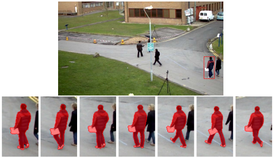
Traitement d'images par équations aux dérivées partielles et méthodes variationnelles

Julien MILLE

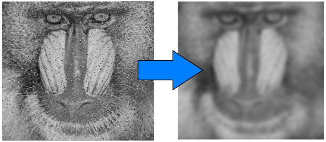
M2R IGI

25 novembre 2014

Applications

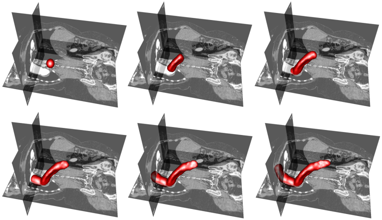


Segmentation et suivi



Filtrage

Estimation du mouvement



Reconstruction 3D

Plan

Plan

Image

- Modélisation continue de l'ensemble de départ (espace) et de l'ensemble d'arrivée (intensité/couleur)
- Domaine de définition (support) de l'image : un sous-domaine de $\mathbb{R}^2$

$$\mathcal{D} \subset \mathbb{R}^2$$

- Image à valeurs scalaires (intensité = niveau de gris)

$$f : \mathcal{D} \to \mathbb{R}$$

- $f(\boldsymbol{p})$ = intensité en un point $\boldsymbol{p} = (x, y) \in \mathcal{D}$
- Pour une image à valeurs vectorielles (couleur, souvent 3 composantes), on écrirait :

$$\begin{aligned} &\mathbf{f} : \mathcal{D} \to \mathbb{R}^3 \\ &\mathbf{f}(\boldsymbol{p}) = [f_1(\boldsymbol{p}) \; f_2(\boldsymbol{p}) \; f_3(\boldsymbol{p})]^\mathsf{T} \end{aligned}$$

Image

- La fonction image est supposée continue et dérivable (au moins deux fois)
- f est au moins de classe C^2 sur $\mathcal{D}$
  $\rightarrow$ses dérivées partielles $\dfrac{\partial f}{\partial x}, \dfrac{\partial f}{\partial y}, \dfrac{\partial^2 f}{\partial x^2}, \dfrac{\partial^2 f}{\partial x \partial y}, \dfrac{\partial^2 f}{\partial y^2}$ sont toutes continues sur $\mathcal{D}$
- Elle est intégrable sur n'importe quel sous-domaine Ω de $\mathcal{D}$

$$\int_{\Omega} f(\boldsymbol{p}) \mathrm{d}\boldsymbol{p} \leq +\infty$$

Principes et intérêt

- Modélisation de l'image comme une quantité physique continue en espace, et également en temps dans le cas d'un modèle basé sur une évolution
- Intérêt : accès aux outils mathématiques d'analyse des fonctions (dérivation, intégration, équations aux dérivées partielles, calcul des variations, etc.) et de concepts issus de la physique (diffusion, densité, énergie, etc.)
- En aval du raisonnement mathématique continu, discrétisation spatiale et temporelle pour permettre la résolution numérique et son implémentation algorithmique
- Dans ce cours, présentation du cheminement complet : des équations au code...

Dérivation : premier ordre

- Dérivées partielles d'ordre 1, gradient

$$\nabla f = \left[\frac{\partial f}{\partial x} \;\; \frac{\partial f}{\partial y}\right]^{\mathsf{T}}$$

- Le gradient donne la direction et l'amplitude de variation de l'image (analogie avec la pente d'un relief)
- Sa norme donne l'amplitude de variation $\rightarrow$utilisée pour la détection des contours

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

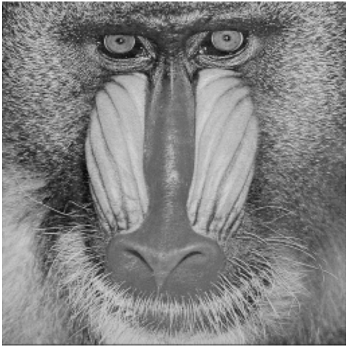
- Dérivée directionnelle selon un vecteur $\mathbf{u}$

$$\begin{aligned} \nabla_{\mathbf{u}} f(\boldsymbol{p}) &= \left.\frac{\mathrm{d} f(\boldsymbol{p} + \epsilon \mathbf{u})}{\mathrm{d}\epsilon}\right|_{\epsilon=0} \\ &= \nabla f \cdot \mathbf{u} \end{aligned}$$

où $\cdot$ est le produit scalaire

Détection de contours

- Exemple de traitement basique : détection de contours par calcul de la norme du gradient



f

$\|\nabla f\|$

- En pratique, lissage préalable

Dérivation : second ordre

- Dérivées partielles d'ordre 2, laplacien, hessienne...
- Hessienne $\mathbf{H}$: matrice des dérivées partielles d'ordre 2

$$\mathbf{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

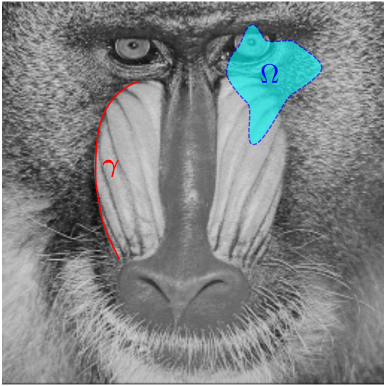
- Laplacien ∇^2 (noté également Δ)

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

- $\mathbf{H}_f(\boldsymbol{p})$ et $\nabla^2 f(\boldsymbol{p})$ sont calculables en tout point $\boldsymbol{p}$

Intégration

- Intégration d'un terme dépendant de l'image sur un sous-ensemble du domaine image $\mathcal{D}$
- Le long d'une courbe γ, sur une région (sous-domaine) Ω, ...



