURE 5 – A gauche : évolution de la carte des distances. A droite : évolution du front, en noir les points visités, en gris le front d'onde et en blanc les points non visités

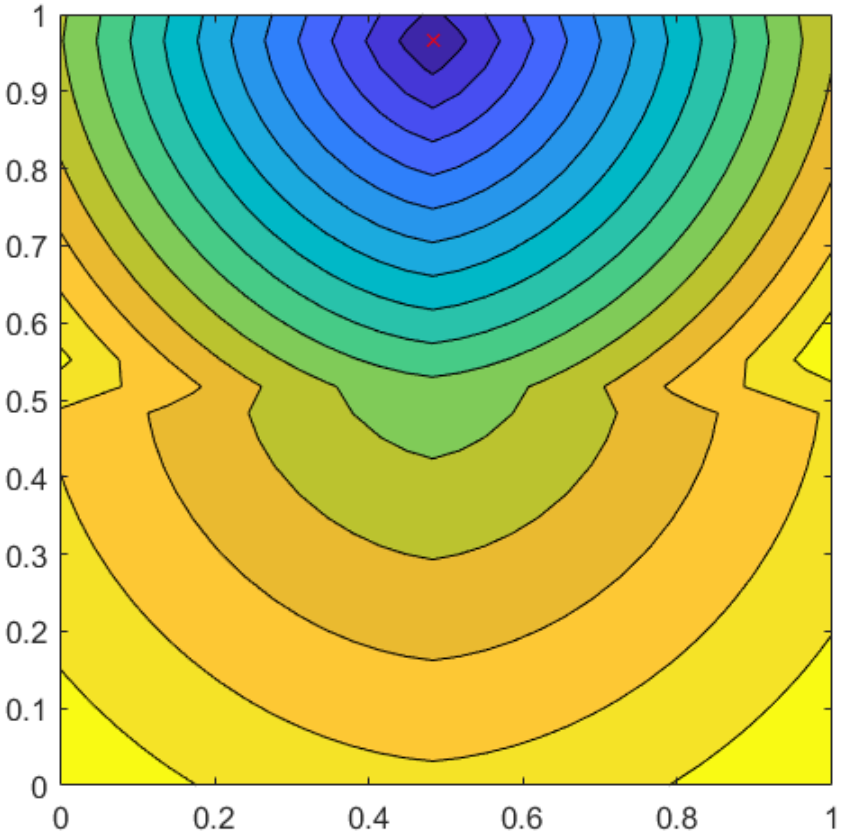

FIGURE 6 – Carte des distances avec isocontours. Le point initial en rouge

Ici le point de départ est dans une zone où les poids sont importants (zone noire sur la carte de potentiel). Ainsi la vitesse sera la même partout dans cette zone, et les distances seront 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.

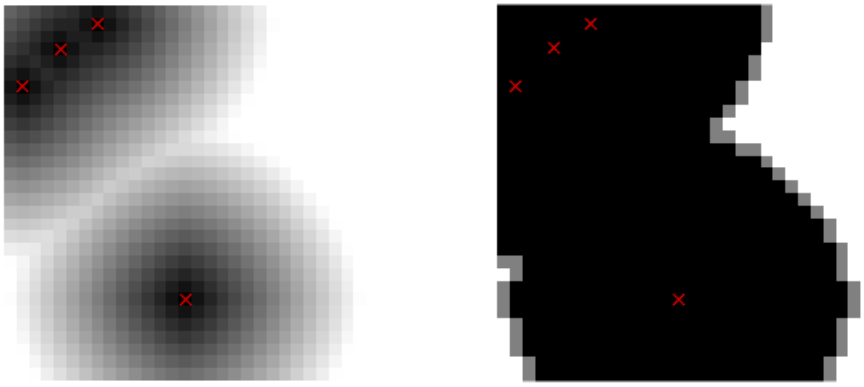

FIGURE 7 – A gauche : évolution de la carte des distances. A droite : évolution du front, en noir les points visités, en gris le front d'onde et en blanc les points non visités

