SAPIENZA
UNIVERSITÀ DI ROMA

UNIVERSITÀ DEGLI STUDI "LA SAPIENZA" DI ROMA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Master di II livello in Calcolo Scientifico

RAPPORTO FINALE DI TIROCINIO

Algoritmi bayesiani e adattivi per la segmentazione delle immagini

Candidato:
Francesco Cola
Matricola 1769172

Tutor accademico:
Prof. Camillo Cammarota

Tutor aziendale:
Prof.ssa Simona Perotto

Anno Accademico 2016 - 2017

Indice

Capitolo 1

Introduzione

Uno dei problemi classici nel mondo dell'Image Science è la segmentazione delle immagini, ovvero l'individuazione dei contorni degli oggetti raffigurati in un'immagine. In letteratura sono presenti due grandi categorie di metodi per risolvere questo tipo di problema: i metodi *edge-based* e i metodi *region-based*. In questo elaborato, presenterò il lavoro, su un metodo *region-based*, che ho svolto presso il Politecnico di Milano durante il mio tirocinio, con l'aiuto e la supervisione della professoressa Simona Perotto e del dottor Matteo Giacomini. I modelli e l'algoritmo descritti in questo testo sono una parte fondamentale di un loro articolo in fase di sottomissione.
Lo scopo di questo lavoro è stato quello di implementare un codice in FreeFem++ che sfruttasse la formulazione bayesiana del problema di segmentazione e la risoluzione di equazioni alle derivate parziali con l'utilizzo del metodo degli elementi finiti con mesh adattate e anisotrope. In particolare la formulazione bayesiana è una formulazione *region-based* che permette di dividere l'immagine I in diverse regioni. In ognuna di queste regioni la luminosità dei pixel avrà una determinata distribuzione statistica. La formulazione bayesiana permette di scrivere un funzionale che tiene conto di queste distribuzioni e il cui minimo è raggiunto nel momento in cui partizioniamo l'immagine in regioni i cui bordi sono proprio quei contorni degli oggetti che vogliamo identificare. Abbiamo scelto il modello bayesiano perché è semplice e in fase di implementazione ed esecuzione non richiede la calibrazione di parametri a differenza di altri modelli. Inoltre abbiamo usato mesh adattate e anisotrope, perché con queste mesh si riesce ad approssimare meglio funzioni poco regolari, specialmente nei punti in cui queste hanno un gradiente molto grande in modulo. Questa proprietà è molto utile nel problema di segmentazione, poiché le immagini, viste come superfici bidimensionali, hanno un gradiente molto alto proprio in prossimità dei contorni degli oggetti o delle figure che vogliamo individuare.

Il mio punto di partenza è stato un codice ibrido Matlab e FreeFem++ scritto da Matteo Giacomini. In questo codice era implementato un algoritmo basato su un funzionale misto: aveva una componente bayesiana e una componente RSFE, un altro modello sempre di tipo *region-based*. Nella prima parte di questo testo esporremo dal punto di vista matematico sia il modello basato sul funzionale RSFE che quello basato sul funzionale bayesiano. Successivamente vedremo come siamo passati dal modello ad un algoritmo, sottolineando le analogie e le differenze tra i due modelli. In questo capitolo parleremo in particolare del metodo di split di Bregman, dell'equazione di Eulero-Lagrange, dell'operatore di shrinkage

usati per minimizzare prima il funzionale RSFE e poi il funzionale bayesiano. Tratteremo in dettaglio il problema della costruzione di una mesh anisotropa, che si adatti al meglio alla soluzione finale del problema di minimizzazione, e vedremo anche il modo in cui questo algoritmo è stato implementato utilizzando l'ambiente e le funzioni offerte da FreeFem++. Infine, una volta arrivati ad un codice puramente bayesiano, adattivo e anisotropo, esporremo qualche risultato, prima su immagini test per calibrare i parametri e per verificare il corretto funzionamento del codice. Successivamente testeremo il codice su immagini reali per sperimentare le diverse versioni del codice. In particolare ci preme verificare se l'algoritmo bayesiano, adattivo con mesh anisotropa sia effettivamente efficace e migliore rispetto ad altri algoritmi più semplici.
Nel modello iniziale le diverse distribuzioni di probabilità dei pixel nelle diverse regioni sono state descritte con istogrammi. Più precisamente abbiamo usato delle funzioni di densità non parametriche ricavate a partire dagli istogrammi delle frequenze. Parte del lavoro che ho svolto è stato quello di passare all'uso di densità parametriche, ovvero di densità la cui forma è definita a priori. Durante l'esecuzione del codice, tramite il metodo di massima verosimiglianza, si stimano i parametri di modo che le densità usate approssimino al meglio le distribuzioni statistiche dell'immagine. Purtroppo, per limiti di tempo, siamo risuciti ad implementare solo densità gaussiane.

Capitolo 2

Segmentazione e modello matematico

Storicamente ci sono stati due diversi approcci alla segmentazione delle immagini [4]. Il primo approccio comprende un insieme di metodi detti *edge-based methods*. Questi metodi prevedono l'evoluzione nel tempo di una curva (solitamente il level set zero di una funzione) che, al raggiungimento di uno stato stazionario, identificherà i bordi dell'immagine. Un esempio classico di questo tipo di modelli è la segmentazione basata sull'evoluzione dell'equazione iconale nel tempo:

$$\partial_t u + c|\partial_x u| = 0 \qquad \text{dove } c = \frac{1}{1+|\nabla I|^p}$$

L'altro insieme di metodi sono invece chiamati *region-based methods*. Questi metodi prevedono di segmentare l'immagine in base alle proprietà delle regioni in cui essa sarà suddivisa. Un tipico esempio di questo tipo di modelli è quello di Mumford-Shah [11] oppure la sua versione semplificata, il modello di Chan-Vese [1]. Il modello di Mumford-Shah ha la seguente formulazione variazionale, e prevede la minimalizzazione del funzionale

$$\min_{u,K}\left\{\int_{\Omega\setminus K}(u-I)^2dx + \int_{\Omega\setminus K}|\nabla u|^2dx + \mathcal{H}^1(K)\right\}$$

dove $K \subset \Omega$ è un sottoinsieme del dominio che indentifica i bordi dell'immagine, cioè i punti di discontinuità della funzione intensità luminosa I, mentre $u \in H^1(\Omega \setminus K)$ e rappresenta la parte *regolare* della funzione I.
Nel nostro lavoro abbiamo usato modelli del tipo *region-based*, in particolare un modello basato sulla minimizzazione del funzionale Region-Scalable Fitting Energy e un modello bayesiano.

2.1 Funzionale Region-Scalable Fitting Energy

Descriviamo ora brevemente il funzionale Region-Scalable Fitting Energy (RSFE) [10], [7] e analizziamo il motivo per cui la sua minimizzazione porta ad ottenere la segmentazione di un'immagine.
Data un'immagine, essa può essere rappresentata da un insieme $\Omega \subset \mathbb{R}^2$ e da una funzione $I : \Omega \longrightarrow \mathbb{R}$, dove Ω e I sono rispettivamente il domino e l'intensità luminosa dell'immagine. Consideriamo una funzione $\phi : \Omega \longrightarrow \mathbb{R}$ il cui insieme

di livello 0 rappresenta i bordi delle figure all'interno dell'immagine. Stiamo descrivendo in tal modo i contorni delle figure in modo implicito, quindi vorremmo che la funzione ϕ rispetti le seguenti condizioni

$$\begin{cases} \phi(x) > 0 & x \in \Omega_I \\ \phi(x) = 0 & x \in \partial\Omega_I \\ \phi(x) < 0 & x \in \Omega_E \end{cases}$$

dove Ω_I e Ω_E sono rispettivamente la regione interna e la regione esterna della figura e $\partial\Omega_I$ rappresenta i bordi. In tal modo, assegnando il valore della funzione ϕ in ogni punto del dominio, otteniamo una possibile segmentazione dell'immagine.

Ora vogliamo costruire un funzionale $\mathcal{F}_{RSFE}$ il cui minimo sia raggiunto proprio per quella funzione ϕ che meglio rappresenta la segmentazione "reale" dell'immagine. Il funzionale sarà costituito da tre termini, i primi due saranno i cosiddetti termini di fedeltà e il terzo sarà un termine di regolarizzazione del bordo. I termini di fedeltà indicano per la regione interna e per quella esterna, quanto il valore dei pixel si discosta dalla media dei valori vicini. Se l'intensità luminosa di un pixel è sensibilmente diversa dal valore medio dei pixel vicini ad esso ed appartenenti alla stessa regione (cioè in cui ϕ ha lo stesso segno) significherà che il pixel, probabilmente, apparterrà all'altra regione. Per questo motivo, il termine di fedeltà in ogni pixel della regione interna sarà calcolato come la media ponderata tramite una gaussiana delle differenze al quadrato tra l'intensità luminosa del pixel, e la media dei pixel vicini e interni alla regione. Quindi per ogni $x \in \Omega_I$ abbiamo

$$e_I(x) = \int_\Omega K_\sigma(x-y)\left|I(x) - \frac{K_\sigma(y) * H(\phi(y))I(y)}{K_\sigma(y) * H(\phi(x))}\right|^2 dy \tag{2.1}$$

dove $*$ rappresenta il prodotto di convoluzione, H è la funzione di Heaviside e K_σ è il kernel gaussiano di media zero e varianza σ^2

$$K_\sigma(y) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(\frac{|x|^2}{2\sigma^2}\right)$$

Analogamente per la regione esterna avremo che per ogni $x \in \Omega_E$

$$e_E(x) = \int_\Omega K_\sigma(x-y)\left|I(x) - \frac{K_\sigma(y) * (1-H(\phi(y)))I(y)}{K_\sigma(y) * (1-H(\phi(x)))}\right|^2 dy \tag{2.2}$$

Infine sommando il contributo di tutti i punti di Ω (i pixel) e pesando in modo diverso la parte interna dalla parte esterna, otteniamo il seguente funzionale

$$\mathcal{F}_{\mathrm{RSFE}}(\phi) = \mu_I \int_\Omega e_I(x)\phi(x)dx - \mu_E \int_\Omega e_E(x)\phi(x)dx + \\ + \nu \int_\Omega \frac{|\nabla\phi(x)|}{1+\beta|\nabla I(x)|^2}dx$$

l'ultimo termine del funzionale si chiama *Total Variation* e serve per regolarizzare il bordo. Quindi per ottenere la segmentazione dell'immagine cerchiamo una funzione ϕ^* tale che

$$\phi^* = \underset{\phi\in\mathcal{A}}{\operatorname{argmin}}\Big\{\mathcal{F}_{\mathrm{RSFE}}(\phi)\Big\} \tag{2.3}$$

dove

$$\mathcal{A} = \left\{ \phi \in H^1(\Omega) \,\middle|\, -\alpha \leq \phi \leq \alpha \text{ q.o. su } \Omega,\ \nabla\phi \cdot \mathbf{n} = 0 \text{ su } \partial\Omega \right\}$$

è l'iniseme delle funzioni ammissibili. Abbiamo scelto funzioni limitate tra $-\alpha$ e α perché in realtà per il nostro scopo non serve sapere l'esatto valore della funzione ϕ ma solo quando è positiva o negativa, inoltre non conoscendo come siano i bordi della figura, imponiamo al bordo delle condizioni di Neumann omogenee, che sembrano le più naturali possibili.

In buona sostanza il modello RSFE è una evoluzione del modello di Chan-Vese, i termini di fedeltà sono calcolati come la distanza dai valori medi dei pixel vicini anzichè come la distanza dal valore medio dei pixel su tutta la regione. Questo modello dovrebbe essere migliore perché, per effetto di ombre o sfumature, l'intesità luminosa dei pixel all'interno di una sagoma potrebbe cambiare sensibilmente, senza però dare luogo a bordi netti.
Sebbene questo modello sia stato il nostro punto di partenza, a causa dei numerosi parametri e della difficile calibrazione di essi, siamo passati gradualmente al modello bayesiano: dapprima usando un funzionale ibrido e poi abbandonando definitivamente il funzionale RSFE.

2.2 Modello bayesiano

Data un'immagine I, sia $\mathcal{P}(\Omega)$ una partizione dell'immagine, il modello bayesiano [2], [4], [12], [9] definisce la probabilità con cui questa partizione è una "corretta" segmentazione dell'immagine. Pertanto per trovare la miglior segmentazione vogliamo trovare quella partizione che, data un'immagine, massimizza a posteriori la probabilità

$$\max\left\{ p(\mathcal{P}(\Omega)|I) \right\}$$

Nel 1996 Zhu e Yuille hanno dimostrato che questo approccio è del tutto equivalente al modello di Mumford-Shah.