Intégration

- Exemple 1 : moyenne et variance d'intensité sur une région Ω

$$\begin{aligned} \mu[\Omega] &= \frac{1}{|\Omega|}\int_\Omega f(\boldsymbol{p})\mathrm{d}\boldsymbol{p} \\ \sigma^2[\Omega] &= \frac{1}{|\Omega|}\int_\Omega \left(f(\boldsymbol{p}) - \mu[\Omega]\right)^2 \mathrm{d}\boldsymbol{p} \end{aligned}$$

$\rightarrow\sigma^2$ mesure l'hétérogénéité d'une région (beaucoup d'intensités différentes ?)

- Exemple 2 : amplitude du gradient le long d'une courbe $\boldsymbol{\gamma} : [0,1] \rightarrow \mathcal{D}$

$$g[\boldsymbol{\gamma}] = \int_0^1 \|\nabla f(\boldsymbol{\gamma}(s))\| \left\| \frac{\mathrm{d}\boldsymbol{\gamma}}{\mathrm{d}s} \right\| \mathrm{d}s$$

$\rightarrow g$ mesure l'adéquation de la courbe aux contours présents dans l'image

Plan

Equation aux dérivées partielles

- Application d'**équations aux dérivées partielles** (EDP) au traitement d'images : l'image ou une ou plusieurs fonctions associées sont soumises à des lois physiques modélisées sous forme d'équations d'évolution
- Introduction de la dimension temporelle t
- Schéma général :
$$\frac{\partial f(\boldsymbol{p}, t)}{\partial t} = \ldots$$
où le membre de droite représente l'évolution de la fonction f en chaque point au cours du temps
- Equation accompagnée de condition(s) initiale(s) : l'image initiale (à $t = 0$) est donnée
$$f(\boldsymbol{p}, 0) = f_0(\boldsymbol{p})$$
- Résolution numérique et itérative, avec critère d'arrêt (seuil sur la variation, nombre d'itérations fixé, etc.)

Equation de diffusion de chaleur

- Exemple 1 d'EDP : équation de diffusion de la chaleur

$$\frac{\partial f}{\partial t} = \nabla^2 f$$

- Appliquer itérativement cette équation revient à effectuer des lissages laplaciens successifs

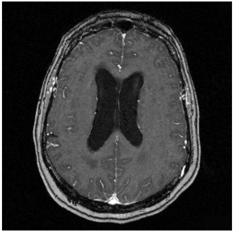


Image initiale

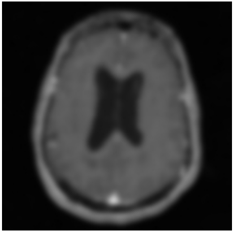
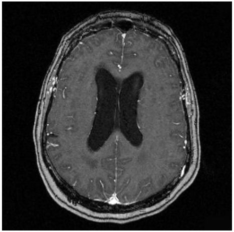


Image finale



Evolution

Equation de diffusion de chaleur

- Utilisation : débruitage (inconvénient : dégradation des contours)

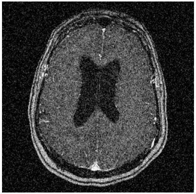


Image initiale
(+ bruit gaussien)

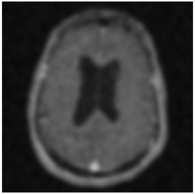
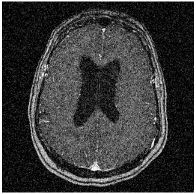


Image finale



Evolution

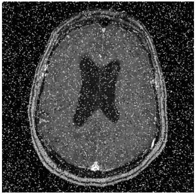


Image initiale
(+ bruit impulsionnel)

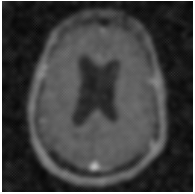
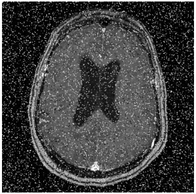


Image finale



Evolution

Equation de diffusion anisotrope

- Exemple 2 d'EDP : équation de diffusion anisotrope
$$\frac{\partial f}{\partial t} = \nabla c \cdot \nabla f + c \nabla^2 f$$
où c est le coefficient de diffusion (décroissant avec l'amplitude du gradient)
$$c(\boldsymbol{p}, t) = \frac{1}{1 + \|\nabla f(\boldsymbol{p}, t)\|}$$
- Appliquer itérativement cette équation revient à effectuer des lissages successifs, avec conservation des contours saillants

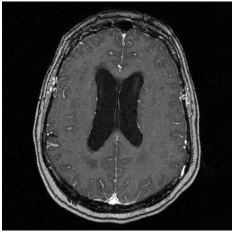


Image initiale

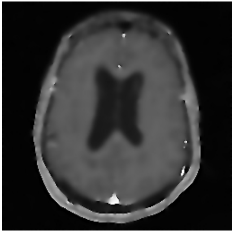
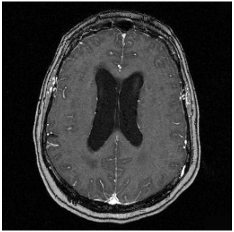


Image finale



Evolution

Equation de diffusion anisotrope

- Utilisation : débruitage avec préservation des contours saillants

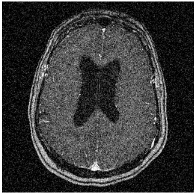


Image initiale
(+ bruit gaussien)

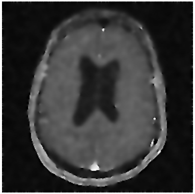
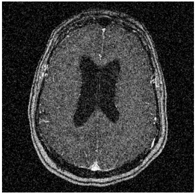


Image finale



Evolution

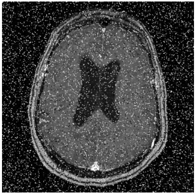


Image initiale
(+ bruit impulsionnel)

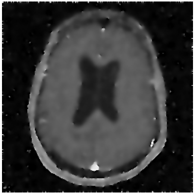
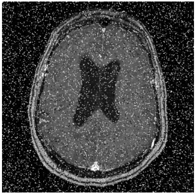