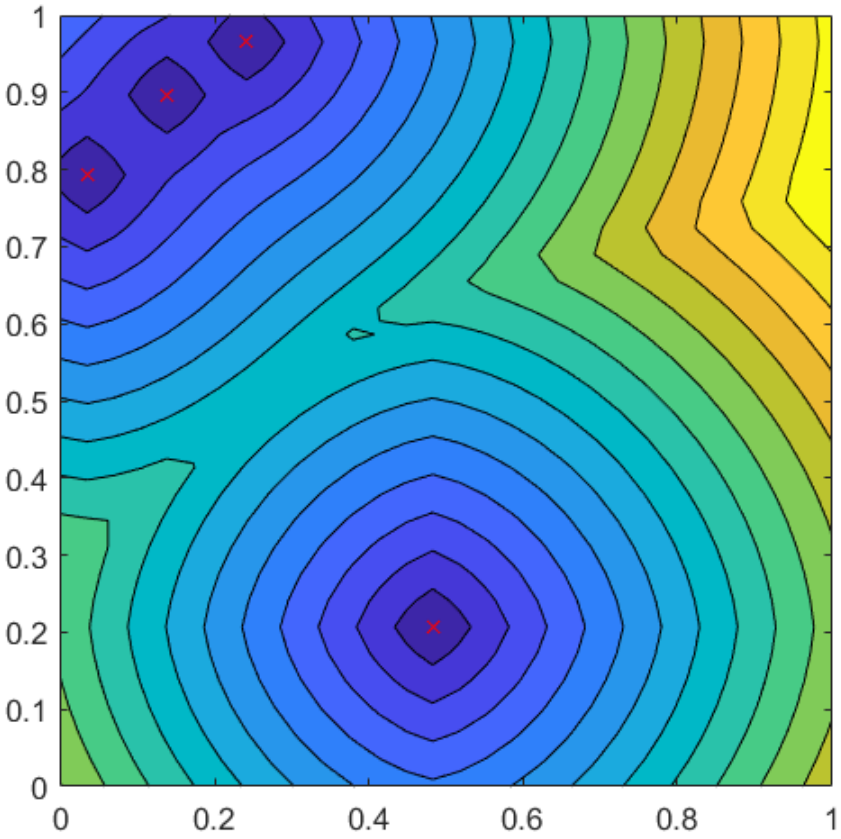

FIGURE 8 – Carte des distances avec isocontours. Les points initiaux en rouge

L'objectif est d'avoir des isocontours complètement lisse. Or d'après la figure précédente on voit très bien que le résultat est impacté par notre solveur qui ne calcule qu'en fonction des quatre plus proches voisins. Il est possible d'utiliser un solveur plus performant mais cela impliquerait d'avoir des calculs plus complexes et également de ralentir le programme.

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. Pour l'améliorer il est possible d'utiliser un ordre de discrétisation supérieur ou d'augmenter le nombre de voisins.

Pour un voisinage de 8 points 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. Ainsi les coordonnées d'un point (x_D, y_D) du nouveau système de coordonnées s'écrivent

$$\begin{aligned} x_D &= x\cos\theta + y\sin\theta \\ y_D &= -x\sin\theta + y\cos\theta \end{aligned}$$

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

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

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. 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.

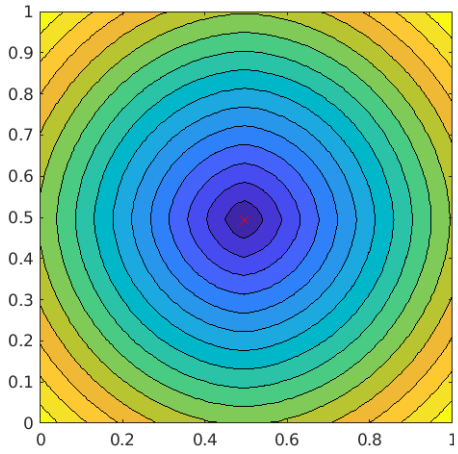

FIGURE 9 – Avec solveur à 4 voisins

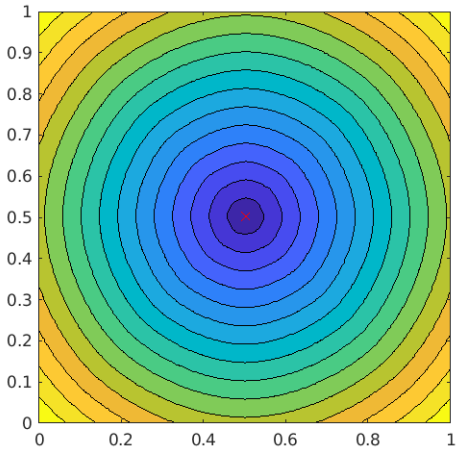

FIGURE 10 – Avec solveur à 8 voisins

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 Entraînement et structure du réseau (cas non pondéré)

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.