Ora proviamo a ricondurre questo modello alla minimizzazione di un funzionale. Innanzitutto notiamo come massimizzare una probabilità sia equivalemte a minimizzare l'opposto del logaritmo della probabilità

$$\max\left\{ p(\mathcal{P}(\Omega)|I) \right\} = \min\left\{ -\log p(\mathcal{P}(\Omega)|I) \right\}$$

a questo punto, visto che stiamo minimizzando una probabilità condizionata, possiamo provare ad applicare il teorema di Bayes per separare le informazioni che conosciamo a priori da quelle che invece conosceremo solo a posteriori. Dati due eventi A e B il teorema di Bayes afferma che

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \qquad p(B) \neq 0$$

applicando il teorema al nostro caso otteniamo che

$$p(\mathcal{P}(\Omega)|I) = \frac{p(I|\mathcal{P}(\Omega))p(\mathcal{P}(\Omega))}{p(I)}.$$

Non prenderemo in considerazione la probabilità $p(I)$ di avere una determinata immagine, essa è solamente una costante che non ci interessa determinare, poiché è irrilevante ai fini della minimizzazione. $p(\mathcal{P}(\Omega))$ è la probabilità di avere una

determinata partizione a prescindere dall'immagine. Includiamo in questa probabilità il termine di regolarizzazione: vogliamo che la partizione abbia un bordo regolare e possibilmente corto. La scelta migliore per penalizzare le partizioni con bordi lunghi e frastagliati è porre

$$p(\mathcal{P}(\Omega)) \propto e^{-\nu TV_g(\phi)}$$

dove TV_g è il termine di *Total Variation*

$$TV_g(\nabla\phi) = \int_\Omega g(I(x))|\nabla\phi|^2 dx \qquad\qquad g(y) = \frac{1}{1+\beta|\nabla y|^2}$$

Infine $p(I|\mathcal{P}(\Omega))$ è la probabilità di avere una determinata immagine data una partizione a priori. L'utilità dell'utilizzo del teorema di Bayes sta proprio nel fatto che è più comodo assegnare una probabilità ad un'immagine, data una partizione, piuttosto che viceversa. Supponendo che non ci sia correlazione tra le due regioni in cui divido la partizione possiamo affermare che

$$p(I|\mathcal{P}(\Omega)) = p(I|\{\Omega_I, \Omega_E\}) = p(I|\Omega_I)\, p(I|\Omega_E)$$

Infine, supponendo che i pixel all'interno di una stessa regione non siano altro che realizzazioni indipendenti e identicamente distribuite (iid) di una stessa variabile aleatoria con densità di probabilità $f_i(x)$ con $i = I, E$ otteniamo che il metodo bayesiano consiste nella minimizzazione della seguente quantità

$$-\prod_{x\in\Omega_I} f_I(I(x)) \prod_{x\in\Omega_E} f_E(I(x))\, e^{\nu TV_g(\phi(x))}$$

da cui, passando ai logaritmi, otteniamo il seguente funzionale

$$\mathcal{F}_B(\mathcal{P}(\Omega)) = -\int_{\Omega_I} \phi(x) \log f_I(I(x))dx + \\ + \int_{\Omega_E} \phi(x) \log f_E(I(x))dx + \nu TV_g(\nabla\phi(x))dx \quad (2.4)$$

I primi due termini sono i termini di fedeltà della regione interna e di quella esterna, mentre l'ultimo termine serve per regolarizzare il contorno. Anche in questo caso come nel caso RSFE, per le stesse motivazioni, restringiamo lo spazio delle possibili soluzioni all'insieme

$$\mathcal{A} = \left\{\phi \in H^1(\Omega) \,\middle|\, -\alpha \le \phi \le \alpha \text{ q.o. su } \Omega,\ \nabla\phi\cdot\mathbf{n} = 0 \text{ su } \partial\Omega\right\}$$

Ora manca solo una questione da risolvere, come è possibile definire le densità di probabilità interna f_I e esterna f_E? Le possibilità sono due e le abbiamo utilizzate entrambe. La prima è quella di stimare le densità di probabilità con un metodo non parametrico, ovvero, poiché il numero dei pixel è elevato, si può supporre che l'istogramma sia una buona approssimazione della densità di probabilità. In questo caso scegliamo

$$f_i(s) = K_\tau * \left(\frac{1}{|\Omega_i|}\int_{\Omega_i} \mathbb{1}_{\{I(x)=s\}} dx\right)$$

ove $\mathbb{1}_S$ è la funzione indicatrice dell'insieme S, $|S|$ è la misura dell'insieme S e K_τ è un kernel gaussiano di media 0 e varianza τ^2 che usiamo per rendere più liscio l'istogramma prima di usarlo come densità di probabilità.
L'altra possibilità è quella di usare una densità parametrica, ovvero definire a priori la forma della densità e poi di stimarne i parametri. Ad esempio si può supporre che i pixel all'interno di una stessa regione siano generati da una distribuzione normale e quindi che le densità f_i con $i = I, E$ abbiano la seguente forma

$$f_i(s) = \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left(\frac{|x - \mu_i|^2}{2\sigma_i^2}\right)$$

in tal modo la minimizzazione del funzionale $\mathcal{F}_B$ dipenderà anche dai parametri μ_i e σ_i oltre che dalla funzione ϕ. Il vantaggio di questo modello è che non contiene parametri da stimare o da calibrare, il che rende il suo utilizzo più facile ed immediato. Inoltre le conoscenze a priori sull'immagine, come ad esempio il tipo di rumore, possono essere tenute in considerazione quando scegliamo una densità di probabilità parametrica, in modo che questa si adatti il più possibile al tipo di immagine che vogliamo segmentare.

Capitolo 3

Algoritmo e implementazione

Lo scopo di questo capitolo sarà quello di descrivere il metodo risolutivo usato concretamente per minimizzare i due funzionali, quello RSFE e quello bayesiano. Innanzitutto facciamo notare che in realtà i due funzionali sono molto simili. Infatti possono essere scritti quasi allo stesso modo. Se poniamo

$$r_{\mathrm{RSFE}}(x) = \mu_I e_I(\phi(x)) - \mu_E e_E(\phi(x)) \tag{3.1}$$

allora il funzionale RSFE può essere scritto nel seguente modo

$$\mathcal{F}_{\mathrm{RSFE}}(\phi) = \langle \phi, r_{\mathrm{RSFE}} \rangle + \nu T V_g(\nabla \phi)$$

dove $\langle \quad , \quad \rangle$ indica il prodotto scalare nello spazio $L^2(\Omega)$.
Analogamente se poniamo

$$r_B(x) = -H(\phi) \log f_I(I(x)) + \big(H(\phi(x) - 1)\big) \log f_E(I(x)) \tag{3.2}$$

otteniamo che il funzionale bayesiano si scrive allo stesso modo.

$$\mathcal{F}_B(\phi) = \langle \phi, r_B \rangle + \nu T V_g(\nabla \phi)$$

Visto che i due funzionali si scrivono praticamente allo stesso modo li risolveremo usando lo stesso algoritmo, l'unica differenza sarà il calcolo del residuo r_{RSFE} o r_B a seconda del modello usato. Per questo motivo d'ora in poi indicheremo con r il residuo senza fare distinzione tra il tipo di funzionale utilizzato.

3.1 Algoritmo

Nel calcolo delle variazioni, uno dei metodi più semplici per trovare il minimo di un funzionale è usare l'equazione di Eulero-Lagrange. Tuttavia il termine di regolarizzazione della "total variation" potrebbe creare dei problemi, perché a tutti gli effetti è una norma $L^1(\Omega)$. Pertanto prima di ricavare l'equazione di Eulero-Lagrange utilizziamo l'algoritmo di split di Bregman [8], [7], metodo ampiamente diffuso in letteratura per la risoluzione di questo tipo di problemi di minimizzazione.
Per prima cosa introduciamo una variabile ausiliaria $\mathbf{d}$ che rappresenta il gradiente della funzione ϕ. In questo modo il funzionale da minimizzare diventa

$$\min_{\phi,d} \left\{ \langle \phi, r \rangle + \nu T V_g(\mathbf{d}) + \frac{\mu}{2} \|\mathbf{d} - \nabla \phi\|^2_{L^2(\Omega)} \right\}$$

dove $\mu > 0$ è un moltiplicatore di Lagrange e l'ultimo termine è la norma $L^2(\Omega)$ che garantisce il rispetto del vincolo $\mathbf{d}(x) = \nabla\phi(x)$ per quasi ogni x. Ora rilassiamo le condizioni del vincolo, permettendo che $\mathbf{d}(x) - \nabla\phi(x) = \mathbf{b}(x)$, per poter risolvere il problema di minimizzazione in modo iterativo. In questo modo possiamo trovare il minimo del funzionale minimizzando prima la coppia $(\phi, \mathbf{d})$ e successivamente aggiornando il termine di rilassamento $\mathbf{b}$.

$$
\begin{aligned}
(\phi^{k+1}, \mathbf{d}^{k+1}) &= \min_{\phi,\mathbf{d}} \left\{ \left\langle \phi, r^k \right\rangle + \nu TV_g(\mathbf{d}) + \frac{\mu}{2} \left\| \mathbf{d} - \nabla\phi \right\|^2_{L^2(\Omega)} \right\} \\
\mathbf{b}^{k+1} &= \mathbf{b}^k + (\nabla\phi^{k+1} - \mathbf{d}^{k+1})
\end{aligned}
$$

A questo punto introduciamo l'algoritmo di split di Bregman, che ci permette di aggiornare in modo alternato i valori di ϕ^k e $\mathbf{d}^k$ che convergono al minimo del funzionale. Nel funzionale si separa la parte in norma $L^2(\Omega)$ dalla parte in norma $L^1(\Omega)$ mantenendo in entrambi il vincolo sul gradiente imposto precedentemente. Quindi si minimizza prima la norma $L^2(\Omega)$ rispetto alla funzione ϕ, successivamente la norma $L^1(\Omega)$ rispetto al suo gradiente $\mathbf{d}$ e infine si aggiorna il termine di rilassamento $\mathbf{b}$.

$$
\begin{aligned}
\phi^{k+1} &= \min_{\phi} \left\{ \left\langle \phi, r^k \right\rangle + \frac{\mu}{2} \left\| \mathbf{d}^k - \nabla\phi - \mathbf{b}^k \right\|^2_{L^2(\Omega)} \right\} \\
\mathbf{d}^{k+1} &= \min_{\mathbf{d}} \left\{ \nu TV_g(\mathbf{d}) + \frac{\mu}{2} \left\| \mathbf{d} - \nabla\phi^{k+1} - \mathbf{b}^k \right\|^2_{L^2(\Omega)} \right\} \\
\mathbf{b}^{k+1} &= \mathbf{b}^k + (\nabla\phi^{k+1} - \mathbf{d}^{k+1})
\end{aligned}
$$

A questo punto per minimizzare i due funzionali spezzati possiamo procedere utilizzando l'equazione di Eulero-Lagrange, che riportiamo di seguito per il caso n-dimensionale

$$
\mathcal{L}_\phi - \sum_{i=1}^{n} \partial_{x_i} \mathcal{L}_{\phi_{x_i}} = 0 \qquad \text{dove} \qquad \mathcal{L}_\phi = \frac{\partial \mathcal{L}}{\partial \phi}, \quad \mathcal{L}_{\phi_{x_i}} = \frac{\partial \mathcal{L}}{\partial \phi_{x_i}}, \quad \phi_{x_i} = \frac{\partial \phi}{\partial x_i}
$$

Nel primo caso, per trovare ϕ^{k+1}, considerando come lagrangiana

$$
\mathcal{L}(\phi, \nabla\phi, x) = \phi(x) r^k(x) + \frac{\mu}{2} \left(\left(d_1^k(x) - \partial_{x_1}\phi - b_1^k(x) \right)^2 + \left(d_2^k(x) - \partial_{x_2}\phi - b_2^k(x) \right)^2 \right)
$$

otteniamo la seguente equazione alle derivate parziali

$$
-\Delta\phi^{k+1} = -\frac{1}{\mu} r^k + \operatorname{div}\left(\mathbf{b}^k - \mathbf{d}^k\right)
$$

alla quale dobbiamo accoppiare le condizioni al bordo di Neumann omogenee e la costrizione che $|\phi| \leq \alpha$ quasi ovunque su Ω che derivano dalle condizioni che abbiamo imposto all'insieme delle soluzioni ammissibili $\mathcal{A}$. Purtroppo, anche non considerando la limitatezza della funzione a valori nell'intervallo $[-\alpha, \alpha]$ questa equazione non ha soluzione.
Abbiamo quindi interpretato la precedente equazione di Poisson con condizioni al bordo di Neuamann omogeee come lo stato stazionario di un'equazione del calore (parabolica) con le stesse condizioni.

$$
\begin{cases}
\partial_t \Phi - \Delta\Phi = -\frac{1}{\mu} r^k + \operatorname{div}\left(\mathbf{b}^k - \mathbf{d}^k\right) & \text{su } \Omega \times (0, T] \\
\nabla\Phi \cdot \mathbf{n} = 0 & \text{su } \partial\Omega \times (0, T] \\
\Phi(\cdot, 0) = \phi^k & \text{su } \bar{\Omega} \times \{0\}
\end{cases} \tag{3.3}
$$

Una volta risolta l'equazione precedente porremo $\phi^{k+1}(x) = \Phi(x, T)$. Ma quale dovrebbe essere il giusto valore di T affinchè $\Phi(x, T)$ sia una soluzione stazionaria? Poiché stiamo sviluppando un algoritmo iterativo, non ci interessa arrivare alla soluzione in un solo passo, pertanto ad ogni iterazione calcoleremo la soluzione dell'equazione del calore prendendo un intervallo temporale $(0, T]$ fissato. Ovviamente, per terminare l'algoritmo il criterio di stop sarà strettamente legato allo stato stazionario di questa equazione del calore. Dopo ogni iterazione dell'equazione del calore tronchiamo la funzione ϕ, ovvero poniamo

$$\phi(x) = \begin{cases} -\alpha & \phi(x) < -\alpha \\ \phi(x) & \phi(x) \in [-\alpha, \alpha] \\ \alpha & \phi(x) > \alpha \end{cases}$$

In questo modo a fine algoritmo la funzione ϕ apparterrà all'insieme delle soluzioni ammissibili $\mathcal{A}$, ricordando che non è importante il valore della funzione ϕ nella regione interna od esterna ma il suo segno.

Nel secondo caso, per trovare $\mathbf{d}^{k+1}$, consideriamo come lagrangiana

$$\begin{aligned} \mathcal{L}(\mathbf{d}, \nabla\mathbf{d}, x) = \frac{\nu|\mathbf{d}|}{1 + \beta|\nabla I(x)|^2} + \\ + \frac{\mu}{2}\left(\left(d_1(x) - \partial_{x_1}\phi^{k+1} - b_1^k(x)\right)^2 + \left(d_2(x) - \partial_{x_2}\phi^{k+1} - b_2^k(x)\right)^2\right) \end{aligned}$$

come si può facilmente notare la lagrangiana in questo caso non dipende da $\nabla\mathbf{d}$, pertanto si può trovare il minimo semplicemente ponendo $\mathcal{L}_{d_1} = 0$ e $\mathcal{L}_{d_2} = 0$. Otteniamo così un sistema di due equazioni e due incognite, le due componenti del vettore $\mathbf{d}$

$$\begin{cases} \frac{\nu g(I)}{\mu}\frac{d_1}{|\mathbf{d}|} + d_1 - \partial_{x_1}\phi^{k+1} - b_1^k = 0 \\ \frac{\nu g(I)}{\mu}\frac{d_2}{|\mathbf{d}|} + d_2 - \partial_{x_2}\phi^{k+1} - b_2^k = 0 \end{cases}$$

Questo sistema si risolve in modo diretto, prima ponendo $\mathbf{f} = \nabla\phi^{k+1} + \mathbf{b}^k$ e $\gamma = \frac{\nu}{\mu}g(I)$ e successivamente passando a variabili polari.

$$\begin{cases} d_1 = |\mathbf{d}|\cos(\theta) \\ d_2 = |\mathbf{d}|\sin(\theta) \end{cases}$$

la soluzione che si ottiene è facile e in letteratura prende il nome di *shrinkage*.

$$\mathbf{d}^{k+1} = \text{shrink}\Big(\mathbf{b}^k + \nabla\phi^{k+1}, \frac{\nu}{\mu}g(I)\Big) \tag{3.4}$$

dove l'operatore di *shrinkage* è così definito:

$$\text{shrink}(\mathbf{f}, \gamma) = \frac{\mathbf{f}}{|\mathbf{f}|}\max\big(|\mathbf{f}| - \gamma, 0\big)$$

Da notare come le quantità $\mathbf{f}$ e γ sono le stesse che abbiamo usato per risolvere il sistema precedente.

