PROGETTO **Marco Fariselli mat.0000851125**

Il progetto presenta la trattazione del problema dell'eliminazione di rumore in immagini digitali tramite l'utilizzo di equazioni alle derivate parziali. In particolare la risoluzione dell'equazione proposta da Perona e Malik per la diffusione:

$$\begin{cases} \dfrac{\partial I}{\partial t} = \nabla \cdot (c(x,y,t)\nabla I) \\ condizioni\ iniziali: I(x,y,0) = I_0(x,y,) \\ condizioni\ al\ contorno: \dfrac{\partial I}{\partial t} = 0 \end{cases}$$

Questa equazione modella il problema della diffusione anisotropa del calore, nella quale cioè la diffusività g=c non è una costante ma è funzione del gradiente dell'immagine. Per discretizzare questa equazione si possono usare schemi espliciti o semi-impliciti, a seguire verranno presentate entrambe le soluzioni applicate a due immagini, una medica e una fotografica, alle quali è stato aggiunto un rumore additivo di tipo gaussiano a valor medio nullo e varianza pari a 0,01, esponendo pro e contro di ogni schema.

SEMI-IMPLICITO

In questa parte viene presentato il codice Matlab relativo alla risoluzione dell'equazione di Perona Malik tramite un metodo semi implicito. L'equazione discretizzata per questo metodo diventa quindi:

$$\frac{(I_k)^{t+1} - (I_k)^t}{dt} = DIV(g^t \nabla (I_k)^{t+1})$$

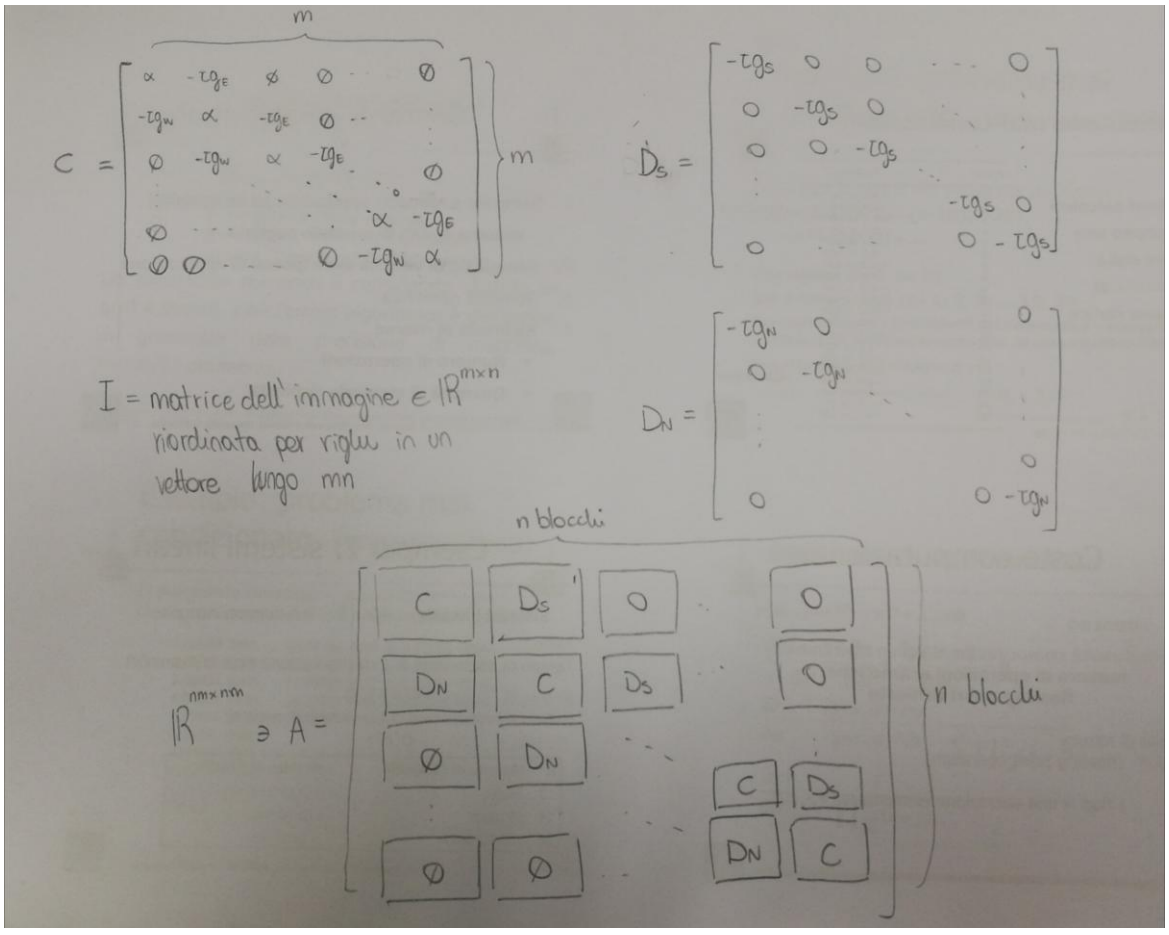
Dopo alcuni passaggi il termine a destra dell'equazione diventa:

