# Regolarizzazione di Tikhonov con sampling per problemi inversi

Luigina Mazzone

8 ottobre 2021

**Sommario**

In questo lavoro andremo ad analizzare metodi iterativi basati su tecniche di campionamento di dati per la risoluzione di sistemi lineari mal posti di grossa dimensione. Le tecniche di campionamento utilizzate renderanno computazionalmente maneggevole il sistema lineare originario, la cui soluzione avrebbe altresì un costo computazionale proibitivo, analizzando un più piccolo sottoinsieme di righe della matrice associata. Sfrutteremo inoltre due diversi metodi atti alla ricerca di un adeguato parametro di regolarizzazione, che consentirà convergenza del metodo iterativo ad una soluzione regolarizzata del problema di Tikhonov.

# Indice



# 1 Introduzione

Per il problema di nostro interesse, consideriamo un sistema lineare del tipo

$$b = Ax + \varepsilon \tag{1}$$

dove $x \in \mathbb{R}^n$ rappresenta l'incognita, $b \in \mathbb{R}^m$ il vettore contenente i dati osservati, $A \in \mathbb{R}^{m \times n}$ la matrice mal posta che modella il processo di acquisizione dati ed $\varepsilon$ rumore o eventuali errori nel processo di acquisizione. In questo lavoro assumeremo che $\varepsilon$ abbia media zero e secondo momento finito.

Il metodo più comunemente usato per la risoluzione di problemi mal posti è la regolarizzazione di Tikhonov che consiste nell'aggiungere un parametro di regolarizzazione $\lambda > 0$ nella formula dei minimi quadrati

$$\min_x f(x) = \|Ax - b\|_2^2$$

allo scopo di migliorarne il condizionamento. Otteniamo perciò il problema

$$\min_x f(x) = \|Ax - b\|_2^2 + \lambda \|Lx\|_2^2 \tag{2}$$

dove $\lambda$ è il parametro di regolarizzazione ed $L$ matrice opportunamente scelta di rango massimo che chiameremo "matrice di regolarizzazione". In questa relazione spesso considereremo $L = I_{m \times n}$

La convergenza alla soluzione di Tikhonov regolarizzata sarà fornita nel nostro caso da un particolare sistema iterativo della forma:

$$x_{k+1} = x_k - B_k g_k(x_k), \qquad k \in \mathbb{N} \tag{3}$$

La struttura di $B_k$ e $g_k$ sarà presentata nella sezione 1.2.

## 1.1 Campionamento dei dati

Come accennato in precedenza, l'obiettivo del lavoro è quello di risolvere problemi lineari mal posti di ingenti dimensioni e questo è computazionalmente possibile grazie ad un campionamento tramite apposite matrici. Definiamo pertanto le matrici di campionamento:

**Definizione:**
Siano $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ e supponiamo che solo un determinato numero di righe di $A$ e di elementi di $b$ diventi disponibile ad ogni iterazione. Per un certo $M \in \mathbb{N}$ fissato, definiamo un insieme di matrici (dette matrici di campionamento) $\{W_i\}_{i=1}^M$ tali che:

1. $\forall i \in \{1, ..., M\},\ W_i \in \mathbb{R}^{m \times l}$, dove $l = \frac{m}{M}$ (considereremo sempre $l, m, M$ interi);
2. $\sum_{i=1}^M W_i W_i^T = I_m$;

e indichiamo con $A_k = W_k^T A$ e $b_k = W_k^T b$ rispettivamente, gli elementi di $A$ e $b$ disponibili durante l'iterazione $k$. Spesso useremo come matrici $W_k$ colonne della matrice identità $I_{m \times n}$

## 1.2 Due metodi a confronto: rrls vs sTik

In questa sezione poniamo a confronto due metodi iterativi comunemente usati e due tecniche di campionamento ciclico. Il primo metodo iterativo che consideriamo è il metodo $rrls$ (regularized recursive least squares), le cui iterazioni sono della forma:

$$y_k = y_{k-1} - B_k A_k^T (A_k y_{k-1} - b_k), \qquad k \in \mathbb{N}$$

dove $B_k = (\lambda L^T L + \sum_{i=1}^k A_i^T A_i)^{-1}$. Questo metodo, come vedremo nell'esempio, verrà scartato in favore di un metodo più efficiente che consente di aggiornare ad ogni iterazione il parametro di regolarizzazione $\lambda = \sum_{i=1}^k \lambda_k$.