Riassumiamo ora quello che sarà il nostro algoritmo per trovare la segmentazione dell'immagine I, utilizzando il metodo di split di Bregman. Definiamo un

contorno iniziale attraverso una funzione $\phi^0 = \phi_0$, poniamo $\mathbf{d}^0 = \nabla\phi_0$ e $\mathbf{b}^0 = 0$ e siamo pronti per iniziare le iterazioni. Ad ogni iterazione calcoliamo la funzione ϕ^{k+1} risolvendo uno step dell'equazione del calore, poi attraverso l'operatore di shrinkage aggiorniamo il termine $\mathbf{d}^{k+1}$ e successivamente il termine di rilassamento $\mathbf{b}^{k+1}$. A questo punto verifichiamo il criterio di stop e, se non è rispettato, eseguiamo una nuova iterazione. Da notare che ad ogni iterazione il residuo r cambia perché cambia la funzione ϕ che determina i bordi dell'immagine.

Per quanto riguarda il criterio di stop, abbiamo usato due metodi diversi a seconda del tipo di funzionale usato. Nel caso del funzionale RSFE, fissata una tolleranza "tol", abbiamo usato come criterio di stop l'errore relativo in norma L^2 tra un'iterazione e la successiva

$$\frac{\|\phi^{k+1} - \phi^k\|_{L^2(\Omega)}}{\|\phi^k\|_{L^2(\Omega)}} < \text{tol}$$

per quanto riguarda il funzionale bayesiano invece abbiamo elaborato un sistema più sofisticato, abbiamo l'errore in norma L^2 tra una iterazione e la successiva del modulo della differenza tra le probabilità interna ed esterna. In formula risulta più semplice

$$\big\| |f_I^{k+1}(I) - f_E^{k+1}(I)| - |f_I^k(I) - f_E^k(I)| \big\|_{L^2(\Omega)} < \text{tol}$$

dove f_I^k e f_E^k sono rispettivamente le densità di probabilità interna ed esterna alla k-esima iterazione.

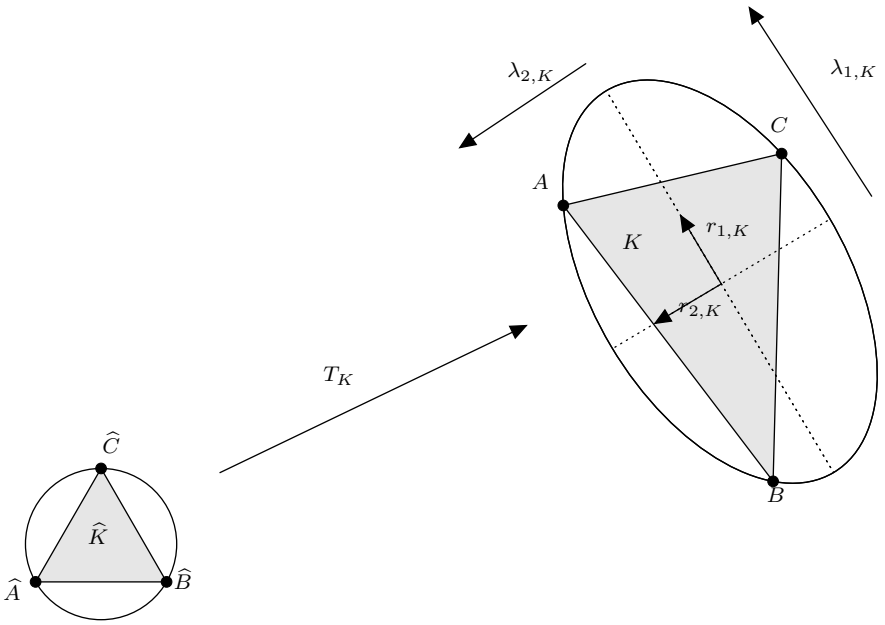
3.2 Mesh isotrope e anisotrope

Per risolvere l'equazione del calore abbiamo usato il framework degli elementi finiti. In particolare abbiamo usato il programma FreeFem++ che dispone di un'ampia gamma di strumenti per trattare gli elementi finiti e non solo. Un aspetto importante per cui si sceglie il metodo degli elementi finiti anziché le differenze finite è la possibilità di utilizzare mesh non regolari. Il metodo degli elementi finiti (FEM) infatti ci permette di usare oltre alla classica mesh regolare, costituita da nodi equispaziati e da elementi regolari (stessa forma e dimensione) anche mesh adattate isotrope o anisotrope [5], [6]. Una mesh *isotropa adattata* è una mesh i cui elementi hanno tutti la stessa forma, ma diversa dimensione, per poter catturare meglio il dettaglio della soluzione nei punti dove essa è più irregolare. Una mesh *anisotropa adattata* è un mesh i cui elementi hanno forma e dimensione diversa, per poter descrivere la soluzione con maggior dettaglio tenendo conto anche della direzione di massima pendenza e non solo dei punti di maggior irregolarità. Lo scopo delle mesh adattate alla soluzione è quello di usare il minor numero di elementi finiti per descrivere la soluzione in modo che l'errore sia equidistribuito su tutti gli elementi.

Ogni triangolo della mesh adattata o no, isotropa o anisotropa è legato al triangolo o elemento di riferimento da una mappa affine. La scelta più comune, come triangolo di riferimento, è il triangolo $\widehat{K}$ di vertici $A(-\sqrt{3}/2, -1/2)$, $B(\sqrt{3}/2, -1/2)$ e $C(0, 1)$. Allora per ogni elemento $K \in \mathcal{T}_h$ esite una mappa affine $T_K \colon \widehat{K} \longrightarrow K$ tale che

$$\forall x \in K, \quad \exists!\, \widehat{x} \in \widehat{K} \quad \text{tale che} \quad x = T_K\widehat{x} = A_K\widehat{x} + v_K$$

dove $A_K \in \mathbb{R}^{2\times 2}$ è la parte lineare della trasformazione mentre $v_K \in \mathbb{R}^2$ è la traslazione.



L'immagine sopra mostra come agisce la mappa T_K sul triangolo di riferimento $\widehat{K}$ per ottere il triangolo K. Se fattorizziamo la matrice A_K usando la decomposizione polare otteniamo che $A_K = B_K\, Z_B$ dove Z_K è una matrice ortonormale mentre B_K è una matrice simmetrica definita positiva. Per il teorema spettrale, la matrice B_K può essere ulteriromente scomposta

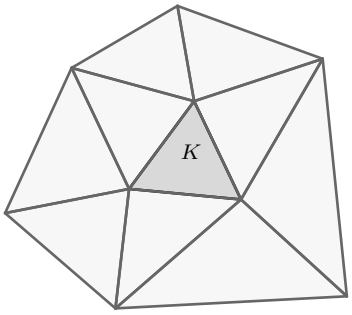
$$B_K = R_K^T\, \lambda_K\, R_K$$

dove

$$\Lambda_K = \begin{bmatrix} \lambda_{1,K} & 0 \\ 0 & \lambda_{2,K} \end{bmatrix} \qquad R_K^T = \begin{bmatrix} r^1_{1,K} & r^1_{2,K} \\ r^2_{1,K} & r^2_{2,K} \end{bmatrix} \quad \mathbf{r}_{1,K} = \begin{bmatrix} r^1_{1,K} \\ r^2_{1,K} \end{bmatrix} \quad \mathbf{r}_{2,K} = \begin{bmatrix} r^1_{2,K} \\ r^2_{2,K} \end{bmatrix} \tag{3.5}$$

Λ_K è la matrice diagonale contente gli autovalori di B_K, mentre R_K^T è la matrice ortonormale le cui colonne sono gli autovettori relativi agli autovalori della matrice B_K. Quindi abbiamo fattorizzato $A_K = R_K^T\, \Lambda_K\, R_K\, Z_K$. La matrice R_K^T è una matrice ortonormale e contiene le direzioni lungo le quali il triangolo subisce la dilatazione più ampia e la dilatazione più piccola, la matrice Λ_K contiene i fattori di dilatazione lungo le direzioni prestabilite dalla matrice R_K^T, infine la matrice Z_K contiene la rotazione che subisce il triangolo. Per lo sviluppo di una mesh anisotropa non ci interessa l'orientazione del triangolo, ma solamente le direzioni e i fattori di stretching di esso, contenuti esclusivamente nella matrice $B_K = R_K^T\, \Lambda_K\, R_K$.
Al fine di elaborare una mesh anisotropa, la matrice B_K può essere vista come una metrica, ovvero un tensore definito sul dominio Ω che punto per punto rappresenta il prodotto scalare e che può essere espresso come una matrice simmetrica definita positiva. Non parleremo del modo in cui si possa costruire una

mesh a partire da una metrica, perché è abbastanza complicato e inoltre il programma FreeFem++ fornisce la funzione `adaptmesh` che svolge questo compito in automatico, ma ci concentreremo solo sulla costruzione della metrica.
Lo scopo di avere una mesh adattata anisotropa è quello di approssimare al meglio la soluzione esatta ϕ^*. Se indichiamo con ϕ_h la soluzione approssimata ad una qualche iterazione vorremo quindi che l'errore H^1 sia il più piccolo possibile. In particolare vorremo che la componente L^2 dell'errore sul gradiente sia piccola, perché l'errore $\|\phi_h - \phi^*\|_{L^2}$ è generalmente più piccolo. Quindi vogliamo costruire una mesh tale che l'errore $\|\nabla\phi_h - \nabla\phi^*\|_{L^2}$ sia il più piccolo possibile. Ora per ridurre l'errore ci sono due possibilità, o amunentare il numero degli elementi della mesh, renderli più piccoli per cogliere meglio le variazioni di ϕ^* oppure adattare la mesh alla soluzione ϕ^*, in paricolar modo dobbiamo infittirla laddove commettiamo un errore maggiore. Ora noi non abbiamo il valore esatto di ϕ^* per cui dobbiamo usare una stima a posteriori, lo stimatore dell'errore di Zienkeiwicz-Zhu basato sulla ricostruzione del gradiente [5], [6]. Nel nostro modello la funzione ϕ_h è una funzione lineare a tratti, ovvero appartiene a $\mathbb{P}_1(\mathcal{T}_h)$. Pertatno il gradiente, che è il vettore delle derivate di ϕ_h sarà una funzione vettoriale costante su ogni elemento della mesh. Il gradiente ricostruito $P^K_{ZZ}\phi_h$ su un triangolo K è definito come la media pesata del gradiente di ϕ_h sui triangoli adiacenti. L'insieme dei triangoli adiacenti a K, chiamato *patch*, è indicato con il simbolo Δ_K ed è l'insieme di tutti i triangoli che hanno in comune almeno un vertice con il triangolo K.



Δ_K

Di seguito riportiamo la formula del gradiente ricostruito

$$P^K_{ZZ}\phi_h = \frac{1}{|\Delta_K|}\sum_{T\in\Delta_K} |T|\,\nabla\phi_h\Big|_T$$

Nonostante il gradiente ricostruito sia comunque una costante e non una funzione lineare, nonostante sia soltano una media pesata dei valori nei triangoli vicini, le simulazioni dimostrano che effettivamente funziona e riesce in qualche modo a raccogliere più informazione del gradiente approssimato $\nabla\phi_h$. Ora non ci resta che definire lo stimatore ZZ dell'errore basato sul gradiente ricostruito:

$$\left(\eta^I_K\right)^2 = \frac{|K|}{|\Delta_K|}\int_{\Delta_K}\left|P^K_{ZZ}\phi_h - \nabla\phi_h\right|^2 dx$$

la stima dell'errore η^I_K sul K-esimo elemento è semplicemente una norma L^2 fatta però su tutta la patch Δ_K e non solo sul singolo triangolo K, per questo motivo abbiamo introdotto il fattore di scaling $|K|/|\Delta_K|$. Ovviamente la stima

dell'errore su tutto il dominio Ω sarà

$$\left(\eta^I\right)^2 = \sum_{T\in\mathcal{T}_h}\left(\eta_T^I\right)^2$$

dove la I sta ad indicare la versione isotropica. Se vogliamo la versione anisotropica dobbiamo introdurre la metrica, e quindi il prodotto scalare su ogni triangolo K. Per ottenere la versione anisotropica dello stimatore ZZ dell'errore si scompone il vettore $P^K_{ZZ}\phi_h - \nabla\phi_h$ lungo i due autovettori ortonormali della metrica $\mathbf{r}_{1,K}$ e $\mathbf{r}_{2,K}$, si calcola la norma L^2 al quadrato, si moltiplicano per gli autovalori $\lambda_{1,K}$ e $\lambda_{2,K}$ della metrica e infine si sommano le componenti. Naturalmente bisognerà moltiplicare per un fattore di scaling che in questo caso sarà $(\lambda_{1,K}\lambda_{2,K})^{-1}$.

$$\left(\eta_k^A\right)^2 = \frac{|K|}{|\Delta_K|}\frac{1}{\lambda_{1,K}\lambda_{2,K}}\sum_{i=1}^{2}\lambda_{i,K}^2\int_{\Delta_K}\left|\left(P^K_{ZZ}\phi_h - \nabla\phi_h\right)\mathbf{r}_{i,K}\right|^2 dx \tag{3.6}$$

come si può facilmente calcolare ponendo $\lambda_{1,K} = \lambda_{2,K}$ si riottiene la versione isotropica dello stimatore ZZ. E, analogamente al caso isotropico, abbiamo che

$$\left(\eta^A\right)^2 = \sum_{T\in\mathcal{T}_h}\left(\eta_T^A\right)^2$$

Ponendo

$$s_K = \frac{\lambda_{1,K}}{\lambda_{2,K}} \qquad |K| = |\widehat{K}|\lambda_{1,K}\lambda_{2,K}$$
$$\left(G_{\Delta_K}\right)_{ij} = \frac{1}{|\Delta_K|}\int_{\Delta_K}\left(P^K_{ZZ}\phi_h - \nabla\phi_h\right)_i\left(P^K_{ZZ}\phi_h - \nabla\phi_h\right)_j dx$$

otteniamo la seguente espressione dello stimatore ZZ anisotropo dell'errore

$$\left(\eta_K^A\right)^2 = \lambda_{1,K}\lambda_{2,K}|\widehat{K}|\left(s_K\mathbf{r}_{1,K}^T G_{\Delta_K}\mathbf{r}_{1,K} + s_K^{-1}\mathbf{r}_{2,K}^T G_{\Delta_K}\mathbf{r}_{2,K}\right)$$

