Alma Mater Studiorum · Università di Bologna

facoltà di scienze matematiche, fisiche e naturali

Corso di Laurea Magistrale in Matematica

ANALISI E PROCESSI DI IMMAGINI ASTRONOMICHE

Tesi di Laurea in Calcolo Numerico

Relatore:
Chiar.ma Prof.
Carla Guerrini

Presentata da:
Alberto Landuzzi

Sessione III
Anno Accademico 2013-2014

A tutti quelli
che mi hanno sostenuto.

Introduzione

La scelta di questo argomento per la tesi é dovuta all'intenzione di mettere insieme due grandi passioni della mia vita, la matematica e l'astronomia.

Durante l'ultimo mezzo secolo gli sviluppi della tecnologia digitale hanno fatto fare enormi progressi agli scienziati di ogni settore, soprattutto in campo astronomico. Sin dalla loro introduzione alla fine degli anni 60, i dispositivi CCD (dall'inglese *charge-coupled device*), strumenti in grado di trasformare un segnale elettromagnetico in impulsi elettrici e questi a loro volta in un'immagine digitale, hanno dimostrato la loro grande potenzialitá rispetto alla fotografia tradizionale. Non solo il mondo dell'astronomia si é aperto anche al settore amatoriale, ma gli astronomi professionisti hanno avuto a loro disposizione uno strumento col quale erano possibili studi ed elaborazioni prima di allora impensabili. In campo astronomico quindi, l'elaborazione di immagini rappresenta un settore importante, ma che presenta innumerevoli problemi: da questioni piú semplici come il poter confrontare tra loro immagini dello stesso oggetto ripreso ad istanti diversi, immagini che risultano quindi distinte a causa per esempio della rotazione terrestre, fino a questioni piú delicate come il recupero di una immagine reale a partire da una degradata a causa di fenomeni di interferenza come l'azione dell'atmosfera sulla luce o difetti meccanico/elettronici della strumentazione. In questo elaborato vedremo una panoramica sulle tecniche di analisi ed elaborazione digitale di una immagine, a partire dalle operazioni piú elementari fino ai piú moderni algoritmi di deconvoluzione.

Nel dettaglio, nel primo capitolo vedremo le operazioni di base che si possono eseguire su una immagine; esse comprendono le trasformazioni geometriche classiche, con le quali é possibile archiviare una immagine, e le operazioni matematiche fondamentali quali per esempio addizione o moltiplicazione. Esamineremo nel dettaglio l'algoritmo di Larson-Sekanina, un metodo per lo studio delle comete che fa uso solo di queste operazioni. Nel secondo capitolo parleremo di operazioni puntuali, una categoria di operazioni che servono a modificare il contenuto di una immagine agendo direttamente sui valori di ogni pixel attraverso apposite funzioni dette *transfer function*. Nel terzo capitolo parleremo di convoluzione e di come questa operazione agisca sui pixel di una immagine. Esamineremo le proprietá principali dei *kernel* e ne daremo una classificazione in base all'effetto. Il quarto capitolo riguarderá la Trasformata di Fourier: partiremo dalla definizione nel caso continuo e ne vedremo alcune proprietá fondamentali, e poi sposteremo il discorso al caso discreto, analizzando nel dettaglio la costruzione dell'algoritmo FFT (*Fast Fourier Transform*) e il suo costo computazionale. Infine vedremo alcune sue applicazioni. Nel quinto capitolo parleremo di deconvoluzione, ovvero di quell'insieme di tecniche con cui tentare di ripristinare una immagine che é stata degradata a causa di vari fattori. In particolare studieremo nel dettaglio due algoritmi deconvolutivi e la loro convergenza: l'algoritmo di Richardson-Lucy e l'algoritmo di Van Cittert.

Buona parte delle immagini presenti nell'elaborato sono state create col software di elaborazione di immagini AIP4WIN disponibile con [1].

Indice

Introduzione **iii**

1 Operazione di base **1**

1.1 Trasformazioni geometriche 1

1.1.1 Applicazione: archiviazione di immagini 9

1.2 Operazioni matematiche 11

1.2.1 Applicazione: algoritmo di Larson-Sekanina 15

2 Operazioni Puntuali **25**

2.1 Isolare il range 27

2.2 Transfer functions 31

2.3 Esempi 33

3 Convoluzione **35**

3.1 Proprietá del *kernel* 38

3.2 Filtri spaziali 40

3.2.1 Filtri lowpass 41

3.2.2 Filtri highpass 42

3.3 Rilevamento bordi e gradienti 45

4 Trasformata di Fourier **47**

4.1 Serie di Fourier 48

4.2 Trasformata di Fourier 50

4.2.1 Fourier e convoluzione 51

4.3 Trasformata di Fourier discreta (DTF) 55
4.4 Proprietá della trasformata 58
4.5 Trasformata di Fourier veloce (FFT) 60
4.5.1 Algoritmo FFT 60
4.6 Applicazioni 64
4.6.1 Filtri lowpass 67
4.6.2 Filtri highpass 69
4.6.3 Filtri bandpass e bandstop 71

5 Deconvoluzione **73**
5.1 Algoritmo di Van Cittert 75
5.1.1 Convergenza dell'algoritmo di Van Cittert 76
5.1.2 Esempio 82
5.2 Algoritmo di Richardson-Lucy 82
5.2.1 Esempio 84

Bibliografia **85**

Capitolo 1

Operazione di base

1.1 Trasformazioni geometriche

In questo capitolo parleremo di una classe di operazioni che non agisce sul contenuto dell'immagine, bensí sul contenitore, ovvero sull'*array* dentro cui l'immagine é memorizzata. Esamineremo le piú comuni trasformazioni geometriche e vedremo per ognuna un possibile algoritmo per l'implementazione al calcolatore. Tra le trasformazioni geometriche figurano:

- traslazione;
- rotazione;
- scalamento;
- ribaltamento;
- taglio e aggiunta;

Nonostante il risultato di queste operazioni sembri abbastanza ovvio, anche in virtú della familiaritá di ognuno con queste ultime, non é invece altrettanto semplice la manipolazione effettiva dell'immagine, che ricordiamo nient'altro é che una struttura discreta di pixel, e puó portare in alcuni casi anche ad una perdita di informazioni dell'immagine.

Traslazione

L'operazione di traslazione agisce sull'immagine spostandola verso una nuova posizione, lasciando al contempo invariato il contenuto dell'immagine, e la sua applicazione piú frequente si ha in fase di inserimento di un set di immagini, per poterle portare tutte allo stesso sistema di riferimento ed essere per esempio combinate. Ricordando che possiamo pensare una immagine come una funzione a due variabili $I(x, y)$ dove x e y rappresentano rispettivamente le coordinate orizzontali e verticali di ogni pixel $P(x, y)$, le equazioni per traslare un punto sono date da:

$$\begin{cases} x' = x + x_t \\ y' = y + y_t \end{cases} \tag{1.1}$$

dove x' e y' sono le coordinate di P nella nuova immagine, x_t e y_t sono gli spostamenti lungo i due assi. Per quanto riguarda l'implementazione algoritmica, la trasformazione 1.1 presenta alcuni svantaggi: infatti potrebbe succedere che le nuove coordinate caschino fuori dai limiti massimi dell'array di arrivo, lasciando quindi parti della nuova immagine non definite. Si procede quindi implementando un'analisi inversa a partire dalle coordinate di arrivo e verificando se ad esse corrispondono coordinate ammissibili nell'immagine di partenza. Dalla 1.1 esplicitando le variabili di arrivo ricaviamo:

$$\begin{cases} x = x' - x_t \\ y = y' - y_t \end{cases} \tag{1.2}$$

Fino a questo punto abbiamo supposto di traslare una immagine di un numero intero (x_t, y_t) di pixel, ma non sempre é cosí. In questo caso non ci sará nessun pixel corrispondente alle coordinate non intere nella vecchia immagine, dovremo quindi interpolare i valori dei quattro pixel che circondano il punto non intero nella vecchia immagine e assegnare il valore trovato al pixel corrispondente nella nuova immagine, come mostrato dalla seguente figura:

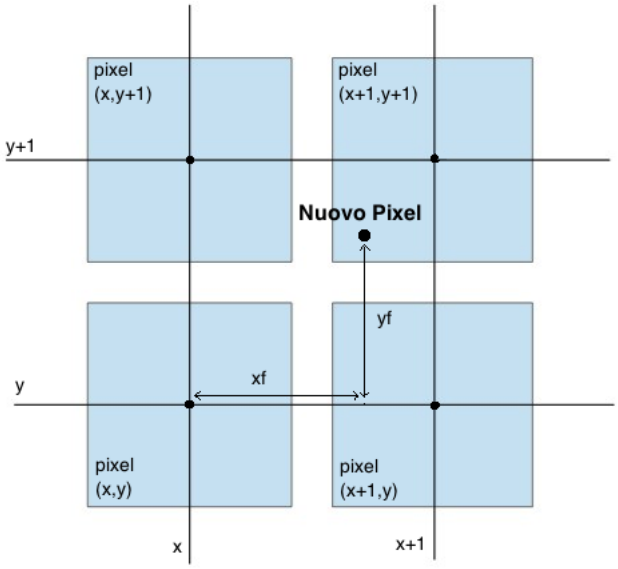


Figura 1.1: Interpolazione tra pixel.

L'algoritmo seguente esegue la traslazione tenendo conto di tutte le considerazioni fin quí fatte:

```
for yp: 0 to ymax
for xp: 0 to xmax
x = xp - x_t;
y = yp - y_t;
if int(x)<0 | int(x)+1>xmax | int(y)<0 | int(y)+1>ymax
new(xp,yp) = 0
elseif
xf = x - int(x);
yf = y - int(y);
a = old(int(x),int(y));
b = old(int(x),int(y)+1);
c = old(int(x)+1,int(y));
d = old(int(x)+1,int(y)+1);
new(xp,yp) = a*(1-xf)*(1-yf)+b(1-xf)*yf+c*xf*(1-yf)+d*xf*yf;
```

```
end
end
end
```

dove *xmax* e *ymax* sono le dimensioni (in pixel) dell'immagine, *xp* e *yp* le coordinate dei pixel di arrivo, *int* é la funzione che mi restituisce l'intero ottenuto troncando il numero *floating point* ed a, b, c, d i valori dei quattro pixel che circondano le coordinate non intere nella vecchia immagine.

Rotazione

Analogamente alla traslazione, quando si ha a che fare con l'elaborazione di immagini spesso risulta necessario ruotare una immagine per far coincidere l'orientazione con un'altra immagine al fine di eseguire altre operazioni. Dato un punto (x_0, y_0) ed un angolo θ, le equazioni che determinano una rotazione di angolo θ e centro (x_0, y_0) sono date da

$$\begin{cases} x' = x_0 + (x - x_0)cos\theta + (y - y_0)sin\theta \\ y' = y_0 - (x - x_0)sin\theta + (y - y_0)cos\theta \end{cases} \tag{1.3}$$

Notiamo che il punto (x_0, y_0) non si muove, occupando lo stesso posto sia nella vecchia che nella nuova immagine. Per quanto detto nel paragrafo precedente occorre peró partire dalle nuove coordinate ed effettuare un controllo sulle coordinate di partenza, ed in seguito interpolare il valore dei quattro pixel circostanti per ottenere il valore del pixel di arrivo. La trasformazione inversa, che si ottiene sostituendo $x \to x'$, $y \to y'$ e $\theta \to -\theta'$, é la seguente

$$\begin{cases} x = x_0 + (x' - x_0)cos\theta - (y' - y_0)sin\theta \\ y = y_0 + (x' - x_0)sin\theta + (y' - y_0)cos\theta \end{cases} \tag{1.4}$$

e l'algoritmo risultante é:

```
sinth = sin(theta);
costh = cos(theta);
for yp: 0 to ymax
```

```
for xp: 0 to xmax
x = x0 + (xp-x0)*costh - (yp-y0)*sinth;
y = y0 + (xp-x0)*sinth + (yp-y0)*costh;
if int(x)<0 | int(x)+1>xmax | int(y)<0 | int(y)+1>ymax
new(xp,yp) = 0
elseif
xf = x - int(x);
yf = y - int(y);
a = old(int(x),int(y));
b = old(int(x),int(y)+1);
c = old(int(x)+1,int(y));
d = old(int(x)+1,int(y)+1);
new(xp,yp) = a*(1-xf)*(1-yf)+b(1-xf)*yf+c*xf*(1-yf)+d*xf*yf;
end
end
end
```

Notiamo che é necessario calcolare le funzioni seno e coseno soltanto una volta, riducendo di molto il costo computazionale rispetto al calcolo di queste funzioni per ogni singolo pixel.

Scalamento

Scalare un'immagine significa cambiarne la dimensione. Per semplicitá assumiamo che l'immagine venga scalata dello stesso fattore in entrambi gli assi, cioé che si mantengano le proporzioni. Fissiamo un punto (x_0, y_0) dell'immagine, le equazioni per scalare l'immagine intorno al suddetto punto sono date da

$$\begin{cases} x' = x_0 + (x - x_0) * s \\ y' = y_0 + (y - y_0) * s \end{cases} \tag{1.5}$$

dove s rappresenta il fattore di scala. Come per la rotazione il punto (x_0, y_0) resta fisso. Possiamo quindi ricavare la trasformazione inversa che mi fornisce

il valore del punto per la nuova immagine

$$\begin{cases} x = x_0 + \frac{(x'-x_0)}{s} \\ y = y_0 + \frac{(y'-y_0)}{s} \end{cases} \tag{1.6}$$

L'algoritmo per scalare un'immagine é essenzialmente lo stesso della rotazione, tranne l'accorgimento di rimpiazzare le equazioni di rotazione con le equazioni seguenti

```
x = x0 + (xp - x0) / s;
y = y0 + (yp - y0) / s;
```

Quanto detto finora sullo scalare una immagine si esprime al meglio in termini di qualitá quando si cerca di ingrandire una immagine. Per esempio ingrandire una immagine di un fattore di scala ≥ 2 significa che ogni pixel dell'immagine originale deve essere campionato per determinare un nuovo pixel almeno 4 volte, cosicché tutte le informazioni contenute nell'immagine originale sono trasferite alla nuova.
Per fattori di ingrandimento ≤ 2 invece, i pixel dell'immagine originale sono campionati meno di 4 volte, inoltre per pixel adiacenti bisogna calcolare una media, rendendo quindi l'immagine leggermente piú sfocata.
Per fattori di scala vicini a 1, a causa del fatto che alcuni pixel sono calcolati come media di 4 pixel mentre altri prendono il loro valore quasi interamente da un singolo pixel, possono crearsi degli artifici sulla nuova imamgine. Per esempio in immagini giá affette da rumore (cioé come quasi tutte le immagini astronomiche ottenute dalla superficie della terra a casua dell'interferenza dell'atmosfera) questo si traduce nella creazione sulla nuova immagine di zone liscie e smussate (dovute al calcolo dei pixel come media di molti altri) alternate a zone in cui i pixel risentono ancora del rumore preesistente.
Con fattori di scala $\leq \frac{1}{2}$ il rimpicciolimento causa invece la perdita totale di alcuni pixel, ma poiché la nuova immagine é piú piccola dell'originale spesso essa risulta piú nitida anche se si é verificata una perdita di informazione.

Ribaltamento

Questa categoria di trasformazioni molto utilizzata nell'elaborazione di immagini consiste nello specchiare un immagine verticalmente o orizzontalmente lungo il proprio asse. Questo tipo di trasformazioni possiedono proprietá geometriche particolari, ovvero non possono essere ottenute singolarmente come composizione di traslazioni e rotazioni. Mentre ribaltare verticalmente e successivamente orizzontalmente una immagine equivale ad eseguire una rotazione di 180° rispetto al centro dell'immagine stessa. Le equazioni che descrivono per esempio il ribaltamento verticale (cioé con asse orizzontale) sono le seguenti

$$\begin{cases} x' = x \\ y' = -y \end{cases} \tag{1.7}$$

Sfortunatamente queste equazioni, che sono corrette dal punto di vista matematico, non si prestano appieno al nostro contesto in quanto, come si vede facilmente, piazzerebbero l'immagine specchiata al di fuori dei limiti dell'immagine originale, quindi l'immagine va successivamente traslata per riportarla entro suddetti limiti. L'equazione corretta é la seguente

$$\begin{cases} x' = x \\ y' = y_{max} - y \end{cases} \tag{1.8}$$

In modo del tutto analogo si ottengono le equazioni per il ribaltamento orizzontale (cioé specchiare l'immagine lungo un asse verticale) date da

$$\begin{cases} x' = x_{max} - x \\ y' = y \end{cases} \tag{1.9}$$

L'algoritmo per implementare tali trasformazioni é molto semplice, in quanto le coordinate restano sempre intere (non é quindi richiesta una interpolazione come nei casi precedenti) e inoltre la nuova immagine giace entro i bordi dell'immagine originale. Non é quindi richiesto un controllo preliminare sulle coordinate dei pixel. Mostriamo solo l'algoritmo per il ribaltamento orizzontale

```
for yp: 0 to ymax
for xp: 0 to xmax
x = xmax - x;
y = y;
new(xp,yp) = old(x,y);
end
end
```

Taglio e Aggiunta

Tagliare una immagine consiste nel copiare parte dell'immagine originale in una nuova immagine e questo procedimento puó essere utile per eliminare porzioni di immagine alle quali non siamo interessati. Aggiungere invece consiste nel copiare tutta un'immagine all'interno di una nuova piú grande, per poter poi eseguire per esempio una rotazione senza rischiare di perdere gli angoli dell'immagine durante il lavoro. In entrambi i casi comunque la nuova immagine ha dimensioni diverse da quella originale.
Per tagliare un'immagine bisogna prima determinare sugli assi x e y il minimo e il massimo valore che determinano l'area in cui copiare l'immagine tagliata. Siano questi valori rispettivamente $maxx$, $minx$, $maxy$, $miny$, e assumendo che tali punti appartengano all'immagine originale, l'algoritmo che esegue la trasformazione "tagliare" é il seguente

```
xmax = maxx - minx;
ymax = maxy - miny;
for y: 0 to ymax
for x: 0 to xmax
new(x,y) = old(x+ minx,y + miny);
end
end
```

La trasformazione di aggiunta invece richiede di definire prima quante colonne aggiungere a destra e sinistra e quante righe aggiungere sopra e sotto

all'immagine. Chiamiamo queste quantitá *left*, *right*, *up* e *down*, e siano *oldxmax* e *oldymax* le dimensioni dell'immagine originale. Allora l'algoritmo che esegue l'operazione di "aggiunta" é il seguente

```
xmax = left + oldxmax + right;
ymax = up + oldymax + down;
for y: 0 to ymax
for x: 0 to xmax
if x>=left & x<(xmax-right) & y>=down & y<(ymax-up)
new(x,y) = old(x+ minx,y + miny);
elseif
new(x,y) = 0;
end
end
end
```

1.1.1 Applicazione: archiviazione di immagini

Quando si studiano certi fenomeni/oggetti astronomici come per esempio comete, asteroidi, stelle variabili, supernove ed altro ancora, gli astronomi necessitano di confrontare tra loro immagini relative allo stesso fenomeno/oggetto ma catturate a diversi istanti di tempo. Purtroppo il sistema di puntamento dei telescopi non é mai *esattamente* lo stesso in due immagini distinte, questo a causa di innumerevoli fattori, e quindi le immagini risultano molto spesso traslate, ruotate e scalate l'una rispetto all'altra. Quindi per poter confrontare tra loro un insieme di immagini occorre prima trasformarle in un unico sistema di coordinate. Questa procedura é detta *archiviazione*, e fa largo uso delle traformazioni geometriche descritte in precedenza. Generalmente la procedura richiede di scegliere una immagine di riferimento detta *master* e fissare su di essa due punti di riferimento (generalmente si scelgono stelle molto luminose). Dopodiché si ricercano gli stessi oggetti scelti come riferimento sulle altre immagini (dette *slave*) e attraverso opportune trasformazioni applicate in sequenza (generalmente traslazioni, rotazioni e

scalamento) si cerca di sovrapporli a quelli presenti sull'immagine *master*.