Il metodo $sTik$ è invece costituito dalle seguenti iterazioni:

$$x_k = x_{k-1} - B_k(A_k^T(A_k x_{k-1} - b_k) + \lambda_k L^T L x_{k-1}), \quad k \in \mathbb{N}$$

dove $B_k = (\sum_{i=1}^k \lambda_k L^T L + \sum_{i=1}^k A_i^T A_i)^{-1}$.

Riportiamo qui di seguito l'enunciato di un teorema dimostrato in [**1**]:

**Teorema 1** Siano $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Sia $L \in \mathbb{R}^{s\times n}$ una matrice a rango pieno e $W_i \in \mathbb{R}^{m\times l}$, $\forall i \in \{1, ..., k\}$ una arbitraria sequenza di matrici.

(i) Per $\lambda > 0$ e $y_0 \in \mathbb{R}^n$ arbitrario, l' iterazione di $rrls$ è la soluzione al problema dei minimi quadrati

$$\min_x f(x) = \|[W_1, ..., W_k]^T(Ax - b)\|_2^2 + \lambda\|L(x - y_0)\|_2^2$$

(ii) Per $\lambda = \sum_{i=1}^k \lambda_k > 0$ per ogni k e $x_0 \in \mathbb{R}^n$ arbitrario, l'iterazione $sTik$ è soluzione del problema dei minimi quadrati:

$$\min_x g(x) = \|[W_1, ..., W_k]^T(Ax - b)\|_2^2 + \lambda\|Lx\|_2^2$$

.

Questo risultato è valido per ogni scelta delle matrici $W_i$. Definiamo ora due metodi per il campionamento: campionamento randomico e campionamento ciclico randomico. Nel primo caso, all'iterazione $k$-esima, consideriamo $W_{\tau(k)}$ dove $\tau(k)$ rappresenta una variabile randomica che definisce una tecnica di campionamento, ponendo ad esempio $\tau(k)$ variabile con sistema di distribuzione uniforme nell'insieme $\{1, ..., M\}$ ne otteniamo un esempio. Il campionamento ciclico randomico corrisponde invece a porre $\forall j \in \mathbb{N}$, $\{\tau(k)\}_{jM+1}^{(j+1)M}$ uguale ad una permutazione dell'insieme $\{1, ..., M\}$. L'esempio più comune è porre: $\tau(k) = k \bmod M$. Introduciamo il concetto di epoch prima del prossimo enunciato: una epoch corrisponde al numero di iterazioni necessarie per analizzare ogni elemento della matrice $A$ e del vettore $b$; in altre parole, una epoch finisce quando nel campionamento ciclico randomico ho analizzato tutte le matrici $W_i$ al variare di $\{\tau(k)\}_{jM+1}^{(j+1)M}$. Presentiamo ora un altro risultato la cui dimostrazione è in [**1**]:

**Teorema 2** Siano $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Sia $L \in \mathbb{R}^{s\times n}$ una matrice a rango pieno e $W_i \in \mathbb{R}^{m\times l}$, $\forall i \in \{1, ..., k\}$ matrici con la proprietà $\sum_{i=1}^M W_i W_i^T = I_m$; e sia $\tau(k)$ una variabile randomica t.c. $\forall j \in \mathbb{N}$, $\{\tau(k)\}_{jM+1}^{(j+1)M}$ uguale ad una permutazione dell'insieme $\{1, ..., M\}$.
(i) Se $\lambda > 0$, $y_0 = 0$ e la sequenza $\{y_k\}$ è $rrls$, allora l'iterazione alla $j$-esima epoch è data come $y_{jM} = x(\frac{1}{j}\lambda)$.
(ii)Sia $\{\lambda_k\}$ una sequenza infinita con la proprietà che $\lambda = \sum_{i=1}^k \lambda_k > 0$. Se $x_0$ è arbitrario e la sequenza $\{x_k\}$ è definita come $sTik$, allora alla $j$-esima epoch, ho che $x_{jM} = x(\frac{1}{j}\lambda_{jM})$.