A questo punto fissata una tolleranza τ, vogliamo che lo stimatore ZZ dell'errore anisotropico sia sempre minore o uguale a τ. Allora per usare meno elementi possibili bisogna massimizzare l'area $|K| = |\widehat{K}|\lambda_{1,K}\lambda_{2,K}$ per ogni $K \in \mathcal{T}_h$, ma questo corrisponde a minimizzare la quantità

$$\underset{s_K,\mathbf{r}_{1,K}}{\operatorname{argmin}}\left(s_K\mathbf{r}_{1,K}^T G_{\Delta_K}\mathbf{r}_{1,K} + s_K^{-1}\mathbf{r}_{2,K}^T G_{\Delta_K}\mathbf{r}_{2,K}\right)$$

imponendo le condizioni $s_K \geq 1$, poiché è il rapporto tra l'autovalore più grande e l'autovalore più piccolo della metrica, e $\mathbf{r}_{1,K}\cdot\mathbf{r}_{2,K} = 0$ con $\|\mathbf{r}_{1,K}\| = \|\mathbf{r}_{2,K}\| = 1$, perché sono i due autovettori normalizzati relativi alla matrice della metrica. Risolvendo questo problema di minimo avremo così ottenuto una nuova metrica che ci permetterà di usare il minor numero di elementi per descrivere la funzione ϕ con un errore τ equidistribuito su ogni elemento.
Posti $\mathbf{v}_1$ e $\mathbf{v}_2$ gli autovettori di norma unitaria della matrice G_{Δ_K} e $\sigma_1 \geq \sigma_2$ i rispettivi autovalori, la soluzione al problema precedente sarà:

$$s_K = \sqrt{\frac{\sigma_1}{\sigma_2}} \qquad \mathbf{r}_{1,K} = \mathbf{v}_{2,K} \qquad \mathbf{r}_{2,K} = \mathbf{v}_{1,K}$$

in questo modo abbiamo determinato la matrice R_K^T e per determinare la metrica ci serve ricavare la matrice Λ_K ovvero la matrice diagonale contente gli autovalori. Poichè s_K è il rapporto tra gli autovalori della metrica ci basta ricavare il prodotto degli autovalori dalla (3.6) e poi risolvere un sistema di due equazioni e due incognite.

$$\lambda_{1,K}\lambda_{2,K} = \frac{\tau^2}{2|\widehat{K}|\sqrt{\sigma_1\sigma_2}}$$

da cui segue immediatamente che

$$\lambda_{1,K} = \sqrt{\frac{\tau^2 s_K}{2|\widehat{K}|\sqrt{\sigma_1\sigma_2}}} \qquad \lambda_{2,K} = \sqrt{\frac{\tau^2}{2s_K|\widehat{K}|\sqrt{\sigma_1\sigma_2}}}. \tag{3.7}$$

Per quanto riguarda il caso isotropo che avevamo lasciato in sospeso, basta semplicemente porre $s_K = 1$ da cui si ottiene

$$\lambda_{1,K} = \lambda_{2,K} = \sqrt{\frac{\tau^2}{2|\widehat{K}|\sqrt{\sigma_1\sigma_2}}}$$

3.3 Implementazione

In questa sezione proveremo a descrivere brevemente il modo in cui abbiamo implementato l'algoritmo e le criticità che abbiamo incontrato in questa operazione. Sia per il modello RSFE che per il modello bayesiano l'algoritmo e l'implementazione sono praticamente simili, ad eccezione della parte in cui si calcola il residuo r_{RSFE} oppure r_B. Per tale motivo descriveremo l'implementazione una volta sola per entrambi i modelli, tranne per la parte che concerne il calcolo del residuo r.

Come già detto precedentemente il programma principale usato è stato FreeFem++ anche se alcune parti sono state implementate in Matlab per comodità. Ad esempio per importare i dati dell'immagine e per disegnare graficamente la segmentazione finale abbiamo usato Matlab, mentre per tutto il resto abbiamo usato FreeFem++. Inizialmente anche il calcolo delle funzioni di densità e il calcolo della metrica erano implementati in Matlab, ma per ridurre le tempistiche sono stati tradotti in FreeFem++. Utilizzare Matlab durante ogni iterazione, comportava costi aggiuntivi anche in termini di I/O, poiché i dati venivano passati tra i due programmi usando file di testo. Le operazioni di I/O sono abbastanza lente, specialmente se eseguite in Matlab, pertanto abbiamo cercato di limitare queste operazioni all'inizio e alla fine del calcolo. Una volta importata l'immagine come matrice di pixel, si definiscono la mesh e il tipo di elementi finiti da utilizzare. Poiché i pixel sono distribuiti uniformemente nell'immagine, la mesh iniziale non può che essere una mesh regolare in cui ogni pixel rappresenta un nodo, quindi abbiamo generato la mesh $\mathcal{T}_h$ usando il comando `square` di FreeFem++. Successivamente abbiamo definito gli spazi a cui apparterranno le funzioni che useremo durante il codice, ad esempio `fespace Vh(Th,P1)`. Infine, prima di iniziare le iterazioni abbiamo definito una funzione ϕ_0 di partenza e inizializzato $\mathbf{d} = 0$ e $\mathbf{b} = 0$. A questo punto iniziamo il ciclo delle iterazioni. Prima di poter risolvere l'equazione (3.3) dobbiamo calcolare il residuo r.

3.3.1 Residuo r_{RSFE}

Il calcolo del residuo r_{RSFE} è abbastanza semplice, si tratta solo di applicare la formula (3.1). L'unica difficolà è il calcolo dei prodotti di convoluzione che compaiono nei termini e_I ed e_E, rispettivamente formula (2.1) e (2.2).

Poiché stiamo usando FreeFem++, programma costruito per risolvere PDE, il miglior modo per calcolare un prodotto di convoluzione di una funzione f con una gaussiana K_σ è quello di vederlo come soluzione di un'equazione del calore con $\Delta T = 1$, con dato iniziale la funzione f e condizioni al contorno di Neumann omogenee. Di seguito riporto il frammento di codice utilizzato.

```
// Heat equation to compute the convolution products for the RSFE model
 varf convolveMatrix(u,v) = int2d(Th)(u*v)
                          + int2d(Th)(dt*sigmaK^2./2.*grad(u)'*grad(v));

// RHS for all the concolution products
 varf convolveOne(u,v) = int2d(Th)(1.*v);
 varf convolveU0(u,v) = int2d(Th)(Im0*v);
 varf convolveNum(u,v) = int2d(Th)(Im0*Heps*v);
 varf convolveDen(u,v) = int2d(Th)(Heps*v);
 varf convolveReg1(u,v) = int2d(Th)(r1*v);
 varf convolveReg2(u,v) = int2d(Th)(r2*v);

// Precompute the matrix associated to all convolution products
 matrix A=convolveMatrix(Vh,Vh);
 set(A,solver=sparsesolver);
 real[int] Arhs(Vh.ndof);
// Ksigma * 1 and Ksigma * Im0
 Arhs=convolveOne(0,Vh); kId[]=A^-1*Arhs;
 Arhs=convolveU0(0,Vh); kU0[]=A^-1*Arhs;

// Compute the components of the RHS
 Arhs=convolveNum(0,Vh); kNum[]=A^-1*Arhs; // Convolution at numerator
 Arhs=convolveDen(0,Vh); kDen[]=A^-1*Arhs; // Convolution at denominator
// Functions associated with the values inside the two regions
 f1 = kNum/kDen;
 f2 = (kU0-kNum)/(kId-kDen);
// Discrepancies between the image u0 and the functions fi's
 r1 = lambda1*f1^2. - lambda2*f2^2.;
 r2 = lambda1*f1 - lambda2*f2;;

// Convolution products of the discrepancies
 Arhs=convolveReg1(0,Vh); kReg1[]=A^-1*Arhs; // Convolution region #1
 Arhs=convolveReg2(0,Vh); kReg2[]=A^-1*Arhs; // Convolution region #2

// RHS for RSFE
 RSFE = (lambda1-lambda2)*Im0*Im0*kId + kReg1 - 2.*Im0*kReg2;
```

dove `Im0` è l'immagine I, `Heps` è la funzione di Heaviside approssimata $H_\epsilon(x) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{\phi(x)}{\epsilon}\right)$, `lambda1` e `lambda2` sono i parametri μ_I e μ_E. Non mi sembra il caso di addentrarci ulteriormente nell'analisi del funzionale RSFE, dato che non ho scritto in prima persona il codice relativo e soprattutto perché è stato abbandonato in favore di un algoritmo puramente bayesiano.

3.3.2 Residuo r_B

Il calcolo del residuo r_B dipende dalle statistiche dell'immagine, in particolare dalle funzioni di densità di probabilità delle regioni interna ed esterna. Tuttavia, noi non conosciamo a priori queste densità, ma dobbiamo stimarle. A seconda del metodo usato per stimare le densità otterremo dei risultati diversi. In primo luogo dobbiamo scegliere se usare delle funzioni di densità parametriche oppure non parametriche. Tipicamente il metodo non parametrico è da preferirsi quando non si hanno informazioni sull'immagine e quindi sulla possibile distribuzione dei pixel. Al contrario il metodo parametrico offre grossi vantaggi se si hanno informazioni a priori sull'immagine da segmentare. Come già detto nel secondo capitolo il modello non parametrico consiste nel costruire gli istogrammi della luminosità dei pixel nella regione interna ed esterna ed usarli come funzioni di densità. Un passaggio importante è la regolarizzazione delle densità tramite la convoluzione con una gaussiana, per evitare problemi numerici ma sopratutto per far in modo che l'istogramma assomigli il più possibile ad una densità di probabilità continua. Per quanto riguarda il modello parametrico si sceglie semplicemente una densità di probabilità parametrica e si stimano i parametri con il metodo di massima verosimiglianza. Nel nostro caso abbiamo scelto una densità gaussiana perché per cominciare è la più facile da implementare e sopratutto perché ha due parametri, media μ_i e varianza σ_i, che si calcolano in modo diretto senza dover massimizzare il funzionale di massima verosimiglianza ($i = I, E$).

$$\mu_i = \frac{1}{N_i} \sum_{\substack{k=1 \\ x_k \text{ nodo di } \mathcal{T}_h}}^{N_i} I(x_k) \qquad \sigma_i = \frac{1}{N_i - 1} \sum_{\substack{k=1 \\ x_k \text{ nodo di } \mathcal{T}_h}}^{N_i} \big(x_k - \mu_i\big)^2$$

dove N_i è il numero dei nodi della mesh $\mathcal{T}_h$ interni alla regione Ω_i.

Tuttavia sia per il modello parametrico che per quello non parametrico sorge un problema con l'adattazione della mesh. Infatti, adattando la mesh, i nodi non sono più equispaziati, quindi bisognorebbe introdurre un peso per poter stimare i parametri o anche per costruire l'istogramma delle frequenze. Inoltre le stime e gli istogrammi portano all'esatta determinazione dei parametri o delle densità solo per $N_i \to \infty$, ma se l'adattazione di mesh porta ad una diminuzione del numero di nodi, le nostre stime potrebbero peggiorare sensibilmente. Per tale motivo abbiamo deciso di calcolare gli stimatori o di costruire gli istogrammi considerando sempre la mesh iniziale costruita a partire dai pixel.

```
mesh Pixel = square(nx,ny,[nx*x,ny*y],flags=3);  // mesh made of pixels
fespace Ph(Pixel,P1);
```

Un'altra accortezza utilizzata in entrambi i metodi è stata quella di troncare i valori troppo piccoli alla precisione macchina. Questo perché, dovendo calcolarne il logaritmo, valori più piccoli della precisione macchina avrebbero prodotto un errore a runtime.

Per quanto rigurda la regolarizzazione degli istogrammi, come detto precedentemente, abbiamo usato la convoluzione con una gaussiana. Tuttavia, a differenza di quanto visto per il calcolo del residuo r_{RSFE}, non abbiamo potuto sfruttare le potenzialità di FreeFem++ perché il dominio degli istogrammi è monodimensionale e non bidimensionale, e il programma permette l'utilizzo di elementi finiti solo in dimensione 2 o 3. Per tale motivo abbiamo calcolato la convoluzione dell'istogramma con un kernel gaussiano come soluzione di un'equazione del calore con $\Delta T = 1$, condizioni al bordo di Neumann omogenee e dato iniziale

proprio l'istogramma, utilizzando il metodo alle differenze finite Eulero implicito, in modo da non avere condizioni di stabilità da controllare.

```
/* we are smoothing the probability density function using the heat
equation. Since the domain fo these PDFs is 1-dim, we use one step
of Euler implicit method (Finite Difference) applied to the histograms
(internal and external) with sigma equals to 2.0 */

int N = 256; // histograms have 256 beans

//build the matrix of the liner system.
real[int,int] A(N,N);
A = 0;
A(0,0) = 1 + 2*sigma;
A(N-1,N-1) = 1 + 2*sigma;
A(0,1) = -2*sigma;
A(N-1,N-2) = -2*sigma;

for (int i=1; i<(N-1); i++)
{
    for (int j=0; j<N; j++)
    {
        if (i==j) A(i,j) = 1+2*sigma;
        if ((i==(j+1))||(i==(j-1))) A(i,j) = -sigma;
    }
}
// transofrn the matrix A into a sparse matrix (A is a tridiagronal
// positive defined matrix)
matrix AA = A;
set(AA, solver=sparsesolver);

//Build continuous probability density functions
real[int] vecInt(256), vecExt(256);

vecInt = AA^-1*histogramInt;
vecExt = AA^-1*histogramExt;
```

Per quanto riguarda il calcolo delle densità di probabilità f_I ed f_E non ci sono altre criticità, ed è possibile leggere il resto del codice usato in appendice. Ora non ci resta che calcolare il residuo r_B utilizzando le densità di probabilità come illustrato dalla formula (3.2). Di seguito riporto il codice usato

```
// RHS for the Bayesian algorithm
rBayes = log(pExt) - log(pInt);
//rBayes = ((1. - Heps)*log(pExt) - Heps*log(pInt));
```

come si può facilemente notare la riga di codice corretta sarebbe la terza, mentre noi abbiamo usato la seconda, quella che non utilizza la funzione di Heaviside. A causa di un errore abbiamo implementato la versione del codice diversa dal modello, ma ci siamo accorti, facendo le simulazioni, che questa versione convergeva più rapidamente quindi l'abbiamo tenuta così. In pratica il funzionale che andiamo a minimizzare è il seguente

$$\mathcal{F}_B(\phi) = \int_\Omega \phi(x) \log \frac{f_E(I(x))}{f_I(I(x))} dx + \nu \int_\Omega \frac{|\nabla \phi(x)|}{1 + \beta |\nabla I(x)|^2} dx \tag{3.8}$$

La convergenza è più rapida perché il funzionale tiene in cosiderazione anche la probabilità che il pixel si trovi nell'altra regione.