$$\begin{aligned} \frac{\partial}{\partial x}\Big(gf_x\Big)(x,y) + \frac{\partial}{\partial y}\Big(gf_y\Big)(x,y) =& \Big(gf_x\Big)(x+0.5,y) - \Big(gf_x\Big)(x-0.5,y) \\ &+ \Big(gf_y\Big)(x,y+0.5) - \Big(gf_y\Big)(x,y-0.5) \\ =& g(x+0.5,y)\Big(f(x+1,y) - f(x,y)\Big) \\ &+ g(x-0.5,y)\Big(f(x-1,y) - f(x,y)\Big) \\ &+ g(x,y+0.5)\Big(f(x,y+1) - f(x,y)\Big) \\ &+ g(x,y-0.5)\Big(f(x,y-1) - f(x,y)\Big) \\ =& g_E \nabla_E f + g_W \nabla_W f + g_S \nabla_S f + g_N \nabla_N f \end{aligned}$$

L'immagine al passo t+1 con τ=dt diventa quindi:

$$
\begin{aligned}
(I_k)_{i,j}^{t+1}\left(1+\tau(g_N^t+g_S^t+g_W^t+g_E^t)\right) =& (I_k)_{i,j}^t \\
&+\tau g_N^t\left((I_k)_{i-1,j}^{t+1}\right) \\
&+\tau g_S^t\left((I_k)_{i+1,j}^{t+1}\right) \\
&+\tau g_W^t\left((I_k)_{i,j-1}^{t+1}\right) \\
&+\tau g_E^t\left((I_k)_{i,j+1}^{t+1}\right)
\end{aligned}
$$

Da queste formule viene derivata la matrice del sistema che, se ordiniamo immagine nuova e immagine vecchia per righe, avrà una forma di questo tipo:



Dove:

m = numero di righe dell'immagine;

n = numero colonne dell'immagine;

$$\alpha = 1 + dt(g_E + g_W + g_N + g_S)$$

Il sistema avrà la forma:

$$A\, I_k^{t+1} = I_k^t$$

Vediamo il codice associato a questo approccio con relativi commenti:

```
close all
clear all

M=menu('Seleziona il tipo di immagine', 'fotografica (cameraman)', 'medica (radiografia mano)');

switch M
    case 1
        I_original=imread('Cameraman256.png'); % load image
        imgstrg='cameraman semi implicito ';
    case 2
        I_original=imread('mano.jpg'); % load image
        imgstrg='mano semi implicito ';
end

I_original=I_original(:,:,1);
I_noise=imnoise(uint8(I_original),'gaussian',0,0.01);
I_noise=double(I_noise); % cut a piece, convert to double
I_original=double(I_original);

[rows,cols]=size(I_noise);

I_iter=I_noise;
numpassi=10;
peack_snr=zeros(1,numpassi);
peack_snr_iniziale=psnr(I_noise,I_original,255);
edge_mean=zeros(1,numpassi);

for h=1:numpassi
```

```
    tic
%creo la matrice dell'immagine con le condizioni di Neumann
    xi=2:rows+1;
    yi=2:cols+1;
    I_neumann_pre_filt=zeros(rows+2,cols+2);
    I_neumann_pre_filt(xi,yi) = I_iter(:,:,1);

    I_neumann_pre_filt(1,yi)= I_neumann_pre_filt(3,yi);
    I_neumann_pre_filt(end,yi)= I_neumann_pre_filt(end-2,yi);
    I_neumann_pre_filt(xi,1)= I_neumann_pre_filt(xi,3);
    I_neumann_pre_filt(xi,end)= I_neumann_pre_filt(xi,end-2);

   % faccio un filtraggio blando con un filtro gaussiano
   I_neumann_post_filt=imgaussfilt(I_neumann_pre_filt,0.001);

    %calcolo il valore medio del gradiente e lo uso per determinare k
    [edge_mean(h), modgrad] = edge_grad_implicito(I_neumann_post_filt);

    switch M
        case 1
            %per immagine normale
```

```
            K=0.75*edge_mean(h);
            dt=0.75;
        case 2
            %per immagine medica
            K=3*edge_mean(h);
            dt=0.75;
    end

    b=1/K^2;  %b=1/k^2

    [gn,ge,gs,gw]=my_g(I_neumann_post_filt,b);
    mat_term_noti=I_iter;
```