Image finale



Evolution

Plan

Différences finies

- Rappel de la définition de la dérivée partielle (ordre 1) :

$$\frac{\partial f(x_0, y_0)}{\partial x} = \lim_{h \to 0} \frac{f(x_0 + h, y_0) - f(x_0, y_0)}{h}$$

- Une différence finie est une approximation de la dérivée
- Trois schémas de discrétisation possibles (par rapport à x, dans cet exemple)
  - Différence finie arrière (à gauche) :

$$f(x, y) - f(x - 1, y)$$

  - Différence finie avant (à droite) :

$$f(x + 1, y) - f(x, y)$$

  - Différence finie centrée :

$$\frac{f(x + 1, y) - f(x - 1, y)}{2}$$

Différences finies

- Rappel de la définition de la dérivée partielle (ordre 2) :

$$\frac{\partial^2 f(x_0, y_0)}{\partial x^2} = \lim_{h \to 0} \frac{f(x_0 + h, y_0) - 2f(x_0, y_0) + f(x_0 - h, y_0)}{h^2}$$

- Pour la discrétisation de la dérivée seconde, on utilise généralement la différence finie centrée :

$$\frac{\partial^2 f(x, y)}{\partial x^2} \approx f(x + 1, y) - 2f(x, y) + f(x - 1, y)$$

  (idem par rapport à y)

- Pour la dérivée croisée, deux différences finies centrées d'ordre 1 :

$$\frac{\partial^2 f(x, y)}{\partial x \partial y} \approx \frac{f(x+1, y+1) - f(x-1, y+1) - f(x+1, y-1) + f(x-1, y-1)}{4}$$

Discrétisation d'une EDP

- Discrétisation temporelle d'Euler = approximation par différence finie arrière de la dérivée par rapport au temps (membre de gauche)

$$\frac{\partial f}{\partial t} \approx \frac{f^{(t+1)} - f^{(t)}}{\Delta t}$$

- Notation : temps discret placé en exposant
- Mise sous forme d'un schéma explicite : calcul de $f^{(t+1)}$ en fonction de $f^{(t)}$
- Exemple avec l'équation de diffusion de la chaleur

$$\begin{aligned} \frac{\partial f}{\partial t} &= \nabla^2 f \\ \Rightarrow \quad f^{(t+1)} &= f^{(t)} + \Delta t \nabla^2 f^{(t)} \end{aligned}$$

où $\nabla^2 f^{(t)}$ est discrétisé (en espace) par différences finies

Implémentation en C

- Pour les images en niveaux de gris, on considèrera une représentation en tableau 2D de réels (peu efficace, mais génère du code très lisible !)

```
...

typedef struct
{
  unsigned int width, height;
  float **pixels;
} ImageGrayscale;

void initImage(ImageGrayscale *, unsigned int, unsigned int);
void freeImage(ImageGrayscale *);
void copyImage(ImageGrayscale *, const ImageGrayscale *);
...
```

- Dans la fonction d'initialisation, on suppose que `pixels` est alloué de telle sorte que `pixels[x][y]` corresponde à $f(x, y)$

Implémentation en C : lissage Laplacien

- Discrétisation de l'opérateur Laplacien

$$\nabla^2 f(x,y) \approx f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

```
void smoothImageLaplacian(const ImageGrayscale *pImgInput,
                          ImageGrayscale *pImgOutput, float delta_t)
{
  unsigned int x, y;
  float **f, **g;

  initImage(pImgOutput, pImgInput->width, pImgInput->height);
  f = pImgInput->pixels;
  g = pImgOutput->pixels;
  ...
  for (y=1; y<pImgInput->height-1; y++)
    for (x=1; x<pImgInput->width-1; x++)
      g[x][y] = f[x][y] + delta_t*
        (f[x+1][y] + f[x-1][y] + f[x][y+1] + f[x][y-1] - 4*f[x][y]);
}
```

- Remarque : bords non traités
- A appeler dans une boucle jusqu'à vérification d'un critère d'arrêt

Plan

Des équations sur l'image...
- Calcul différentiel et intégral sur les images
- Equations aux dérivées partielles
- Discrétisation et implémentation

Méthodes variationnelles
- Principe général
- Exemple 1 : modèle de débruitage
- Exemple 2 : modèle de contour actif

Plan

Problème d'optimisation

- Le traitement à réaliser à partir de l'image est formulé comme un **problème d'optimisation**, qui comporte :
  - des donnée(s) : une ou plusieurs fonction(s) (dont l'image f)
  - des variable(s) : une ou plusieurs fonction(s) (notée u par défaut)
  - une fonction objective : une fonctionnelle J de u à minimiser
  - éventuellement, des contraintes : équation(s) ou inéquation(s) faisant intervenir u
- Exemple : recherche d'une image lisse proche de f et nulle sur les bords

$$\min_{u \in F} \int_{\mathcal{D}} \underbrace{\|\nabla u(\boldsymbol{p})\|}_{\text{régularisation}} + \lambda \underbrace{(u(\boldsymbol{p}) - f(\boldsymbol{p}))^2}_{\text{attache aux données}} \mathrm{d}\boldsymbol{p}$$
$$\text{t.q. } u(\boldsymbol{p}) = 0 \ \forall \boldsymbol{p} \in \partial\mathcal{D}$$

  où F est un espace de fonctions vérifiant certaines propriétés sur $\mathcal{D}$ et $\lambda \in \mathbb{R}^+$ est un paramètre de la fonctionnelle