3.3.3 Calcolo dei nuovi valori di ϕ^{k+1} e $\mathbf{d}^{k+1}$

Ora che abbiamo calcolato il residuo r possiamo procedere con l'iterazione e calcolare ϕ^{k+1} e $\mathbf{d}^{k+1}$. Per quanto riguarda ϕ^{k+1} abbiamo risolto l'equazione (3.3) utilizzando il metodo agli elementi finiti per discretizzare il laplaciano, e il metodo di Eulero implicito per la derivata temporale. Il metodo di Eulero implicito è stata una scelta obbligata, dato che non ha condizioni di stabilità e quindi funziona per qualsiasi tipo di mesh. Questo passaggio non presenta delle criticità, quindi riporto semplicemente la porzione di codice usato.

```
//////////  Variational forms  //////////
// Split Bregman - FIRST STEP
varf parabolic(phi,psi) = int2d(Th)(phi*psi)
                        + int2d(Th)(dt*grad(phi)'*grad(psi));

varf parabolicRHS(phi,psi) = int2d(Th)(tmp*psi)
                           + int2d(Th)(dt*div(b)*psi)
                           - int2d(Th)(dt*div(d)*psi)
                           - int2d(Th)(dt*rBayes/lambda*psi);

 matrix C=parabolic(Vh,Vh);
 set(C,solver=sparsesolver);
 real[int] Crhs(Vh.ndof);
 Crhs=parabolicRHS(0,Vh);
 phi[]=C^-1*Crhs;
```

ancora più immediato è l'aggiornamento del termine $\mathbf{d}^{k+1}$ che avviene in modo diretto utilizzando la formula 3.4 dell'operatore di shrinkage. Infine, prima di iniziare una nuova iterazione si aggiorna il termine $\mathbf{b}^{k+1} = \mathbf{b}^k + \nabla\phi^{k+1} - \mathbf{d}^{k+1}$.

3.3.4 Calcolo della metrica e adattazione della mesh

Della costruzione della metrica abbiamo già ampiamente parlato nel capitolo precedente, e sostanzialmente l'implementazione segue al dettaglio l'algoritmo descritto. Quindi per prima cosa abbiamo calcolato la matrice degli errori quadratici G_{Δ_K}, successivamente abbiamo ricavato autovalori e autovettori. Quest'ultima operazione risulta particolarmente rapida poiché G_{Δ_K} è una matrice simmetrica 2×2. In questo modo abbiamo direttamente $\mathbf{r}_{1,K}$ e $\mathbf{r}_{2,K}$ e applicando la (3.7) otteniamo direttamente anche gli autovalori della metrica $\lambda_{1,K}$ e $\lambda_{2,K}$ L'unica accortezza che bisogna avere è quella di assicurarsi che gli autovalori non siano troppo piccoli, per evitare i classici errori a runtime e per evitare di avere triangoli eccessivamente allungati. Per tale motivo si imposta un limite massimo al valore $s_K < 10^6$, in tal modo siamo sicuri che l'autovalore $\lambda_{2,K} \geq 10^6\lambda_{1,K}$ e che il fattore di stretching dei triangoli sia al massimo $\sqrt{s_K} = 10^3$.
A questo punto per costruire la metrica basta ricostruire la matrice B_K per ogni elemento $K \in \mathcal{T}_h$ semplicemente moltiplicando le tre matrici

$$B_K = R_K^T \, \Lambda_K \, R_K$$

dove le matrici R_K e Λ_K sono definire dalla (3.5).

Infine per costruire la mesh abbiamo semplicemente usato un comando già integrato in FreeFem++, il comando `adaptmesh`.

```
Th = adaptmesh(Th, nbvx = max(500000,nx*ny),
               metric = [m11[],m12[],m22[]],
               hmax=H,
               keepbackvertices=false);
```

dove `nbvx` indica il numero massimo di vertici con cui posso costruire la nuova mesh, `metric` è la metrica definita su ogni vertice della mesh precedente, `hmax` è il diametro massimo dei triangoli della nuova mesh, e `keepbackvertece = false` indica che non vogliamo mantenere i vecchi vertici, ma costruire una mesh completamente nuova.

Questo conclude il capitolo e i punti interessanti della scrittura del codice, nel prossimo capitolo vedremo i risultati ottenuti con le diverse versioni del codice e parleremo in dettaglio dei parametri che regolano l'algoritmo, come ad esempio il passo temporale dt nella risoluzione della 3.3 o la tolleranza τ dello stimatore ZZ dell'errore sul gradiente $\nabla\phi_h$.

Nella stesura di questo lavoro ho usato $\phi(x)$, ϕ^*, ϕ_h, ϕ^k o $\Phi(x,T)$ per descrivere la funzione level-set nei suoi vari aspetti, soluzione esatta, approssimata, k-esima iterazione, ecc. spero che a seconda dei casi non ci sia stata confusione sul significato del simbolo utilizzato.

Capitolo 4

Risultati

In questo capitolo vorrei esporre i rusultati ottenuti con varie simulazioni. Le simulazioni sono state fatte utilizzando solo l'algoritmo bayesiano, cambiando i parametri iniziali e/o il metodo risolutivo. Prima di illustrare gli esperimenti fatti, vorrei elencare tutti i paramentri e tutti i possibili metodi utilizzati. Per farlo vorrei partire dalla parte iniziale del codice

```
bool adaptivityON = true;
bool gaussDistributionON = false;
bool anisON = true;  // if TRUE, adaptmesh will produce an
                     // anisotropic mesh

// Parameters
real toll = 1.e-3;

int maxIter=500;
real dt = 1.e0; // Time step (Time is fictitious, but... ...)
int NN = 1; // Number of iterations of heat-PDE
real beta = 1.e2; // Parameter for edge detecting function
real nu = 1.e0; // Coefficient for the force term
real eta = 1.e-15; // Small parameter to avoid zero elements
                   // (i.e. in matrix inversion)
real epsilon = 1.e-1; // parameter of Heaviside Smooth Function
                      // (NOT used at the moment)
real lambda = 1.e-3;  //parameter of Split-Bregman Algoritm
real sigma = 2.; // std deviation of Gauss smoothing kernel for
                 // PROBABILITIES DENSITY

real H = 3.; // Max size for mesh adaptivity
real hplus = 0.1; // Increasing max size for mesh adaptivity
real tau = 0.2; // tolerance to compute metric
real maxstretch = 1.e6;  // ratio between autovalues of metric matrix
                         // (the real stretching factor is the sqrt of
                         // this value)
```

le variabili booleane `adaptivityON`, `gaussDistributionON` e `anisON` servono semplicemente per scegliere il tipo di algoritmo usato. Se `adaptivityON` è vera l'algoritmo effettua l'adattazione della mesh, altrimenti usa una mesh fissa, quella iniziale determinata dai pixel dell'immagine. Se `gaussDistributionOn` è vera l'algoritmo usa le densità di probabilità parametriche, in particolare trova le due densità gaussiane che meglio si adattano ai valori di luminosità interni ed

esterni, altrimenti usa le densità non parametriche, ciò quelle ottenute a partire dagli istogrammi. Se stiamo usando l'algoritmo adattivo inoltre possiamo scegliere se produrre una mesh isotropa o anisotropa, ponendo `anisON` vera otteniamo la versione anisotropa, altrimenti ottenimo quella isotropa.

Passiamo ora a descrivere i parametri. `toll` indica la tolleranza, ovvero il valore del criterio di stop per il quale fermiamo l'algoritmo. `maxIter` è il numero massimo di iterazioni. `dt` indica il passo temporale per la risoluzione dell'equazione del calore (3.3) ed `NN` invece indica il numero di passi che facciamo, e avremo che $T = NN \cdot dt$. `beta` è il parametro della funzione *edge-detecting* $g(I(x)) = \frac{1}{1+\beta|\nabla I(x)|^2}$ e lo poniamo uguale a 200, durante le simulazioni lo abbiamo lasciato invariato. `eta` è semplicemente il valore di threshold per evitare errori a runtime dovuti a numeri troppo piccoli al denominatore. `epsilon` è il parametro della funzione di Heaviside H_ϵ regolarizzata, che tuttavia non usiamo più in quanto abbiamo sostituito il funzionale $\mathcal{F}(\phi)$ con il residuo (3.2) con il funzionale (3.8). `lambda` corrisponde a μ, il moltiplicatore di Lagrange dell'algoritmo di split di Bregman. In teoria avremmo dovuto includere anche μ come variabile nell'algoritmo di minimizzazione, tuttavia abbiamo notato che considerandolo come parametro e cambiandone il valore non si riscontravano differenze significative nella segmentazione finale, per tale motivo lo abbiamo fissato a 10^{-3}. `sigma` è la deviazione standard del kernel gaussiano di media 0, che abbiamo usato per smussare gli istogrammi ed ottenere le densità di probabilità non parametriche.

Gli ultimi parametri non riguardano l'algoritmo in sé, ma la costruzione della mesh adattata. `H` è il diametro massimo degli elementi della mesh, mentre `hplus` è l'incremento di `H` ogni volta che costruiamo una nuova mesh. In particolare, man mano che l'algoritmo procede consentiamo alla mesh di avere triangoli più lunghi. `tau` è la tolleranza dell'errore stimato tramite il gradiente ricostruito su ogni elemento $K \in \mathcal{T}_h$. Infine `maxstetch` è il quadrato del massimo fattore di stretching degli elementi della nuova mesh.

L'ultimo parametro che dobbiamo settare prima di iniziare l'algoritmo è la funzione iniziale ϕ_0, nella maggior parte delle simulazioni abbiamo scelto la funzione

$$\phi_0(x) = \begin{cases} \frac{1}{2} & x \in Q \\ -\frac{1}{2} & x \notin Q \end{cases}$$

dove $Q = \{\, x \in \Omega \big| \frac{1}{3}m < x_1 < \frac{2}{3}m \wedge \frac{1}{3}n < x_2 < \frac{2}{3}n \,\}$ ed $m \times n$ sono le dimensioni in pixel dell'immagine I.

```
// Initial level-set function
real a0= -0.5;
real b0= 0.5;
func phi0 = a0 +
            (b0-a0)*(x>nx/3. && x<2.*nx/3. && y>ny/3. && y<2.*ny/3.);
```

La prima serie di simulazioni che ho fatto è stata per testare che effettivamente il codice ripulito dall'algoritmo RSFE e da alcne parti ancora in MATLAB funzionasse correttamente. Questa fase è durata più del previsto, a causa forse dei troppi bug e di una programmazione poco modulare. Questa serie di test è stata fatta utilizzando un codice adattivo anisotropo con distribuzioni di probabilità non parametriche. Una volta corretti tutti i bug, però il risultato non è stato dei migliori, la segmentazione era corretta, ma il contorno individuato era eccessivamente lontano la quello vero. In figura 4.1 è riportato il risultato finale

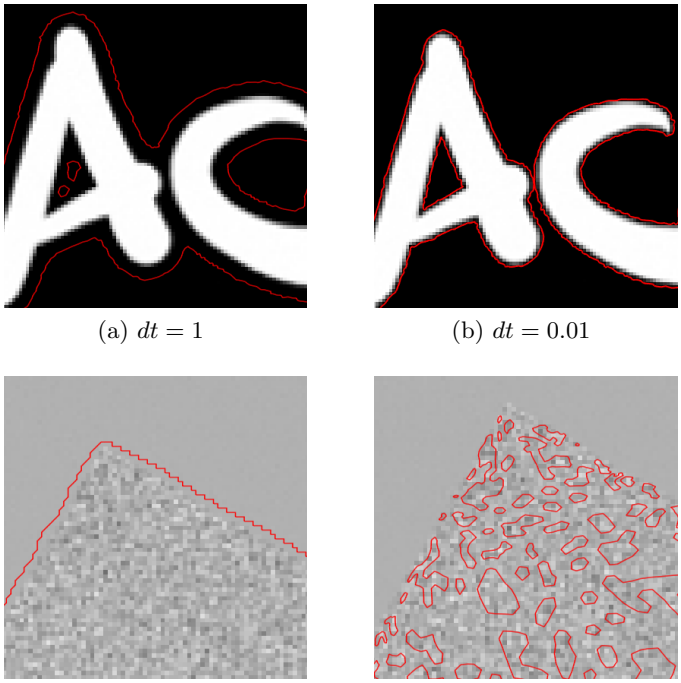
Figura 4.1: risultato della segmentazione con algoritmo adattivo bayesiano, densità non parametrica e $dt = 1$.

della segmentazione di un'immagine sintetica, effettuata con i seguenti parametri e impostazioni: algoritmo adattivo anisotropo, densità parametrica gaussiana, $dt = 1$, $NN = 1$, $H = 3$, $h^+ = 0.1$ e $\tau = 0.2$.

Per trovare la segmentazione finale ϕ abbiamo trasformato un'equazione di Poisson in un'equazione parabolica, di cui cerchiamo lo stato stazionario. In questa situazione il passo temporale è fittizio e lo avevamo fissato a $dt = 1$. Tuttavia, in seguito a queste simulazioni, abbiamo provato a modificare il passo temporale per vedere se fosse possibile migliorare il risultato finale. In figura 4.2 sono riportati i dettagli di due immagini, entrambe segmentate con lo stesso algoritmo, gli stessi parametri, ma con diversi passi temporali. Come si può notare il passo temporale che va bene per una immagine non è detto che funzioni anche con un'altra. In generale, il passo temporale dt è legato al termine diffusivo dell'equazione, pertanto in immagini con poco rumore è bene prendere un passo temporale $dt \approx 10^{-2}$, mentre per immagini con un livello alto di rumore la scelta migliore è prendere $dt \approx 1$, soprattutto quando si usa le versione dell'algoritmo in cui si stimano le densità di probabilità in modo non parametrico.