Nella sezione sopra si può notare come vengono settati i parametri principali come il numero di output (numpassi=10) e il parametro k, che compare nella espressione della diffusività g (b=1/k^2):

$$g(|\nabla u|) = \frac{1}{1 + \frac{|\nabla u|^2}{k^2}}$$

k viene scelto in funzione (variabile a seconda del tipo di immagine: medica, fotografica, ecc..) del valore medio del modulo del gradiente dell'immagine da trattare (calcolato tramite la funzione edge_grad_implicito()) in modo che, nell'equazione differenziale, g risulti circa 0 nei punti in cui l'immagine presenta un gradiente molto più grande rispetto alla sua media su tutta l'immagine, e circa 1 quando il gradiente risulta molto più piccolo. In questo modo quando il gradiente è molto elevato, e quindi ho degli edge che non voglio filtrare (esempio il bordo di un dettaglio della fotografia), g diventa molto piccolo e il filtro non agisce particolarmente, mentre quando ho degli edge piccoli, cioè rumore che voglio eliminare, il filtro agisce e uniforma la zona.

Viene inoltre creata la matrice su cui calcolare i gradienti (I_neumann_pre_filt) che non è altro che l'immagine originale con i bordi ricopiati in una cornice più esterna (ghost points), in questo modo il gradiente sui bordi della matrice è uguale a 0 coerentemente con le condizioni di Neumann (vengono ricopiati i bordi interni cioè seconda e penultima riga e colonna perché nel calcolo del gradiente sono state usate derivate centrali). A tale matrice viene poi applicato un prefiltraggio tramite la funzione imgaussfilt() già presente in Matlab, per eliminare in maniera preliminare parte del rumore in modo uniforme.

Vengono poi calcolate le g relative ai 4 punti intermedi (N,S,W,E) dei vari pixel della mia nuova matrice tramite la funzione my_g.

Si noti come, in questo tipo di approccio (semi-implicito), non ci siano vincoli su dt per la stabilità (mentre nell'esplicito è necessario imporre un dt<0,25) è però ragionevole prendere valori di dt non troppo grandi e, piuttosto, applicare più volte il metodo. Per le immagini considerate ho trovato, tramite prove sperimentali, un valore ottimo di dt pari a 0,75. Con questo dt si trovano oltre a psnr con valori più alti, anche immagini visivamente più uniformi nelle zone prima rumoreggiate.

Impongo le Neumann nel vettore dei termini noti

```
    %Aggiungo i 4 vettori che andranno poi nei termini noti per imporre le Neumann

    for i=1:rows
```

```
        mat_term_noti(i,1)=I_iter(i,1)+dt*gw(i,1)*I_iter(i,1);
        mat_term_noti(i,cols)=I_iter(i,cols)+dt*ge(i,cols)*I_iter(i,cols);
    end
    for i=1:cols
        mat_term_noti(1,i)=I_iter(1,i)+dt*gn(1,i)*I_iter(1,i);
        mat_term_noti(rows,i)=I_iter(rows,i)+dt*gs(rows,i)*I_iter(rows,i);
    end
```

Nella sezione sopra creo una matrice dei termini noti (quella che poi verrà ordinata in un vettore) e aggiungo nei bordi di tale matrice dt*gx*bordo_x dell'Immagine, coerentemente con le condizioni di Neumann imposte dal problema. (Questa formula è stata verificata manualmente su matrici di piccole dimensioni).