Remarques

- Le problème d'optimisation est de **dimension infinie**, car la recherche de la solution se fait dans un **espace de fonctions**
  - A l'inverse, dans les problèmes d'optimisation de dimension finie, on cherche un nombre fini de variables (exemple : optimisation discrète, problème linéaire simple, ...)
  - Exemple de problème d'optimisation discret en traitement d'images : champs de Markov $\rightarrow$une variable par pixel
- La **fonctionnelle** à minimiser (souvent appelée **énergie** ou **coût**) est la plupart du temps une intégrale d'un terme dépendant de f, u et de leurs dérivées
- L'énergie est une traduction mathématique des propriétés recherchées de la ou des variable(s)
- **Difficulté** : savoir écrire mathématiquement les propriétés/phénomènes que l'on souhaite pénaliser (exemple : pénaliser la norme du gradient revient à encourager une fonction lisse)

Optimisation de fonctionnelles : un exemple simple

- Existence de cas d'écoles, pour lesquels une solution analytique existe
- Exemple : trouver le plus court chemin (une courbe) allant d'un point $\boldsymbol{a}$ à un point $\boldsymbol{b}$ dans le plan continu $\mathbb{R}^2$
  - Soit $\mathcal{C}$ l'espace de fonction décrivant toutes les courbes planes possibles, de $[0, 1]$ dans $\mathbb{R}^2$
  - Fonction à déterminer : une courbe $\boldsymbol{\gamma} \in \mathcal{C}$
  - Fonctionnelle d'énergie J : la longueur de la courbe, $J : \mathcal{C} \to \mathbb{R}$
  - Contraintes : $\boldsymbol{\gamma}(0) = \boldsymbol{a}$ et $\boldsymbol{\gamma}(1) = \boldsymbol{b}$
- En d'autres termes :

$$\min_{\gamma \in \mathcal{C}} \int_0^1 \left\| \frac{\mathrm{d}\boldsymbol{\gamma}(s)}{\mathrm{d}s} \right\| \mathrm{d}s \qquad \text{t.q.} \qquad \boldsymbol{\gamma}(0) = \boldsymbol{a} \text{ et } \boldsymbol{\gamma}(1) = \boldsymbol{b}$$

- Solutions triviales : segment de droite $[\boldsymbol{ab}]$, mais démontrable par le calcul. Une solution possible est :

$$\boldsymbol{\gamma}(s) = s\boldsymbol{b} + (1 - s)\boldsymbol{a}$$

- Dans notre cas, les problèmes ne pourront jamais être résolus analytiquement ! $\rightarrow$résolution numérique

Principe

- Etude des extrema locaux de la fonctionnelle d'énergie à l'aide du **calcul des variations**
- Résolution numérique (le plus souvent itérative) pour atteindre un **minimum local** de l'énergie : méthode de **descente de gradient**
- En résumé, le traitement d'image par méthode variationnelle suit les étapes suivantes :
  1. Introduction d'une énergie
  2. Calcul des variations de l'énergie
  3. Formulation de l'EDP
  4. Discrétisation et implémentation de la descente de gradient de l'EDP
  5. Tests sur un ensemble d'images

Extrema locaux

Extrema d'une fonction d'une seule variable réelle

- Lorsqu'on cherche à minimiser une fonction $\rho : \mathbb{R} \to \mathbb{R}$ par rapport à une variable réelle x,
$$\min_{x \in \mathbb{R}} \rho(x)$$
la première étape consiste à étudier les extrema locaux de ρ, donc résoudre
$$\frac{\partial \rho}{\partial x} = 0$$

Extrema d'une fonctionnelle

- Ici, on cherche à minimiser une fonctionnelle $J : F \to \mathbb{R}$ par rapport à une fonction $u \in F$
$$\min_{u \in F} J[u] = \int L\left(x, u(x), \frac{\partial u(x)}{\partial x}\right) \mathrm{d}x$$
où F est l'espace des fonctions étudiées
- Besoin de caractériser les fonctions u entrainant un extremum local de $J[u]$ $\rightarrow$nécessité d'étendre la notion de dérivée *classique* aux fonctionnelles

Calcul des variations

- Admettons que $J[u]$ soit dans un minimum local lorsque $u = u_0$ $\rightarrow$pour n'importe quelle infime perturbation $\eta \in F$ ajoutée à u, l'énergie augmentera : $J[u_0 + \eta] > J[u_0]\ \forall \eta$
- On suppose que J est **différentiable** $\forall u$
- Introduction d'une variation infinitésimale η et dérivation $\rightarrow$**calcul des variations**
- On obtient la **variation première** de J en u pour une variation η :

$$\lim_{\epsilon \to 0} \frac{J[u + \epsilon\eta] - J[u]}{\epsilon} = \left.\frac{\mathrm{d}J[u + \epsilon\eta]}{\mathrm{d}\epsilon}\right|_{\epsilon=0}$$

- Remarque : ressemble à une dérivée directionnelle

Equation d'Euler-Lagrange

Cas de fonctions d'une seule variable réelle

- L'espace F contient les fonctions $u : \mathbb{R} \to \mathbb{R}$ de classe C^2
- Soit la fonctionnelle J définie sur $[a, b] \subset \mathbb{R}$,

$$J[u] = \int_a^b L(x, u(x), u'(x))\mathrm{d}x \quad \text{avec} \quad u'(x) = \frac{\partial u(x)}{\partial x}$$

- Par le calcul des variations, on peut montrer que si $J[u]$ est un extremum local de J, alors l'**équation d'Euler-Lagrange** est vérifiée :

$$\frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\left\{\frac{\partial L}{\partial u'}\right\} = 0 \quad \forall\, x \in [a, b]$$

- Les dérivées partielles de L sont calculées en considérant x, u et u' comme des variables indépendantes (par abus de notation, $\frac{\partial L}{\partial u}$ et $\frac{\partial L}{\partial u'}$ désignent les dérivées partielles de L par rapport à ses 2ème et 3ème paramètre)