A seguito di questi esperimenti eseguiti cambiando il passo temporale, abbiamo apportato una piccola modifica al codice che permettesse di fare NN passi temporali per l'equazione (3.3) per ogni iterazione.

```
for(int i=0; i<NN; i++)
{
    matrix C=parabolic(Vh,Vh);
    set(C,solver=sparsesolver);
    real[int] Crhs(Vh.ndof);
    Crhs=parabolicRHS(0,Vh);
    tmp[]=C^-1*Crhs;
    cout << endl << "iteration " << i +1 << " of "<< NN << endl;
}
phi = tmp;
```



(a) $dt = 1$ (b) $dt = 0.01$

(c) $dt = 1$ (d) $dt = 0.1$

Figura 4.2: alcuni esempi in cui il passo temporale dt influisce sulla segmentazione finale.

Tuttavia non si sono registrate differenze significative tra fissare il passo temporale ed eseguire NN passi, piuttosto che fissare direttamente il passo temporale uguale a $dt \cdot NN$ ed eseguire un solo passo. Per questo motivo, in quasi tutte le simulazioni fatte successivamente, abbiamo usato $NN = 1$. Inoltre risolvere più volte l'equazione del calore durante una singola iterazione avrebbe comportato un costo computazionale notevole dato che proprio la risoluzione del sistema lineare legato a questa equazione è uno dei punti critici dell'algoritmo. La scelta migliore sarebbe stata quella di non decidere a priori il numero di passi, ma trovare un indicatore o un criterio di stop che segnalasse quando "fermare il tempo" dell'equazione del calore e proseguire con l'algoritmo. Purtroppo non siamo riusciti a trovare un criterio di stop adeguato, anche perché la funzione ϕ che si trova dopo NN passi non ha nulla di diverso rispetto a quella trovata al passo $NN - 1$, è semplicemente un risultato intermedio, e ovviamente non possiamo usare lo stesso criterio di stop che usiamo per terminare l'intero algoritmo.

Dopo queste considerazioni sul passo temporale fittizio dt abbiamo introdotto la porzione di codice che ci permette di calcolare le densità di probabilità in modo parametrico. Inizialmente avremmo voluto introdurre più tipi di distribuzioni, oltre alla gaussiana, ma purtroppo non c'è stato tempo. Per testare questo algoritmo abbiamo costruito un'immagine *ad hoc*: formata da due sole regioni con intensità luminosa uniforme e nota (figura 4.3). e abbiamo eseguito l'algoritmo di segmentazione una volta utilizzando le densità di probabilità parametriche e la seconda volta invece no. In entrambi i casi sono stati usati gli stessi parametri iniziali

```
real toll = 1.e-3;
```

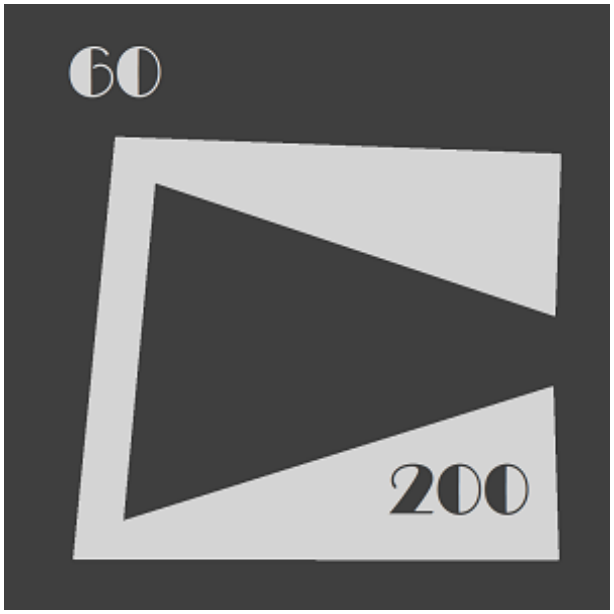
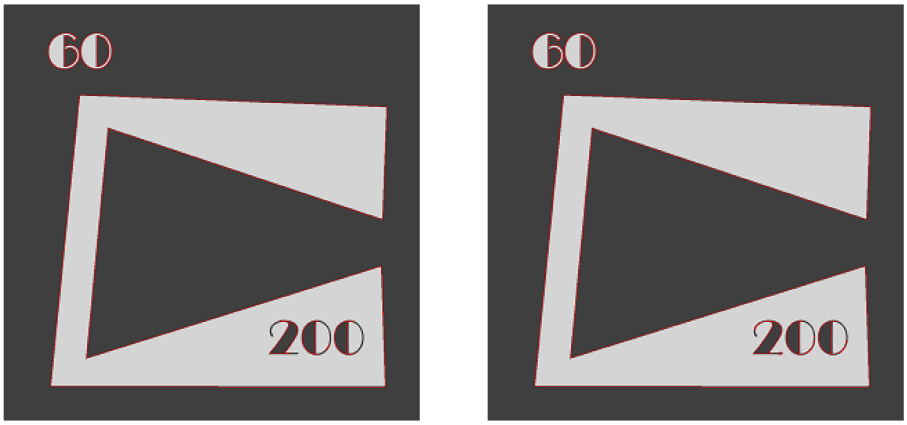


Figura 4.3: immagine sintetica formata da due regioni distinte, una con luminosità 60 (regione scura) e l'altra con luminosità 200 (regione chiara).

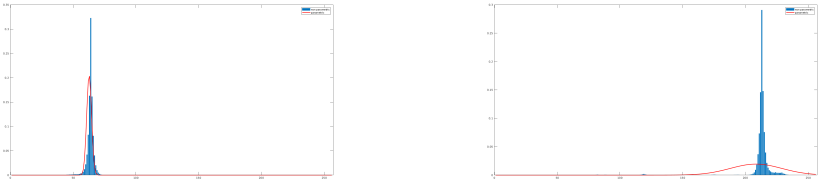
```
int maxIter=500;
real dt = 1.e-2; // Time step (Time is fictitious, but... ...)
int NN = 1; // Number of iterations of heat-PDE
real beta = 1.e2; // Parameter for edge detecting function
real nu = 1.e0; // Coefficient for the force term
real lambda = 1.e-3;  //parameter of Split-Bregman Algoritm

real H = 3.; // Max size for mesh adaptivity
real hplus = 0.1; // Increasing max size for mesh adaptivity
real tau = 0.2; // tolerance to compute metric
real maxstretch = 1.e6;  // ratio between autovalues of metric matrix
```

Per l'algoritmo parametrico il tempo computazionale è stato di 779 secondi, il numero di iterazioni è stato 71 e la mesh finale era costituita da 148328 triangoli e 74307 vertici, mentre la mesh iniziale era formata da 178802 triangoli e 90000 vertici. Per quanto riguarda invece l'algoritmo non parametrico il tempo di calcolo è stato di 487 secondi, il numero di iterazioni 31 e la mesh finale era costituita da 210720 triangoli e 105537 vertici, ovviamente la mesh iniziale è uguale per entrambi gli algoritmi perché è la mesh formata dai pixel. I risultati finali della segmentazione sono in figura 4.4. Dalla figura tuttavia è praticamente impossibile stabilire quale dei due algoritmi ha lavorato meglio. Sicuramente quello non parametrico è stato più veloce quindi sarebbe da preferire. E infatti guardando anche la figura 4.5 che riporta gli istogrammi e le densità di probabilità gaussiane nella regione interna e nella regione esterna vediamo che l'algoritmo non parametrico ha lavorato meglio. Questo risultato era proprio quello che ci aspettavamo, infatti l'immagine non ha una distribuzione di intensità gaussiana all'interno delle due regioni, direi piuttosto che la distribuzione che meglio identifica la situazione esatta è la delta di Dirac, che viene catturata meglio dagli istogrammi delle frequenze. La situazione invece dovrebbe cambiare quando aggiungiamo del rumore gaussiano additivo all'immagine. Con l'utilizzo di MATLAB abbiamo aggiunto



(a) gaussDistributionON = true (b) gaussDistributionON = false

Figura 4.4: risultato finale della segmentazione: con algoritmo parametrico e non parametrico



(a) regione interna Ω_I (b) regione esterna Ω_E

Figura 4.5: confronto tra istogrammi e distribuzioni gaussiane nelle due regioni Ω_I e Ω_E in cui abbiamo segmentato l'immagine

all'immagine un rumore gaussiano con media 0 e varianza 0.01. La funzione Matlab imnoise che aggiunge il rumore considera l'intensità luminosa dell'immagine su una scala da 0 a 1 e non da 0 a 255 come invece stiamo facendo in FreeFem++, pertanto nella scala usata da noi (in FreeFem++) la varianza del rumore gaussiano introdotto corrisponde a $\sigma^2 = 655.36$. Abbiamo applicato i due algoritmi precedenti, con gli stessi parametri e le stesse impostazioni, alla figura 4.6 con aggiunta di rumore. Il risultato finale della segmentazione è stato paragonabile (figura 4.7, ma questa volta l'algoritmo parametrico ha lavorato meglio: il tempo di calcolo è stato 155 secondi, il numero di iterazioni 10 e la cardinalità della mesh finale era 153758 triangoli e 27099 vertici. Mentre per l'algoritmo non parametrico il tempo di elaborazione è stato di 167 secondi, il numero di iterazioni 11 e la mesh finale era formata da 53098 triangoli e 26769 vertici. Tuttavia rimane ancora una perplessità guardando il confronto tra gli istogrammi e le distribuzione gaussiane stimate (figura 4.8), infatti per i valori di luminosità 0 o 255, la gaussiana non riesce a seguire l'istogramma. Questo capita perché i valori di luminosità delle immagini sono limitati all'intervallo $[0, 255]$ mentre la gaussiana è definita su tutta la retta reale $\mathbb{R}$, per ovviare a questo problema, nei prossimi lavori, si potrebbe modificare la distribuzione gaussiana in modo tale che sia troncata a sinistra di 0 e a destra di 255 per catturare meglio l'informazione statistica dell'immagine. Infine abbiamo testato i due algoritmi, parametrico e non, su un'immagine reale. In questo caso la "vittoria" dell'algoritmo parametrico è stata netta. Non abbiamo confrontato istogrammi con la stima delle gaussiane, dal momento che il risultato finale della segmentazione è

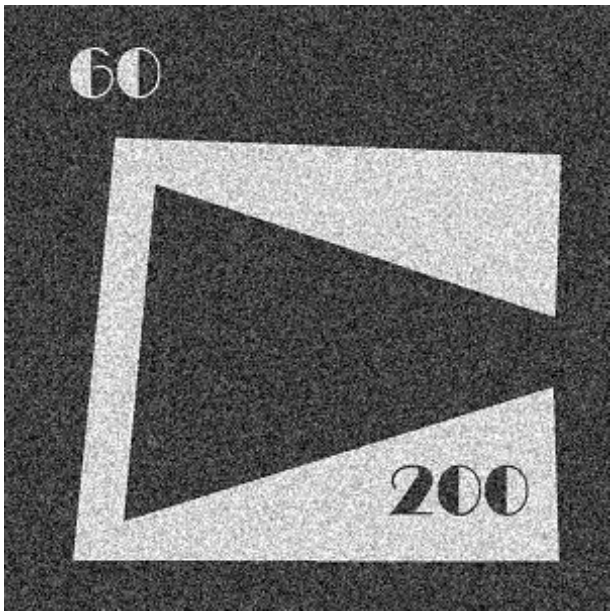
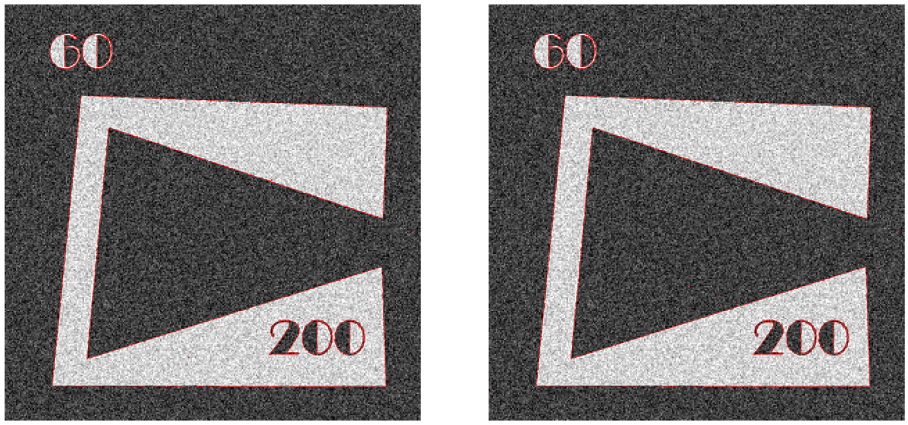
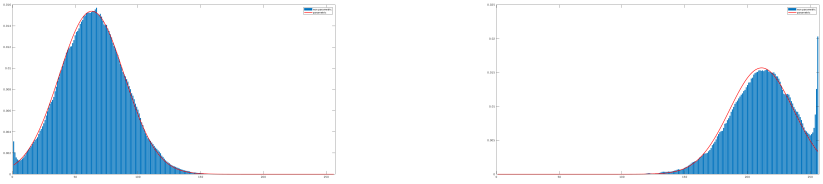


Figura 4.6: immagine sintetica formata da due regioni distinte, una con luminosità 60 (regione scura) e l'altra con luminosità 200 (regione chiara) con aggiunta di rumore gaussiano additivo $\mu = 0$ e $\sigma^2 = 0.01$.



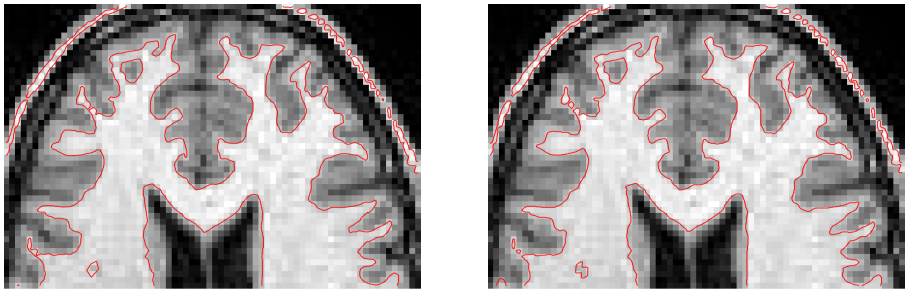
(a) GAUSSDISTRIBUTIONON = TRUE (b) GAUSSDISTRIBUTIONON = FALSE

Figura 4.7: risultato finale della segmentazione: con algoritmo parametrico e non parametrico nel caso di immagine con rumore gaussiano additivo.