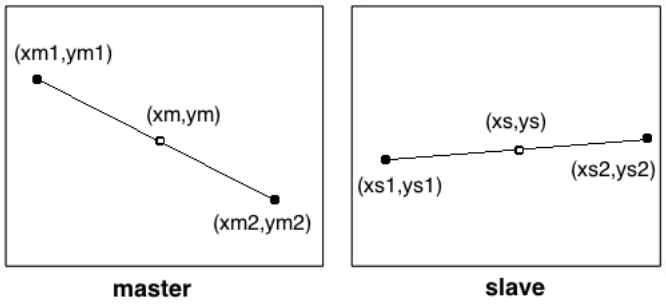


Figura 1.2: Immagine *master* (a sinistra) e *slave* (a destra).

Piú in dettaglio, con riferimento alla figura 1.2, l'archiviazione comincia misurando le coordinate di due punti di riferimento sull'immagine *master*, (x_{m1}, y_{m1}) e (x_{m2}, y_{m2}), e dei due corrispondenti punti di riferimenti sull' immagine *slave*, (x_{s1}, y_{s1}) e (x_{s2}, y_{s2}). Per una migliore accuratezza dell'operazione, i due punti dovrebbero essere scelti in due angoli diametralmente opposti dell'immagine il piú distanti possibile. Nonostante uno di questi punti possa essere scelto come centro per l'operazione di rotazione e di scalamento, generalmente si predilige scegliere come centro il punto medio del segmento individuato dai due punti, che risulta essere lo stesso in ognuna delle immagini esaminate. Le sue coordinate nell'immagine *master* sono date da

$$\begin{cases} x_m = \frac{x_{m1}+x_{m2}}{2} \\ y_m = \frac{y_{m1}+y_{m2}}{2} \end{cases} \tag{1.10}$$

mentre le sue coordinate nell'immagine *slave* sono date da

$$\begin{cases} x_s = \frac{x_{s1}+x_{s2}}{2} \\ y_s = \frac{y_{s1}+y_{s2}}{2} \end{cases} \tag{1.11}$$

tenendo quindi come riferimento i punti appena trovati, il vettore di traslazione sará $\overrightarrow{v} = (-\Delta x, -\Delta y)$ con

$$\begin{cases} \Delta x = x_m - x_s \\ \Delta y = y_m - y_s \end{cases} \tag{1.12}$$

a questo punto per trovare l'angolo di rotazione $\Delta\theta$ da applicare, teniamo a mente che, indipendentemente dall'orientazione, esso sará dato dalla differenza tra l'inclinazione del segmento scelto sull'immagine *master* e quella sull'immagine *slave*, ovvero sará dato da

$$\Delta\theta = \theta_m - \theta_s = \arctan(\frac{x_{m1} - x_{m2}}{y_{m1} - y_{m2}}) - \arctan(\frac{x_{s1} - x_{s2}}{y_{s1} - y_{s2}})$$

resta ora soltanto da determinare il fattore di scala dell'immagine *slave*. Come é logico aspettarsi, esso sará proporzionale al rapporto tra la distanza dei punti di riferimento scelti nell'immagine *master* e la distanza dei corrispondenti sull'immagine *slave*, ovvero sará dato da

$$s = \frac{d_m}{d_s}$$

con

$$d_m = \sqrt{(x_{m1} - x_{m2})^2 + (y_{m1} - y_{m2})^2}$$

e

$$d_s = \sqrt{(x_{s1} - x_{s2})^2 + (y_{s1} - y_{s2})^2}$$

Ricapitolando, per far combaciare l'immagine *slave* all'immagine *master*, si dovranno applicare alla prima, in sequenza, una traslazione di vettore $\overrightarrow{v} = (-\Delta x, -\Delta y)$, una rotazione di angolo $-\Delta\theta$ e uno scalamento di fattore $\frac{1}{s}$, con gli algoritmi spiegati precedentemente.

1.2 Operazioni matematiche

Nel paragrafo precedente abbiamo esaminato alcune operazioni che non modificano il contenuto di una immagine, ma soltanto la sua struttura. In questa sezione invece vedremo le operazioni piú semplici che vanno ad agire direttamente sull'informazione contenuta nell'immagine. Come il titolo suggerisce si tratta delle usuali operazioni matematiche che tutti conoscono, applicate al contesto dell'analisi di immagini. Ci limiteremo a trattare soltanto alcune di esse:

Addizione

Il risultato della somma tra due immagini é un'altra immagine in cui il valore di ogni pixel é dato dalla somma dei valori dei pixel corrisponenti nelle due immagini addende secondo la seguente legge

$$N(x, y) = O_1(x, y) + O_2(x, y)$$

Tenendo presente la terminologia introdotta nel paragrafo precedente, un possibile algoritmo che esegua la somma tra due immagini é dato dal seguente codice:

```
for y: 0 to ymax
for x: 0 to xmax
sum = old1(x,y) + old2(x,y);
if sum <= vmax
new(x,y) = sum;
elseif
new(x,y) = vmax;
end
end
end
```

dove *old*1 e *old*2 sono le due immagini originali, *new* é la nuova immagine somma delle due e *vmax* rappresenta il massimo valore di pixel rappresentabile con il numero di *bit* in cui stiamo lavorando. Questa operazione é molto semplice come implementazione, ma richiede un controllo iniziale sul range di valori dei pixel delle immagini originali, affinché durante il processo di addizione, la somma non superi il valore *vmax* causando una consistente perdita di informazione.

Sottrazione

La sottrazione tra due immagini puó essere descritta dalla relazione

$$N(x, y) = O_1(x, y) - O_2(x, y)$$

e un possibile algoritmo é il seguente:

```
for y: 0 to ymax
for x: 0 to xmax
diff = old1(x,y) - old2(x,y);
if diff <= 0
new(x,y) = 0;
elseif
new(x,y) = diff;
end
end
end
```

dove analogamente all'algoritmo precedente, in caso di risultato negativo della sottrazione, si pone il valore del nuovo pixel pari a 0, poiché valori negativi di luce non sono contemplati.

Moltiplicazione

Questa operazione é tra le piú versatili in ambito di elaborazione di immagini, puó essere per esempio usata per evidenziare i *range* di luminositá di una immagine oppure, come vedremo in seguito, usata per creare delle maschere che modifichino la trasformata di Fourier di una immagine. Precisiamo che con moltiplicazione intendiamo una operazione punto per punto e non una moltiplicazione tra immagini pensate come matrici. Poiché durante una moltiplicazione quasi tutti i valori dei pixel risulteranno maggiori del valore limite ammissibile, prima di eseguire tale operazione sará quindi necessario operare una normalizzazione dei valori dei pixel fino a portarli a valori compresi tra 0 e 1, per poi riscalare il risultato ottenuto dopo aver

eseguito la moltiplicazione. A seguito l'algoritmo (a titolo dimostrativo, ma che risulta poco efficente dal punto di vista computazionale) che esegue tale operazione:

```
for y: 0 to ymax
for x: 0 to xmax
old1n = old1(x,y) / vmax;
old2n = old2(x,y) / vmax;
    new(x,y) = old1n * old2n * vmax;
end
end
```

Fusione

La fusione tra due immagini é una operazione simile all'addizione ma molto piú versatile, permette infatti di sommare tra loro due immagini moltiplicate per degli opportuni coefficienti a scelta dell'utente (non é nient'altro che una combinazione lineare tra le immagini), in modo da enfatizzare certe caratteristiche di una o dell'altra:

$$N(x, y) = \alpha_1 * O_1(x, y) + \alpha_2 * O_2(x, y)$$

e l'algoritmo che mi calcola la fusione tra due immagini é il seguente:

```
for y: 0 to ymax
for x: 0 to xmax
sum = a1 * old1(x,y) + a2 * old2(x,y);
switch sum
case sum < 0
new(x,y) = 0;
case sum > vamx
new(x,y) = vmax;
otherwise
new(x,y) = sum;
```

```
end
end
end
```

1.2.1 Applicazione: algoritmo di Larson-Sekanina

In questo paragrafo esamineremo uno degli strumenti piú efficaci per lo studio degli oggetti astronomici che piú hanno affascinato l'umanitá nel corso della sua storia: le comete. Questi corpi celesti sono costituiti da nuclei composti da un misto di ghiaccio e polveri e risiedono in una zona di spazio molto distante dal Sole chiamata *Nube di Oort*. Occasionalmente la gravitá solare "strappa" questi corpi dalla loro zona di quiete e ne causa l'addentrarsi nel sistema solare interno. Quando una cometa passa a 0.1 U.A. dalla Terra si ha l'opportunitá di osservare in maniera molto dettagliata dei particolari interni alla chioma, chioma che come sappiamo é il prodotto diretto dell'emissione di polveri e gas dal nucleo della cometa a causa dell'azione del vento solare. Tuttavia le fotografie, anche ad alta risoluzione, ben difficilmente possono evidenziare questi particolari poiché i lunghi tempi d'esposizione finiscono per sovraesporre la zona del nucleo e della chioma; d'altro canto le osservazioni visuali sono molto piú dettagliate grazie alle capacitá dell'occhio umano, ma hanno il grosso difetto di essere troppo soggettive e dipendenti dalle condizioni ambientali e strumentali. Solo di recente con l'utilizzo di alcune tecniche di elaborazione digitale delle immagini astronomiche queste difficoltá sono state in parte superate. Ma perché siamo cosí interessati a questi particolari delle chiome? Poiché le strutture della chioma contengono informazioni sul vettore di *spin* (rotazione) del nucleo, con importanti conseguenze per lo studio della dinamica di questi oggetti celesti.

L'algoritmo in questione fú descritto per la prima volta in un articolo dell'*Astronomical Journal* del 1984 degli astronomi Steven M. Larson del *Lunar and Planetary Laboratory* in Arizona e Zdenek Sekanina del *Jet Propulsion Laboratory* in California. Nell'articolo in questione i due ricercatori hanno utilizzato questa

tecnica di image-processing su delle lastre ad alta risoluzione della chioma della cometa di Halley riprese nel maggio-giugno del 1910 e poi digitalizzate. Giá a quei tempi si utilizzavano varie tecniche, come le derivate direzionali dell'intensitá luminosa, per evidenziare i particolari meno contrastati della chioma; queste peró erano limitate dal fatto che risaltavano le caratteristiche presenti solo nelle direzioni considerate dalle derivate. Larson e Sekanina invece misero a punto un algoritmo che permetteva l'applicazione delle derivate direzionali dell'intensitá luminosa di un'immagine in qualsiasi direzione operando una semplice trasformazione di coordinate.

Come sappiamo un'immagine digitale si puó rappresentare con una funzione bidimensionale $I(x, y)$ dove ad ogni coordinata discreta (x, y) rappresentata da un pixel viene associato un valore di intensitá dato dal valore della conversione analogico-digitale dell'immagine, coordinate, come abbiamo detto, riferite ad un sistema di coordinate cartesiane con l'origine rappresentata da uno dei quattro pixel posti nei quattro angoli dell'immagine. Se ci portiamo in un sistema di coordinate polari possiamo scrivere la nostra immagine in modo del tutto analogo come $B(r, \theta)$, dove r é la distanza del punto dall'origine e θ é l'angolo che individua la direzione del punto rispetto all'asse x di riferimento. L'origine di questo nuovo sistema di coordinate non sará piú il pixel di coordinate $(0, 0)$ bensí un pixel generico di nostra scelta che indicheremo con (x_0, y_0). Ricordando la trigonometria delle scuole superiori, é facile ricavare le formule di trasformazione dal sistema cartesiano a quello polare

$$\begin{cases} x = r * cos(\theta) \\ y = r * sen(\theta) \end{cases} \longrightarrow \begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \arctan(\frac{y}{x}) \end{cases} \tag{1.13}$$

Il sistema di coordinate polari é piú conveniente nel caso delle rappresentazioni di oggetti che presentano una simmetria polare come appunto le comete. Se infatti individuiamo il pixel (x_0, y_0) piú luminoso nel centro della chioma e presupponiamo che esso coincida con il falso nucleo della cometa, l'immagine digitalizzata la possiamo rappresentare con il nuovo sistema di coordinate

$B(r, \theta)$ con origine nel falso nucleo (x_0, y_0). Facendo riferimento a questa rappresentazione, l'algoritmo di Larson-Sekanina si puó scrivere come:

$$B'(r, \theta, \Delta r, \Delta\theta) = 2B(r, \theta) - B(r - \Delta r, \theta + \Delta\theta) - B(r - \Delta r, \theta - \Delta\theta)$$

In sostanza dall'immagine originale, opportunamente duplicata nelle intensitá, vengono sottratte due immagini modificate geometricamente: alla prima, rappresentata dal secondo termine del secondo membro, viene effettuato uno spostamento radiale $-\Delta r$ ed uno rotazionale $\Delta\theta$ relativamente al centro del sistema di coordinate polari (rappresentato come dicevamo prima dal pixel piú luminoso della chioma della cometa ovvero dal falso nucleo); alla seconda, rappresentata dal terzo termine del secondo membro, viene effettuato uno spostamento radiale della stessa grandezza e direzione ed uno spostamento rotazionale in direzione opposta. L'immagine risultante, poi riconvertita in coordinate cartesiane, sará una mappa delle variazioni delle intensitá luminose avente le zone con un gradiente positivo di luminositá scure mentre quelle con un gradiente negativo saranno visualizzate piú chiare.

Per capirne meglio il funzionamento consideriamo per esempio la figura 1.3. Essa rappresenta (primo diagramma) una linea di pixel orizzontale della nostra immagine da trattare nella quale sono presenti due picchi di luminositá: questi picchi possono essere anche molto deboli e non distinguibili con le normali visualizzazioni poiché il contrasto é molto debole ed impercettibile dall'occhio umano. Traslando di un pixel verso sinistra tutta la linea (secondo diagramma) e sottraendo pixel per pixel i valori delle intensitá dal grafico non traslato, otteniamo (terzo diagramma) una mappa di chiaroscuri altamente contrastata ed indicativa della presenza di queste deboli variazioni di intensitá (in sostanza é un'applicazione del concetto di derivata come variazione). É chiaro quindi che l'immagine risultante perderá tutte le eventuali informazioni fotometriche presenti nell'originale ma indicherá con precisione le zone che presentano delle variazioni di luminositá anche deboli o nascoste dallo scarso contrasto generalmente presente nell'intorno piú luminoso della chioma.

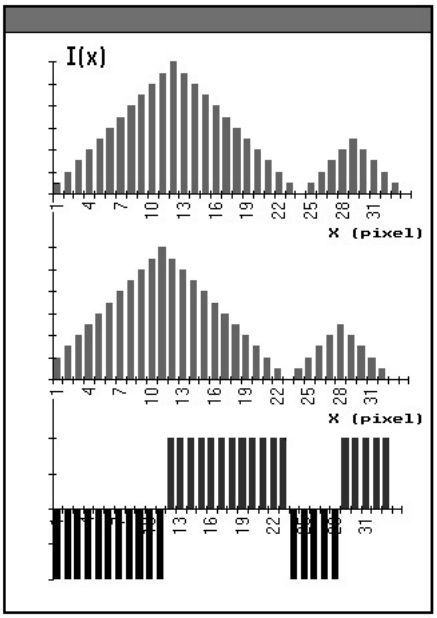
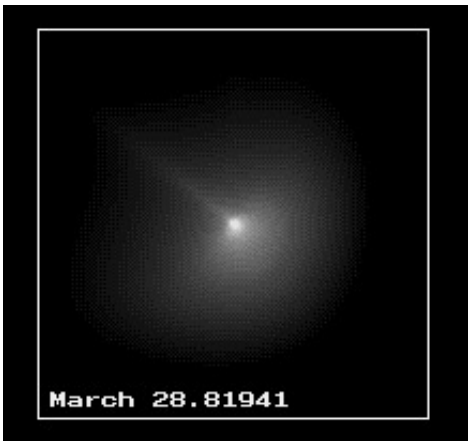


Figura 1.3: Diagrammi di luminositá.

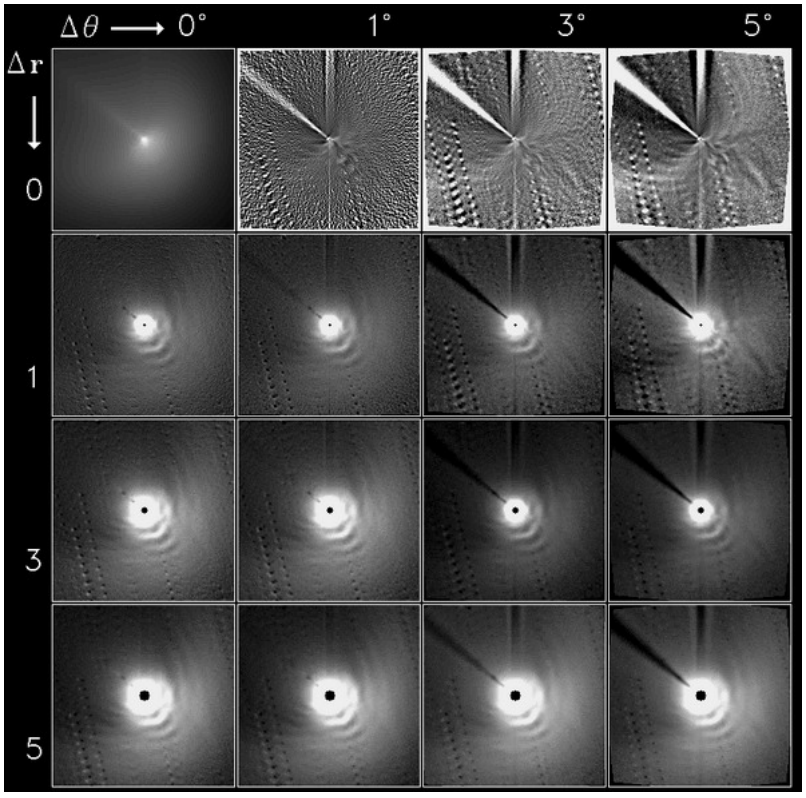
Consideriamo ora come caso di studio la seguente immagine della cometa *Hyakutake* ripresa il 28 marzo 1996 dall'osservatorio astronomico di Cavezzo:



Teniamo presente che la grandezza degli spostamenti Δr e $\Delta\theta$ non é data a priori, ma viene stabilita empiricamente con delle prove e dipende in gran parte dall'esperienza dell'analista e dal confronto con l'immagine originale

che deve sempre rappresentare un punto di riferimento per l'osservatore; non é difficile infatti incorrere in artefatti che non corrispondono ad alcuna caratteristica morfologica della chioma in esame: in genere questi artefatti sono ben distinguibili in quanto presentano una certa simmetria rispetto all'entitá degli spostamenti effettuati. Tale entitá dipende comunque anche dalla scala dell'immagine, ovvero dalla grandezza dei particolari della chioma che si cerca di evidenziare.

Una indicazione di massima la si puó ottenere compilando una tabella come quella mostrata di seguito:



Questa tabella mostra gli effetti dell'algoritmo di Larson-Sekanina al variare dei parametri Δr e $\Delta\theta$ sull'immagine considerata. Le righe rappresentano uno spostamento radiale costante Δr contrapposto ad uno spostamento rotazionale variabile $\Delta\theta$. Analogamente le colonne rappresentano uno spostamento rotazionale $\Delta\theta$ costante rispetto a uno spostamento radiale Δr

variabile. L'immagine in alto a sinistra con $\Delta r = 0$ e $\Delta\theta = 0$ rappresenta ovviamente la nostra immagine originale non trattata.