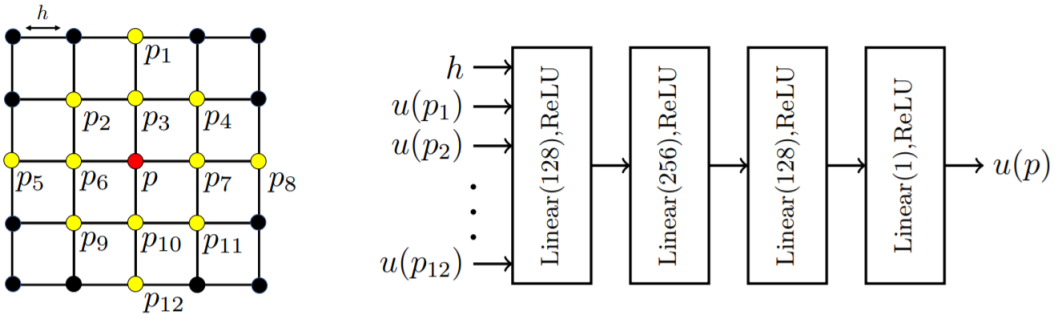

FIGURE 11 – Architecture du réseau

Pour l'entraînement on donne au réseau 10000 exemples correspondants à di-

verses 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. Pour l'entraînement il faut également fournir la valeur que doit retourner le réseau pour chaque patch. Ici la distance de vérité terrain u_{gt} entre deux points sera 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 la distance de vérité terrain du point q est plus faible que celle du point p et puisque 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'informations est indigeste pour le réseau de neurones, il faut donc simplifier les entrées. On dimensionne donc nos données vers $[0; 1]$ en soustrayant 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.

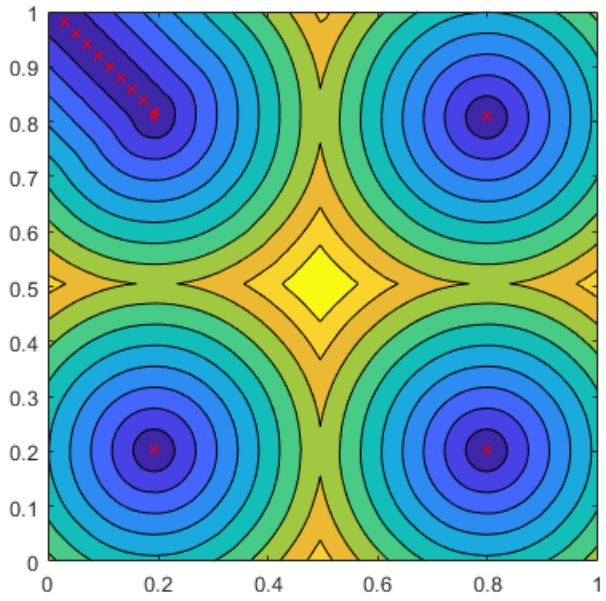

FIGURE 12 – Résultat du Deep Eikonal Solver

TODO : Superposer les résultats précédents avec ceux de cette section et la distance de vérité terrain.

3.2.2 Cas pondéré

La partie précédente résout l'équation eikonale pour $f = 1$ or il est intéressant de pouvoir résoudre l'équation pour f faiblement variable. 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.

Rien ne diffère entre le cas pondéré et non pondéré dans la construction du réseau... Sauf la distance de vérité terrain ! La distance utilisée ne peut plus être la distance euclidienne car chaque point est maintenant pondéré. On procède donc de cette manière :

TODO

4 Intégration de librairies

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 enfants. 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 $\lfloor \frac{k}{2} \rfloor$.

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 égale 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 en Matlab, néanmoins Matlab n'est absolument pas optimisé pour la gestion de ce type de donné comparé à d'autres langages de programmations plus classiques. C'est pourquoi dans la version finale le codage de l'algorithme est réalisé en C++ et est intégré dans du code Matlab à l'aide de la librairie Mex. On utilise par ailleurs le réseau créé avec PyTorch dans le code C++ à l'aide de la librairie LibTorch.

5 Conclusion

TODO