TP1 : Distances géodésiques isotropes et anisotropes [1]

Nous utilisons dans le calcul de distances et de géodésiques Riemanniennes dans le plan, via la librairie HamiltonFastMarching de l'auteur, dans le cadre de différentes techniques d'application au traitement des données - sur des exemples synthétiques.

Les paramètres données décrivant nos EDP seront stockées dans un dictionnaire, `hfmIn`, dont les clefs et valeurs seront présentées sous la forme `'clef':valeur`. Toutes les listes et tableaux renseignés de cette manière doivent être au format `numpy.array`. On importera les librairies suivantes [2] :

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.image
import HFMUtils as utils # Wrapper pour la librairie HFMpy + divers
```

1 Shape from shading / Métriques isotropes

Le problème du shape from shading, très classique en image, a été l'une des premières motivations du développement des méthodes de fast marching et autres solveurs d'équations eikonales. Cette approche a aujourd'hui un intérêt principalement historique, compte tenu de ses limitations, notamment l'impossibilité de reconstruire des creux à cause du principe de sélection de la solution de viscosité.

Dans le cadre idéal d'une surface illuminée verticalement et photographiée verticalement, uniformément réfléchissante, la luminosité captée suit l'expression suivante :

$$I(x,y) = \frac{I_0}{\sqrt{1 + \|\nabla z(x,y)\|^2}}$$

où x, y sont des coordonnées planaires, et z désigne l'élévation de la surface.

1. Montrer que l'élévation satisfait $\|\nabla z(x,y)\| = c(x,y)$, où c s'exprime en fonction de I et I_0.
2. Importer l'image [3] et définir une variable `gradNorm` par l'expression précédente [4] avec $I_0 = 1$.

```
intensity = matplotlib.image.imread("vaseShading.png")[:,:,0]
```

Nous allons (tenter de) reconstruire la hauteur $z(x,y)$ en calculant la solution de viscosité [5] d'une équation eikonale, donnée ci-dessous

$$\|\nabla z\| = c \text{ dans } \Omega, \qquad z = 0 \text{ dans } \Gamma, \tag{1}$$

où $\Gamma \subset \partial\Omega$. Des conditions de flot sortant sont imposées sur le reste de $\partial\Omega$.

3. Définir dans `hfmIn` les entrées suivantes, qui désignent le type d'équation eikonale résolue (métrique isotrope bi-dimensionnelle), et le second membre.

1. Dernière version : `dl.dropbox.com/s/8hwnj80gyhj919x/TP1.pdf`
2. Librairies HFMUtils : `dl.dropbox.com/s/sr5gluvg5nv7kf4/HFMUtils.py`
3. Image test : `dl.dropbox.com/s/61va8befr9f8nk5/vaseShading.png`
4. A laquelle on rajoutera $\varepsilon > 0$, pour éviter des problèmes de dégénérescence, avec e.g. $\varepsilon = 10^{-5}$.
5. Un choix de solution qui peut être qualifié d'arbitraire

`'model':'Isotropic2', 'cost':gradNorm, 'sndOrder':1,`

La dernière clef met en place la HAFMM (High Accuracy Fast Marching Method, Sethian et al), utilisant lorsque c'est possible des différences finies d'ordre 2 dans le schéma numérique.

Construction du domaine

4. Le domaine de résolution de l'EDP (1) sera $\Omega =]0, n_x[\times]0, n_y[$, où (n_x, n_y) est la taille en pixels de l'image traitée. Le définir par les clefs :

   `'dims':np.array(intensity.shape), 'gridScale':1,`

5. Définir `'seeds':[[x0,y0],[x1,y1],...]`, qui sont les "graines" à partir desquelles le front sera propagé, c'est à dire l'ensemble Γ dans (1). Compte tenu de l'instance spécifique traitée, on se peut se contenter de deux graines situées près des bords droit et gauche du domaine. (Pourquoi ?)

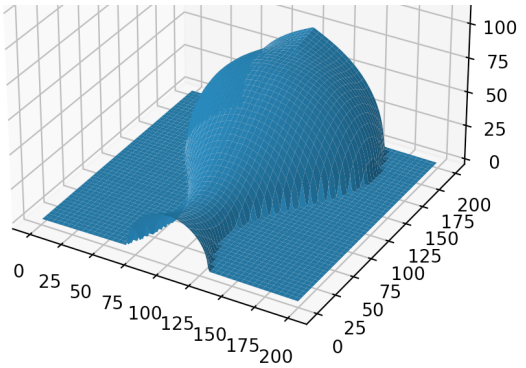
Lancement de l'algorithme, et visualisation

6. Demander l'export de la solution de (1), dans la convention d'axes de matplotlib [6], avec

   `'exportValues':1, 'arrayOrdering':'YXZ_RowMajor'`

   Execution puis visualisation via mplot3d