Equation d'Euler-Lagrange

Cas de fonctions de plusieurs variables réelles

- L'espace F contient les fonctions $u : \mathbb{R}^n \to \mathbb{R}$ de classe C^2
- Un point est noté $\boldsymbol{x} = [x_1 \ x_2 \ ... \ x_n]$
- Soit la fonctionnelle J définie sur $\mathcal{D} \subset \mathbb{R}^n$,

$$J[u] = \int_{\mathcal{D}} L(\boldsymbol{x}, u(\boldsymbol{x}), u_{x_1}(\boldsymbol{x}), u_{x_2}(\boldsymbol{x}), ..., u_{x_n}(\boldsymbol{x}))\mathrm{d}\boldsymbol{x}$$

avec $u_{x_i}(\boldsymbol{x}) = \dfrac{\partial u(\boldsymbol{x})}{\partial x_i}$

(autre écriture possible : $L(\boldsymbol{x}, u(\boldsymbol{x}), \nabla u(\boldsymbol{x}))$)

- Equation d'Euler-Lagrange correspondante :

$$\frac{\partial L}{\partial u} - \sum_{i=1}^{n} \frac{\mathrm{d}}{\mathrm{d}x_i} \left\{ \frac{\partial L}{\partial u_{x_i}} \right\} = 0 \quad \forall \ \boldsymbol{x} \in \mathcal{D}$$

Dérivée fonctionnelle

- Le membre de gauche de l'équation d'Euler-Lagrange est appelé **dérivée fonctionnelle** de J par rapport à u.
- La dérivée fonctionnelle de

$$J[u] = \int_a^b L(x, u(x), u'(x))\mathrm{d}x$$

s'écrit

$$\frac{\delta J[u]}{\delta u} = \frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\left\{\frac{\partial L}{\partial u'}\right\}$$

- Intuition pour la minimisation numérique de J : résoudre itérativement l'équation d'Euler-Lagrange en chaque point x $\rightarrow$à chaque itération, prendre la **direction opposée à la dérivée fonctionnelle** (principe de **descente de gradient**)

Lien entre variation première et dérivée fonctionnelle

- Calculons la variation première de

$$J[u] = \int_a^b L(x, u(x), u'(x))\mathrm{d}x$$

pour une variation η. On obtient : (détail au tableau)

$$\left.\frac{\mathrm{d}J[u+\epsilon\eta]}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = \int_a^b \eta(x)\frac{\partial L}{\partial u} + \eta'(x)\frac{\partial L}{\partial u'}\mathrm{d}x$$

- En intégrant par partie et en posant $\eta(a) = 0$ et $\eta(b) = 0$, (détail au tableau)

$$\left.\frac{\mathrm{d}J[u+\epsilon\eta]}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = \int_a^b \underbrace{\left(\frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\left\{\frac{\partial L}{\partial u'}\right\}\right)}_{\text{dérivée fonctionnelle}} \eta(x)\mathrm{d}x$$

Lien entre dérivée fonctionnelle et variation première

- Si $J[u]$ est un extremum local de J, alors sa variation première doit s'annuler quelle que soit la fonction η

$$\left.\frac{\mathrm{d}J[u+\epsilon\eta]}{\mathrm{d}\epsilon}\right|_{\epsilon=0} = \int_a^b \left(\frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\left\{ \frac{\partial L}{\partial u'} \right\} \right) \eta(x)\mathrm{d}x = 0 \ \ \forall \eta$$

- Lemme fondamental du calcul des variations $\rightarrow$si la condition précédente est vraie, alors on a :

$$\frac{\delta J[u]}{\delta u} = \frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\left\{ \frac{\partial L}{\partial u'} \right\} = 0 \ \ \forall x \in [a,b]$$

- On retombe ainsi sur l'équation d'Euler-Lagrange

Exercice

- Soit une fonction $u : \mathbb{R} \to \mathbb{R}$ de classe C^2 sur l'intervalle $[a, b] \subset \mathbb{R}$
- Soit la fonctionnelle

$$J[u] = \int_a^b (u'(x))^2 + (u(x) - f(x))^2 \mathrm{d}x$$

- Calculer la variation première de J pour une variation η (nulle en a et b) et la dérivée fonctionnelle correspondante

Exercice

- Soit une fonction $u : \mathbb{R} \to \mathbb{R}$ de classe C^2 sur l'intervalle $[a, b] \subset \mathbb{R}$
- Soit la fonctionnelle

$$J[u] = \int_a^b (u'(x))^2 + (u(x) - f(x))^2 \mathrm{d}x$$

- Calculer la variation première de J pour une variation η (nulle en a et b) et la dérivée fonctionnelle correspondante

Solution : (les (x) sont parfois omis)

$$\begin{aligned}
\left.\frac{\mathrm{d}J[u+\epsilon\eta]}{\mathrm{d}\epsilon}\right|_{\epsilon=0} &= \int_a^b \frac{\mathrm{d}}{\mathrm{d}\epsilon} \left.\left\{(u' + \epsilon\eta')^2 + (u + \epsilon\eta - f)^2\right\}\right|_{\epsilon=0} \mathrm{d}x \\
&= \int_a^b \left.\left\{2\eta'(u' + \epsilon\eta') + 2\eta(u + \epsilon\eta - f)\right\}\right|_{\epsilon=0} \mathrm{d}x \\
&= \int_a^b 2\eta' u' + 2\eta(u - f)\mathrm{d}x \\
&= \int_a^b (-2u''(x) + 2(u(x) - f(x)))\eta(x)\mathrm{d}x
\end{aligned}$$

$$\frac{\delta J[u]}{\delta u(x)} = 2(u(x) - f(x) - u''(x))$$

Descente de gradient

- L'équation d'Euler-Lagrange ne peut être résolue analytiquement
- Fonctionnelle d'énergie souvent non-convexe $\rightarrow$présence de nombreux minima locaux
- Introduction d'une EDP effectuant une **descente de gradient** de l'énergie :

$$\frac{\partial u(x,t)}{\partial t} = -\frac{\delta J[u]}{\delta u(x)} \; \forall x$$