Notiamo che il parametro di regolarizzazione nel metodo $rrls$ tende a diminuire all'aumentare delle iterazioni $k$ e perciò, in particolare, se la matrice $A$ avesse rango

massimo, avremmo $lim_{j\to\infty} y_{jM} = x(0)$, che corrisponde alla soluzione non regolarizzata di Tikhonov. Il metodo $sTik$ converge invece alla soluzione regolarizzata di Tikhonov in quanto alla $j$-esima epoch ($j = k/M$), $x_{jM} = x_k = x(M/k\lambda_k)$ e $M/k\lambda_k > 0$ . Il nostro lavoro sarà dunque incentrato sul metodo $sTik$.

D'ora in avanti scriveremo $W_k$, $A_k$, $b_k$ per indicare $W_{\tau(k)}$, $A_{\tau(k)}$, $b_{\tau(k)}$.

# 2 Parametro di regolarizzazione

Uno dei motivi per cui preferiamo utilizzare il metodo sTik è dovuto alla possibilità di aggiornare il parametro di regolarizzazione ad ogni iterazione. Supponiamo ora di avere a disposizione $\forall i \in \{1, ..., k-1\}$ il parametro $\lambda_i$ e di voler ricavare il parametro $\lambda_k$. Osserviamo preliminarmente che la generica $k$-esima iterazione $sTik$ può essere scritta come:

$$x_k(\lambda) = C_k(\lambda)b, \tag{4}$$

$$C_k(\lambda) = ((\lambda + \sum_{i=1}^{k} \lambda_i)L^T L + \sum_{i=1}^{k} A_k^T A_k)^{-1} \sum_{i=1}^{k} A_k^T W_k^T. \tag{5}$$

presentiamo due metodi che useremo: sUpre(sampled Unbiased predictive risk estimator) e sGCV (sampled generalized cross validation), entrambi modifiche su un più piccolo insieme di algoritmi già esistenti per la ricerca di parametri di regolarizzazione.

## 2.1 sUpre

Il metodo sUpre è una variazione del metodo Upre ed è solitamente usato nel caso in cui è nota la varianza $\sigma^2$ della distribuzione normale corrispondente all'errore $\epsilon$ nei dati. La quantità $\mathbb{E}\|W_k^T(Ax_k(\lambda) - Ax_{\text{true}})\|_2^2$ è chiamata "rischio predittivo campionato" e il nostro obiettivo è quello di minimizzare questa quantità al variare di lambda. Il valore $\lambda_k$ cercato è il valore che minimizza il rischio. Possiamo riformulare la quantità nella forma equivalente:

$$\mathbb{E}\|W_k^T(Ax_k(\lambda) - b)\|_2^2 + \sigma^2 \mathbb{E}tr(W_k W_k^T A C_k(\lambda)) - \sigma^2 l,$$

da cui ricaviamo la funzione da minimizzare $U_k$:

$$U_k(\lambda) = \|W_k^T(Ax_k(\lambda) - b)\|_2^2 + \sigma^2 tr(W_k W_k^T A C_k(\lambda)) - \sigma^2 l$$

.

da cui $\lambda_k = \min_\lambda U_k(\lambda)$

Nel paragrafo 3 vedremo quale valore assegnare a $\sigma^2$.