```
hfmOut = utils.Run(hfmIn)

# -------- Plotting -------
ax = plt.figure().add_subplot(111, projection='3d'
ax.set_zlim(0,gradNorm.shape[0])
ax.plot_surface(X, Y, hfmOut['values'])
plt.show()
```



7. Varier les conditions au bord, en retirant l'un des points, ou en définissant `'seedValues':[a0,a1,...]`.

Script complet : `dl.dropbox.com/s/lk178slthnpd9g3/TP1_ShapeFromShading.py`

2 Géodésiques sur une surface paramétrée / Met. Riemanniennes

Nous considérons dans cette partie une surface paramétrée $(x, y) \in \Omega \mapsto (x, y, z(x, y)) \in \mathbb{R}^3$, sur laquelle nous calculons des distances et géodésiques. Pour cela, nous résolvons une équation eikonale anisotrope sur le domaine Ω, muni de la métrique Riemannienne définie par le plongement précédent, à savoir

$$\mathcal{M}(x, y) := \mathrm{Id} + \nabla z(x, y) \otimes \nabla z(x, y)$$

6. `np.meshgrid` → `'YXZ_RowMajor'`, `np.mgrid` → `'RowMajor'`, et Matlab → `'YXZ_ColumnMajor'`.

Domaine et modèle

1. Définir le domaine Ω, par exemple avec la commande suivante dont les deux premiers arguments sont les coins du rectangle utilisé

   ```
   hfmIn = utils.MakeHFMRect([-0.5,-0.5],[0.5,0.5],gridScale=0.01)
   ```

2. Définir le modèle utilisé (Riemannien bi-dimensionnel, HAFMM) et la convention d'axes

   ```
   'model':'Riemann2', 'sndOrder':1, 'arrayOrdering':'YXZ_RowMajor'
   ```

Surface et métrique, extrémités des géodésiques

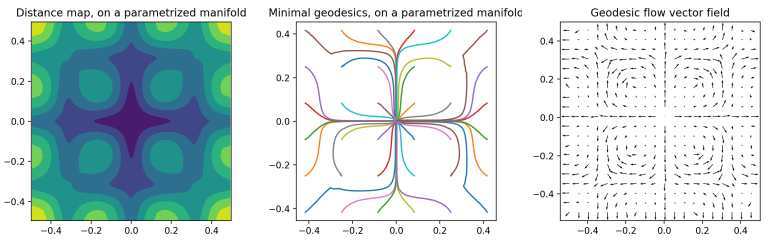
3. Construire un système de coordonnées par `X,Y = utils.GetGrid(hfmIn)`, et l'utiliser pour définir l'élévation `Z` souhaitée, par ex. $\frac{3}{4}\sin(3\pi x)\sin(3\pi y)$ dans les illustrations. Calculer ensuite son gradient et l'utiliser pour définir la métrique du problème

   ```
   DyZ,DxZ = np.gradient(Z,hfmIn['gridScale']) #Note the YXZ axes ordering

   'metric':np.stack((1.+DxZ**2, DxZ*DyZ, 1.+DyZ**2),2),
   ```

4. Définir un (ou plusieurs) point source pour la propagation de front, comme en §1-5, ainsi que les extrémités `'tips':[[x0,y0],[x1,y1],...]` des géodésiques à extraire.
5. Executer `hfmOut = utils.Run(hfmIn)` après avoir éventuellement défini `'exportValues':1,'exportGeodesicFlow':1` (pour le gradient Riemannien de la solution)

Exemples de visualisation :

```
plt.contourf(X,Y,hfmOut['values']);