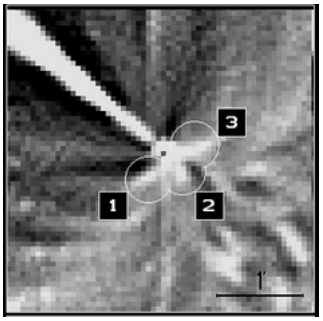
Possiamo a questo punto fare le seguenti considerazioni:

- $\Delta r = 0$
  In questo caso l'equazione di Larson-Sekanina si riduce a

$$B'(r, \theta, \Delta r, \Delta\theta) = 2B(r, \theta) - B(r, \theta + \Delta\theta) - B(r, \theta - \Delta\theta)$$

  Poiché si tratta di eseguire una rotazione con centro il pixel rappresentante il falso nucleo della cometa, questo procedimento puó essere eseguito anche senza la trasformazione in coordinate polari.
  Per spostamenti radiali Δr nulli (prima riga in alto), modificando il valore di $\Delta\theta$, si aumenta il contrasto di tutti i particolari che hanno un gradiente angolare di luminositá rispetto all'origine del nostro sistema polare di coordinate (falso nucleo); i particolari che si evidenziano con questo tipo di spostamento sono in genere i jet e le fontane (getti di materia proveniente dal nucleo cometario che costituiscono elementi della chioma): é evidente per esempio il jet principale di plasma che attraversa tutto il quadrante in alto a sinistra dell'immagine a circa 45° (figura sottostante), e anche i piccoli jet di gas e polveri (evidenziati sempre nella figura sottostante) che si sviluppano in direzione del Sole e quindi nella parte attiva del nucleo. Precisiamo inoltre la presenza di un falso jet verticale dovuto a un difetto di ripresa dell'immagine al momento dell'acquisizione, che é stato enfatizzato dall'uso dell'algoritmo di Larson-Sekanina, infatti tale algoritmo aumenta notevolmente il contrasto tra le disuniformitá di un'immagine a scapito della dinamica dell'immagine stessa; inoltre, operando con un sistema di coordinate polari, bisogna sempre sospettare fortemente dei particolari che presentano una certa simmetria o gradienti molto ripidi di luminositá. Nel nostro caso un buon valore di variazione rotazionale puó essere $\Delta\theta = 3°$.
  La figura sottostante rappresenta un ingrandimento del nucleo dell'im-

magine ottenuta con tali parametri, e si evidenziano i jet di materia analizzati sopra:



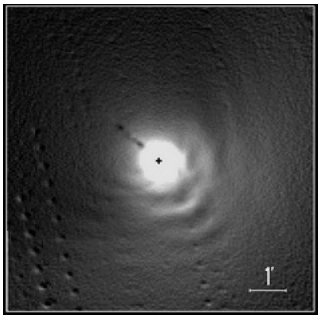
- $\Delta\theta = 0$

  In questo caso l'equazione di Larson-Sekanina si riduce a

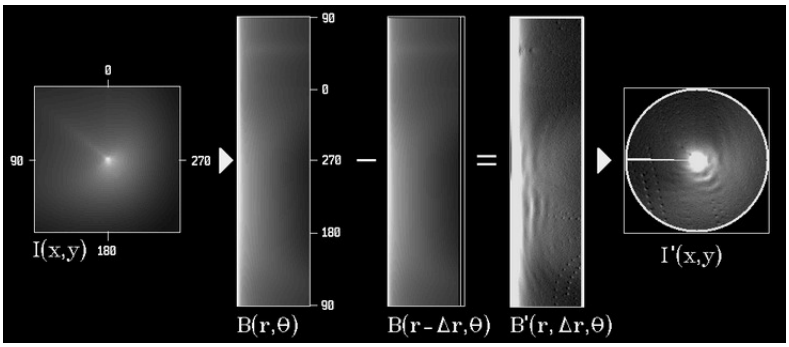
  $$B'(r, \theta, \Delta r, \Delta\theta) = 2[B(r, \theta) - B(r - \Delta r, \theta)]$$

  e l'algoritmo consiste in una differenza tra una immagine e la sua traslata, il tutto ovviamente in coordinate polari.
  Per spostamenti rotazionali $\Delta\theta$ nulli (prima colonna a sinistra), modificando il valore di Δr, si aumenta il contrasto di tutti i particolari che hanno un gradiente radiale di luminositá rispetto al falso nucleo. Essendo nulli gli spostamenti rotazionali, tutti i jet che si protendono radialmente dal nucleo verso l'esterno (particolari che, ripetiamo, hanno un gradiente angolare di luminositá rispetto all'origine del nostro sistema polare di coordinate) non sono piú visibili. Questo tipo di elaborazione permetterá di mettere in evidenza aloni, strutture a spirale e gusci di polvere e gas che compongono gli strati piú interni della chioma. Lo studio di questi particolari é importante per determinare la forma ed il periodo di rotazione del nucleo ed é quindi fondamentale per la conoscenza della dinamica dei nuclei cometari (metodo messo a punto da F.L. Whipple).
  Anche in questo caso la scala dei fenomeni che sono oggetto di studio é fondamentale per determinare la giusta grandezza dello spostamento

Δr da operare sull'immagine originale: come si vede per esempio analizzando la solita tavola predecente, per Δr crescente diventano sempre meno distinguibili i gusci a spirale che avvolgono il nucleo della cometa e si ingrandisce sempre di piú la zona centrale "bruciata" da questo tipo di elaborazione. Uno spostamento $\Delta r = 1$ pixel (mostrato nella figura seguente) é sufficiente per evidenziare una notevole quantitá di strutture nella parte piú interna della chioma della *Hyakutake*, come per esempio il particolare in corrispondenza dei due picchi di luminositá nel jet principale di ioni giá visto precedentemente e causato appunto da queste due ripide variazioni del gradiente di luminositá in direzione del jet.



Concludiamo con una figura che rappresenta "visivamente" l'applicazione dell'algoritmo di Larson-Sakenina in quest'ultimo caso particolare considerato, utile per comprenderne meglio il funzionamento. Si noti che, avendo considerato nella trasformazione polare un raggio pari al semilato del quadrato dell'immagine originale, durante l'antitrasformazione per tornare in coordinate cartesiane non sono piú riproducibili gli spigoli dell'immagine.


0
90
270
180
I(x,y)
90
0
270
180
90
B(r,θ)
B(r-Δr,θ)
B'(r,Δr,θ)
I'(x,y)

Capitolo 2

Operazioni Puntuali

Le immagini astronomiche molto spesso contengono un range di luminositá molto elevato (ovvero in range di valori dei pixel molto ampio), ma le caratteristiche a cui siamo realmente interessati molto spesso risiedono dentro un range di luminositá piú limitato. Altre volte invece accade che il range sia troppo elevato per poter essere visualizzato correttamente sul monitor di un computer. In queste situazioni e tante altre vengono in nostro aiuto le operazioni puntuali. Esse si chiamano cosí perché intervengono direttamente a modificare il valore di ogni singolo pixel in modo da eliminare determinate caratteristiche dell'immagine oppure enfatizzarne altre.
In questo capitolo impareremo dove trovare informazioni utili nascoste in un range troppo alto di valori di pixel e alcune delle tecniche base che permettono di alterare la scala di luminositá di un'immagine per mostrare l'informazione in essa contenuta piú chiaramente. I passi da seguire sono quindi:

- identificare il range di valori di pixel interessante a seconda del contesto
- alterare il valore dei pixel per un *output* ottimale

Il primo punto puó essere svolto lavorando direttamente sull'immagine impostando a propria scelta i valori di pixel da associare al nero e al bianco (ovvero i colori estremi) e lavorare sui pixel dai valori intermedi a questi (metodi *direct*), oppure creando un istogramma che contenga i numeri di

pixel corrispondenti ai diversi valori e scegliere su quale percentuale di pixel operare (metodi *histogram*).
L'alterazione dei valori di pixel avviene invece generalmente ad opera di un calcolatore usando generalmente diversi tipi di funzioni matematiche dette *transfer function* a seconda del risultato che si vuole ottenere. Esamineremo piú nel dettaglio queste funzioni in uno dei paragrafi seguenti. I quattro tipi base di operazioni puntuali si distinguono tra loro principalmente per il metodo usato per identificare il range di valori di pixel di interesse. Essi sono:

- **direct function specification**
  Con questo metodo si decide direttamente quale trasformazione matematica eseguire sui pixel (per esempio: $newvalue = \sqrt{oldvalue - 100}$) senza esplicitare su quale range applicare tale funzione.

- **direct endpoint specification**
  Con questo metodo si ispeziona l'immagine e si decide il range di interesse scegliendo a quale valore associare il nero e il bianco (per esempio: imposta il nero al valore 26 e il bianco al valore 2100), per poi scegliere quale funzione applicare a tutti i pixel con un valore intermedio rispetto agli estremi (*endpoints*) appena fissati.

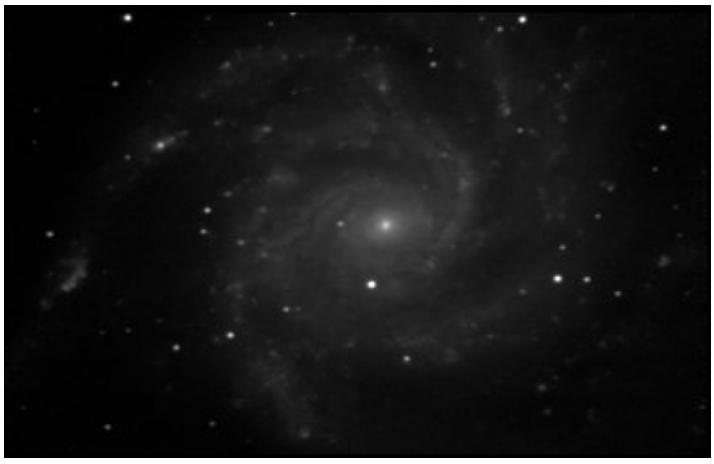
- **histogram endpoint specification**
  Con questo metodo l'utente stabilisce solamente le percentuali di pixel da saturare a nero e a bianco (per esempio satura a nero lo 0.1% dei pixel e satura a bianco il2.5%), e il calcolatore provvederá in maniera automatica a trovare i valori (*endpoints*) corrispondenti a questi colori, per poi applicare la funzione scelta ai pixel che si trovano in quell'intervallo.

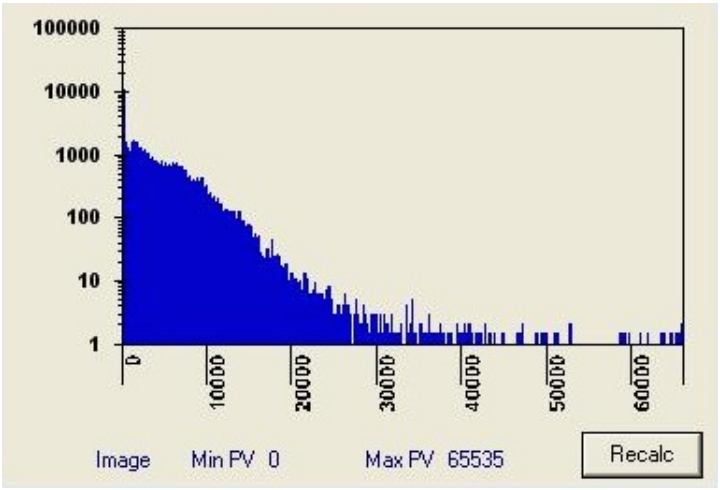
- **histogram specification** Con questo ultimo metodo, chiamato anche *histogram shaping*, l'utente decide invece quale forma (esponenziale, gaussiana, etc...) vuole che assuma l'istogramma dell'immagine finale

modificata, ed il calcolatore provvederá in automatico a calcolare l'istogramma dell'immagine originale e a trovare la funzione adatta con cui trasformare il valore dei pixel in modo da ottenere una immagine con un istogramma con le caratteristiche richieste.

2.1 Isolare il range

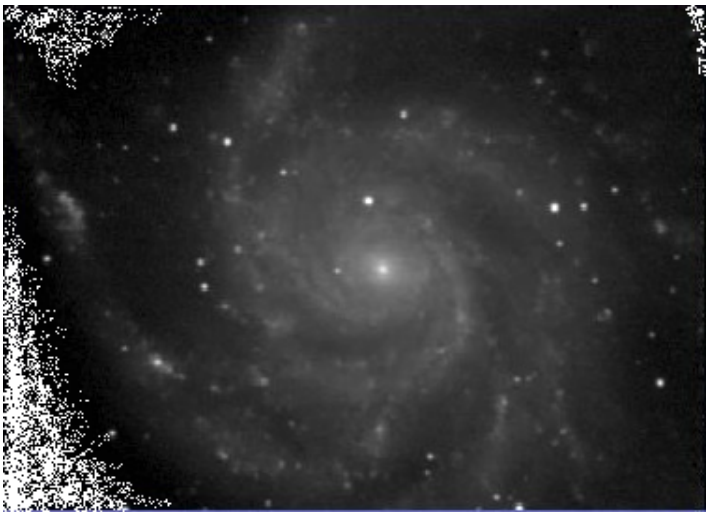
Sappiamo da quanto detto che il valore di ogni pixel nell'immagine é determinato proporzionalmente alla quantitá di luce che ha colpito la CCD, e che prima di agire con determinate funzioni sull'immagine é opportuno eseguire una scelta su quali pixel intervenire. Tale scelta puó essere effettuata impostando valori direttamente da utente (metodi *direct*) oppure effettuando una analisi preventiva sull'istogramma corrispondente all'immagine (metodi *histogram*). L'istogramma non é nient'altro che una rappresentazione su di un piano cartesiano in cui sull'asse delle x si trovano i valori dei pixel da 0 al valore massimo (che varia a seconda dei bit utilizzati, 255 per dati a 8 bit, 65535 per dati a 16 bit e cosi via...), mentre sull'asse delle y troviamo il numero di pixel corrispondenti a un determinato valore. Generalmente sull'asse delle ordinate si usa una scala logaritmica, in quanto in molte delle immagini astronomiche il numero di pixel corrispondenti a un determinato valore puó essere molto maggiore rispetto ad altri valori. Si pensi per esempio alle immagini di campi stellari in cui la quantitá di pixel di valore basso (corrispondenti alle porzioni di cielo prive di stelle) é enormemente maggiore rispetto alla quantitá di pixel con valori piú elevati (corrispondenti alle stelle ed altri oggetti celesti luminosi).
Vediamo con il seguente esempio come una scelta opportuna dei valori estremi é importante per un risultato ottimale.
Consideriamo la seguenti immagini:



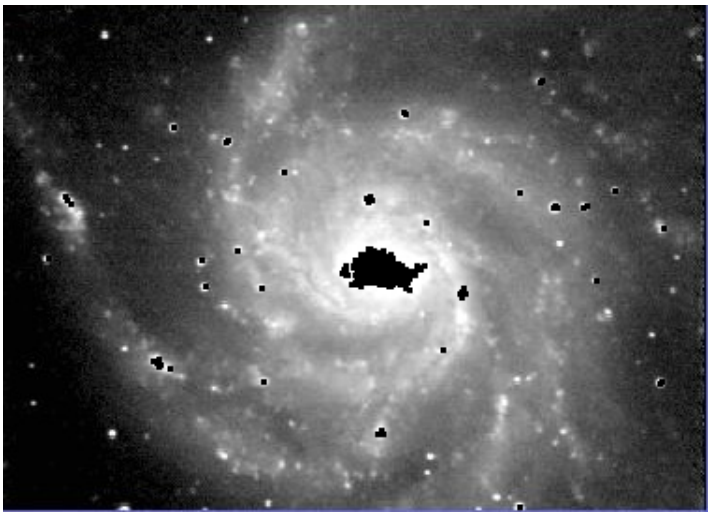


La prima immagine rappresenta una istantanea della galassia M101, mentre la seconda il relativo istogramma. Osservando la figura 2.1 si nota immediatamente che la maggioranza dei pixel dell'immagine 2.1 assumono valori bassi. É questo il motivo per cui, in una scala di valori di pixel da 0 a 4095, l'immagine appare cosí scura. Infatti circa il 10% dei pixel assumono un valore inferiore a 100, il 99% hanno un valore inferiore a 1108 mentre solo lo 0.1% (ovvero 1 pixel su 1000) presenta un valore superiore a 2100. Quasi tutti i pixel di valore basso provengono dal cielo di sfondo e dai tenui bracci a spirale della galassia, mentre i pixel di valore piú alto dalle stelle e dal centro della galassia stessa. Supponiamo ora di voler saturare in nero (*endpoint* inferiore) il 10% dei pixel dell'immagine (ovvero quel 10% di pixel

che avevano un valore inferiore a 100), quello che si ottiene é mostrato nella seguente figura, dove tutti i pixel totalmente neri sono stati colorati di bianco per rendere maggiormente visibile la zona saturata.



Si capisce subito che tale operazione ha il grosso vantaggio di rendere piu evidenti oggetti di maggiori dimensione, ma puó causare la scomparsa di oggetti piú piccoli.
La saturazione del bianco (*endpoint* superiore) invece é un passaggio piú complicato e su cui prestare piú accorgimenti, poiché sono proprio gli oggetti luminosi quelli a cui siamo interessati quando si tratta di immagini stellari. Abbiamo visto che nell'immagine 2.1 soltanto l'1% dei pixel hanno un valore superiore a 1108, e questi rappresentano il nucleo galattico e le stelle piú luminose dell'immagine. Quindi se impostiamo una saturazione del bianco al 99%, ovvero saturiamo a bianco tutto il restante 1%, rischiamo di perdere moltissime informazioni sull'immagine, per esempio forma e dimensioni del nucleo galattico o luminositá stellare. Tale risultato é mostrato nella figura seguente, dove sono state colorate di nero le zone saturate a bianco per enfatizzarne la vista



Una buona rimappatura dell'immagine tale da renderla piú visibile senza perdere troppe informazione si puó ottenere impostando l'*endpoint* del nero a 1% e quello del bianco al 99.95%, e utilizzando poi una *transfer function* gamma-logaritmica (descritta nella sezione seguente). Aggiungiamo che questi valori di impostazione sono molto buoni per la maggior parte delle immagini dello spazio profondo, in quanto esse presentano quasi sempre un istogramma con andamento di tipo esponenziale (ricordiamo che la scala delle ordinate é logaritmica), ovvero con la stragrande maggioranza dei pixel di valore basso. L'immagine ottenuta é mostrata nella figura che segue

2.2 Transfer functions

Una volta selezionato il range di valori sul quale intervenire, lo strumento matematico al quale ci si affida per alterare il valore dei vari pixel sono le *transfer functions*. Esse non sono altro che funzioni matematiche

$$f(p) = q$$

che prendono in ingresso il valore di un pixel e restituiscono in output il nuovo valore. Nei programmi di elaborazione di immagini tali funzioni sono generalmente pre-calcolate, e i loro valori $(f(p_1), f(p_2), \ldots, f(p_{max}))$ memorizzati in tabelle di ricerca per aumentare la velocitá di calcolo. Se per esempio vogliamo modificare la luminositá di una immagine di dimensione 1024×1024 pixel, si dovrebbe richiamare la funzione per ben 1048576 volte. Ma con una tabella di ricerca con i valori della funzione pre-calcolati sui 65536 valori possibili di un pixel, basta richiamare la tabella 1048576 volte e leggere il valore corrispondente, con un calo del costo computazionale. Infatti il costo computazionale di 1048576 confronti sulla tabella é assia inferiore al calcolo della funzione stessa per 1048576 volte.
Indichiamo con p_n e p_b rispettivamente l'*endpoint* del nero e del bianco, e con p_{max} il massimo valore possibile di un pixel. Alcune delle principali *transfer function* sono le seguenti:

- **transfer function lineare**
  Con questo tipo di funzione, i valori dei pixel sono divisi uniformemente tra i due valori di *endpoint* bianco e nero. La sua equazione é data da

$$f(p) = \begin{cases} 0, & p \leq p_n \\ \frac{p-p_n}{p_b-p_n} p_{max}, & p_n < p < p_b \\ p_{max}, & p \geq p_b \end{cases} \tag{2.1}$$