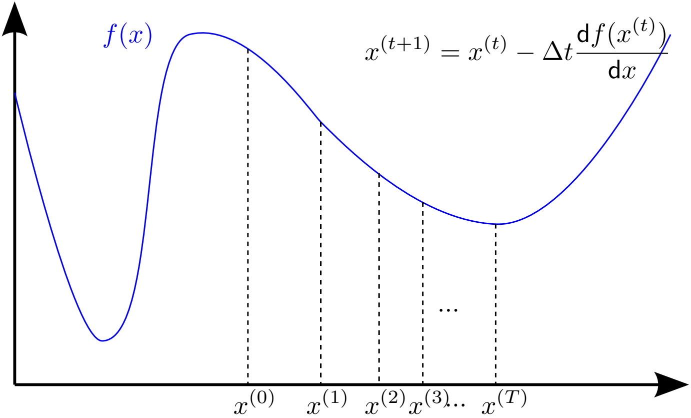
- Convergence attendue vers un minimum local relativement proche d'une solution initiale fournie $u_0 = u(.,0)$
- Discrétisation temporelle d'Euler :

$$u^{(t+1)}(x) = u^{(t)}(x) - \Delta t \frac{\delta J[u^{(t)}]}{\delta u^{(t)}(x)} \; \forall x$$

- Implémentation la plus simple : l'équation d'évolution est appliquée en parallèle **point par point**
- Nécessité de définir un critère de stabilité (= d'arrêt)
- Inconvénient : dépendance par rapport à l'initialisation

Descente de gradient

- Illustration pour une fonction d'une variable réelle



- Difficile de représenter graphiquement le même processus pour une fonctionnelle !

Plan

Modèle de débruitage

- Recherche d'une image débruitée (lisse et proche de l'image d'entrée)
- La fonction recherchée u a les mêmes ensembles de départ et d'arrivée que l'image : $u : \mathcal{D} \to \mathbb{R}$
- Energie de débruitage :

$$J[u] = \int_{\mathcal{D}} \|\nabla u(\boldsymbol{p})\|^2 + (u(\boldsymbol{p}) - f(\boldsymbol{p}))^2 \mathrm{d}\boldsymbol{p}$$

Modèle de débruitage

- Recherche d'une image débruitée (lisse et proche de l'image d'entrée)
- La fonction recherchée u a les mêmes ensembles de départ et d'arrivée que l'image : $u : \mathcal{D} \to \mathbb{R}$
- Energie de débruitage :

$$J[u] = \int_{\mathcal{D}} \|\nabla u(\boldsymbol{p})\|^2 + (u(\boldsymbol{p}) - f(\boldsymbol{p}))^2 \mathrm{d}\boldsymbol{p}$$

Calcul des variations : (les $(\boldsymbol{p})$ sont parfois omis)

$$\begin{aligned}
\left.\frac{\mathrm{d}J[u+\epsilon\eta]}{\mathrm{d}\epsilon}\right|_{\epsilon=0} &= \int_{\mathcal{D}} \frac{\mathrm{d}}{\mathrm{d}\epsilon} \left.\left\{\|\nabla u + \epsilon\nabla\eta\|^2 + (u + \epsilon\eta - f)^2\right\}\right|_{\epsilon=0} \mathrm{d}\boldsymbol{p} \\
&= \int_{\mathcal{D}} \left.\{2\nabla\eta \cdot (\nabla u + \epsilon\nabla\eta) + 2\eta(u + \epsilon\eta - f)\}\right|_{\epsilon=0} \mathrm{d}\boldsymbol{p} \\
&= \int_{\mathcal{D}} 2\nabla\eta \cdot \nabla u + 2\eta(u - f)\mathrm{d}\boldsymbol{p} \\
&= \int_{\mathcal{D}} \left(-2\nabla^2 u(\boldsymbol{p}) + 2(u(\boldsymbol{p}) - f(\boldsymbol{p}))\right) \eta(\boldsymbol{p})\mathrm{d}\boldsymbol{p}
\end{aligned}$$

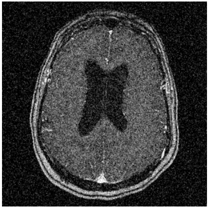
$$\frac{\delta J[u]}{\delta u(\boldsymbol{p})} = 2(u(\boldsymbol{p}) - f(\boldsymbol{p}) - \nabla^2 u(\boldsymbol{p}))$$

Modèle de débruitage

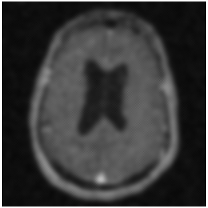
- Equation d'évolution discrétisée en temps (pour minimiser J) :

$$\begin{aligned} u^{(t+1)}(\boldsymbol{p}) &= u^{(t)}(\boldsymbol{p}) - \Delta t \frac{\delta J[u^{(t)}]}{\delta u^{(t)}(\boldsymbol{p})} \\ &= u^{(t)}(\boldsymbol{p}) + 2\Delta t \left[\nabla^2 u^{(t)}(\boldsymbol{p}) - u^{(t)}(\boldsymbol{p}) + f(\boldsymbol{p}) \right] \end{aligned}$$

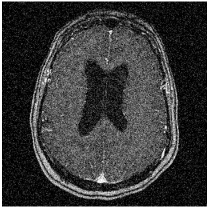
- Initialisation pertinente : $u^{(0)}(\boldsymbol{p}) = f(\boldsymbol{p})$
- Exemple :