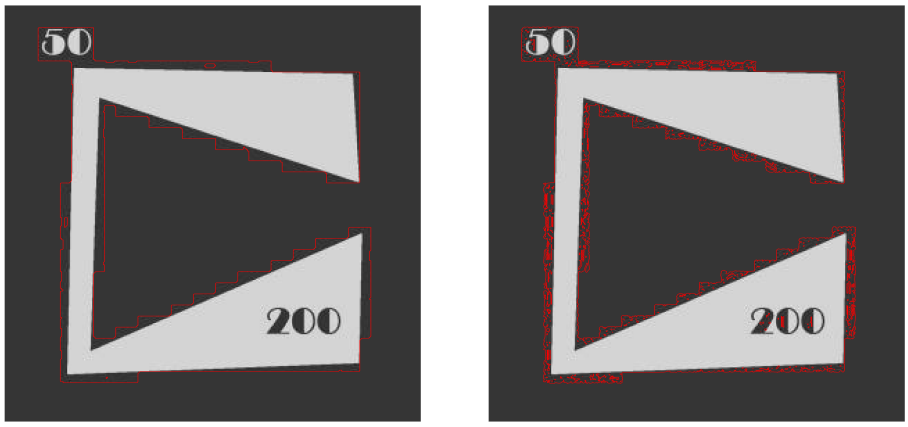
(a) regione interna Ω_I (b) regione esterna Ω_E

Figura 4.8: confornto tra istogrammi e distribuzioni gaussiane nelle due regioni Ω_I e Ω_E in cui abbiamo segmentato l'immagine, caso di immagine con rumore gaussiano additivo.



(a) GAUSSDISTRIBUTIONON = TRUE (b) GAUSSDISTRIBUTIONON = FALSE

Figura 4.9: risultato finale della segmentazione: con algoritmo parametrico e non parametrico su immagine reale (sezione di una risonanza magnetica al cervello).



(a) SENZA ADATTAZIONE DI MESH (b) CON ADATTAZIONE DI MESH

Figura 4.10: immagine sintetica salvata in formato jpeg e successivamente segmentata.

stato praticamente lo stesso anche per le simulazioni con immagini sintetiche. La differenza schiacciante è stata nel tempo di calcolo e quindi nel numero di iterazioni, e nella cardinalità della mesh. Nel primo caso il tempo di calcolo è stato di 73 secondi con 25 iterazioni e una mesh formata da 50688 triangoli e 25486 vertici, mentre nel secondo caso il tempo computazionale è stato di addirittura 2202 secondi con 125 iterazioni e una mesh formata da 109562 triangoli e 54956 vertici. Nella figura 4.9 riportiamo per dover di cronaca il risultato finale della segmentazione. Guardando le immagini del cervello ci si potrebbe domandare perché entrambi gli algoritmi abbiano separato l'immagine in quel modo e non in un altro, per esempio includendo la sostanza grigia all'interno della regione più chiara. Questo potrebbe essere un difetto del modello, ma principlamente dipende dal fatto che le regioni da segmentare sono in realtà tre, ma noi stiamo applicando un algoritmo che riesce a separare solamente due regioni. Tuttavia, il caso che andrò a descrivere ora, fa riflettere e forse indica che ci sia ancora qualcosa da sistemare nell'algoritmo o nel modello, anche se per immagini reali sembra funzionare tutto molto bene. La prima versione dell'immagine 4.3 è stata salvata utilizzando il formato JPEG e il risultato della segmentazione è stato diverso da quanto atteso (figura 4.10). Come si può facilmente vedere la segmentazione non ha funzionato, questo perché salvando un'immagine sintetica, costituita da due regioni a luminosità costante, si genera il cosidetto fenomeno di Gibbs, dovuto all'approssimazione dell'immagine con serie di coseni. In pratica vicino ai bordi

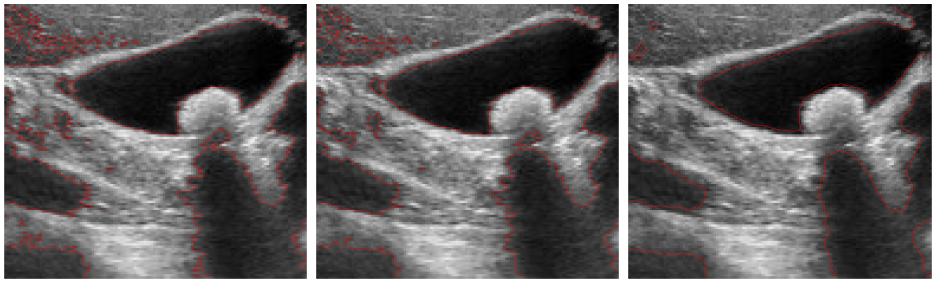
delle regioni di diverso colore si generano dei pixel che hanno dei picchi di intensità maggiore di circa il 20% rispetto ai pixel vicini nel caso ci troviamo nella regione chiara o minori del 20% sempre rispetto ai pixel vicini nel caso in cui ci troviamo nella regione scura. L'algoritmo usato, sia con adattazione di mesh che senza, sia con denstià parametriche che non, produce sempre un risultato simile a quello in figura 4.10, sembra infatti che l'algoritmo tenda a identificare la regione interna con i pixel in cui c'è solo un'intensità luminosa, quella più frequente, mentre tutto il resto fa parte della regione esterna. E questo è un aspetto che va sicuramente indagato più a fondo.

Ora passiamo alle simulazioni fatte cambiando i parametri relativi alla mesh e all'algoritmo di adattazione. Per quanto riguarda la tolleranza τ abbiamo notato che valori troppo alti $\tau > 0.5$ portano ad una mesh praticamente isotropa e regolare, i triangoli si allargano tutti indistintamente, facendo perdere risoluzione alla segmentazione finale. Invece per valori di $\tau < 0.1$ si ha l'effetto opposto, la mesh si infittisce a dismisura, raggiungendo cardinalità elevate (numero dei vertici ≈ 500000) rendendo di fatto lentissima la computazione e quindi impraticabile. Per questo motivo dopo alcune prove ci è sembrato opportuno fissare $\tau = 0.2$ e lasciarlo invariato in tutte le simulazioni, tuttavia anche valori tra 0.3 e 0.1 possono essere usati. Anche il fattore di stretching massimo riveste un ruolo simile a τ infatti un valore troppo alto porta all'eccessivo allungamento dei triangoli e quindi ad un aumento eccessivo dei triangoli della mesh, con un costo computazionale enorme, viceversa un fattore di stretching troppo basso porta ad avere una mesh praticamente isotropa, che quindi non si adatta al meglio ai contorni delle immagini. Infine è stato necessario introdurre il parametro H per controllare la grandezza dei triangoli, sopratutto con immagini con tanto rumore. Per immagini senza rumore infatti il parametro H che determina il diametro massimo dei triangoli della mesh è del tutto inutile e rallenta l'esecuzione dell'algoritmo. Invece per le immagini con tanto rumore il parametro H evita il formarsi di triangoli troppo grossi, i cui vertici cadendo su un pixel danneggiato dal rumore, potrebbero inficiare il risultato, perché comprometterebbero una regione molto grande.

Infine abbiamo notato come la scelta della funzione level-set iniziale non produce differenze significative sulla segmentazione finale. Al posto del rettangolo centrale abbiamo provato a prendere un cerchio centrato in diversi punti dell'immagine, oppure numerosi cerchi distribuiti su tutta l'immagine, ma il risultato finale della segmentazione non è cambiato, per questo motivo ci è sembrato comodo mantere come ϕ_0 la funzione che vale 0.5 nel rettangolo centrale all'immagine e -0.5 altrove.

Per concludere questo capitolo di test e simulazioni, vediamo come si comportano le varie versioni del codice con un'immagine reale, un'ecografia della cistifellea. In tutte queste simulazioni abbiamo usato i seguenti parametri fissi

```
int maxIter=500;
int NN = 1; // Number of iterations of heat-PDE
real nu = 1.e0; // Coefficient for the force term
real eta = 1.e-15; // Small parameter to avoid zero elements
real lambda = 1.e-3;  //parameter of Split-Bregman Algoritm
real sigma = 2.; //std dev of Gauss smoothing kernel for PDFs

real H = 3.; // Max size for mesh adaptivity
real hplus = 0.1; // Increasing max size for mesh adaptivity
```



(a) $dt = 0.01$ (b) $dt = 0.1$ (c) $dt = 1$

Figura 4.11: segmentazione di un'ecografia alla cistifellea con algoritmo adattivo anisotropo, con densità gaussiane.

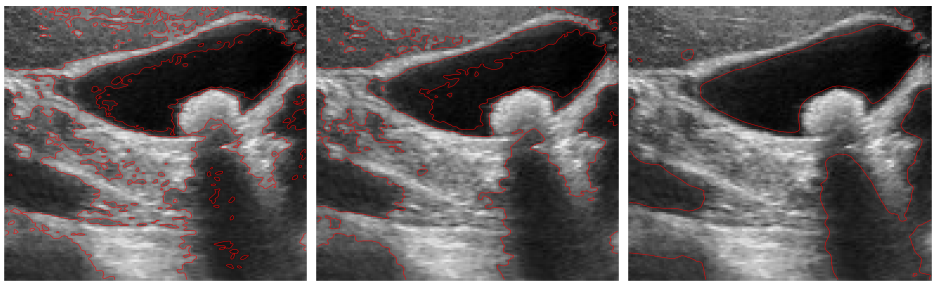
Tabella 4.1: dati relativi all'algoritmo adattivo anisotropo, con densità gaussiana, cfr. figura 4.11.

figura	passo dt	tempo computaz. (s)	# iterzioni	# mesh (vertici)
(a)	0.01	65	24	11341
(b)	0.10	47	19	6674
(c)	1.00	37	18	3110

```
real tau = 0.3; // tolerance to compute metric
real maxstretch = 1.e6;  // ratio between autovalues of metric matrix
```

a differenza di quanto detto precedentemente, abbiamo usato una tolleranza τ per la metrica leggermente più alta per ridurre il tempo computazionale ($\tau = 0.3$), ma costante in tutte le 18 simulazioni su questa immagine. Invece abbiamo fatto variare il passo temporale dt e le varie tipologie di algoritmo: con adattazione della mesh o senza, in caso di adattazione isotropa o anisotropa, con densità parametriche o non parametriche.

La prima osservazione è che in generale l'algoritmo con densità parametriche (gaussiane nel nostro caso) riconosce meglio le regioni e i contorni delle immagini, rispetto ad un algoritmo identico ma con densità non parametriche. La seconda osservazione riguarda il passo temporale dt. Sebbene avessimo criticato la scelta

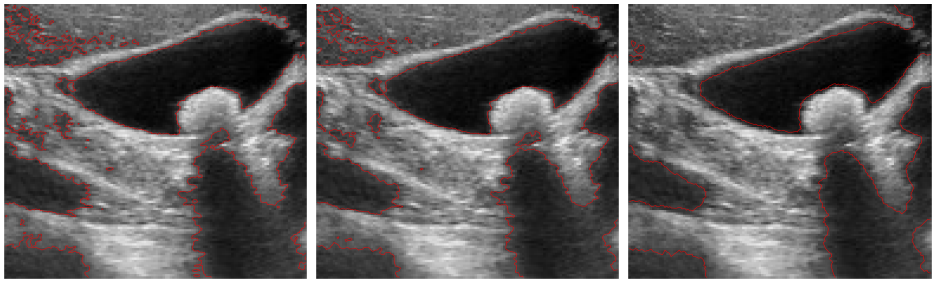


(a) $dt = 0.01$ (b) $dt = 0.1$ (c) $dt = 1$

Figura 4.12: segmentazione di un'ecografia alla cistifellea con algoritmo adattivo anisotropo, con densità non parametriche.

Tabella 4.2: dati relativi all'algoritmo adattivo anisotropo, con densità non parametrica, cfr. figura 4.12.

figura	passo dt	tempo computaz. (s)	# iterzioni	# mesh (vertici)
(a)	0.01	765	65	109362
(b)	0.10	934	110	67340
(c)	1.00	243	135	7832

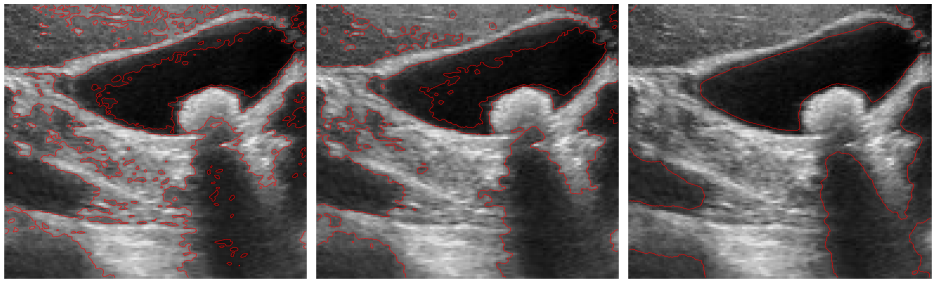


(a) $dt = 0.01$ (b) $dt = 0.1$ (c) $dt = 1$

Figura 4.13: segmentazione di un'ecografia alla cistifellea con algoritmo adattivo isotropo, con densità gaussiane.