- **transfer function gamma**
  La *transfer function gamma* ricopre un ruolo importante tra le operazioni puntuali, in quanto permette di modificare i mezzi toni di una

immagine e renderli piú/meno luminosi senza andare a intaccare in maniera significativa i toni chiari o scuri. Tenendo presente che é possibile variare la costante γ a seconda della situazione, l'equazione di tale funzione é data da

$$f(p) = \begin{cases} 0, & p \leq p_n \\ (\frac{p-p_n}{p_b-p_n})^{\frac{1}{\gamma}} p_{max}, & p_n < p < p_b \\ p_{max}, & p \geq p_b \end{cases} \tag{2.2}$$

- **transfer function logaritmica**

  La *transfer function logaritmica* si presta in modo ottimale quando si lavora con la luce stellare. La scala di magnitudine stellare (ovvero la scalla di luminositá) é infatti una scala logaritmica, in quanto é logaritmica la scala con cui l'occhio umano percepisce la luce proveniente dalle stelle. L'equazione di tale funzione é data da

$$f(p) = \begin{cases} 0, & p \leq p_n \\ \frac{log(p-p_n)}{log(p_b-p_n)} p_{max}, & p_n < p < p_b \\ p_{max}, & p \geq p_b \end{cases} \tag{2.3}$$

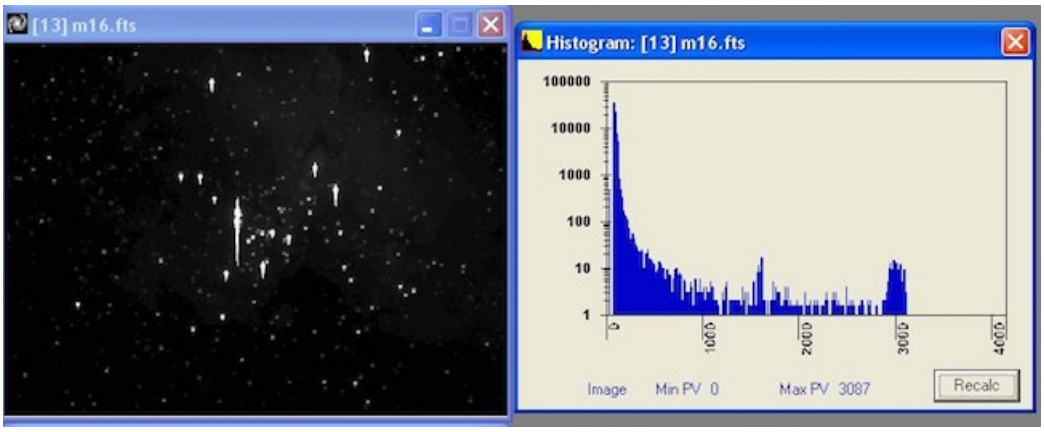
- **transfer function negativa**

  Si tratta di un tipo particolare di *transfer function* lineare che trasforma l'immagine nel suo negativo fotografico. La grande utilitá sta nel fatto che su un negativo risultano molto piú visibili dettagli deboli dell'immagine originale. A differenza delle funzioni precedente, in questo caso i ruoli degli *endpoints* sono invertiti. La sua equazione é data da

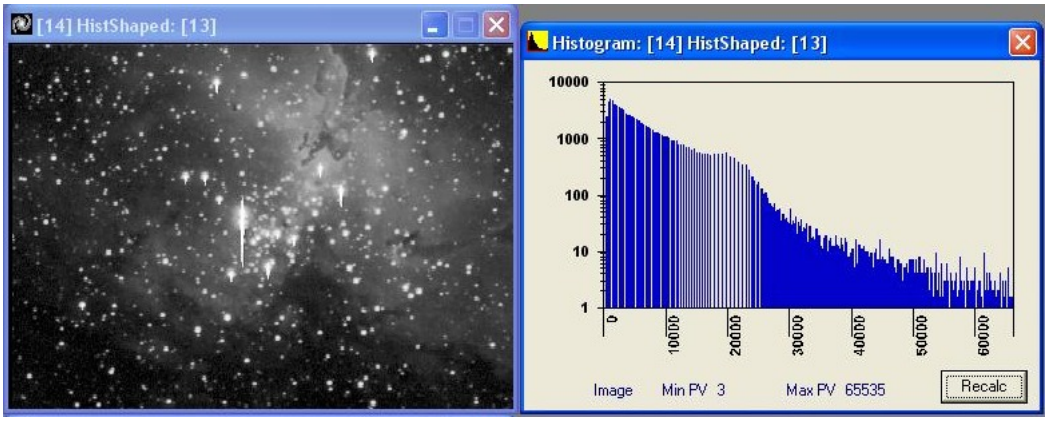
$$f(p) = \begin{cases} p_{max}, & p \leq p_n \\ p_{max} - p, & p_n < p < p_b \\ 0, & p \geq p_b \end{cases} \tag{2.4}$$

2.3 Esempi

Con il programma AIP4WIN carichiamo una immagine e costruiamo il suo istogramma.



Come si vede, l'immagine é molto scura e non si riesce a distinguere l'oggetto osservato. Operando quindi sull'istogramma dell'immagine (stiamo quindi eseguendo una *histogram specification*) andiamo a modificarne la forma tramite una funzione a nostra scelta. In questo caso abbiamo scelto di dare all'istogramma la forma di una gaussiana. Il risultato ottenuto é mostrato nella seguente:



A destra il nuovo istogramma modificato e a sinistra l'immagine alterata corrispondente. Si vede chiaramente come l'oggetto osservato, la nota *Nebulosa dell'Aquila*, da impercettibile che era, risulta ora ben visibile.

Capitolo 3

Convoluzione

Abbiamo giá visto nel capitolo sulle trasformazioni geometriche come a volte si debba trovare il nuovo valore di un pixel interpolando il pixel originale con quelli adiacenti. In questa sezione esamineremo tecniche di elaborazione di immagini che sfruttano proprio questo principio, concentrandoci particolarmente sulla convoluzione.

Date due funzioni continue $f,\ g : \mathbb{R} \longrightarrow \mathbb{R}$, si definisce la *convoluzione* tra queste due funzioni come

$$(f * g)(x) = \int_{-\infty}^{+\infty} f(u)g(x-u)du \tag{3.1}$$

In modo del tutto analogo al caso unidimensionale, possiamo definire la convoluzione tra due funzioni $f,\ g : \mathbb{R}^2 \longrightarrow \mathbb{R}$ come

$$(f * g)(x,y) = \int_{-\infty}^{+\infty} f(u,v)g(x-u,y-v)dudv \tag{3.2}$$

Osserviamo che la convoluzione é un operatore simmetrico, ovvero

$$f * g = g * f$$

basta infatti operare in 3 la sostituzione $z = x - y$. Quanto appena detto per il caso di funzioni continue si puó facilmente generalizzare al caso di nostro interesse di funzioni discrete, sappiamo infatti che possiamo considerare le

immagini come funzioni a due variabili a valori discreti. In questo caso le equazioni 3 e 3 diventano rispettivamente

$$(f * g)(n) = \sum_{k \in K} f(k)g(n-k) \tag{3.3}$$

e

$$(f * g)(n, m) = \sum_{(k,h) \in K'} f(k, h)g(n-k, m-h) \tag{3.4}$$

dove $K = \{k \mid f(k) \neq 0\}$ e $K' = \{(k, h) \mid f(k, h) \neq 0\}$. Nella nostra trattazione la funzione g é la nostra immagine, mentre la funzione f sará una matrice detta *kernel* (nucleo) di convoluzione.

Per cercare di illustrare in che modo agisce la convoluzione su di una immagine partiamo dal caso piú semplice, ovvero supponiamo che la nostra immagine sia costituita da un *array* unidimensionale estratto da una immagine piú grande:

$$[\ldots 10\ 10\ 10\ 10\ 10\ 10\ 10\ 90\ 90\ 90\ 90\ 90\ 90\ 90 \ldots]$$

possiamo pensare questi valori come un improvviso cambio di luminositá nell'immagine, si passa infatti da una serie di pixel con valore 10 a una con valore 90. E consideriamo poi il nostro *kernel* come un *array* unidimensionale di questo tipo:

$$[0.25\ \mathbf{0.50}\ 0.25]$$

possiamo pensare che questo array ci dica che il valore del nuovo pixel di riferimento (indicato in grassetto) venga calcolato a partire da 3 pixel, con un contributo del 50% da parte del pixel di riferimento originale e del 25% da parte dei due pixel adiacenti. In termini matematici la convoluzione é data da:

$$N_x = \frac{\sum_{i=-r}^{r} k_i O_{x+i}}{\sum_{i=-r}^{r} k_i} \tag{3.5}$$

dove N_x rappresenta l'x-esimo pixel nella immagine convoluta, O_x l'x-esimo pixel nella immagine originale, e k_i l'i-esimo elemento nel *kernel* che si estende

da $-r$ a $+r$. Notiamo dalla formula che il prodotto della convoluzione viene poi normalizzato dividendo per la somma degli elementi del *kernel* in modo da avere nella immagine convoluta valori in scala con quelli dell'immagine originale. Tale normalizzazione puó essere in ogni caso omessa se si vuole far risaltare dettagli dell'immagine enfatizzandone la tonalitá. Il *kernel* agisce quindi spostandosi pixel dopo pixel sull'immagine originale e generando di volta in volta un nuovo pixel nell'immagine convoluta. Tornando all'esempio proposto prima, l'immagine risultante a seguito della convoluzione sará il seguente *array*

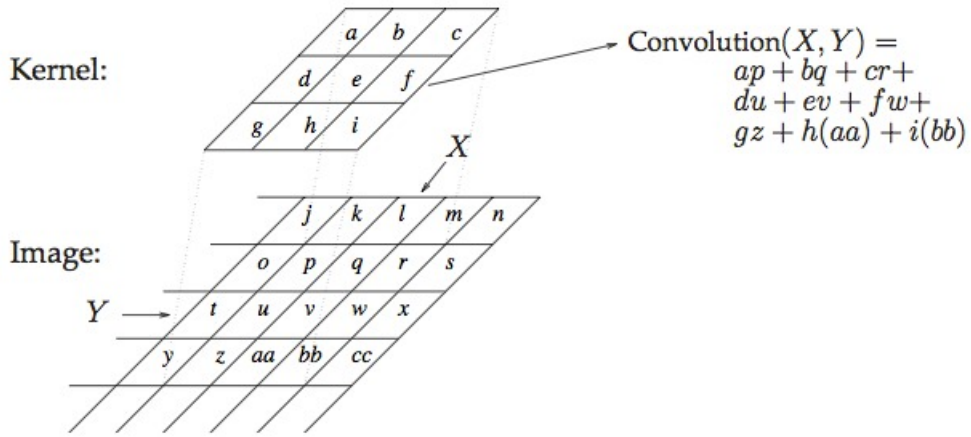
$$[\ldots 10\ 10\ 10\ 10\ 10\ 10\ 30\ 70\ 90\ 90\ 90\ 90\ 90\ 90\ldots]$$

con l'effetto che il bordo di separazione che prima era netto, ora risulta piú diffuso e smussato, passando da valori estremi 10 e 90 attraverso valori intermedi 30 e 70.

In modo del tutto analogo al caso unidimensionale possiamo dare l'espressione matematica della convoluzione in due dimensioni:

$$N_{x,y} = \frac{\sum_{i=-r}^{r} \sum_{j=-r}^{r} k_{i,j} O_{x+i,y+j}}{\sum_{i=-r}^{r} \sum_{j=-r}^{r} k_{i,j}} \tag{3.6}$$

In questo caso il *kernel* si sposta lungo tutti gli elementi della matrice originale ed agisce in maniera analoga al caso precedente come descritto dalla seguente figura:



Convolvendo la seguente immagine con un *kernel*

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 50 & 90 & 90 \\ 10 & 10 & 10 & 10 & 10 & 10 & 50 & 90 & 90 & 90 \\ 10 & 10 & 10 & 10 & 10 & 50 & 90 & 90 & 90 & 90 \\ 10 & 10 & 10 & 10 & 50 & 90 & 90 & 90 & 90 & 90 \\ 10 & 10 & 10 & 50 & 90 & 90 & 90 & 90 & 90 & 90 \\ 10 & 10 & 50 & 90 & 90 & 90 & 90 & 90 & 90 & 90 \end{bmatrix} \otimes \begin{bmatrix} 0.06 & 0.13 & 0.06 \\ 0.13 & \mathbf{0.24} & 0.13 \\ 0.06 & 0.13 & 0.06 \end{bmatrix}$$

otteniamo

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 12 & 25 & 50 & 75 & 88 \\ 10 & 10 & 10 & 10 & 12 & 25 & 50 & 75 & 88 & 90 \\ 10 & 10 & 10 & 12 & 25 & 50 & 75 & 88 & 90 & 90 \\ 10 & 10 & 12 & 25 & 50 & 75 & 88 & 90 & 90 & 90 \\ 10 & 12 & 25 & 50 & 75 & 88 & 90 & 90 & 90 & 90 \\ 12 & 25 & 50 & 75 & 88 & 90 & 90 & 90 & 90 & 90 \end{bmatrix}$$

Concludiamo dicendo che l'operazione di convoluzione con un *kernel puó* causare problemi lungo i bordi dell'immagine. Se consideriamo l'esempio sopra, quando il *kernel* scorre lungo la prima riga, succede che ai valori della prima riga del *kernel* non corrisponde nessun pixel sull'immagine originale. In questo caso il calcolatore esegue lo stesso l'operazione facendo finta che sopra la prima riga di pixel dell'immagine sia presente una riga di 0, che quindi durante il processo di convoluzione non danno contributo. Ma poiché questa operazione non coincide con la convoluzione che si ha su ogni altro pixel dell'immagine, essa puó condurre ad artefatti sull'immagine convoluta lungo i bordi.

3.1 Proprietá del *kernel*

Dal momento che fondamentalmente la convoluzione non é nient'altro che una ripetizione di moltiplicazioni ed addizioni, essa condivide molte delle pro-

prietá di queste ultime. In particolare la convoluzione é un processo lineare e invariante per traslazioni.

- **Linearitá.** Questa proprietá significa che é possibile equivalentemente agire su una immagine per intero oppure suddividerla in piú parti e agire su ognuna di esse separatamente e sommare i risultati alla fine. Significa inoltre che se si esegue una convoluzione ripetuta con diversi *kernel*, essa equivale a convolvere l'immagine con un *kernel* i cui elementi sono la somma dei corrispondenti elementi dei due *kernel* precedenti. In formula:

$$G \otimes \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} + G \otimes \begin{bmatrix} k & l & m \\ n & o & p \\ q & r & s \end{bmatrix} = G \otimes \begin{bmatrix} a+k & b+l & c+m \\ d+n & e+o & f+p \\ g+q & h+r & i+s \end{bmatrix}$$

- **Invarianza per traslazione.** Questa proprietá garantisce che il risultato di una convoluzione con un *kernel* sia lo stesso indipendentemente da dove un certo gruppo di pixel si trovi all'interno di una immagine. Per esempio la convoluzione di una immagine con un pianeta produrrá lo stesso risultato sia che il pianeta si trovi al centro dell'immagine oppure vicino al bordo.

- **Commutativitá.** Questa proprietá garantisce che in caso di applicazione di vari *kernel* ad una immagine, non é importante l'ordine di applicazione. In formula:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

- **Associativitá.** Questa proprietá mi garantisce che componendo tra loro due *kernel* piú piccoli a formarne uno piú grande, il risultato che si ha applicando in successione i due *kernel* piú piccoli sia lo stesso che si ottiene applicando il *kernel* generato dai due. Ovvero dati due *kernel*

k_1 e k_2 e chiamato $k_3 = k_1 \otimes k_2$ la loro convoluzione si ha

$$(G \otimes k_1) \otimes k_2 = G \otimes (k_1 \otimes k_2) = G \otimes k_3$$

- **Separabilitá.** L'applicazione successiva di due *kernel* unidimensionali produce lo stesso effetto dell'applicazione di un *kernel* bidimensionale. nell'esempio sottostante un *kernel* 5×5 é ottenuto da due di dimensione 5×1

$$\begin{bmatrix} 1 \\ 4 \\ \mathbf{6} \\ 4 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 4 & \mathbf{6} & 4 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & \mathbf{36} & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

  Ma eseguendo la convoluzione con un *kernel* di dimensione 5×5, per ogni pixel si devono eseguire 25 moltiplicazioni e 25 addizioni, al contrario eseguendo l'equivalente convoluzione consecutiva con due *kernel* di dimensione 5×1 si devono eseguire soltanto 10 moltiplicazioni e 10 addizioni, con notevoli benefici dal punto di vista computazionale.

3.2 Filtri spaziali

Come vedremo piú nel dettaglio in un capitolo successivo dedicato alla trasformata di Fourier, una immagine si puó decomporre in una somma di funzioni trigonometriche di varia ampiezza e frequenza. Le componenti ad alta frequenza hanno picchi molto vicini tra loro e contribuiscono ai dettagli piú evidenti dell'immagine, mentre le componenti a bassa frequenza contribuiscono in modo molto minore ai dettagli operando piú sulle caratteristiche ad ampia scala. Abbiamo inoltre visto che la convoluzione nient'altro é che l'azione di un *kernel* su un'immagine. Quindi *kernels* che agiscono in modo uniforme su pixel adiacenti attenueranno le alte frequenze senza alterare in modo significativo le basse frequenze, questi vengono detti *filtri lowpass*. In-

vece *kernels* che accentuano le differenze tra pixel adiacenti enfatizzeranno le alte frequenze, e vengono chiamati *filtri highpass*.

3.2.1 Filtri lowpass

Questo tipo di filtri abbiamo giá detto che agiscono su ogni pixel trasformandolo in una sorte di "media" con i pixel adiacenti. L'applicazione piú utile di questi filtri é per ridurre il rumore di una immagine, risultando peró molto spesso in una perdita di dettagli dell'immagine. Nonostante ció, in riprese da terra in cui le immagini sono giá affette da perturbazioni dovute all'atmosfera, la scelta di un opportuno filtro puó permettere di ridurre il rumore di fondo senza grosse ripercussioni sulla nitidezza (giá scarsa) dell'immagine. L'effetto che ne deriva é che l'immagine sembra "smussata" e "levigata". Di seguito alcuni esempi di filtri a passo-basso:

il filtro "boxcar" 3×3: $\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$

da ad ogni pixel lo stesso peso, e puó condurre a una ingente perdita di dettagli piccoli in immagini molto nitide.

il filtro "gentle" 3×3: $\frac{1}{16} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{8} & 1 \\ 1 & 1 & 1 \end{bmatrix}$

al contrario del precedente enfatizza il pixel esaminato con un peso pari a 8 volte quelli circostanti, e l'effetto "smussato" che si ottiene é molto tenue.

il filtro "gaussian" 3×3: $\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & \mathbf{4} & 2 \\ 1 & 2 & 1 \end{bmatrix}$

dove lo smussamento é passato anche ai pixel vicini.

Questi sono solo alcuni esempi di filtri, abbiamo inoltre riportato solo filtri 3×3, ma essi possono essere di ogni dimensione.