$u^{(0)}$



$u^{(t_{\text{end}})}$



Itérations

Modèle de débruitage : implémentation

- Discrétisation spatiale de l'expression $2\left[\nabla^2 u^{(t)}(\boldsymbol{p}) - u^{(t)}(\boldsymbol{p}) + f(\boldsymbol{p})\right]$ $\rightarrow$vitesse en $\boldsymbol{p}$
- Principe : calcul de la vitesse en chaque point, puis ajout de la vitesse en chaque point (l'algorithme est parallélisable)
- Choix du critère d'arrêt : nombre d'itérations fixé

```
void smoothImageVariational(const ImageGrayscale *pImgInput,
                  ImageGrayscale *imgOutput, float delta_t, int nb_iter)
{
  unsigned int x, y, t;
  float **f, **u, **s;
  ImageGrayscale imgU, imgSpeed;

  copyImage(&imgU, pImgInput);
  initImage(&imgSpeed, pImgInput->width, pImgInput->height);

  f = pImgInput->pixels;
  u = imgU.pixels;
  s = imgSpeed.pixels;
```

Modèle de débruitage : implémentation

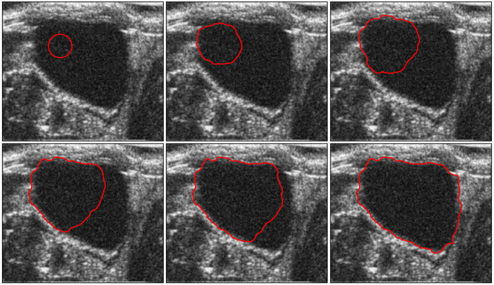
```
  for (t=0; t<nb_iter; t++)
  {
    for (y=1; y<pImgInput->height-1; y++)
      for (x=1; x<pImgInput->width-1; x++)
        s[x][y] = 2*((u[x+1][y] + u[x-1][y] + u[x][y+1] + u[x][y-1]
                  - 4*u[x][y]) - u[x][y] + f[x][y]);

    for (y=1; y<pImgInput->height-1; y++)
      for (x=1; x<pImgInput->width-1; x++)
        u[x][y] += delta_t*s[x][y];
  }

  copyImage(pImgOutput, &imgU);
  freeImage(&imgU);
  freeImage(&imgSpeed);
}
```

- Bords non traités dans cet exemple

Plan

Modèle de contour actif

- Principe : déformation d'une courbe de manière à segmenter (isoler) progressivement un objet en particulier dans l'image
- Initialisation fournie par l'utilisateur
- Connaissance *a priori* sur l'emplacement de l'objet
- Particulièrement utilisé en imagerie médicale

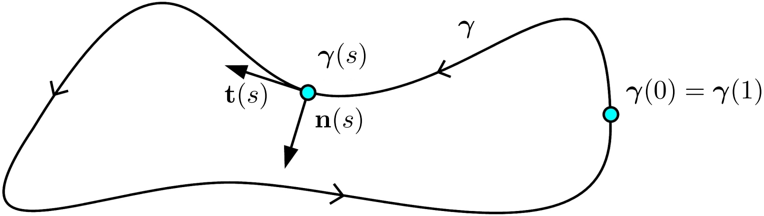


Modèle de contour actif

- L'énergie dépend d'une courbe *superposée* à l'image
- Soit γ une courbe de paramètre s :

$$\begin{aligned} \boldsymbol{\gamma} : [0,1] &\to \mathbb{R}^2 \\ s &\mapsto [\gamma_1(s)\ \gamma_2(s)]^\mathsf{T} \end{aligned}$$

- Courbe simple (sans intersection) et fermée : $\boldsymbol{\gamma}(0) = \boldsymbol{\gamma}(1)$



$\mathbf{t}$ = tangente, $\mathbf{n}$ = normale intérieure

Modèle de contour actif

- Objectif : segmentation d'un objet $\rightarrow$la courbe doit s'ajuster aux frontières de l'objet recherché
- Les contours de l'objet correspondent à des zones de fort gradient
- Recherche d'une courbe à la fois lisse et localisée sur les contours $\rightarrow$l'énergie pénalise une courbe étirée ou située sur des zones où la norme du gradient est faible

$$J[\boldsymbol{\gamma}] = \int_0^1 \alpha \frac{1}{2} \|\boldsymbol{\gamma}'(s)\|^2 - \|\nabla f(\boldsymbol{\gamma}(s))\| \,\mathrm{d}s$$

avec $\boldsymbol{\gamma}'(s) = \dfrac{\mathrm{d}\boldsymbol{\gamma}(s)}{\mathrm{d}s}$

- Version simplifiée du modèle original (qui comporte un terme dépendant de la dérivée seconde $\rightarrow$pénalisation de la courbure)
- Le calcul des variations donne :

$$\frac{\delta J[\boldsymbol{\gamma}]}{\delta \boldsymbol{\gamma}(s)} = -\alpha \boldsymbol{\gamma}''(s) - \nabla \|\nabla f(\boldsymbol{\gamma}(s))\|$$

Modèle de contour actif : discrétisation

- Equation d'évolution discrétisée en temps :

$$\boldsymbol{\gamma}^{(t+1)}(s) = \boldsymbol{\gamma}^{(t)}(s) + \Delta t \left[\alpha \boldsymbol{\gamma}^{(t)''}(s) + \nabla \left\| \nabla f(\boldsymbol{\gamma}^{(t)}(s)) \right\| \right]$$

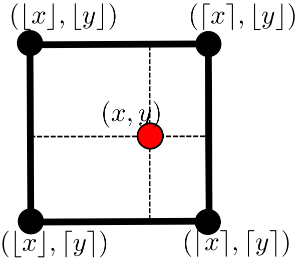
- Discrétisation spatiale $\rightarrow$échantillonnage de la courbe
- Représentation simple : polygone défini par une séquence de n sommets $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n\}$
- Les coordonnées des sommets sont réelles : $\mathbf{v}_i \in \mathbb{R}^2$
- Equation d'évolution discrétisée en temps et espace (sommet par sommet) :

$$\mathbf{v}_i^{(t+1)} = \mathbf{v}_i^{(t)} + \Delta t \left[\alpha \left(\mathbf{v}_{i+1}^{(t)} + \mathbf{v}_{i-1}^{(t)} - 2\mathbf{v}_i^{(t)} \right) + \nabla \left\| \nabla f(\mathbf{v}_i^{(t)}) \right\| \right]$$