Sistemo la matrice in un vettore (row-wise)

```
    vett_term_noti=reshape(mat_term_noti',cols*rows,1);
```

Costruzione della matrice e risoluzione del sistema lineare

```
%costruisco la matrice A e risolvo il sistema A* vett_new_image=vett_term_noti
%utilizzando il metodo del gradiente coniugato precondizionato già implementato da Matlab

    A  = semi_implicit_matrix( I_noise, dt, gn, ge, gs, gw );

    tol=1e-7;
    maxit=50;
    [vett_new_image,flags,relres,iter]=pcg(A,vett_term_noti,tol,maxit); % senza precondizionatore
```

La matrice A viene calcolata da semi_implicit_matrix, questa, essendo di dimensioni mnxmn, è troppo grande per essere salvata normalmente, il fatto che la maggior parte degli elementi di A siano 0 però ci permette di salvarla in modalità sparsa tramite la funzione sparse di Matlab. Per costruzione A è simmetrica, poiché avendo calcolato le gx con i punti intermedi, la ge e la gw (e anche gs e gn) di due punti adiacenti risultano uguali. E' anche definita positiva poiché a diagonale dominanza (α>dt*(ge+gs+gn+gw)). Per risolvere il sistema posso quindi usare un metodo per matrici SPD come il gradiente coniugato. Risolvo il sistema con la funzione di matlab pcg(), senza precondizionatore perché risultato più performante rispetto a dover anche calcolare una matrice di precondizionamento (A già ben condizionata).

Riordino il vettore creando una matrice

```
%costruisco l'immagine nuova risistemando il vettore new_image in una
%matrice della stessa dimensione di quella di partenza

    new_image=reshape(vett_new_image,rows,cols);
    new_image=new_image';

    I_iter=new_image;
    peack_snr(h)=psnr(I_iter,I_original,255);

    figure
    imshow(uint8(new_image)),title(strcat(imgstrg,strcat(' ',string(h),' iterazione'))),
xlabel(strcat('psnr=',string(peack_snr(h))));
toc
```

Avendo ora la soluzione del mio filtraggio in un vettore lungo mn questo viene riordinato in una matrice mxn, ovvero la mia immagine filtrata, che viene quindi stampata a video.

cameraman semi implicito originale

cameraman semi implicito con rumore

psnr=20.3494

cameraman semi implicito1 iterazione

psnr=23.413

cameraman semi implicito2 iterazione

psnr=24.8684

```
Elapsed time is 1.220526 seconds.
```

```
Elapsed time is 0.567297 seconds.
```

cameraman semi implicito3 iterazione

psnr=25.4036

cameraman semi implicito4 iterazione

psnr=25.513

```
Elapsed time is 0.515122 seconds.
```

```
Elapsed time is 0.464305 seconds.
```

cameraman semi implicito5 iterazione

psnr=25.4442

Elapsed time is 0.407885 seconds.

cameraman semi implicito6 iterazione

psnr=25.3099

Elapsed time is 0.377392 seconds.

cameraman semi implicito7 iterazione

psnr=25.1544

Elapsed time is 0.367613 seconds.