3.2.2 Filtri highpass

Questo tipo di filtri aumentano la differenza tra il pixel centrale esaminato e i pixel circostanti enfatizzando il contrasto dei dettagli nell'immagine, ma accentuando anche un eventuale rumore presente. La maggior parte di questi filtri sono strutturati in modo che la somma degli elementi del *kernel* sia 1. Consideriamo il seguente filtro:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{9} & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Notiamo come abbiamo giá detto che la somma dei sui elementi é pari ad 1. Ma come agisce questo filtro?
Per rispondere alla domanda consideriamolo come una matrice e scomponiamolo nel modo seguente:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{9} & -1 \\ -1 & -1 & -1 \end{bmatrix} = \underbrace{\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{8} & -1 \\ -1 & -1 & -1 \end{bmatrix}}_{A} + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{B}$$

Abbiamo quindi scomposto il filtro come somma di due filtri A e B, in cui A ha somma degli elementi nulla. Per quanto abbiamo detto precedentemente sul calcolo esplicito della convoluzione ne segue che l'azione di A e B sull immagine puó essere riassunta cosí:

- la convoluzione con B, poiché essa é formata solo da 0 e da un 1 nella posizione centrale, non fa altro che creare una copia esatta dell'immagine originale;

- nel caso i pixel circostanti al pixel esaminato abbiano il suo stesso valore l'azione di A é nulla perché la sommatoria dei prodotti fa 0;
- nel caso il pixel esaminato abbia valore maggiore o minore dei pixel circostanti, il suo valore viene enfatizzato di un fattore 8 da A.

Riassumendo, l'azione di questo filtro puó essere vista come l'aggiunta all'immagine originale (azione di B) di una immagine in cui sono evidenziate le differenze dei vari pixel dalla media dei pixel adiacenti.
Possiamo peró scomporre il filtro anche in un altro modo:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & \mathbf{9} & -1 \\ -1 & -1 & -1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{10} & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{C} - \underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 1 & \mathbf{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}}_{D}$$

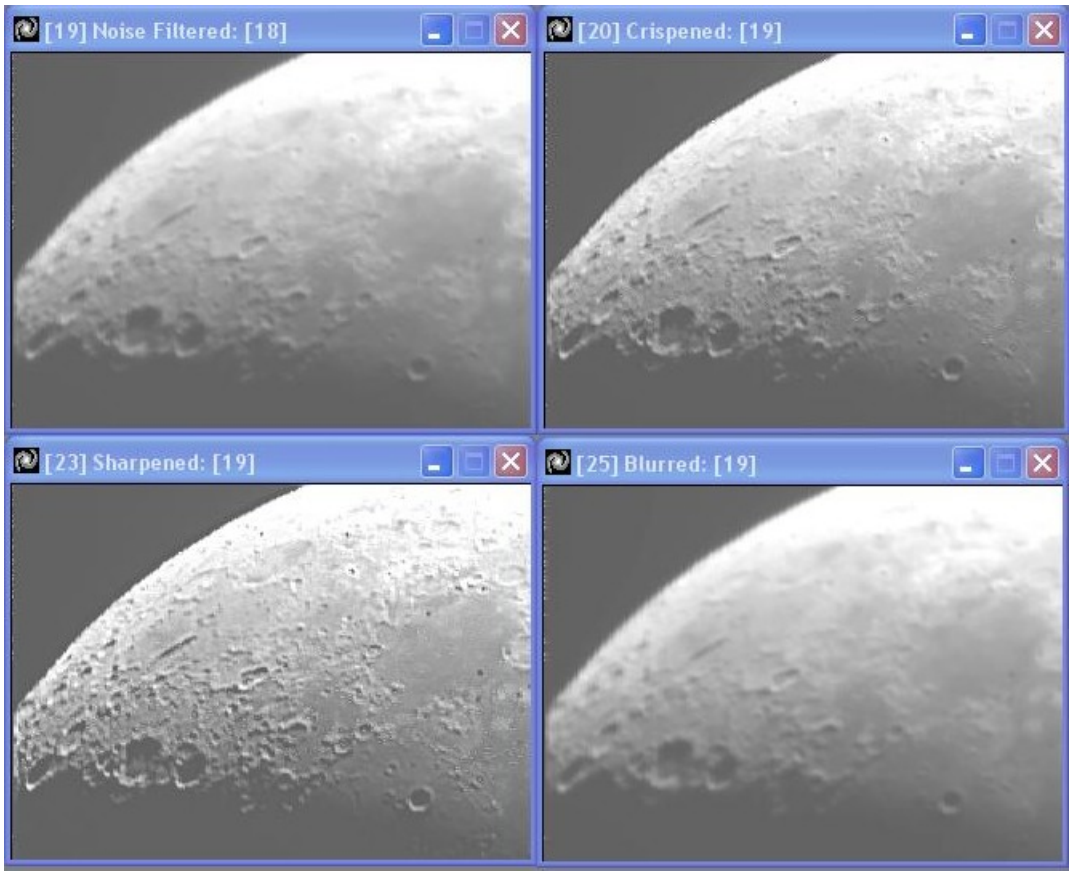
In questo caso abbiamo scomposto il filtro come differenza di due *kernels* separati C e D. L'azione di C crea una copia dell'immagine originale enfatizzandone i valori di un fattore 10, a cui va poi sottratta l'immagine ottenuta dall'azione di D, che nient'altro é che un filtro a passo-basso.

Questo tipo di filtri 3×3 sono molto adatti quando si lavora con immagini planetarie perché ricche di dettagli, ma hanno grossi deficit di prestazioni quando si ha a che fare con immagini di campi stellari. Tendenzialmente infatti i pixel rappresentanti la stella hanno valori molto maggiori (in quanto sorgenti luminose) dei pixel circostanti, e quando il filtro va ad agire su questi ultimi, i fattori negativi del filtro amplificano in negativo il contributo dei pixel della stella, creando attorno alle stelle una sorta di "alone nero". Si preferisce in questo caso usare filtri di maggiore dimensione come ad esempio

$$\begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -4 & -16 & -24 & -16 & -4 \\ -6 & -24 & \mathbf{220} & -24 & -6 \\ -4 & -16 & -24 & -16 & -4 \\ -1 & -4 & -6 & -4 & -1 \end{bmatrix}$$

in modo che il filtro lavori su un'area di pixel piú ampia.

Esempio 3.1. Usando il programma AIP4WIN applichiamo vari tipi di filtri ad una immagine della superficie lunare. quello che otteniamo é riportato nella figura seguente



L'immagine in alto a sinistra rappresenta la mia immagine originale. Nella immagine in alto a destra e in basso a sinistra abbiamo applicato due *filtri highpass* con intensitá crescente, si vede come l'immagine risulti piú nitida e compaiano molti dettagli dei crateri lunari prima non visibili. Nell'immagine in basso a destra abbiamo applicato invece un *filtro lowpass*, ottenendo una immagine piú sfocata e meno ricca di dettagli.

3.3 Rilevamento bordi e gradienti

Nelle applicazioni dei processi di immagini rivestono un ruolo importante metodi per il rilevamento dei bordi degli oggetti nelle immagini. Nel caso di immagini astronomiche questo aspetto puó applicarsi solo a immagini di superfici planetarie, mentre nella stra grande maggioranza dei casi si ha a che fare con immagini di galassie o nebulose, i cui confini si sfumano dolcemente con il cielo di sfondo, e quindi perde di significato e rilevanza parlare di "bordi". In ogni caso faremo una breve panoramica su questa importante applicazione della convoluzione mediante *kernel*.

Nel caso si voglia per esempio cercare di enfatizzare le linee orizzontali presenti in una immagine, e quindi rilevarne la presenza si puó utilizzare un *kernel* modificato per l'evenienza:

$$\begin{bmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix}$$

Strutturato in questo modo, l'azione per convoluzione con questo *kernel* enfatizza elementi che si trovano lungo una direzione orizzontale. In modo del tutto analogo si puó costruirre un *kernel* che agisca lungo una direzione arbitrario.

Un'altra categoria di *kernel* molto usati sono i cosiddetti *operatori a bassorilievo*, e sono di queste forma:

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & -1 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & -1 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & -1 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ -1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 0 & 0 \\ -1 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Prendiamo per esempio il primo e ragioniamo in modo analogo a quanto fatto per i *filtri highpass*. Possiamo scomporre il *kernel* come:

$$\begin{bmatrix} 0 & -1 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{A} - \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{B}$$

In questo caso quindi l'azione dell'operatore di bordo si puó interpretare come l'azione di A che crea una copia dell'immagine originale a cui viene sottratta una immagine traslata di 1 pixel verso il basso, generata dall'azione di B. L'effetto tridimensionale che si crea assomiglia alla presenza di un gradiente luminoso dall'alto verso il basso.

Capitolo 4

Trasformata di Fourier

In questo capitolo studieremo uno strumento potentissimo nell'analisi di immagini, la *trasformata di Fourier*. L'idea fondamentale che sta dietro a tutta la teoria é che un segnale, nel nostro caso una immagine, puó essere decomposto in un unico spettro di frequenze spaziali, modificato in questo spazio per poi essere riconvertito in una nuova immagine con caratteristiche modificate. Con appropriate alterazioni dello spettro possiamo aumentare i dettagli di una immagine, modificarne la distribuzione di luminositá, o rimuoverne il rumore presente. Queste sono solo alcune delle innumerevoli applicazioni.

La teoria matematica che sta dietro a questa trasformata vede la sua introduzione nel 1822 da parte del matematico francese Baptiste Joseph Fourier $(1768 - 1830)$. Essa tuttavia era ovviamente assai distante dal suo uso (tra gli svariati usi) in campo di processamento di immagini. Vedremo di seguito i passi principali che hanno portato dalla teoria originale alla *Fast Fourier Transform* (FFT) usata in analisi di immagini.

4.1 Serie di Fourier

Sappiamo dall'analisi che una funzione reale f si dice periodica di *periodo* T se $T > 0$ é un numero reale tale che

$$f(x+T) = f(x) \quad \text{per ogni} \quad x \in \mathbb{R}$$

Fourier dimostró che ogni funzione periodica di periodo 2π poteva essere scritta come sommatoria infinita di funzioni trigonometriche elementari nella seguente forma:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}[a_n cos(nx) + b_n sin(nx)] \tag{4.1}$$

con

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)cos(nx)dx \tag{4.2}$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)sin(nx)dx \tag{4.3}$$

i coefficienti descritti in 4.2 e 4.3 prendono il nome di *coefficienti di Fourier*, mentre l'equazione 4.1 si chiama *Serie di Fourier* della funzione f.
Introducendo dei coefficienti complessi c_n e usando la ben nota formula di Eulero

$$e^{ix} = cos(x) + isin(x)$$

possiamo riscrivere la serie di Fourier di f nel seguente modo:

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{inx} \tag{4.4}$$

dove

$$c_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx = \begin{cases} \frac{1}{2}(a_n - ib_n), & \text{se } n > 0 \\ \frac{a_0}{2}, & \text{se } n = 0 \\ \frac{1}{2}(a_{-n} + ib_{-n}), & \text{se } n < 0 \end{cases} \tag{4.5}$$

A seconda delle situazioni puó essere piú conveniente ragionare in termini di serie i Fourier come funzioni trigonometriche oppure come in quest'ultimo caso. Vediamo un esempio numerico per fissare le idee:

Esempio 4.1. Consideriamo la funzione di periodo 2π data $f(x) = x$ sull'intervallo $[-\pi, \pi[$ e calcoliamone i coefficienti

$$\begin{aligned} a_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} xcos(nx)dx = \\ &= \frac{1}{\pi n}\left[xsin(nx)\right]_{-\pi}^{\pi} - \frac{1}{\pi n}\int_{-\pi}^{\pi} sin(nx)dx = \\ &= \frac{1}{n}[sin(n\pi) + sin(-n\pi)] - \frac{1}{\pi n^2}\left[-cos(nx)\right]_{-\pi}^{\pi} = \\ &= 0 \end{aligned}$$

e

$$\begin{aligned} b_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} xsin(nx)dx = \\ &= -\frac{1}{\pi n}\left[xcos(nx)\right]_{-\pi}^{\pi} + \frac{1}{\pi n}\int_{-\pi}^{\pi} cos(nx)dx = \\ &= -\frac{1}{n}[cos(n\pi) + cos(-n\pi)] + \frac{1}{\pi n^2}\left[sin(nx)\right]_{-\pi}^{\pi} = \\ &= -\frac{1}{n}[cos(n\pi) + cos(n\pi)] = \\ &= (-1)\frac{2}{n}cos(n\pi) = \frac{2}{n}(-1)(-1)^n = \frac{2}{n}(-1)^{n+1} \end{aligned}$$

Quindi la nostra funzione f la possiamo scrivere come

$$f(x) = 2\sum_{n=1}^{\infty}\frac{(-1)^{n+1}}{n}sin(nx)$$

Quanto appena detto sulle funzioni 2π-periodiche lo si puó generalizzare al caso di funzioni periodiche di periodo qualsiasi.

Sia f una funzione di periodo $T = 2\pi w$, allora la sua serie di Fourier in forma trigonometrica sará data da:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}[a_n cos(\frac{nx}{w}) + b_n sin(\frac{nx}{w})] \tag{4.6}$$

mentre quella a coefficienti complessi risulta:

$$f(x) = \sum_{n=-\infty}^{+\infty} c_n e^{inwx} \tag{4.7}$$

con

$$c_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)e^{-inwx}dx, \quad w = \frac{2\pi}{T}, \quad n = 0, \pm 1, \pm 2, \ldots$$

4.2 Trasformata di Fourier

Fino a questo punto abbiamo definito la serie di Fourier soltanto per funzioni periodiche, quello che vogliamo ora é generalizzare questo concetto per funzioni continue qualunque.

Sia $f(x)$ una funzione continua di variabile reale x. La *trasformata di Fourier* di $f(x)$, indicata con $\mathfrak{F}\{f(x)\}$, é definita come:

$$\mathfrak{F}\{f(x)\} = F(u) = \int_{-\infty}^{+\infty} f(x)e^{-2\pi iux}dx \tag{4.8}$$

Data $F(u)$, possiamo riottenere la nostra $f(x)$ di partenza usando la *trasformata di Fourier inversa* definita da:

$$\mathfrak{F}^{-1}\{F(u)\} = f(x) = \int_{-\infty}^{+\infty} F(u)e^{2\pi iux}du \tag{4.9}$$

Le equazioni 4.8 e 4.9 precedenti costituiscono quella che viene detta *coppia di trasformate di Fourier*, e si puó dimostrare che esistono se $f(x)$ é continua e integrabile e se $F(u)$ é integrabile. Nella pratica queste condizioni sono quasi sempre verificate. Nei casi che ci interessano, $f(x)$ sará sempre una funzione reale, ma la trasformata di una funzione reale quasi sempre é una funzione a valori complessi, ovvero possiamo scrivere

$$F(u) = R(u) + iI(u) \tag{4.10}$$

dove $R(u)$ e $I(u)$ rappresentano rispettivamente la parte reale e immaginaria di $F(u)$. Alle volte, come visto in precedenza, puó convenire esprimere l'equazione 4.10 in forma esponenziale complessa

$$F(u) = |F(u)|e^{i\phi(u)} \tag{4.11}$$

dove

$$|F(u)| = \sqrt{R^2(u) + I^2(u)}$$

e

$$\phi(u) = arctan\left[\frac{I(u)}{R(u)}\right]$$

La funzione $|F(u)|$ é chiamata *spettro di Fourier* di $f(x)$, mentre $\phi(u)$ é detta *angolo di fase*. E chiamiamo *energia dello spettro* la quantitá $P(u) = |F(u)|^2$. La variabile u che appare nella trasformata di Fourier é chiamata *variabile di frequenza*. Il nome deriva dal fatto che, ricordando la formula di Eulero giá citata in precedenza (qui riportata in un'altra forma)

$$e^{-2\pi iux} = cos(2\pi ux) - isin(2\pi ux)$$

e l'equazione 4.8, quest'ultima puó essere interpretata come limite di una sommatoria di termini discreti, ed é evidente che $F(u)$ é composta da un'infinita somma di termini in seno e coseno, e quindi ogni valore di u determina la *frequenza* della corrispondente coppia seno-coseno.

Quanto detto si puó estendere anche a funzioni a due variabili. Sia $f(x, y)$ una funzione continua e integrabile, la sua trasformata di Fourier é data da

$$\mathfrak{F}\{f(x,y)\} = F(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x,y)e^{-2\pi i(ux+vy)}dxdy \tag{4.12}$$

e analagomante definiamo la sua trasformata inversa

$$\mathfrak{F}^{-1}\{F(u,v)\} = f(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} F(u,v)e^{2\pi i(ux+vy)}dudv \tag{4.13}$$

Tutte le nozioni giá esplicitate per il caso unidimensionale si ricavano in maniera del tutto analoga anche in questo caso.

4.2.1 Fourier e convoluzione

In aggiunta a quanto detto nel capitolo precedente e alla luce dei nuovi risultati, possiamo vedere il legame che esiste tra trasformata di Fourier e

convoluzione, in altre parole, la trasformata di Fourier della convoluzione di due funzioni é il prodotto delle trasformate di Fourier delle due funzioni. Ci limitiamo a enunciare e dimostrare il seguente risultato noto come *Teorema di convoluzione* nel caso unidimensionale, ma esso vale per ogni dimensione dello spazio di definizione delle mie funzioni.

Teorema 4.2 (Teorema di convoluzione)**.** *Siano f e g due funzioni continue, e sia $h(x) = (f * g)(x)$. Allora vale che*

$$\mathfrak{H}\{h(x)\} = \mathfrak{F}\{f(x)\}\mathfrak{G}\{g(x)\}$$

Dimostrazione. Siano f,g e h come da ipotesi, allora:

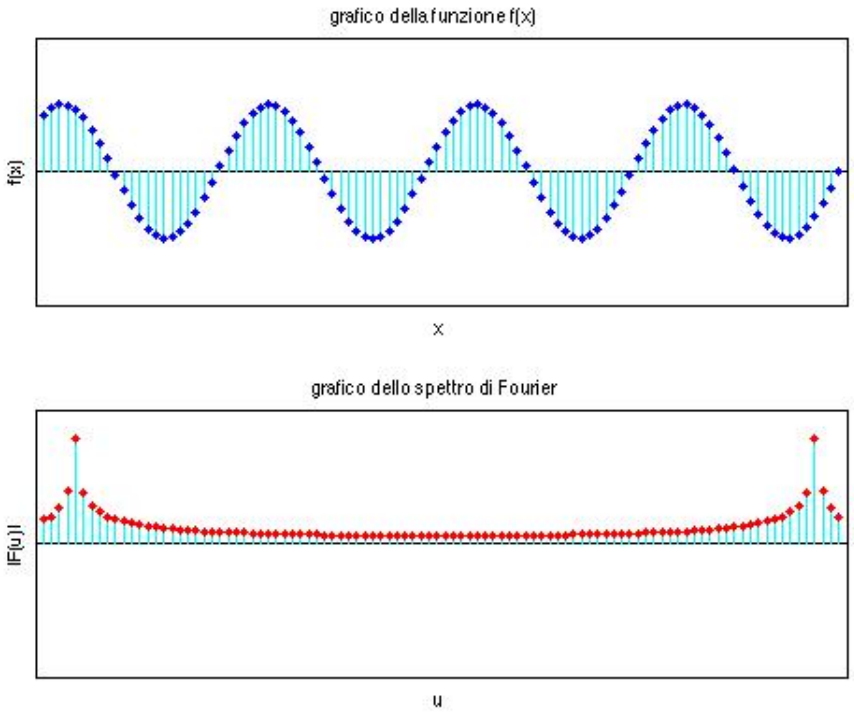
$$\begin{aligned}
\mathfrak{H}\{h(x)\} &= \mathfrak{H}\{(f*g)(x)\} = \\
&= \int_{\mathbb{R}} (f*g)(x) e^{-2\pi i u x} dx = \\
&= \int_{\mathbb{R}} \left(\int_{\mathbb{R}} f(z) g(x-z) e^{-2\pi i u x + 2\pi i u z - 2\pi i u z} dz \right) dx = \\
&= \int_{\mathbb{R}} \left(\int_{\mathbb{R}} f(z) g(x-z) e^{-2\pi i u (x-z)} e^{-2\pi i u z} dx \right) dz = \\
&= \int_{\mathbb{R}} f(z) \left(\int_{\mathbb{R}} g(x-z) e^{-2\pi i u (x-z)} dx \right) e^{-2\pi i u z} dz = \quad \text{pongo } y = x - z \\
&= \int_{\mathbb{R}} f(z) \left(\int_{\mathbb{R}} g(y) e^{-2\pi i u (y)} dy \right) e^{-2\pi i u z} dz = \\
&= \left(\int_{\mathbb{R}} f(z) e^{-2\pi i u z} dz \right) \left(\int_{\mathbb{R}} g(y) e^{-2\pi i u (y)} dy \right) = \\
&= \mathfrak{F}\{f(x)\}\mathfrak{G}\{g(x)\}
\end{aligned}$$

□

Questo teorema importantissimo assicura quindi la possibilitá di calcolare la convoluzione tra due funzioni calcolando prima le loro trasformate, moltiplicandole tra loro e poi applicare la trasformata inversa. Questi semplici passaggi al dominio delle frequenze si rendono enormemente vantaggiosi quando si ha a che fare con *kernel* di grandi dimensioni, in quanto il costo computazionale della trasformata di Fourier (come vedremo successivamente) é inferiore e la sua computazione molto piú rapida e flessibile.