## 2.2 sGCV

La principale differenza tra sGCV e l'algoritmo originario di partenza GCV è data dalle informazioni su A a disposizione, in sGCV abbiamo solo bisogno di una parte dei dati ($A_k, b_k$ e $W_k$). Il parametro $\lambda_k$ cercato sarà il minimizzatore della quantità:

$$G_k(\lambda) = \frac{l\|A_k x_k(\lambda) - b_k\|_2^2}{l - tr(A_k C_k(\lambda) W_k)}$$

# 3 Ricostruzione di immagini

## 3.1 Metodo sTik

Cominciamo con l'implementare su Matlab gli algoritmi visti inizialmente, senza occuparci del consumo di memoria. Prendiamo perciò una immagine di dimensioni 30x30 px che fungerà da $x$ (immagine originale) nel sistema (1).

La sfocatura dell'immagine sarà resa possibile dalla funzione PRblur.m del pacchetto IRTools [**3**], in questo esempio useremo la funzione PRblur che prende in input la dimensione $n$ dell'immagine e la struct $'options'$ data in output dalla funzione PRset. Nella funzione PRset, il livello e il tipo di sfocatura vanno precisati nelle opzioni, noi useremo il tipo di sfocatura di default (quello gaussiano) di livello 'mild' ovvero con $\sigma^2 = 2$.

Otteniamo pertanto la terna $(A, b, x)$ , dove $b = Ax + \varepsilon$, dove $\varepsilon \in \mathcal{N}(0, \sigma^2 I_n)$ come richiesto.

Costruiamo ora uno script in cui implementiamo il metodo sTik e utilizziamo gli algoritmi sUpre e sGCV per selezionare il parametro di regolarizzazione ad ogni iterazione. Per minimizzare le funzioni $U_k(\lambda) e G_k(\lambda)$ usiamo la funzione fminbnd.

### 3.1.1 Blurlevel:mild/medium

Consideriamo ora l'immagine 30x30 px 'Logo.png'. Il nostro obiettivo è quello di trasformare l'immagine in formato ".png" in una matrice 30x30 con valori da 0 a 1.

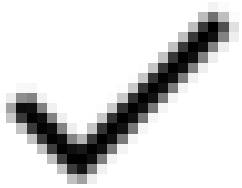

Figura 1: Immagine Originale

```
L=imread('Logo.png');
L=double(L);
L=rgb2gray(L);
L=L-min(L, [], 'all');
```

Il primo comando è necessario per ottenere la matrice L dall'immagine 'Logo', il secondo è necessario per poter operare su elementi della matrice L, mentre l'ultimo ci serve per poter rispettare le condizioni sul bordo, che vedremo in dettaglio nel prossimo paragrafo.

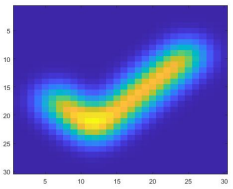

Figura 2: Blurlevel: mild

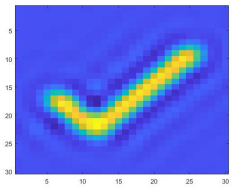

Figura 3: $x_{opt}$ con Blurlevel: mild (sUpre)

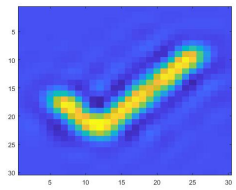

Figura 4: $x_{opt}$ con Blurlevel: mild (sGCV)

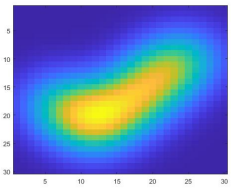

Figura 5: Blurlevel: medium

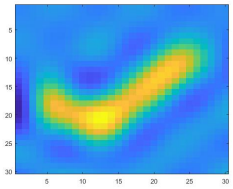

Figura 6: $x_{opt}$ con Blurlevel: medium (sUpre)

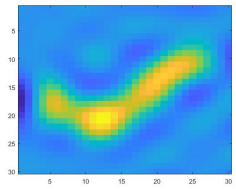

Figura 7: $x_{opt}$ con Blurlevel: medium (sGCV)

Cominciamo con una sfocatura con 'BlurLevel'='mild':

```
options = PRset('trueImage', L, 'Blurlevel', 'mild');
[A, b, x, ProbInfo] = PRblur(30, options);
```

Qui, come detto prima, non precisiamo il tipo di sfocatura e usiamo pertanto il modello di default gaussiano, in cui il valore di $\sigma^2$ (presente nelle funzioni $U_k$ e $G_k$), assume valore 2, 4 e 6 rispettivamente per livello di sfocatura basso, medio e alto.