cameraman semi implicito8 iterazione

psnr=24.9938

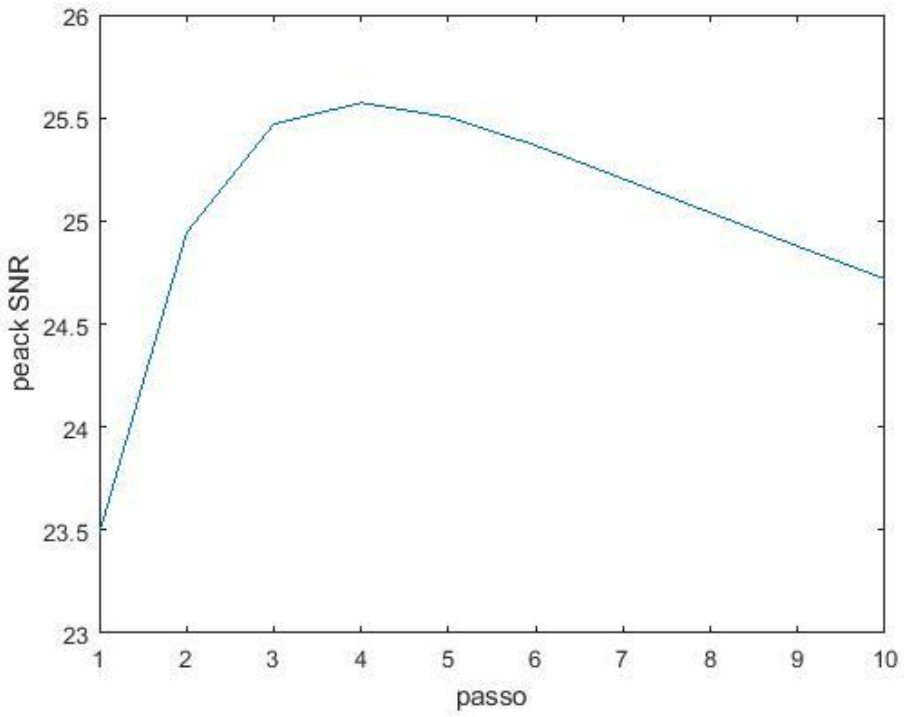
Elapsed time is 0.377667 seconds.

psnr=24.8349

```
Elapsed time is 0.377667 seconds.
```

psnr=24.6802

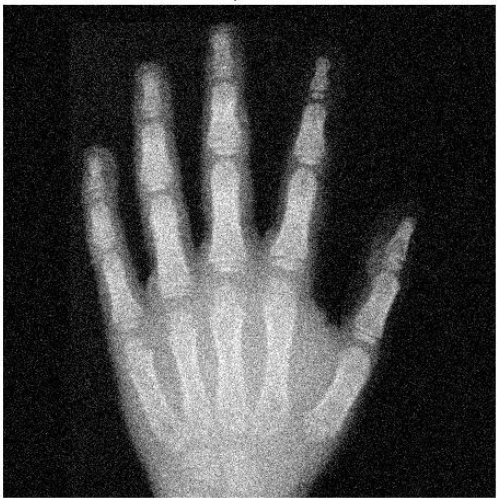
```
Elapsed time is 0.371920 seconds.
```



Sopra sono mostrate tutte le immagini generate ad ogni iterazione con indicati i relativi psnr calcolati rispetto all'immagine originale e il tempo necessario per elaborarla. Come si può vedere dall'andamento del psnr nel grafico sopra riportato, l'immagine che mostra il psnr maggiore risulta quella alla quarta iterazione, sono state mostrate però tutte per dare la possibilità all'utente di scegliere quella più congeniale alla sua applicazione in quanto non sempre l'snr è indice di qualità dell'immagine.