Interpolation

- En pratique, la fonction image n'est donnée que pour des points de coordonnées entières (pixels)
- Pour approximer l'image aux points de coordonnées non-entières $\rightarrow$interpolation bilinéaire

$$\begin{array}{rcll} f(x,y) & \approx & (\lceil x\rceil - x)(\lceil y\rceil - y) & f(\lfloor x\rfloor, \lfloor y\rfloor) \\ & + & (x - \lfloor x\rfloor)(\lceil y\rceil - y) & f(\lceil x\rceil, \lfloor y\rfloor) \\ & + & (\lceil x\rceil - x)(y - \lfloor y\rfloor) & f(\lfloor x\rfloor, \lceil y\rceil) \\ & + & (x - \lfloor x\rfloor)(y - \lfloor y\rfloor) & f(\lceil x\rceil, \lceil y\rceil) \end{array}$$



Modèle de contour actif : implémentation

- Besoin de l'interpolation bilinéaire pour estimer $\|\nabla f\|$ et $\nabla\|\nabla f\|$ aux coordonnées des sommets

```
float interpImage(const ImageGrayscale *pImg, float x, float y)
{
  unsigned int xi, yi, xs, ys;
  float **f;

  xi = (unsigned int)floor(x); yi = (unsigned int)floor(y);
  xs = xi+1; ys = yi+1;
  f = pImg->pixels;

  return (xs-x)*(ys-y)*f[xi][yi] + (x-xi)*(ys-y)*f[xs][yi]
         + (xs-x)*(y-yi)*f[xi][ys] + (x-xi)*(y-yi)*f[xs][ys];
}
```

Modèle de contour actif : implémentation

- Besoin de calculer et conserver l'image $\|\nabla f\|$

```
void gradientNormImage(const ImageGrayscale *pImgInput,
                             ImageGrayscale *pImgGrad)
{
  unsigned int x,y;
  float **f, **g;
  float der_x, der_y;

  initImage(pImgGrad, pImgInput->width, pImgInput->height);
  f = pImgInput->pixels;
  g = pImgGrad->pixels;

  ...
  for (y=1; y<pImgInput->height-1; y++)
    for (x=1; x<pImgInput->width-1; x++)
    {
      der_x = (f[x+1][y]-f[x-1][y])/2;
      der_y = (f[x][y+1]-f[x][y-1])/2;
      g[x][y] = sqrt(der_x*der_x + der_y*der_y);
    }
}
```

Modèle de contour actif : implémentation

- Choix de représentation du contour : tableau de sommets

```
typedef struct {
  float x, y;
} Vertex;

typedef struct {
  Vertex *arrayVertices;
  unsigned int nbVertices;
  float delta_t;
  float alpha;
  ImageGrayscale *pImgGrad;
} ActiveContour;

void initActiveContourCircle(ActiveContour *, float, float, float);
void moveActiveContour(ActiveContour *);
void resampleActiveContour(ActiveContour *);
```

- Fonctions d'initialisation, d'évolution et de ré-échantillonnage (ajout et suppression de sommet afin de conserver une répartition homogène et suffisamment dense le long de la courbe)

Modèle de contour actif : implémentation

- Fonction d'initialisation en cercle

```
void initActiveContourCircle(ActiveContour *pAC,
                             float cx, float cy, float radius)
{
  unsigned int i;
  float _pi, angle, perimeter;

  _pi = 3.14159;
  perimeter = 2*_pi*radius;
  pAC->nbVertices = perimeter/3; /* One vertex every 3 pixels */
  pAC->arrayVertices =
    (Vertex *)malloc(pAC->nbVertices*sizeof(Vertex));

  for (i=0; i<pAC->nbVertices; i++)
  {
    angle = (float)i/(float)pAC->nbVertices*2*_pi;
    pAC->arrayVertices[i].x = cx + radius*cos(angle);
    pAC->arrayVertices[i].y = cy + radius*sin(angle);
  }
}
```

Modèle de contour actif : implémentation

```
void moveActiveContour(ActiveContour *pAC)
{
  unsigned int i, ip1, im1, n;
  Vertex *arraySpeed, *v;
  float x, y, grad_x, grad_y;

  arraySpeed =
    (Vertex *)malloc(pAC->nbVertices*sizeof(Vertex));
  v = pAC->arrayVertices;
  n = pAC->nbVertices;

  for (i=0; i<n; i++)
  {
    grad_x = (interpImage(pImgGrad, v[i].x+1.0, vi[i].y)
             -interpImage(pImgGrad, v[i].x-1.0, vi[i].y))/2;
    grad_y = (interpImage(pImgGrad, v[i].x, vi[i].y+1.0)
             -interpImage(pImgGrad, v[i].x, vi[i].y-1.0))/2;
```

Modèle de contour actif : implémentation

```
      im1 = (i+n-1)%n;
      ip1 = (i+1)%n;

      arraySpeed[i].x = alpha*(v[im1].x + v[ip1].x - 2*v[i].x) + grad_x;
      arraySpeed[i].y = alpha*(v[im1].y + v[ip1].y - 2*v[i].y) + grad_y;
    }

    for (i=0; i<n; i++)
    {
      v[i].x += delta_t*arraySpeed[i].x;
      v[i].y += delta_t*arraySpeed[i].y;
    }

    free(arraySpeed);
  }
```

Modèle de contour actif : implémentation

- Fonction principale

```
ImageGrayscale imgInput, imgGradNorm;
ActiveContour ac;
unsigned int t, nb_iter;

... /* Load input image into imgInput */

gradientNormImage(&imgInput, &imgGradNorm);
ac.alpha = 0.5;
ac.delta_t = 0.25;
ac.pImgGrad = &imgGradNorm;

initActiveContourCircle(&ac, ...);

for (t=0; t<nb_iter; t++)
{
  moveActiveContour(&ac);
  resampleActiveContour(&ac);
}

...
```