Le figure 2-5 sono ottenute tramite il comando imagesc(). Nella Figura 2 poniamo un esempio di sfocatura gaussiana ed un esempio di immagine ricostruita con algoritmo sUpre (Figura 3) e algoritmo sGCV (Figura 4) con $k \sim 30$ iterazioni circa. La matrice $A$ ha dimensione $900 \times 900$, perciò il costo nel calcolo del minimo della funzione $U_k$ tende ad essere proibitivo per un eccessivo numero di iterazioni e il Teorema 2 ci assicura di ottenere un buon risultato per $k \sim n$.

L'operazione più costosa diventa appunto la ricerca del minimo della funzione $U_k$ o della funzione $G_k$. Una possibile soluzione consiste nello studiare l'andamento delle funzioni per fornire una prima stima della posizione del minimo e restringere così l'intervallo.

Un metodo per stimare l'ampiezza dell'intervallo è quindi stampare il grafico per un numero ragionevole di iterazioni e notare se è possibile restringere a priori l'intervallo:

```
if(i<10)
xp=-1:0.001:1;
yp=Plot(xp, U_k);
plot(xp, yp);
hold on
end
```

ovvero, nelle prime 10 iterazioni studiamo il comportamento della funzione sUpre( o sGCV), dove la funzione Plot valuta la funzione $U_k$ sugli elementi contenuti nell'array $x_p$. Nel nostro caso, la funzione sUpre ha un andamento che ci permette facilmente di restringere il dominio in 'fminbnd' :

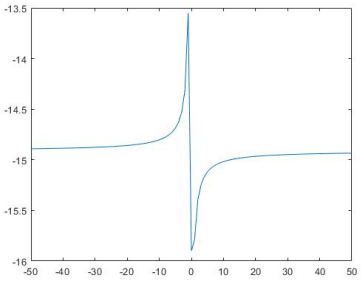


Figura 8: Grafico della funzione $U_k$ valutata in punti distanti 0.1

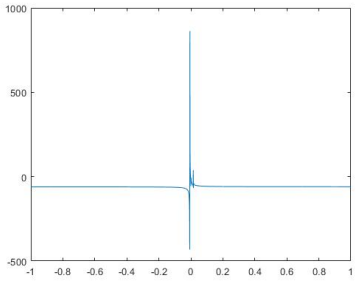


Figura 9: Grafico della funzione $U_k$ valutata in punti distanti 0.01

In questo caso, il minimo della funzione sUpre è in un punto compreso tra 0 e $10^{-3}$. Più difficile è invece ottenere il minimo della funzione $G_k$ di cui mostriamo il grafico per $k < 10$:

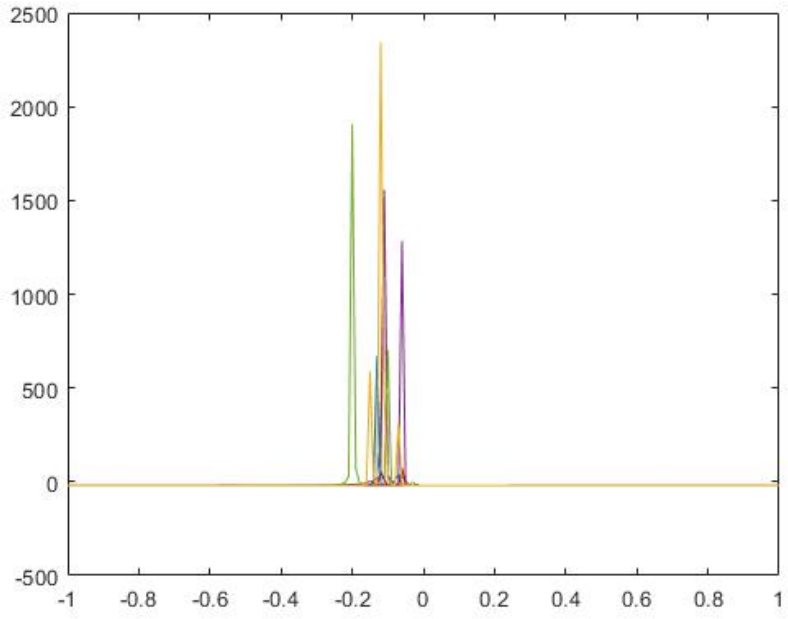


Figura 10: Grafico della funzione $G_k$ per $k < 10$

In questo caso, la difficoltà della minimizzazione della funzione $G_k$ ci porta a

procedere a tentativi, cercando l'ampiezza di un intorno di 0 nel quale l'algoritmo GCV fornisce una ricostruzione accettabile di "Logo.png". La ricerca del parametro $\lambda$ pertanto procede a tentativi.

## 3.2 Blurlevel: 'severe'

Nel caso in cui il livello di sfocatura è piuttosto alto (Figura 11), l'immagine 'Logo.png' diventa difficile da ricostruire a causa delle condizioni al bordo.

Come spiegato in [**2**], l'ideale sarebbe visualizzare la nostra immagine in uno sfondo scuro, ragion per cui utilizziamo inizialmente il comando

```
L=L-min(L, [], 'all');
```

Aggiungendo poi un contorno di 5 px neri all'immagine, l'algoritmo restituisce il risultato in figura (12).

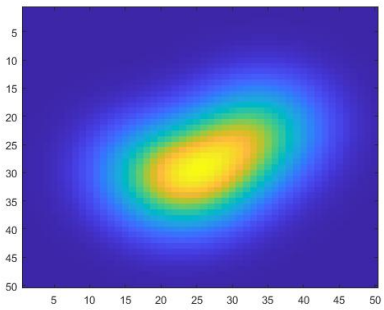

Figura 11: Blurlevel: severe

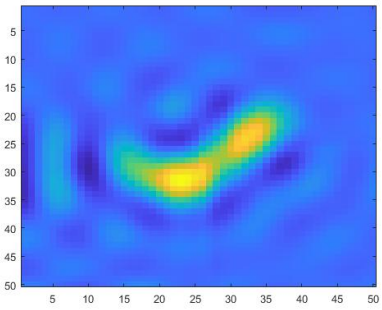

Figura 12: $x_{opt}$ con Blurlevel: severe

La necessità di avere uno sfondo scuro, ovvero di un contorno di pixel con valore pari a zero è evidenziata dall'effetto di "ringing" nel momento in cui andiamo a sfocare con il comando *PRBlur* la nostra immagine originale. Qui presentiamo a scopo puramente illustrativo la differenza tra immagini (di cui quella in Figura 13) lievemente sfocate ('BlurLevel'=mild):

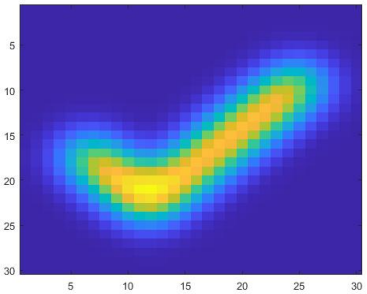

Figura 13: immagine sfocata senza 'ringing' sul bordo

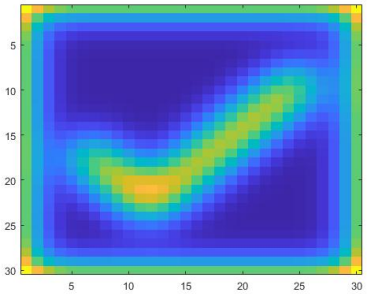

Figura 14: immagine sfocata con 'ringing' sul bordo

## 3.3 Esempi con diverse immagini

Qui di seguito inseriamo altri esempi di ricostruzione dell'immagine in Figura 15 usando l'algoritmo sUpre. Anche in questo caso, usando l'opzione 'severe' nella sfocatura, aggiungiamo un bordo di 5px con valore zero.