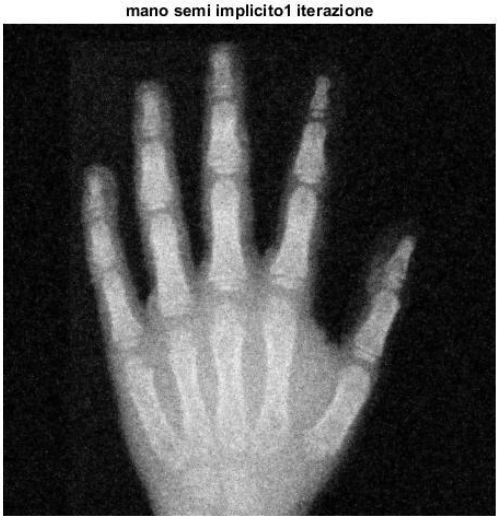
mano semi implicito originale

mano semi implicito con rumore



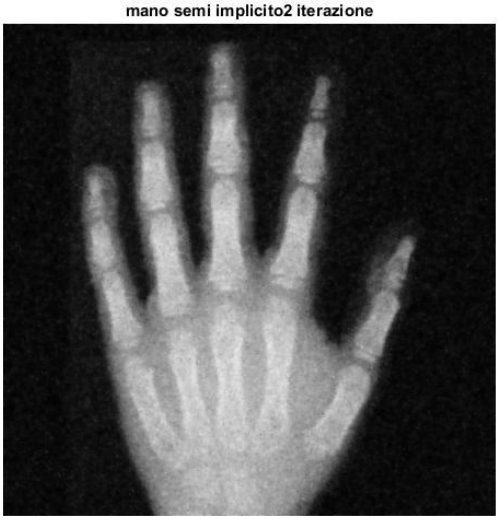
psnr=21.5204

mano semi implicito1 iterazione



psnr=26.756

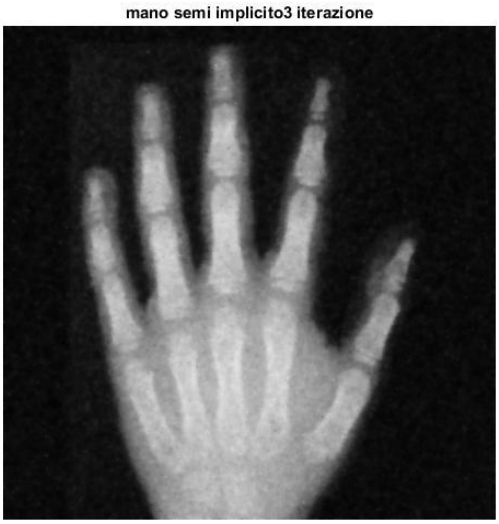
mano semi implicito2 iterazione



psnr=28.437

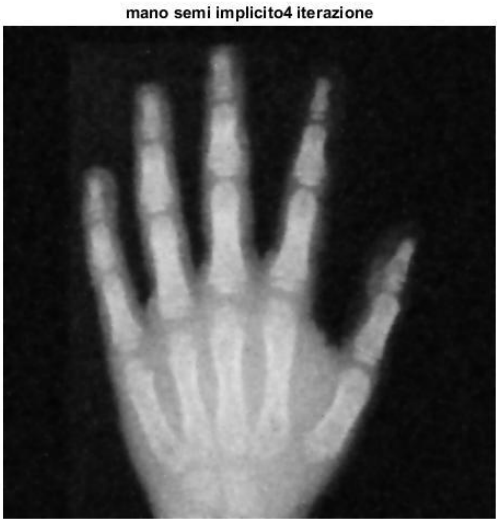
Elapsed time is 1.220526 seconds.

Elapsed time is 0.567297 seconds.



psnr=28.8488

Elapsed time is 0.515122 seconds.



psnr=28.9203

Elapsed time is 0.464305 seconds.

psnr=28.884

Elapsed time is 0.407885 seconds.

psnr=28.8061

Elapsed time is 0.377392 seconds.

psnr=28.71

Elapsed time is 0.367613 seconds.

psnr=28.6057

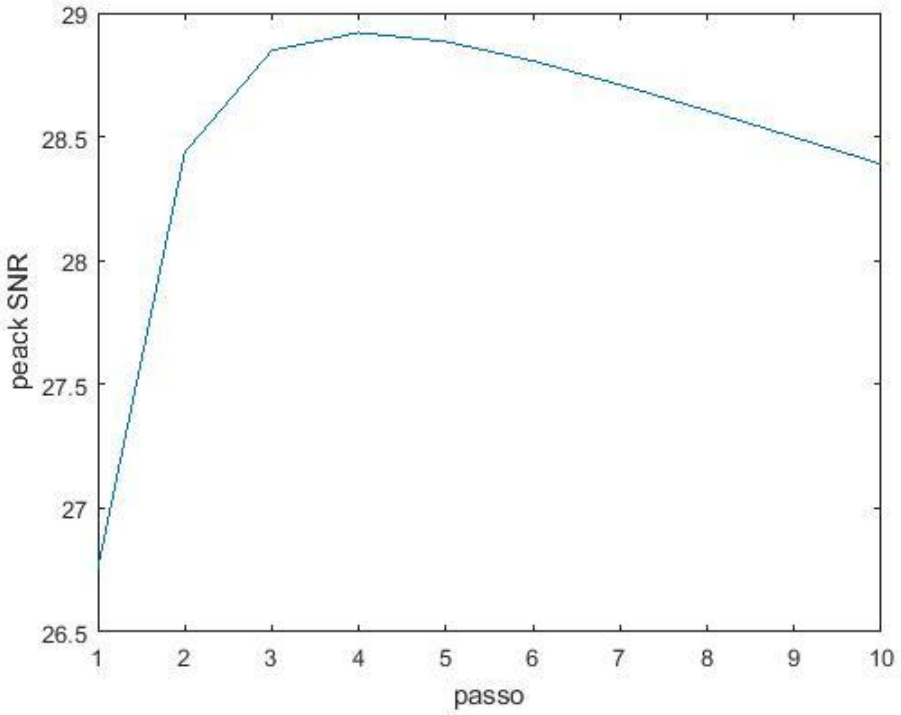
Elapsed time is 0.377667 seconds.

psnr=28.498

Elapsed time is 0.377667 seconds.

psnr=28.3893

Elapsed time is 0.371920 seconds.