for geo in utils.GetGeodesics(hfmOut):
        plt.plot(geo[:,0],geo[:,1])
```



Script complet : `dl.dropbox.com/s/lk178slthnpd9g3/TP1_ShapeFromShading.py`

3 Distance de Fisher-Rao / Optimisation géométrique

Nous calculons la moyenne de Fréchet d'une famille de distribution statistiques, au sens de la métrique de Fisher-Rao. Considérons est une densité de probabilité $p(x;\theta)$ sur un espace mesuré X, pour tout paramètre $\theta \in \Theta \subset \mathbb{R}^d$. La métrique d'information de Fisher, liée à la variation d'entropie relative et la divergence de Kullback-Liebler, est définie sur Θ par l'expression suivante (qui est bien symétrique semi-définie positive)

$$\mathcal{M}(\theta) := -E_\theta[\nabla^2_\theta \ln p] = -\int_X \nabla^2_\theta \ln p(x;\theta)\, p(x;\theta)\mathrm{d}x.$$

La moyenne de Fréchet des paramètres $\theta_1, \cdots, \theta_n \in \Theta$ est alors le minimiseur $\theta_* \in \Theta$ (s'il est unique) de

$$U(\theta) = \sum_{1\leq i\leq n} d_{\mathcal{M}}(\theta,\theta_i)^2. \tag{2}$$

1. Justifier que si $\mathcal{M} \equiv \mathrm{Id}$ sur un domaine $\Theta \subset \mathbb{R}^d$ convexe, alors $\theta_* = (\theta_1 + \cdots + \theta_n)/n$.

Nous nous focalisons dans la suite sur la gaussienne uni-dimensionelle, de paramètres $\theta = (m, \sigma) \in \mathbb{R} \times \mathbb{R}_+^*$, pour laquelle

$$p(x;\theta) := \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{(x-m)^2}{2\sigma^2}\Big), \qquad \mathcal{M}(\theta) = \frac{1}{\sigma^2}\begin{pmatrix} 1 & \\ & 2 \end{pmatrix}.$$

Domaine et modèle

2. Construire un domaine strictement inclus dans $\mathbb{R} \times \mathbb{R}_+^*$, par exemple $[-1.5, 1.5] \times [0.1, 1]$, en adaptant §2-1.

Quitte à faire un changement de variables linéaire (lequel ?), on peut se ramener au modèle $\mathcal{M}(\theta) = \mathrm{Id}\,/\sigma^2$ du demi-plan de Poincaré. C'est à dire à une métrique isotrope de fct coût $1/\sigma$.

3. Définir le modèle, et les caractéristiques d'export de la solution.

```
'model':'Isotropic2','sndOrder':1,'arrayOrdering':'YXZ_RowMajor','exportValues':1
```

4. Construire un système de coordonnées `mu,sig = utils.GetGrid(hfmIn)`, comme en §2-3, puis définir la fonction coût `'cost':1/sig`.
5. Définir une manière arbitraire une/des `'seeds'` et `'tips'`. Quelle est la forme géométrique des géodésiques et des lignes de niveau [7] ?

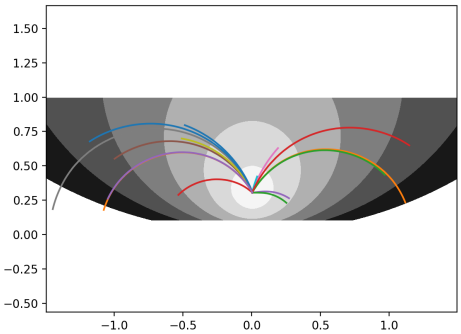
Calcul par recherche exhaustive.

6. Générer des paramètres aléatoires $\theta_1, \cdots, \theta_n \in \Omega$. On les prendra suffisamment loin du bord supérieur toutefois, compte tenu de la forme incurvée des géodésiques.
7. Les choisir tour à tour chaque $(\theta_i)_{i=1}^n$ comme `'seed'` et calculer $U = \sum_{i=1}^n u_i^2$, la somme des carrés des solutions associées. Puis extraire θ_* grâce à (avec `dist2sum`$= U$ et `pMin`$= \theta_*$)

```
iMin = np.argmin(dist2sum)
pMin=[mu.flat[iMin],sig.flat[iMin]]
```

Calcul par optimisation. L'approche du paragraphe précédent nécessite de lancer un calcul de Fast-Marching par paramètre $(\theta_i)_{i=1}^n$ à moyenner, ce qui est excessivement coûteux si ceux-ci sont trop nombreux. Dans ce cas, une approche plus raisonnable est d'évaluer la fonctionnelle (2) en un point $\theta \in \Theta$ par un unique calcul de FastMarching, de graine θ, puis d'appliquer un algorithme d'optimisation. Des méthodes de différentiation automatique, qui seront abordées si le temps le permet, permettent d'accéder à $\nabla U(\theta)$.

```
plt.contourf(mu,sig,hfmOut['values'],cmap='Greys');
for geo in utils.GetGeodesics(hfmOut):
        plt.plot(geo[:,0],geo[:,1])
plt.axis('equal');
```



Script complet : `dl.dropbox.com/s/mxuvxyfhj1whhjm/TP1_FisherRao.py`.

7. Ne pas oublier `plt.axis('equal')`. (Rep : ce sont tous deux des cercles.)