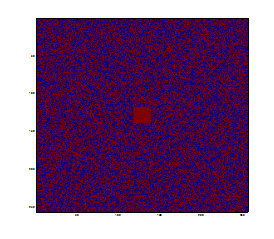
* + - * + *Esercitazione n. 3.9: Compressed sensing applicato ad un’immagine RM*
* Importare il file Matlab I0.mat



* Creare una maschera di campionamento random con fattore di accelerazione pari a 2 (viene campionato metà del K-spazio). Avere cura di riservare un’area fully sampled nel K-spazio, che corrisponde a campionare il K-spazio come in figura.



* Creare gli operatori inline per la realizzazione delle operazioni A\*x e AT\*x. In particolare il primo operatore effettua la trasformata di Fourier e poi applica la maschera di campionamento, mentre il secondo effettua l’antitrasformata di Fourier.

Per realizzare la maschera di campionamento utilizzare le seguenti istruzioni:

R=2; % indice di sottocampionamento

Np =floor(N\*N/R);

mask = 0\*I0;

p=randperm(N\*N);

mask(p(1:Np))=1;

mask(end/2-10:end/2+10,end/2-10:end/2+10)=1;

figure,imagesc(mask)

* Effettuare il campionamento dell’immagine e aggiungere rumore.
* Calcolare l’immagine AT\*y e visualizzarla. Notare come appare l’aliasing incoerente.



* Ricostruire l’immagine usando i seguenti algoritmi:
  + *L’algoritmo iterativo classico*

for it = 1:nIt

res = At(A(Ii)) - Iadj;

Ii = Ii - stepSize\*res;

end

* + *Il soft thresholding nel dominio wavelet*

%% CS nel dominio wavelet

lambda = 1e-3;

stepSize = 1/2;

prevObj = Inf;

c = 1;

Ii = Iadj;

for it = 1:nIt

Ax = A(Ii);

res = At(Ax) - Iadj;

tmpI = Ii - stepSize\*res;

% dominio wavelet

[Cr,S]=wavedec2(real(tmpI),8,'db1'); %Psi\*x

[Ci,S]=wavedec2(imag(tmpI),8,'db1'); %Psi\*x

C = Cr+1j\*Ci;

%soft thresholding

tmp =1-lambda\*stepSize./abs(C);%soft thresh

C2 = tmp.\*(tmp>0).\*C;

% dominio dell’immagine

tmpI = waverec2(real(C2),S,'db1')+1j\*waverec2(imag(C2),S,'db1');%Psi^T\*x

idx = find(mask(:));

Obj = .5\*norm(Ax(idx) -y(idx))^2+lambda\*sum(abs(C2(:)))

Ii = tmpI;

figure(100),subplot(1,2,1),imagesc(abs(Ii-I0)),colormap gray, colorbar, title('Res.'),ylabel(sprintf('iter %d',it))

subplot(1,2,2),imagesc(abs(Ii)),colormap gray, colorbar, title('I'), drawnow

end

ICS= Ii;

figure,imagesc(abs([[I0,Iadj,ILS,ICS];[I0-I0,Iadj-I0,ILS-I0,ICS-I0]])),colormap gray

* *l’algoritmo FISTA\_wavelet.*

%% FISTA

% nIt=50; % 200 tipico

lambda = 10e-3;

rho=1/2;

x=Iadj\*0;

r=Iadj;

t=1;

prevObj=Inf;

xprev=Iadj;

for it=1:nIt

Ar=A(r);

AtAr=At(Ar);

grad\_r=(AtAr-Iadj);

xprev=x;

x=r-rho\*grad\_r;

[cr,s] = wavedec2(real(x),8,'db1');

[ci,s] = wavedec2(imag(x),8,'db1');

c=cr+ci\*1j;

c=(1-2\*rho\*lambda./abs(c)).\*((1-2\*rho\*lambda./abs(c))>0).\*c;%soft Tresh

xr=waverec2(real(c),s,'db1');

xi=waverec2(imag(c),s,'db1');

x= xr+1j\*xi;

tprev=t;

t=(1+sqrt(1+4\*t^2))/2;

r=x+(tprev-1)/t\*(x-xprev);

figure(100),subplot(1,2,1),imagesc(abs(x)),colormap gray,title(sprintf('iter %i%',it)),drawnow

y1 = A(x);

idx=find(mask(:));

lsqrTerm=norm(y(idx)-Ar(idx))^2;

l1Term=norm(c(:),1);

obj=lsqrTerm+lambda\*l1Term;

%if (prevObj<obj && it>20),break,end;

prevObj=obj;

fprintf('Iter %i: obj=%5.4f l1=%5.4f lsqrTerm=%5.4f (est. sigma= %5.4f) \n',it,obj,l1Term,lsqrTerm,std(y(idx)-y1(idx)));

subplot(1,2,2),semilogy(it,(obj),'\*'),hold on, grid on,drawnow,title('obj')

end

ICS2=r;

%%

figure,imagesc(abs([[I0,Iadj,ILS,ICS,ICS2];[I0-I0,Iadj-I0,ILS-I0,ICS-I0, ICS2-I0]])),colormap gray

Risultati attesi



Immagine originale maschera di sottocampionamento del k=spazio immagine sottocampionata



Algoritmo iterative classico (AIC) Algoritmo ISTA



Algoritmo FISTA



Alto da sinistra: immagini originale, sottocampionata, ricostruita con AIC, con ISTA e con FISTA

Basso da sinistra: errori rispetto ai suddetti algoritmi