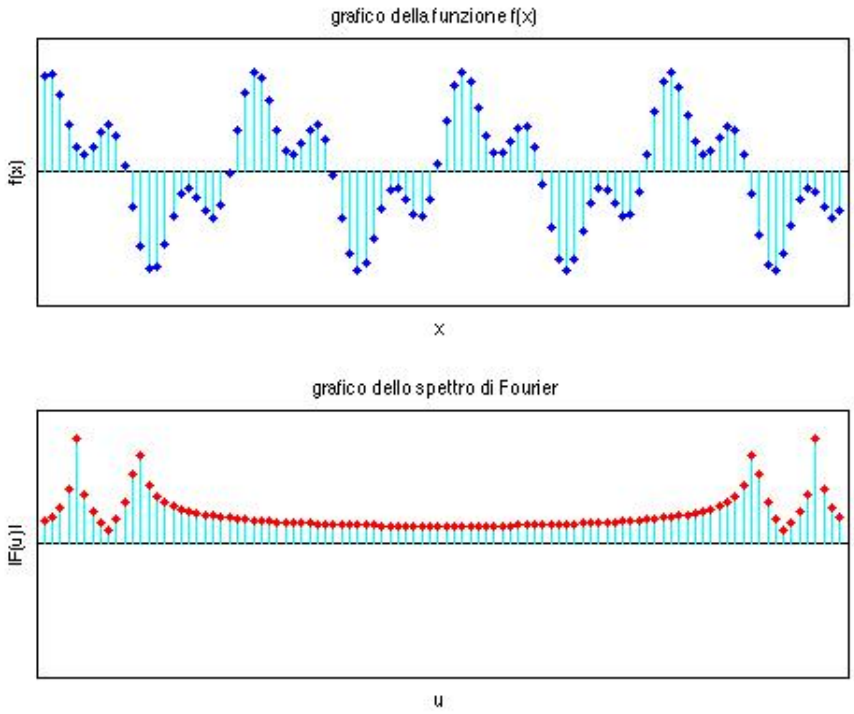
Esempio 4.3. A titolo esplicativo della potenza della trasformata di Fourier in termini di utilitá nell'analisi di immagini, riportiamo alcuni esempi che speriamo siano chiarificatori su quanto svolge la trasformata e le informazioni che da essa si possono trarre. I grafici riportati in seguito sono stati ottenuti con il software MATLAB tramite una opportuna modifica del file "*fftgui.m*" scritto da Cleve Moler e reperibile gratuitamente online su `http://www.mathworks.com`. Si tenga presente che i grafici dello *spettro* risultano simmetrici con asse verticale per la proprietá di essere coniugati dei coefficienti di Fourier simmetrici per posizione.

1. Consideriamo per prima la funzione $f(x) = sin(x)$ e costruiamo il grafico della funzione e il grafico dello spettro.



In questo caso la nostra funzione di partenza é una semplice funzione trigonometrica, e come ci aspettiamo l'energia del segnale é quasi tutta concentrata in corrispondenza di una singola frequenza.

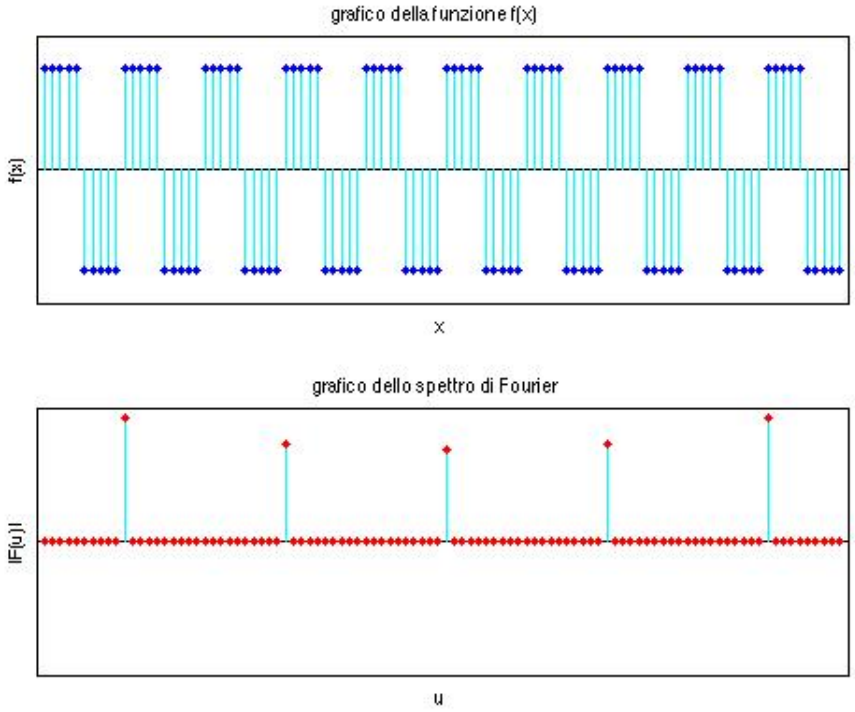
2. Consideriamo ora invece la funzione data da $f(x) = sin(x) - \frac{3}{5}cos(3x)$, ovvero il nostro segnale in questo caso é la risultante della somma di due sinusoidi con frequenze diverse.



Si capisce immediatamente dal grafico della funzione che se non conoscessimo l'espressione analitica della funzione, sarebbe molto difficile intuire le due frequenze di base, cosa che invece osservando lo spettro risulta evidente dai due picchi presenti (a destra o sinistra é indifferente, per quanto detto prima).

3. Consideriamo come ultimo il caso di un'onda quadra (a volte detta anche *treno di impulsi*). Le onde quadre sono un particolare tipo di

onde periodiche non-sinusoidali, che possono essere rappresentate come una infinita somma di onde sinusoidali, e in cui la ampiezza dell'onda si alterna ad una frequenza fissata tra due valori minimo e massimo. Questo particolare tipo di onde si incontrano spesso in elettronica. Il grafico dell'onda quadra é mostrato nel primo disegno dell'immagine sottostante:



Anche in questo caso l'analisi dello spettro di Fourier mette subito in risalto le frequenze dominanti delle sinusoidali che vanno a formare la mia onda quadra, cosa che sarebbe impossibile altrimenti.

4.3 Trasformata di Fourier discreta (DTF)

Cerchiamo ora un modo per passare dalla trasformata di una funzione continua, alla trasformata di una funzione discreta, come sono per esempio

le immagini.

Supponiamo che una funzione continua $f(x)$ sia discretizzata in una sequanza di valori

$$\{f(x_0), f(x_0 + \triangle x, f(x_0 + 2 \triangle x), \ldots, f(x_0 + [N-1] \triangle x)\}$$

ottenuta prendendo N punti di campionamento a distanza $\triangle x$. Nella trattazione seguente useremo x sia come variabile discreta che continua a seconda del contesto. Definiamo quindi

$$f(x) = f(x_0 + \triangle x) \tag{4.14}$$

dove x assume i valori discreti $0, 1, 2, \ldots, N-1$. In poche parole la sequenza

$$f(0), f(1), f(2), \ldots, f(N-1)$$

denoterá qualsiasi N-upla di valori campione uniformemente distanziati a partire dalla corrispondente funzione continua. Possiamo quindi definire la *trasformata di Fourier discreta* come

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} f(x) e^{\frac{-2\pi i u x}{N}} \tag{4.15}$$

con $u = 0, 1, 2, \ldots, N-1$, e la rispettiva trasformata inversa

$$f(x) = \sum_{u=0}^{N-1} F(u) e^{\frac{2\pi i u x}{N}} \tag{4.16}$$

con $x = 0, 1, 2, \ldots, N-1$.
I valori $u = 0, 1, 2, \ldots, N-1$ nella trasformata discreta data da 4.15 corrispondono a valori campione della trasformata continua rispetto ai valori $0, \triangle u, 2 \triangle u, \ldots, (N-1) \triangle u$. Si puó mostrare inoltre che le quantitá $\triangle u$ e $\triangle x$ sono legate dalla relazione

$$\triangle u = \frac{1}{N \triangle x} \tag{4.17}$$

Nel caso di funzione a due variabili, la trasformata di Fourier discreta e la inversa sono date rispettivamente da:

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-2\pi i(\frac{ux}{M}+\frac{vy}{N})} \tag{4.18}$$

con $u = 0, 1, 2, \ldots, M-1$ e $v = 0, 1, 2, \ldots, N-1$, e

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{2\pi i(\frac{ux}{M}+\frac{vy}{N})} \tag{4.19}$$

con $x = 0, 1, 2, \ldots, M-1$ e $y = 0, 1, 2, \ldots, N-1$. Anche in questo caso bidimensionale, la funzione discreta $f(x,y)$ rappresenta campioni della funzione $f(x_0 + x \triangle x, y_0 + y \triangle y)$ con x e y che variano come sopra rispettivamente tra 0 e $M-1$ e tra 0 e $N-1$, e gli incrementi nel dominio spaziale e in quello delle frequenze sono espressi da

$$\triangle u = \frac{1}{M \triangle x} \tag{4.20}$$

e

$$\triangle v = \frac{1}{N \triangle y} \tag{4.21}$$

Abbiamo definito la trasformata discreta nel caso generale con M campioni su un asse e N campioni sull'altro. Nel caso di campionamento equo, ovvero quando $M = N$, come accade generalmente per immagini digitalizzate, le equazioni diventano

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)e^{\frac{-2\pi i(ux+vy)}{N}} \tag{4.22}$$

con $u, v = 0, 1, 2, \ldots, N-1$, e

$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} F(u,v)e^{\frac{-2\pi i(ux+vy)}{N}} \tag{4.23}$$

con $x, y = 0, 1, 2, \ldots, N-1$. concludiamo dicendo che al contrario del caso continuo, sia $F(u)$ che $F(u,v)$ esistono sempre nel caso discreto, e lo si puó dimostrare sostituendo l'equazione 4.16 nell'equazione 4.15 e ottenendo una identitá.

4.4 Proprietá della trasformata

Enunciamo ora alcune delle proprietá principali della trasformata di Fourier. Precisiamo che a seconda di come sia piú conveniente e di facile comprensione, utilizzeremo indifferentemente la notazione continua o discreta, in quanto le proprietá valgono in ogni caso.

- **Linearitá**
  Questa proprietá, che discende direttamente dalla definizione integrale di trasformata, dice che se $f(x)$ e $g(x)$ sono due funzioni trasformabili secondo Fourier con trasformate rispettivamente $\mathfrak{F}\{f(x)\}$ e $\mathfrak{G}\{g(x)\}$, allora anche la loro somma $f(x) + g(x)$ é trasformabile ed ha trasformata pari a $\mathfrak{F}\{f(x)\} + \mathfrak{G}\{g(x)\}$. Conseguenza immediata di questa proprietá é che una funzione é moltiplicata per una costante k (ovvero é k volte la somma di se stessa) anche la sua trasformata risulterá amplificata di un fattore k.

- **Separabilitá**
  La coppia di trasformate di Fourier date dalla 4.22 e 4.23 possono essere espresse nella forma separata

$$F(u,v) = \frac{1}{N}\sum_{x=0}^{N-1} e^{\frac{-2\pi iux}{N}} \sum_{y=0}^{N-1} f(x,y)e^{\frac{-2\pi ivy}{N}} \tag{4.24}$$

  con $u, v = 0, 1, 2, \ldots, N-1$, e

$$f(x,y) = \frac{1}{N}\sum_{u=0}^{N-1} e^{\frac{2\pi iux}{N}} \sum_{v=0}^{N-1} F(u,v)e^{\frac{2\pi ivy}{N}} \tag{4.25}$$

  con $x, y = 0, 1, 2, \ldots, N-1$. Questa proprietá mi garantisce che le trasformate $F(u,v)$ e $f(x,y)$ possano essere calcolate in due passaggi successivi attraverso l'applicazione della trasformata di Fourier (diretta o inversa a seconda del caso) unidimensionale

- **Traslazione**
  Data $f(x)$ e $a \in \mathbb{R}$ si ha che

$$\begin{aligned}\mathfrak{F}\{f(x-a)\} &= \int_{-\infty}^{+\infty} f(x-a)e^{-2\pi iux}dx = \int_{-\infty}^{+\infty} f(X)e^{-2\pi iu(X+a)}dX = \\ &= e^{-2\pi ua}\int_{-\infty}^{+\infty} f(X)e^{-2\pi iuX}dX = e^{-2\pi ua}\mathfrak{F}\{f(x)\}\end{aligned}$$

  Quindi la traslazione dell'argomento nella funzione di partenza si traduce in una moltiplicazione nel dominio delle frequenze. Facciamo notare che la traslazione non modifica lo *spettro* in quanto $|e^{-2\pi ua}| = 1$.

- **Riscalamento** Data $f(x)$ e $a \in \mathbb{R} - \{0\}$ si ha che

$$\mathfrak{F}\{f(ax)\} = \frac{1}{|a|}\mathfrak{F}\{\frac{f(x)}{a}\}$$

  Questa proprietá mi dice che se amplifico le dimensioni di un oggetto nell'immagine di un fattore a, nel dominio delle frequenze di Fourier esso risulterá in una riduzione di fattore $\frac{1}{|a|}$ dello spettro.

- **Teorema di Parseval** Questo teorema mi garantisce un importantissima proprietá della trasformata di Fourier. Esso afferma che l'energia di un segnale rappresentato dalla mia funzione $f(x)$ é la stessa sia che venga calcolata nel dominio della funzione sia che venga calcolata nel dominio delle frequenze della sua trasformata, ovvero la trasformata di Fourier conserva l'energia del segnale in entrambe le direzioni (trasformata diretta e inversa). In formule il teorema afferma che:

$$\int_{-\infty}^{+\infty} |f(x)|^2 dx = \int_{-\infty}^{+\infty} |F(u)|^2 du \tag{4.26}$$

  Questo teorema ha grosse ripercussioni nel campo dell'analisi di immagini. Se infatti vogliamo enfatizzare certi dettagli in una immagine, basterá convertirla nel suo dominio delle frequenze, moltiplicare per un certo fattore l'energia delle frequenze associate a quei dettagli e poi

riconvertire l'immagine usando la trasformata inversa. In virtú del teorema anche i dettagli relativi a quelle frequenze modificate verranno enfatizzati dello stesso fattore.

4.5 Trasformata di Fourier veloce (FFT)

La trasformata di Fourier veloce, ovunque abbreviata con FFT (dall'inglese *Fast Fourier Transform*) é un algoritmo ottimizzato per il calcolo della trasformata di Fourier discreta al calcolatore.

Se consideriamo l'equazione 4.15 che rappresenta la trasformata discreta, abbiamo che il numero di moltiplicazioni e addizioni complesse da svolgere é $\mathcal{O}(N^2)$. Questo si puó facilmente dimostrare considerando che, per ognuno degli N valori di u si devono svolgere N moltiplicazioni complesse tra $f(x)$ e $e^{\frac{-2\pi iux}{N}}$ e successivamente $(N-1)$ addizioni tra i risultati ottenuti. Mostreremo qui di seguito che si puó effettuare una scomposizione dell'equazione 4.15 in modo da ottenere un costo computazionale inferiore pari a $\mathcal{O}(N \log_2 N)$. Questa procedura di scomposizione é chiamata *algoritmo FFT*. Esamineremo l'algoritmo soltanto nel caso unidimensionale, ricordando che, per quanto detto sulla proprietá di separabilitá della trasformata di Fourier, il caso bidimensionale si puó ricondurre a una applicazione in serie dell'algoritmo unidimensionale. Precisiamo che esistono svariati algoritmi FFT sviluppati nel corso della storia dell'calcolo numerico, ognuno basato su una differente scomposizione del numero intero N.

4.5.1 Algoritmo FFT

L'algoritmo FFT che riportiamo di seguito, ideato da Cooley-Tukey, si basa sul procedimento chiamato *metodo dei raddoppiamenti successivi*. Ri-

scriviamo l'equazione 4.15 nella forma

$$F(u) = \frac{1}{N}\sum_{x=0}^{N-1} f(x)W_N^{ux} \tag{4.27}$$

dove abbiamo posto $W_N = e^{\frac{-2\pi i}{N}}$, e assumiamo che N sia della forma $N = 2^n$, con n intero positivo. Sotto queste ipotevi possiamo scrivere

$$N = 2M$$

Sostituendo quest'ultima uguaglianza nella 4.27 otteniamo

$$\begin{aligned} F(u) &= \frac{1}{2M}\sum_{x=0}^{2M-1} f(x)W_{2M}^{ux} = \\ &= \frac{1}{2}\left\{\frac{1}{M}\sum_{x=0}^{M-1} f(2x)W_{2M}^{u(2x)} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)W_{2M}^{u(2x+1)}\right\} \end{aligned}$$

quest'ultimo passaggio, poiché per come é stato definito W_N si ha che $W_{2M}^{2ux} = W_M^{ux}$, puó essere riscritto come

$$F(u) = \frac{1}{2}\left\{\frac{1}{M}\sum_{x=0}^{M-1} f(2x)W_M^{ux} + \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)W_M^{ux}W_{2M}^{u}\right\} \tag{4.28}$$

Definiamo ora

$$F_{pari}(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(2x)W_M^{ux} \tag{4.29}$$

e

$$F_{dispari}(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(2x+1)W_M^{ux} \tag{4.30}$$

con $u = 0, 1, 2, \ldots, M-1$.
L'equazione 4.28 quindi diventa

$$F(u) = \frac{1}{2}\{F_{pari}(u) + F_{dispari}(u)W_{2M}^{u}\} \tag{4.31}$$

inoltre, poiché vale che $W_M^{u+M} = W_M^u$ e $W_{2M}^{u+M} = -W_{2M}^u$ ne segue che anche

$$F(u+M) = \frac{1}{2}\{F_{pari}(u) + F_{dispari}(u)W_{2M}^{u}\} \tag{4.32}$$

Le equazioni dalla 4.29 alla 4.32 ci dicono che una trasformata definita su N punti campione puó essere calcolata scomponendo l'espressione iniziale in due parti, come indicato dalla 4.31 e dalla 4.32. Inoltre il calcolo della prima metá di $F(u)$ richiede la valutazione di una trasformata definita su $\frac{N}{2}$ punti. I valori trovati di $F_{pari}(u)$ e $F_{dispari}(u)$ sono poi sostituiti nell'equazione 4.31 per ottenere valori di $F(u)$ per $u = 0, 1, 2, \ldots, (\frac{N}{2} - 1)$. L'altra metá di $F(u)$ segue immediatamente dall'equazione 4.32 senza ulteriori valutazioni da fare.

Per poter esaminare l'efficienza della procedura appena descritta, chiamiamo $m(n)$ e $a(n)$ rispettivamente il numero di moltiplicazioni ed addizioni complesse da effettuare in questa procedura, e sia, come in precedenza detto, 2^n il numero di campioni, con n intero positivo.