Anche per le immagini mediche, quella con psnr migliore non risulta all'ultima iterazione ma circa alla 4ª. Nel running dello script vengono però mostrate tutte per permettere all'utente di visualizzarle e scegliere quella che si adatta di più al suo scopo.

ESPLICITO

Per il metodo esplicito è stata invece risolta l'equazione in questa forma:

$$\frac{(I_k)^{t+1} - (I_k)^t}{dt} = DIV(g^t \nabla (I_k)^t)$$

Quindi:

$$(I_k)^{t+1} = (I_k)^t + dt\, DIV(g^t \nabla (I_k)^t)$$

Con questo schema non serve sviluppare una matrice di iterazione e risolvere un sistema lineare, ma semplicemente basta trovare ogni pixel della nuova immagine applicando la formula sopra, l'occupazione di memoria è quindi molto più limitata.

Il codice che permette di implementare questo tipo di approccio è il seguente:

```
clear all
close all
format compact
format long

M=menu('Seleziona il tipo di immagine', 'fotografica (cameraman)', 'medica (radiografia mano)');

switch M
    case 1
        I_original=imread('Cameraman256.png'); % load image
        imgstrg='cameraman esplicito ';
    case 2
        I_original=imread('mano.jpg'); % load image
        imgstrg='mano esplicito ';
end


I_original=I_original(:,:,1);
I_noise=imnoise(uint8(I_original),'gaussian',0,0.01);
%aggiungo del rumore all'immagine originale facendo finta
%che sia questa l'immagine a mia disposizione:
%tramite delle elaborazioni dovrò cercare di tornare il più possibile a
%quella di partenza.

I_noise=double(I_noise); % cut a piece, convert to double
I_original=double(I_original);

[ edge_mean2, modgrad2]=edge_grad_2(I_noise);

 switch M
%K consigliato in base a prove sperimentali:
        case 1
            %per immagine normale
            K=edge_mean2/20;
        case 2
            %per immagine medica
            K=edge_mean2/15;
    end
```

```
    b=1/K^2;  %b=1/k^2
%b consigliato in base a prove sperimentali:

[B2, peacksnr2] = PeronaMalik( I_noise, b, 10, strcat(imgstrg,'2') );
%guardando il peacksnr delle immagini filtrate sembrerebbe migliore quello
%di PM21, poichè, anche se di poco, più grande rispetto alle altre immagini.
%All'occhio umano risulta però migliore l'immagine PM25.
```