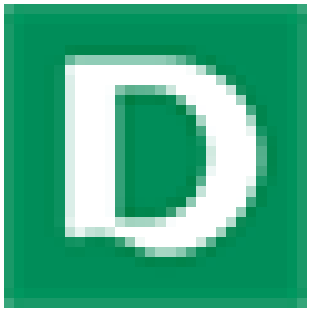

Figura 15: Immagine Originale

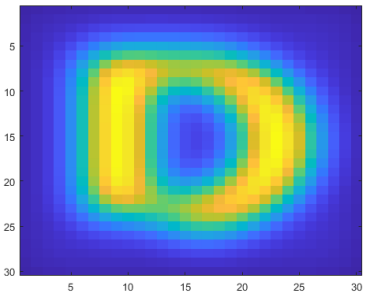


Figura 16: sfocatura 'mild'

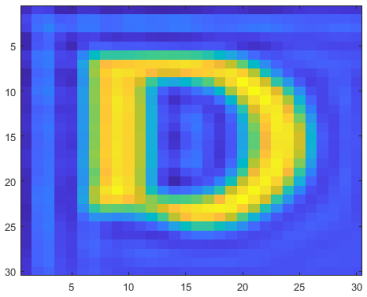


Figura 17: ricostruzione

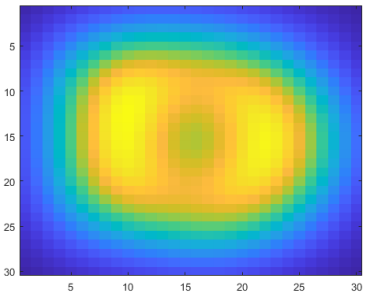


Figura 18: sfocatura 'medium'

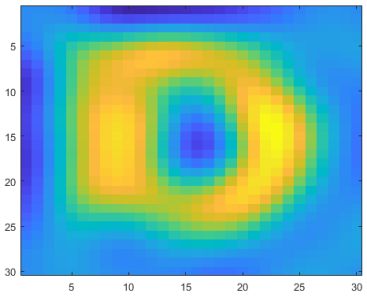


Figura 19: ricostruzione

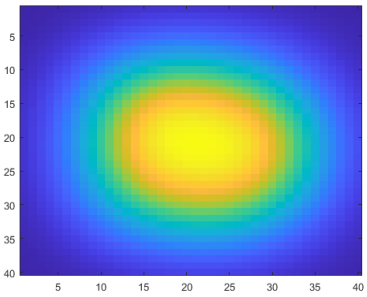


Figura 20: sfocatura 'severe'

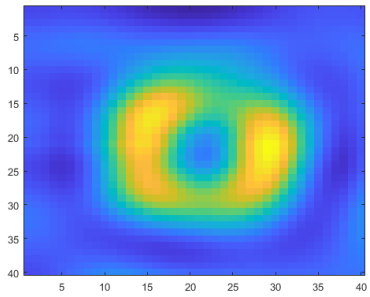


Figura 21: ricostruzione

# Riferimenti bibliografici

[1] J. Tanner Slagel, Julianne Chung, Matthias Chung, David Kozak, Luis Tenorio. *Sampled Tikhonov Regularization for Large Linear Inverse Problems*. `https://arxiv.org/abs/1812.06165`

[2] P C Hansen, J G Nagy, and D P O'Leary. *Deblurring Images: Matrices, Spectra, and Filtering.*. [*3.5 Boundary Conditions*] SIAM, Philadelphia, 2006. pag. 29-31.

[3] Gazzola Silvia, Hansen Per Christian, Nagy James G. *IR Tools - a MATLAB package of iterative regularization methods and large-scale test problems* `http://people.compute.dtu.dk/pcha/IRtools/`