Supponiamo quindi che sia $n = 1$.
La procedure richiede quindi la valutazione di $F(0)$, e $F(1)$ discende direttamente dalla 4.32. Per ottenere $F(0)$ dobbiamo quindi calcolare $F_{pari}(0)$ e $F_{dispari}(0)$. In questo caso peró queste ultime sono trasformazioni su un solo punto campione, e quindi la trasformata di Fourier sará il punto stesso, senza necessitá di moltiplicazioni o addizioni. Per calcolare $F(u)$ dovremo quindi eseguire una moltiplicazione (tra $F_{dispari}(0)$ e W_2^0) e una addizione per ottenere $F(0)$ e una successiva addizione per calcolare $F(1)$ (ricordando che $F_{dispari}(0)W_2^0$ é giá stato calcolato). Riassumendo, il numero di operazioni richieste per calcolare una trasformata su 2 punti campione é di $m(1) = 1$ moltiplicazioni e $a(1) = 2$ addizioni.
Supponiamo ora che sia $n = 2$.
Seguendo il ragionamento svolto prima, una trasformata su 4 punti campione puó essere divisa in due parti. La prima metá di $F(u)$ richiede la valutazione di 2 trasformate su 2 punti campione, e per quanto abbiamo detto richiede quindi un numero di operazioni pari a $2m(1)$ moltiplicazioni e $2a(1)$ addizioni. Due ulteriori moltiplicazioni e addizioni sono a questo punto richieste per calcolare $F(0)$ e $F(1)$ dall'equazione 4.31. Poiché $F_{dispari}(u)W_{2M}^u$ é giá stato

calcolato per $u = \{0, 1\}$, ci basteranno soltanto altre due addizioni per ottenere $F(2)$ e $F(3)$. Il numero di operazioni totali sará quindi $m(2) = 2m(1)+2$ moltiplicazioni e $a(2) = 2a(1) + 4$ addizioni.
Con $n = 3$, si avranno 2 trasformate su 4 punti campione per la valutazione di $F_{pari}(u)$ e $F_{dispari}(u)$ che richiedono complessivamente $2m(2)$ moltiplicazioni e $2a(2)$ addizioni. si dovranno quindi fare 4 ulteriori moltiplicazioni e 8 addizioni per calcolare la trasformata completa, per un totale di $m(3) = 2m(2)+4$ moltiplicazioni e $a(3) = 2a(2) + 8$ addizioni.
Continuando il ragionamento si puó vedere che, per ogni valore intero positivo di n, il numero di operazioni richieste per eseguire la FFT é dato dalle seguenti espressioni ricorsive:

$$m(n) = 2m(n-1) + 2^{n-1} \tag{4.33}$$

e

$$a(n) = 2a(n-1) + 2^n \tag{4.34}$$

dove vale $m(0) = 0$ e $a(0) = 0$ dato che come abbiamo detto la trasformata di un singolo punto non richiede operazioni.

Proviamo ora per induzione su n che il numero di moltiplicazioni e di addizioni necessarie per questo algoritmo FFT é dato rispettivamente da

$$m(n) = \frac{1}{2}2^n log_2 2^n = \frac{1}{2}N log_2 N = \frac{1}{2}Nn \tag{4.35}$$

e

$$a(n) = 2^n log_2 2^n = N log_2 N = Nn \tag{4.36}$$

Dimostrazione. Per $n = 1$ abbiamo

$$m(1) = \frac{1}{2}(2)(1) = 1$$

e

$$a(1) = (2)(1) = 2$$

e quindi l'asserto é vero per quanto verificato precedentemente. Supponiamo quindi i risultati veri per n e proviamolo per $n+1$.
Dall'equazione 4.33 segue che

$$m(n+1) = 2m(n) + 2^n$$

e sostituendovi l'equazione 4.35 che abbiamo supposto vera per n otteniamo

$$m(n+1) = 2(\frac{1}{2}Nn) + 2^n = 2(\frac{1}{2}2^n n) + 2^n = 2^n(n+1) = \frac{1}{2}2^{n+1}log_2 2^{(n+1)}$$

Quindi l'equazione 4.35 é valida per ogni valore intero positivo di n.
Dall'equazione 4.34 segue che

$$a(n+1) = 2a(n) + 2^{n+1}$$

e sostituendovi l'equazione 4.36 che abbiamo supposto vera per n otteniamo

$$a(n+1) = 2Nn + 2^{n+1} = 2(2^n n) + 2^{n+1} = 2^{n+1}(n+1) = 2^{n+1}log_2 2^{(n+1)}$$

e quindi anche 4.36 risulta provata per ogni valore intero positivo di n. □

Quindi per concludere il numero di operazioni totali dell'algoritmo FFT é dato da

$$m(n) + a(n) = \frac{1}{2}Nlog_2 N + Nlog_2 N = \frac{3}{2}Nlog_2 N \simeq \mathcal{O}(Nlog_2 N)$$

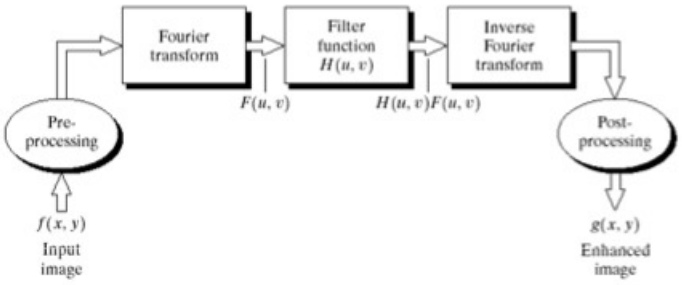
4.6 Applicazioni

La trasformata di Fourier é uno strumento potentissimo a nostra disposizione perché permette di modificare una immagine direttamente nel suo spazio delle frequenze. La maggior parte delle operazioni, come visto nel capitolo precedente, consisterá nell'applicare un filtro all'immagine per enfatizzare certe caratteristiche o affievolirne altre. Il *Teorema di Parseval* mi garantisce che tutte le modifiche in termini di energia vengano ritrasmesse alla immagine originale una volta operata la trasformata inversa, mentre il *Teorema di convoluzione* mi dice che l'applicazione di un filtro (leggasi

convoluzione con un *kernel*) nello spazio di Fourier si traduce con una moltiplicazione tra due immagini.

I passi da eseguire sono quindi i seguenti:

1. Moltiplicare l'immagine $f(x, y)$ per i coefficienti $(-1)^{x+y}$ per centrare lo spettro;
2. Calcolare la trasformata di Fourier $F(u, v)$;
3. Moltiplicare lo spettro $F(u, v)$ per una immagine filtro $H(u, v)$;
4. Calcolare la trasformata inversa $F^{-1}(u, v)$;
5. Rimoltiplicare per i coefficienti $(-1)^{x+y}$ per ottenere il risultato finale.



Ricordiamo che nello spettro di Fourier, le frequenze alte corrispondono ai dettagli piú nitidi di una immagine, dove vi é quindi un rapido cambiamento di contrasto, mentre le frequenze basse corrispondono alle parti dell'immagine che risultano piú "smussate".

I filtri utilizzabili nello spazio delle frequenze di Fourier si dividono in tre principali categorie, a seconda della loro funzione:

- Filtri **lowpass**:

  Questo tipo di filtri hanno la particolaritá di lasciar passare le frequenze basse e bloccare quelle maggiori di un certo valore di soglia d_0 detto

cutoff. L'immagine finale risulterá quindi piú sfocata e verrá privata di tanti piú dettagli quanto piú basso sará il *cutoff*.

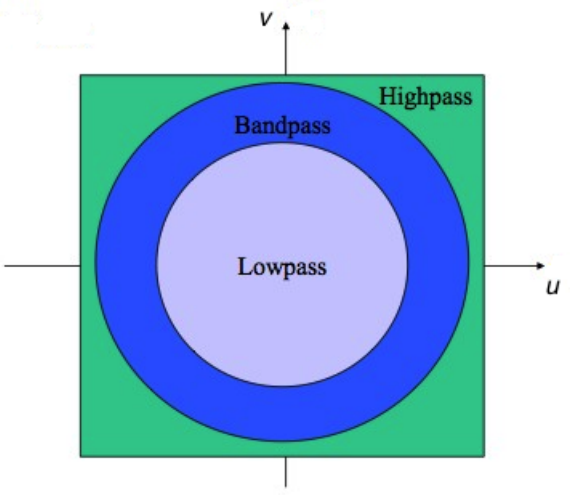
- Filtri **highpass**:

  Al contrario dei precedenti, questi filtri bloccano tutte le basse frequenze inferiori a d_0, lasciando passare soltanto le alte e facendo risultare l'immagine piú nitida e dai bordi piú definiti. Sono molto utili in analisi planetaria perché permettono di mettere in risalto strutture atmosferiche come per esempio la forma delle tempeste di Giove altrimenti poco visibili.

- Filtri **bandpass** e **bandstop**:

  In modo analogo ai precedenti, questi filtri permettono di lasciar passare (*bandpass*) o bloccare (*bandstop*) frequenze comprese tra due valori prestabiliti.

La seguente figura mostra la zona dello spazio delle frequenze (u, v) su cui agiscono positivamente le tipologie di filtri sopra descritti:



Esamineremo ora due tra i principali filtri usati in ambito astronomico, il *filtro Gaussiano* e il *filtro di Butterworth*.

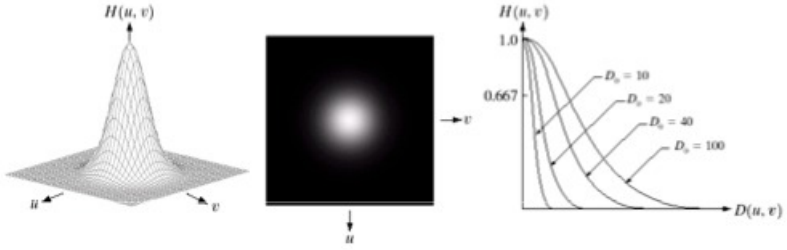
4.6.1 Filtri lowpass

- *Filtro Gaussiano lowpass*

  É descritto dalla seguente equazione:

  $$H_{GLP}(u,v) = e^{-\frac{1}{2}\frac{d^2(u,v)}{d_0^2}}$$

  dove $d(u,v) = \sqrt{(u-\frac{M}{2})^2 + (v-\frac{N}{2})^2}$ é la distanza del punto (u,v) dal centro dello spazio delle frequenze e d_0 é la frequenza di *cutoff*. La figura seguente mostra la visualizzazione del filtro come *mesh* (figura a sinistra), come immagine (figura centrale) e come sezione radiale (figura a destra):

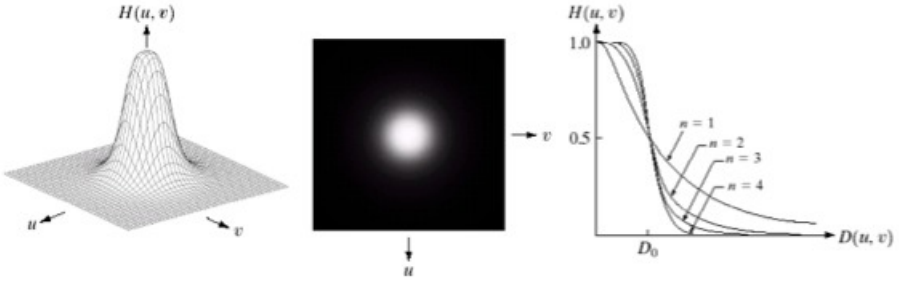
  Osservando per esempio la figura al centro che rappresenta il filtro visto come una immagine e ricordando che nello spazio di Fourier abbiamo una moltiplicazione tra immagini, si vede chiaramente l'azione del filtro. Esso mantiene le frequenze basse moltiplicandole per valori positivi, rappresentati del bianco, e elimina le frequenze negative moltiplicandole per valori nulli, rappresentati dal nero.

- *Filtro Butterworth lowpass*

  É descritto dalla seguente equazione:

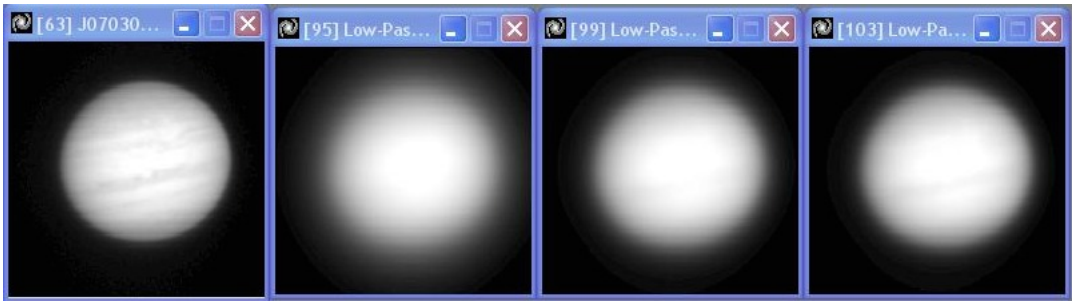
  $$H_{BLP}(u,v) = \frac{1}{1 + \left(\frac{d(u,v)}{d_0}\right)^{2n}}$$

dove le variabili sono le stesse giá esplicitate sopra e n rappresenta un parametro variabile a scelta chiamato *ordine* del filtro. Notiamo dalla equazione che per punti (u, v) vicini al centro dello spazio delle frequenze si ha $d(u, v) << d_0$ e quindi il valore di $H_{BLP}(u, v)$ é molto vicino a 1, lasciando quindi invariate le frequenze basse. Mano a mano invece che ci si allontana radialmente dal centro, il fattore $\frac{d(u,v)}{d_0}$ cresce sempre piú e $H_{BLP}(u, v) \longrightarrow 0$ con il risultato di eliminare le frequenze alte. La figura seguente mostra una visualizzazione del filtro:



Si possono effettuare considerazioni analoghe al filtro Gaussiano.

Esempio 4.4. la figura sottostante mostra l'applicazione di un *filtro Butterworth lowpass* ad una immagine del pianeta Giove. La prima immagine a sinistra é l'immagine originale, mentre procedendo verso destra vi sono le immagini ottenute con ordine rispettivamente uguale a $n = 4, 8, 12$:



Notiamo come il filtro mi blocchi le frequenze piú alte causando una notevole perdita di dettagli nell'immagine, dove del pianeta rimane soltando una macchia di forma pseudosferica.

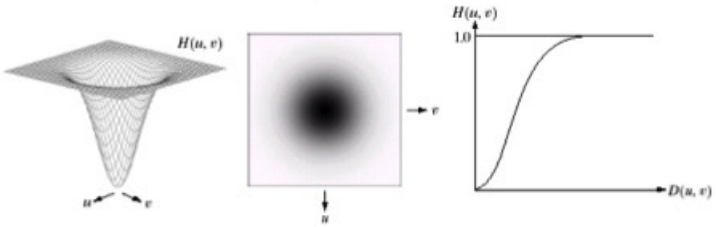
4.6.2 Filtri highpass

- *Filtro Gaussiano highpass*

  In maniera molto intuitiva, esso é definito come:

$$H_{GHP}(u,v) = 1 - H_{GLP}(u,v) = 1 - e^{-\frac{1}{2}\frac{d^2(u,v)}{d_0^2}}$$

  e la figura sottostante mostra la sua forma:

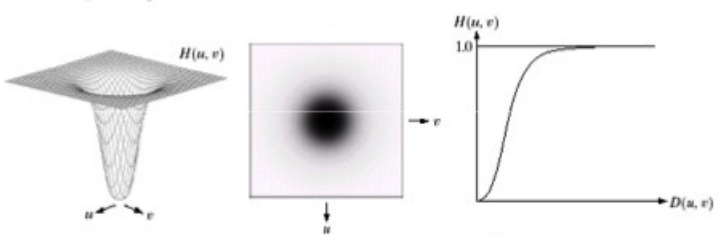


- *Filtro Butterworth highpass*

  Anche esso é definito come:

$$\begin{aligned} H_{BHP}(u,v) &= 1 - H_{BLP}(u,v) = 1 - \frac{1}{\frac{d_0^{2n}+d^{2n}(u,v)}{d_0^{2n}}} = 1 - \frac{d_0^{2n}}{d_0^{2n}+d^{2n}(u,v)} = \\ &= \frac{d^{2n}(u,v)}{d_0^{2n}+d^{2n}(u,v)} = \frac{1}{\frac{d_0^{2n}+d^{2n}(u,v)}{d^{2n}(u,v)}} = \\ &= \frac{1}{1+\left(\frac{d(u,v)}{d_0}\right)^{2n}} \end{aligned}$$

  La figura sottostante mostra la sua forma:

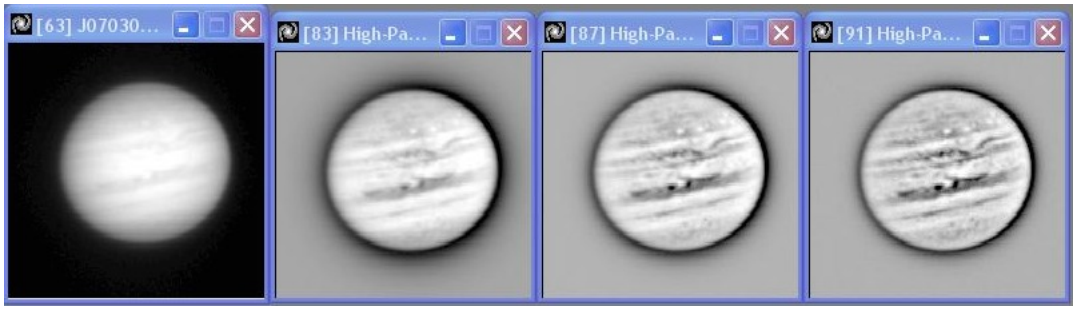


Possiamo eseguire ragionamenti analoghi alla versione *lowpass*. In virtú di come é stato definito, ovvero

$$H_{BHP}(u,v) = 1 - H_{BLP}(u,v)$$

quando si hanno punti (u,v) vicini al centro dello spazio delle frequenze si ha $d(u,v) << d_0$ e quindi il valore di $H_{BLP}(u,v)$ é molto vicino a 1 e di conseguenza $H_{BHP}(u,v) \longrightarrow 0$ col risultato di eliminare le basse frequenza. Allontanandoci radialmente dal centro invece, avevamo che $H_{BLP}(u,v) \longrightarrow 0$, quindi $H_{BHP}(u,v) \longrightarrow 1$ mantenendo le frequenze alte.

Esempio 4.5. Questa volta all'immagine del pianeta Giove applichiamo un *filtro Butterworth highpass*. La figura sottostante mostra i risultati ottenuti con ordine variabile $n = 4, 8, 12$:



Il filtro in questo caso mi blocca le frequenze piú basse eliminando le regioni smussate dell'immagine ed enfatizzandone invece i dettagli. Le caratteristiche fascie orizzontali del pianeta, prima poco visibili risultano invece ora ben marcate.

4.6.3 Filtri bandpass e bandstop

Questo tipo di filtri, come abbiamo giá detto, bloccano tutte le frequenze entro un determinato *range* di valori di frequenza (*bandstop*) o bloccano tutte le frequenze al di fuori del *range* (*bandpass*.

Nel caso di *filtro Gaussiano bandstop* avremo

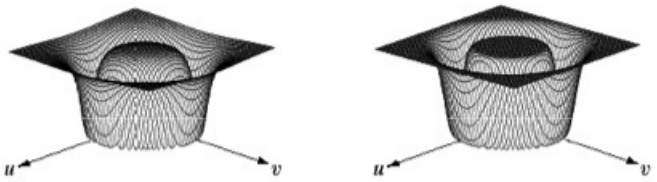
$$H_{GBS}(u,v) = 1 - e^{\left[-\frac{1}{2}\left(\frac{d^2(u,v)-d_0^2}{d_R d(u,v)}\right)^2\right]}$$

dove d_R rappresenta l'ampiezza del *range* che voglio bloccare.