La funzione PeronaMalik che implementa l'effettiva risoluzione dell'equazione, è stata modificata rispetto a quella fornita, in modo che dia in uscita anche il picco del rapporto segnale-rumore di ogni immagine:

```
function [B, peacksnr] = PeronaMalik( A, b, tfinal, outfilename )

fileout = 5;    % numero di passi che si intendono fare

dtout = tfinal/fileout;
time=0;
[m,n,p]=size(A);
xi=2:m+1;
yi=2:n+1;
B=zeros(m+2,n+2);
B(xi,yi) = A(:,:,1);
j=1; %serve per creare il vettore dei peacksnr

while time < tfinal
tic
    % du/dn=0 on the boundary: vengono imposte le condizioni di Neumann al
    % contorno tramite l'allargamento della matrice dell'immagine di una
    % riga sopra, una sotto, una colonna a destra e una a sinistra, e
    % copiando la cornice più esterna dell'immagine originale anche nella
    % nuova cornice creata. In questo modo i pixel nelle prime due e ultime
    % due righe/colonne avranno lo stesso valore --> derivata numerica = 0.

    B(1,yi)= B(2,yi);
    B(end,yi)= B(end-1,yi);
    B(xi,1)= B(xi,2);
    B(xi,end)= B(xi,end-1);

    % g(|grad u|^2)

    C = gaussian(B,2,5);
    % viene prima filtrata in modo più superficiale e uniforme l'immagine
    % originale tramite un filtro gaussiano con matrice di convoluzione
    % settabile (in questo caso 5x5). (si veda function gaussian)

    [gn,ge,gs,gw]=g(C,b);
    % vengono create le varie g riferite ai gradienti della matrice C
    % (destro-w, sinistro-e, superiore-n, inferiore-s). (function g in
    % fondo a questo file)

    Bn = B(xi,yi+1);
    Be = B(xi+1,yi);
    Bs = B(xi,yi-1);
    Bw = B(xi-1,yi);
    Bc = B(xi,yi);
```

```
    % u^k+1 = u^k + dt * div[ g(u) grad(u) ]

    temp = gn.*Bn + ge.*Be + gs.*Bs + gw.*Bw ...
        - (gn+ge+gs+gw).*Bc;

    dt = 0.25*max(max( max(max(max(gn,ge),gs),gw) ));
    %essendo g una funzione sicuramente minore uguale di 1, il dt massimo
    %non supererà 0.25 coerentemente a quanto detto per il metodo esplicito
    dt = min(dt,tfinal-time);
    %viene ulteriormente ridotto nel caso in cui, dopo tot passi, ci si trovi a
    %un passo inferiore di 0.25 dal tfinal.
    time=time+dt;
  %  [time dt tfinal]

    B(xi,yi)=B(xi,yi)+dt*temp;

    % check for intermediate output

    if (floor(time/dtout) ~= floor((time-dt)/dtout) )
        figure
        filename=[outfilename num2str(floor(time/dtout)) '.jpg'];
        imwrite(uint8(B(xi,yi)),filename)
        peacksnr(j)=psnr(B(2:m+1,2:n+1)/255, A/255);
        imshow(uint8(B(xi,yi))), title(filename), xlabel(strcat('psnr = ',string(peacksnr(j))));
        j=j+1;
        toc
    end
end

B=B(2:end-1,2:end-1);

return
```

Nella funzione, temp rappresenta:

$$temp = DIV(g^t \nabla(I_k)^t) = g_N^t\left(I_{i-1,j}^t - I_{i,j}^t\right) + g_S^t\left(I_{i+1,j}^t - I_{i,j}^t\right) + g_W^t\left(I_{i,j-1}^t - I_{i,j}^t\right) + g_E^t\left(I_{i,j-1}^t - I_{i,j}^t\right)$$

Il risultato, sia per immagini fotografiche che per immagini mediche è il seguente, anche qui viene inserito il rapporto segnale rumore e il tempo impiegato per elaborare ogni immagine:

psnr =22.0668

psnr =21.3412

Elapsed time is 1.075296 seconds.

cameraman esplicito23.jpg

psnr =20.9918

Elapsed time is 0.783900 seconds.

cameraman esplicito25.jpg

psnr =20.6126

Elapsed time is 0.853407 seconds.

Elapsed time is 0.777521 seconds.

cameraman esplicito24.jpg

psnr =20.7637

Elapsed time is 0.742066 seconds.

mano esplicito21.jpg

psnr =24.2872

Elapsed time is 2.149213 seconds.

mano esplicito22.jpg

psnr =23.6077

Elapsed time is 1.845072 seconds.

mano esplicito23.jpg

psnr =23.2754

Elapsed time is 1.928993 seconds.

mano esplicito24.jpg

psnr =23.0634

Elapsed time is 1.981814 seconds.

psnr =22.9282

Elapsed time is 1.871359 seconds.

CONCLUSIONI

Dai dati sperimentali ottenuti possiamo concludere che, con il metodo esplicito, siamo in grado di sviluppare algoritmi molto semplici con una occupazione di memoria ridotta, ma che ci portano a risultati significativi solo dopo molto tempo e tante elaborazioni avendo un grosso vincolo di stabilità sul dt. Con lo schema semi-implicito invece ci serve un approccio più complesso con lo sviluppo e risoluzione di sistemi di grandi dimensioni (siamo costretti a trattare con matrici sparse per non incorrere in errori di out of memory) che però ci porta ad una soluzione incondizionatamente stabile (per qualsiasi valore di dt) e quindi arriviamo a risultati evidenti in meno tempo computazionale. E' quindi preferibile l'utilizzo dello schema semi-implicito.