Tabella 4.3: dati relativi all'algoritmo adattivo isotropo, con densità gaussiana, cfr. figura 4.13.

figura	passo dt	tempo computaz. (s)	# iterzioni	# mesh (vertici)
(a)	0.01	78	25	13724
(b)	0.10	59	23	9300
(c)	1.00	49	24	4125

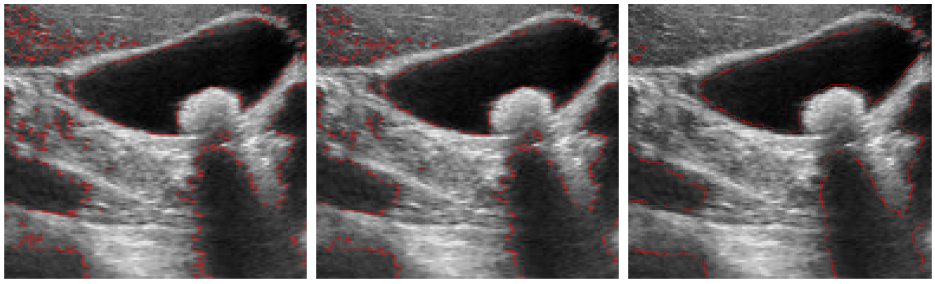


(a) $dt = 0.01$ (b) $dt = 0.1$ (c) $dt = 1$

Figura 4.14: segmentazione di un'ecografia alla cistifellea con algoritmo adattivo isotropo, con densità non parametriche

Tabella 4.4: dati relativi all'algoritmo adattivo isotropo con densità non parametriche, cfr. figura 4.14.

figura	passo dt	tempo computaz. (s)	# iterzioni	# mesh (vertici)
(a)	0.01	208	35	43077
(b)	0.10	184	50	22769
(c)	1.00	355	189	9655

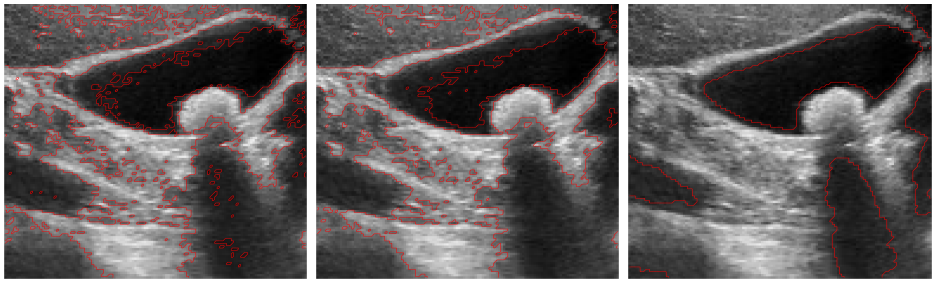


(a) $dt = 0.01$ (b) $dt = 0.1$ (c) $dt = 1$

Figura 4.15: segmentazione di un'ecografia alla cistifellea con algoritmo non adattivo, con densità gaussiane.

Tabella 4.5: dati relativi all'algoritmo non adattivo, con densità gaussiana, cfr. figura 4.15.

figura	passo dt	tempo computaz. (s)	# iterzioni	# mesh (vertici)
(a)	0.01	94	34	14848
(b)	0.10	65	23	14848
(c)	1.00	53	19	14848



(a) $dt = 0.01$ (b) $dt = 0.1$ (c) $dt = 1$

Figura 4.16: segmentazione di un'ecografia alla cistifellea con algoritmo non adattivo, con densità non parametriche.

Tabella 4.6: dati relativi all'algoritmo non adattivo, con densità non parametriche, cfr. figura 4.16.

figura	passo dt	tempo computaz. (s)	# iterzioni	# mesh (vertici)
(a)	0.01	57	23	14848
(b)	0.10	69	23	14848
(c)	1.00	113	45	14848

di porre $dt = 1$, con queste simulazioni ci siamo accorti che in realtà tale criticità si riscontrava quasi esclusivamente con immagini sintetiche. Infatti le segmentazioni dell'ecografia della cistifellea, fatte con diversi algoritmi, hanno portato ad un ottimo risultato anche ponendo $dt = 1$. Quello che abbiamo riscontrato infatti è che con valori di dt troppo piccoli i contorni individuati risultano molto frastagliati, il tempo computazionale è in generale più alto e non sempre il risultato finale è quello ottimale. Infatti un valore di dt più alto fa convergere il metodo più velocemente allo stato stazionario, e produce un risultato con bordi più lisci. Ora a seconda del tipo di immagine da segmentare potremmo desiderare più o meno accuratezza sui contorni. Per esempio nell'immagine della cistifellea è da preferire un bordo liscio, piuttosto che un bordo frastagliato che segue eccessivamente il segnale prodotto dal rumore, invece su un immagine satellitare di una costa sarebbe preferibile seguire più dettagliatamente il profilo della costa, specialmente se frastagliato. Per tale motivo riteniamo che dt sia un parametro che bisogna scegliere in base al tipo di immagine che si vuole analizzare e al livello di rumore. Tuttavia rimane l'impressione che prendendo $dt = 1$ il risultato finale della segmetazione sia leggermente spostato rispetto al bordo reale, specialmente quando si utilizzano densità non parametriche. Per concludere, confrontando tutte le immagini e le tabelle relative ai dati delle segmentazioni, l'algoritmo ottimale risulta essere quello con densità gaussiane, adattivo con mesh anisotropa e $dt = 1$, perché è il più veloce, le dimensioni della mesh sono ridotte e il risultato è comunque uno dei migliori. Il calcolo della metrica comporta un costo computazionale rilevante, pertanto conviene utilizzare una adattazione anisotropa piuttosto che quella isotropa perché in ogni caso si ha il costo computazionale del calcolo della metrica, ma la qualità della mesh è nettamente superiore nel caso anisotropo (minor numero di nodi e miglior approssimazione della soluzione ϕ^*). Il metodo non adattivo produce dei risultati qualitativamente apprezzabili, anche se risentono della struttura rigida della mesh ottenuta ponendo i pixel come vertici (i bordi non sono perfettamente lisci, perché devono seguire i lati o le diagonali dei quadrati formati dai pixel). Inoltre il tempo computazionale è leggermente più alto rispetto al metodo adattivo, questo significa che il costo computazinale del calcolo della metrica viene ammortizzato totalmente dalla riduzione della cardinalità della mesh.

Capitolo 5

Conclusioni

Come detto nell'introduzione, questo lavoro è stato un progetto pilota per testare la possibilità di combinare un approccio bayesiano per la segmentazione delle immagini, con metodi di minimizzazione di funzionali basati sulla risoluzione di PDE con il metodo degli elementi finiti e mesh anisotrope. I risultati sono stati buoni, ma sicuramente migliorabili, segno che la strada da noi intrapresa è effettivamente percorribile.
Dal punto di vista puramente computazionale, abbiamo visto che il codice ci mette circa 40 secondi per segmentare un'immagine medio-piccola, con l'algoritmo da noi proposto (adattivo, anisotropo, con densità gaussiane). Il risultato è positivo ma si può migliorare, infatti nella stesura del codice non abbiamo prestato particolarmente attenzione all'ottimizzazione numerica. In particolare, si potrebbero usare dei risolutori paralleli per risolvere il sistema lineare legato all'equazione del calore, e una programmazione parallela per il calcolo della metrica, le due parti del codice che richiedono più tempo di calcolo.
Per quanto riguarda le densità paramentriche definite a priori abbiamo implementato solo quella gaussiana. Le diverse densità di probabilità che si possono scegliere sono legate al tipo di rumore presente nell'immagine, dato noto a priori. Per esempio si sa che le immagini biomediche presentano un errore di tipo *speckle*, oppure si potrebbe voler analizzare delle immagini corrotte dal rumore *sale e pepe*. Per questi tipi di rumore la densità gaussiana non è quella ottimale, e quindi bisognerà scegliere altri tipi di distribuzioni e implementarli nel codice.

Infine credo che ci sia la necessità di indagare più a fondo il modello matematico usato, anche il relazione alla cattiva segmentazione ottenuta in relazione all'esempio della figura 4.10. L'immagine in questione è un'immagine sintetica salvata in formato jpeg. È fortemente sconsigliato l'utilizzo del formato jpeg per immagini sintetiche a casua del presentarsi del fenomeno di Gibbs. Nella nella pratica corrente, quindi, questo tipo di immagini non dovrebbero mai presentarsi. Tuttavia la scorretta segmentazione di questa immagine potrebbe essere utilie per comprendere più a fondo il modello matematico utilizzato, anche in virtù del fatto che durante l'implementazione è stato cambiato rispetto a quello di partenza e, soprattutto, perché il modello usato è diverso dai modelli classici che si trovano in letteratura. Il modello da noi effettivamente utilizzato è quello bastato sulla minimizzazione seguente

$$\min_{\phi\in\mathcal{A}} \mathcal{F}(\phi) = \min_{\phi\in\mathcal{A}} \left\{ \int_\Omega \phi(x) \log \frac{f_E(I(X))}{f_I(I(x))} dx + \nu \int_\Omega g(I(x)) |\nabla\phi(x)|^2 dx \right\} \qquad (5.1)$$

dove $\mathcal{A} = \left\{\phi \in H^1(\Omega) \,\middle|\, -\alpha \leq \phi \leq \alpha \text{ q.o. su } \Omega,\ \nabla\phi \cdot \mathbf{n} = 0 \text{ su } \partial\Omega\right\}$ è l'insieme delle soluzioni ammissibili. Mentre il modello da cui siamo partiti era leggermene diverso e prevedeva la minimizzazzione di

$$\min_{\phi\in\mathcal{A}} \mathcal{F}(\phi) = \min_{\phi\in\mathcal{A}} \Bigg\{ -\int_{\Omega_I} \phi(x) \log f_I(I(x))dx + \tag{5.2}$$

$$\int_{\Omega_E} \phi(x) \log f_E(I(x))dx + \nu \int_{\Omega} g(I(x))|\nabla\phi(x)|^2 dx \Bigg\} \tag{5.3}$$

Infine, per quanto riguarda i modelli bayesiani, in letteratura si trova un altro modello ancora

$$\min_{\phi\in H^1(\Omega)} \mathcal{F}(\phi) = \min_{\phi\in H^1(\Omega)} \Bigg(\int_{\Omega_I} \log f_I(I(x))dx - \tag{5.4}$$

$$\int_{\Omega_E} \log f_E(I(x))dx + \nu \int_{\Omega} g(I(x))|\nabla\phi(x)|^2 dx \Bigg) \tag{5.5}$$

ovviamente in tutti e tre i casi è implicito che $\Omega_I = \{\, x \in \Omega \,|\, \phi(x) > 0 \,\}$, mentre $\Omega_E = \Omega \setminus \Omega_I$ e $\partial\Omega = \{\, x \in \Omega \,|\, \phi(x) = 0 \,\}$.
Per quanto riguarda il funzionale (5.5) che si trova in letteratura, poiché gli integrali non sono su tutto il dominio Ω, nella funzione lagrangiana compare la funzione di Heaviside $H(\phi(x))$ che solitamente si approssima con la funzione $H_\epsilon(\phi(x))$, dove $H_\epsilon(y) = \frac{1}{2} + \frac{1}{\pi}\arctan\left(\frac{y}{\epsilon}\right)$. Provando a minimizzare questo funzionale tramite l'utilizzo delle equazioni di Eulero-Lagrange si ottengono delle equazioni non lineari, e quindi difficili da risolvere numericamente con il metodo degli elementi finiti.
Per quanto riguarda il modello di partenza (5.3), tramite l'algoritmo di split di Bregman abbiamo trovato il modo, forse non proprio legittimo, di evitare di dover derivare la funzione di Heaviside, ottenendo il tal modo un'equazione lineare, molto più semplice da risolvere numericamente. Tuttavia, questo modello presenta delle problematiche, poiché le densità di probabilità sono funzioni spesso comprese tra 0 e 1 e quando ne facciamo il logaritmo diventano quantità negative. Questo, per esempio, capita sicuramente quando si usa un modello non parametrico delle densità, infatti l'istogramma delle frequenze non può oltrepassare il valore 1. Detto ciò, in questi casi, i tre integrali sono sicuramente non negativi, e prendendo $\phi(x) = 0$ per ogni $x \in \Omega$ il funzionale vale esattamente 0, quindi la soluzione identicamente nulla è il minimo del funzionale, ma evidentemente questo minimo non risolve il nostro problema di segmentazione.
Questo problema è stato risolto considerando le probabilità relative anziché quelle assolute. Per esempio, anziché considerare la probabilità assoluta che un pixel appartenga alla regione interna, si confronta questa probabilità, con la probabilità che appartenga alla regione esterna. Così facendo si ottiene il funzionale (5.1) che abbiamo usato nel codice, esso infatti ci permette di risolvere il problema con un'equazione lineare e non ha problemi di consistenza. Tuttavia, il primo termine di questo funzionale può essere negativo, per tale motivo si è reso necessario limitare le funzioni ammissibili all'insieme $\mathcal{A}$, di modo che $-\alpha \leq \phi(x) \leq \alpha$.

Bibliografia

[1] T. Chan, L. A. Vese *Active Contours Without Edges*, IEEE Transaction on Image Processing (2001) 10, pp. 266-277.

[2] B. Chen, Q. Zou, Y. Li *A new image segmentation model with local statistical characters based on variance minimization*, Applied Mathematical Modelling (2015) 39, pp. 3227-3235.

[3] T. Borx, D. Cremers *On Local Region Models and Statisitcal Interpretation of the Piecwwise Smooth Mumoford-Shah Functional*, Int. J. Comput. Vision (2009) 84, pp. 184-193.

[4] D. Cremers, M. Rousson, R. Deriche *A Review of Statistical Approaches to Level Set Segmentation: Integrating Color, Texture, Motion and Shape*, Int. J. Comput. Vision (2007) 72, pp. 195-215.

[5] L. Dedè , S. Micheletti, S. Perotto *Anisotropic error control for environmental applications*, Applied and Numerical Mathematics (2008) 58, pp. 1320-1339.

[6] P. E. Farrell1, S. Micheletti, S. Perotto *An anisotropic Zienkiewicz–Zhu-type error estimator for 3D applications*, Int. J. for Numerical Methods in Engineering (2011) 85, pp. 671-692.

[7] F. Fedele, E. Faggiano, L. Barbarotta, F. Cremonesi, L. Formaggia, S. Perotto *Semi-Automatic Three-Dimensional Vessel Segmentation Using a Connected Component Localization of the Region-Scalable Fitting Energy*, 9th International Symposium on Image and Signal Processing adn Analysis, IEEE (2015), pp. 72-77.

[8] T. Goldstein, S. Osher *The Split Bregman Method for L1-Regularized Problems* SIAM J. Imaging Science (2009) 2, pp. 323-343.

[9] J. Kim, W. Fisher, A. Yezzi, M. Çetin, A. S. Willsky *A Nonparametric Statistical Method for Image Segmentation Using Information Theory and Curve Evolution*, IEEE Transaction on Image Processing (2005) 14, pp. 1486-1502.

[10] C. Li, C.-Y. Kao, J. C. Gore, Z. Ding *Minimization of Region-Scalable Fitting Energy for Image Segmentation*, IEEE Transaction on Image Processing (2008) 17, pp. 1940-1949.

[11] D. Mumford, J. Shah *Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems*, Commun. on Pure and Applied Math. (1989) 42, pp. 577-685.

[12] A. Sawatzky, D. Tenbrinck, X. Jiang, M. Burger *A Variational Framework for Region-Based Segmentation Incorporating Physical Noise Models*, J. Math. Imaging Vision (2013) 47, pp. 179-209.