Nel caso di *filtro Butterworth bandstop* avremo

$$H_{BBs}(u,v) = \frac{1}{1 + \left[\frac{d_R d(u,v)}{d^2(u,v)-d_0^2}\right]^{2n}}$$

la figura sottostante mostra la forma dei due filtri:



Analogamente ai casi precedenti, le versioni *bandpass* di entrambi i filtri possono essere trovate dalla relazione $H_{bandpass} = 1 - H_{bandstop}$.

Capitolo 5

Deconvoluzione

La deconvoluzione é un procedimento che tenta di ricostruire una immagine che é stata degradata ad opera di diversi fattori. Le immagini astronomiche sono un classico esempio. Infatti per il modo in cui sono ottenute esse sono soggette a diversi agenti che ne modificano la sotanza. Per esempio le immagini ottenute da un telescopio a terra sono soggette a fenomeni di interferenza dovuti alle turbolenze atmosferiche. Problemi di degradazione si riscontrano anche negli stessi processi di acquisizione, dovuti a instabilitá meccaniche, vibrazioni dello strumento. Ma anche lo stesso telescopio orbitale *Hubble* al momento del suo varo nel 1990 ha dato problemi nelle immagini trasmesse a terra a causa della curvatura errata di uno degli specchi riflettori che trasmettono i segnali luminosi, problema che ha causato perdite di parecchi milioni di euro e ha rischiato di compromettere tutta la missione.
Tutti questi fattori contribuiscono a far si che l'immagine che otteniamo non sia quella "reale" ma una immagine perturbata da quello che nell'ambiente dell'analisi di immagini viene chiamato *rumore*.
In termini matematici possiamo esprimere il concetto come segue

$$i(x, y) = (s * o)(x, y) + n(x, y) \tag{5.1}$$

Dove $i(x, y)$ rappresenta l'immagine ottenuta attraverso un apposito strumento di rilevazione, $o(x, y)$ rappresenta l'immagine "reale", $s(x, y)$ é una funzione, chiamata *point spread function (PSF)*, che rappresenta l'insieme di

tutte le interferenze elencate prima, e $n(x, y)$ rappresenta un rumore aggiuntivo casuale. L'operazione $*$ é la convoluzione.

A prima vista il problema sembrerebbe di facile risoluzione. Se infatti conoscessimo s ed n, ricordando quanto detto nel capitolo precedente e indicando con le lettere maiuscole le rispettive trasformate di Fourier, dall'equazione 5.1 avremmo:

$$i(x, y) - n(x, y) = (s * o)(x, y)$$

$$I(u, v) - N(u, v) = S(u, v)O(u, v)$$

da cui segue

$$O(u, v) = \frac{I(u, v) - N(u, v)}{S(u, v)}$$

e quindi potremmo ricavare $o(x, y)$ attraverso la trasformata inversa. Purtroppo nei casi reali di applicazione, la funzione $s(x, y)$ non é nota, o almeno nota per intero, e cosí pure per il rumore casuale $n(x, y)$. Va detto peró che generalmente gli astronomi sono in grado di approssimare abbastanza bene la funzione $s(x, y)$. Infatti le immagini astronomiche generalmente sono piene di sorgenti luminose, e restringendo la zona di studio attorno ad esse (rappresentate in genere da un numero ridotto di pixel isolati) si é in grado di determinare una buona approssimazione della PSF, ovvero $s(x, y)$. La deconvoluzione quindi é uno di quei problemi che i matematici chiamano "mal-posto", perché bastano piccole perturbazioni per causare notevoli cambiamenti nella soluzione. Il punto chiave della deconvoluzione non é ricercare la *unica* immagine originale, ma quella *statisticamente accettabile*. Di algoritmi deconvolutivi ne esistono tantissimi, ed ognuno di essi come molteplici varianti.

Se il rumore $n(x)$ é modellizzato come una distribuzione Gaussiana di densitá

$$\rho_G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

che rappresenta come una buona approssimazione le normali interferenze ambientali casuali, oppure come una distribuzione di Poisson di densitá

$$\rho_P(x) = e^{-x}\frac{x^n}{n!}$$

che rappresenta il tipico rumore prodotto da dispositivi elettronici come i nostri strumenti di acquisizione, allora si possono usare dei procedimenti iterativi per fornire la stima migliore possibile di $o(x, y)$. Tra i principali algoritmi iterativi per stimare una immagine usati in campo astronomico vi sono l'algoritmo di Van Cittert e l'algoritmo di Richardson-Lucy. Nella trattazione seguente per non abusare di notazione considereremo il caso unidimensionale, ma tutto ció che diremo é assolutamente lo stesso anche nel caso bidimensionale.

5.1 Algoritmo di Van Cittert

L'algoritmo deconvolutivo di Van Cittert venne proposto per la prima volta da P.H. Van Cittert in un articolo del 1931. Esso venne poi migliorato successivamente da Landweber e Bialy. Nella notazione introdotta predecentemente, l'idea di Van Cittert é di considerare l'immagine alterata $i(x)$ come una prima buona approssimazione dell'immagine reale $o(x)$, cioé di porre

$$o^{(0)}(x) = i(x) \tag{5.2}$$

L'idea puó sembrare strana, ma in assenza di algoritmi deconvolutivi gli astronomi usavano le immagini e i dati ottenuti come fossero quelli reali. A questo punto si considera una approssimazione convoluta di $i(x)$ data da

$$i^{(0)}(x) = s(x) * i(x) = s(x) * o^{(0)}(x)$$

cioé pensiamo $i^{(0)}(x)$ come la nostra immagine alterata se $o^{(0)}(x)$ fosse la nostra immagine reale. Il passaggio sopra sembra illogico, in quanto noi abbiamo giá la nostra immagine alterata $i(x)$, ma la differenza $i(x) - i^{(0)}(x)$, cioé l'errore nella immagine alterata, é in qualche modo collegato alla differenza $o(x) - o^{(0)}(x)$, cioé all'errore nella immagine reale. L'idea di Van

Cittert fu quella di applicare questo errore come una correzione all'immagine $o(x)$ e ottenere

$$o^{(1)}(x) = o^{(0)}(x) + (i(x) - i^{(0)}(x)) = o^{(0)}(x) + (i(x) - s(x) * o^{(0)}(x))$$

Iterando il procedimento si ottiene l'espressione ricorsiva dell'algoritmo di Van Cittert data da

$$o^{(k+1)}(x) = o^{(k)}(x) + [i(x) - s(x) * o^{(k)}(x)] \tag{5.3}$$

dove $o^{(k+1)}(x)$ é la $(k+1)$-esima stima di $o(x)$.

Alle volte l'equazione 5.3 si puó trovare nella forma:

$$o^{(k+1)}(x) = o^{(k)}(x) + c[i(x) - s(x) * o^{(k)}(x)] \tag{5.4}$$

dove c é un parametro detto *parametro di distensione*.

Uno degli svantaggi dell'algoritmo di Van Cittert é che, facendo uso di addizioni, alle volte possono prodursi risultati privi di senso, come per esempio valori di pixel negativi. A volte si ovvia a ció usando l'algoritmo di Van Cittert a *restrizione positiva*:

$$o^{(k+1)}(x) = a(x)o^{(k)}(x) + c[i(x) - s(x) * a(x)o^{(k)}(x)] \tag{5.5}$$

dove

$$a(x) = \begin{cases} 1 & o^{(k)}(x) \geq 0 \\ 0 & o^{(k)}(x) \leq 0 \end{cases}$$

5.1.1 Convergenza dell'algoritmo di Van Cittert

Come prima indichiamo con le lettere maiuscole le rispettive trasformate di Fourier delle funzioni in gioco. Applicando il teorema 4.2 e la linearitá della trasformata all'equazione 5.3 otteniamo:

$$\begin{aligned} O^{(k+1)}(u) &= O^{(k)}(u) + [I(u) - S(u)O^{(k)}(u)] = \\ &= O^{(k)}(u)[1 - S(u)] + I(u) \end{aligned}$$

Ricordando che $O^{(0)}(u) = I(u)$ (da 5.2), possiamo ricavare il termine k-esimo $O^{(k)}(u)$ della stima nello spazio di Fourier:

$$\begin{aligned}
O^{(k)}(u) &= O^{(k-1)}(u)[1 - S(u)] + I(u) = \\
&= (O^{(k-2)}(u)[1 - S(u)] + I(u))[1 - S(u)] + I(u) = \\
&= O^{(k-2)}(u)[1 - S(u)]^2 + I(u)[1 - S(u)] + I(u) = \\
&= (O^{(k-3)}(u)[1 - S(u)] + I(u))[1 - S(u)]^2 + \\
&\quad + I(u)[1 - S(u)] + I(u) = \\
&= O^{(k-3)}(u)[1 - S(u)]^3 + I(u)[1 - S(u)]^2 + \\
&\quad + I(u)[1 - S(u)] + I(u) = \\
&= \dots \\
&= O^{(0)}(u)[1 - S(u)]^k + I(u)[1 - S(u)]^{k-1} + \dots \\
&\quad \dots + I(u)[1 - S(u)] + I(u) = \\
&= I(u) = \sum_{m=0}^{k} [1 - S(u)]^m
\end{aligned}$$

Usando ora l'dentitá

$$\sum_{m=0}^{k} \xi^m = \frac{1 - \xi^{k+1}}{1 - \xi}$$

e ponendo $\xi = 1 - S(u)$ otteniamo

$$O^k(u) = \frac{I(u)}{S(u)} B(u) \tag{5.6}$$

dove abbiamo posto $B(u) = 1 - [1 - S(u)]^{k+1}$.

Quindi per $|1 - S(u)| < 1$ si ha che

$$[1 - S(u)]^{k+1} \xrightarrow[k \to +\infty]{} 0$$

e di conseguenza

$$B(u) \xrightarrow[k \to +\infty]{} 1$$

Quindi con un numero di iterazioni sempre crescente

$$O^{(k)}(u) \longrightarrow \frac{I(u)}{S(u)}$$

ovvero l'algoritmo converge e posso quindi trovare una buona approssimazione dell'immagine reale operando la trasformata inversa.

Problemi nell'esecuzione dell'algoritmo si potrebbero avere per $S(u) \longrightarrow 0$ in modulo, infatti applicando lo sviluppo di Taylor

$$(1-x)^{\alpha} = 1 - \alpha x + \wr(x)$$

quindi

$$B(u) = 1 - [1 - S(u)]^{k+1} \simeq 1 - (1 - (k+1)S(u)) = (k+1)S(u)$$

e sostituendo nella 5.6 si ha

$$\begin{aligned} O^{(k)}(u) &= \frac{I(u)}{S(u)}(k+1)S(u) = (k+1)I(u) = \\ &= (k+1)[S(u)O(u) + N(u)] = (k+1)N(u) \end{aligned}$$

perché $S(u) \longrightarrow 0$. Questo mi dice che ad ogni iterazione il rumore cresce linearmente (rispetto a k), e l'algoritmo perde di significato.

Possiamo aggiungere ancora qualcosa riguardo la condizione di convergenza. Avevamo trovato che per $k \to +\infty$ l'algoritmo converge se

$$|1 - S(u)| < 1 \tag{5.7}$$

che equivale a dire

$$0 < Re[S(u)] < 2 \tag{5.8}$$

Cerchiamo ora di trovare una condizione anche su $s(x)$.

Osservazione 5.1. Esiste un risultato dell'analisi, che quí non dimostriamo, il quale afferma che se $f(x)$ é una funzione reale, allora $f(x)$ si puó scrivere come somma di due funzioni $f_p(x)$ e$f_d(x)$ rispettivamente pari e dispari, cioé

$$f(x) = f_p(x) + f_d(x)$$

Costruiamo la trasformata di Fourier di $f(x)$:

$$\begin{aligned} F(u) &= \int_{-\infty}^{+\infty} f(x)e^{-2\pi iux}dx = \\ &= \int_{-\infty}^{+\infty} f(x)cos(2\pi ux)dx - i\int_{-\infty}^{+\infty} f(x)sin(2\pi ux)dx \end{aligned}$$

sostituendo $f(x) \to f_p(x) + f_d(x)$ otteniamo

$$\begin{aligned} F(u) &= \int_{-\infty}^{+\infty} f_p(x)cos(2\pi ux)dx + \int_{-\infty}^{+\infty} f_d(x)cos(2\pi ux)dx + \dots \\ &\quad \dots - i\int_{-\infty}^{+\infty} f_p(x)sin(2\pi ux)dx - i\int_{-\infty}^{+\infty} f_d(x)sin(2\pi ux)dx \end{aligned}$$

Poiché il prodotto di una funzione pari per una dispari é una funzione dispari, allora il secondo e il terzo integrale sono nulli perché integrali di funzioni dispari su un intervallo simmetrico. Quindi

$$F(u) = \int_{-\infty}^{+\infty} f_p(x)cos(2\pi ux)dx - i\int_{-\infty}^{+\infty} f_d(x)sin(2\pi ux)dx$$

si verifica facilmente che il primo integrando é parti rispetto a u, mentre il secondo é dispari rispetto a u, quindi in modo analogo a $f(x)$ posso scrivere

$$F(u) = F_p(u) + F_d(u)$$

Qui si conclude l'osservazione.

Quindi se $s(x)$ é una funzione reale, la condizione 5.8 é equivalente a

$$0 < S_p(u) < 2$$

Consideriamo ora il modulo della trasformata inversa di $S_p(u)$, si ha

$$\begin{aligned} |s_p(x)| &= |\int_{-\infty}^{+\infty} S_p(u)e^{2\pi iux}du\ | = \\ &= |\int_{-\infty}^{+\infty} S_p(u)cos(2\pi ux)du + i\int_{-\infty}^{+\infty} S_p(u)sin(2\pi ux)du| = \\ &= |\int_{-\infty}^{+\infty} S_p(u)cos(2\pi ux)du| \end{aligned}$$

perché il secondo integrale é nullo per quanto visto nell'osservazione 5.1. Di conseguenza si ha che

$$\begin{aligned} |s_p(x)| &= |\int_{-\infty}^{+\infty} S_p(u)cos(2\pi ux)du| < \int_{-\infty}^{+\infty} |S_p(u)cos(2\pi ux)|du = \\ &= \int_{-\infty}^{+\infty} S_p(u)du = s_p(0) \end{aligned}$$

quindi abbiamo

$$|s_p(x)| < s_p(0) \quad \forall x \neq 0 \tag{5.9}$$

ovvero $s(x)$ deve avere un massimo unico in $x = 0$. Questo per esempio escluse PSF con forme come quelle nella figura sottostante

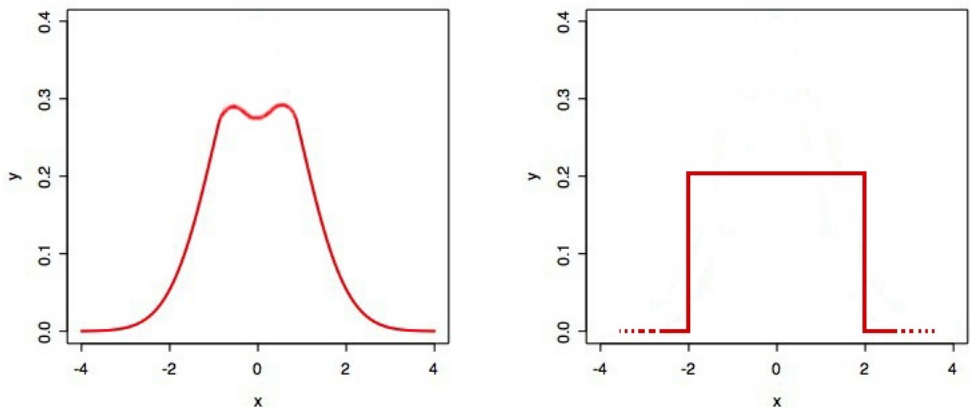


Figura 5.1: *point spread function* non idonee.

Nel caso di algoritmo di Van Cittert con parametro di distensione (*cfr.* 5.4), passando allo spazio di Fourier avremo:

$$\begin{aligned} O^{(k+1)}(u) &= O^{(k)}(u) + c[I(u) - S(u)O^{(k)}(u)] = \\ &= O^{(k)}(u) + [cI(u) - cS(u)O^{(k)}(u)] \end{aligned}$$

e possiamo ricavare nuovamente $B(u)$:

$$\begin{aligned}
O^{(k)}(u) &= O^{(k-1)}(u) + c[I(u) - S(u)O^{(k)}(u)] = \\
&= O^{(k-1)}(u)[1 - cS(u)] + cI(u) = \\
&= \dots \\
&= I(u)[1 - cS(u)]^k + cI(u) \sum_{m=0}^{k-1} [1 - cS(u)]^m = \\
&= I(u)[1 - cS(u)]^k + cI(u) \frac{1 - (1 - cS(u))^k}{cS(u)} = \\
&= \frac{I(u)}{S(u)} \Big(S(u)[1 - cS(u)]^k + 1 - (1 - cS(u))^k \Big) = \\
&= \frac{I(u)}{S(u)} \Big(1 + [1 - cS(u)]^k (S(u) - 1) \Big) = \\
&= \frac{I(u)}{S(u)} \Big(\underbrace{1 - [1 - cS(u)]^k (1 - S(u))}_{B(u)} \Big)
\end{aligned}$$

Alternativamente, cambiando la scelta iniziale $o^{(0)}(x) = i(x)$ con

$$o^{(0)}(x) = ci(x)$$

si puó ottenere una espressione piú semplice di $B(u)$ con calcoli analoghi

$$B(u) = 1 - [1 - cS(u)]^{k+1}$$

da cui discende la nuova condizione

$$|1 - cS(u)| < 1$$

Diciamo per completezza che alle volte si puó anche prendere il parametro c non costante, ma dipendente dal pixel in esame. Questo perché si vuole una convergenza piú lenta (e quindi migliore approsimazione) della iterazione nelle zone maggiormente affette da rumore come per esempio il cielo di sfondo con valori di pixel molto bassi, e una convergenza piú rapida in quelle zone meno affette da rumore come i pixel brillanti che rappresentano le stelle e che hanno quindi una convergenza di base piú lenta.

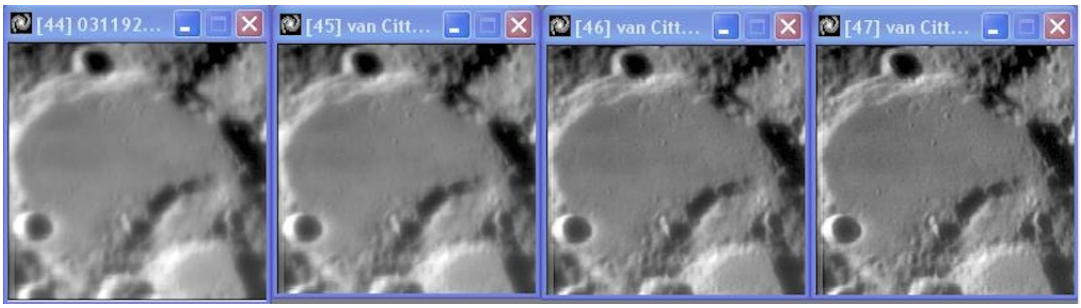
Se indichiamo con p il valore del pixel (x, y) e, con la notazione introdotta precedentemente, p_b e p_n rispettivamente l'endpoint del bianco e del nero, un parametro che agisce allo scopo é dato da

$$c(p) = \begin{cases} 0 & p < p_n \\ \left(sin(\frac{\pi}{2} \frac{p - p_n}{p_b - p_n} \right)^{\gamma} & p_n \leq p \leq p_b \\ 1 & p > p_b \end{cases} \tag{5.10}$$

dove il seno varia tra 0 e 1 perché l'argomento varia tra 0 e $\frac{\pi}{2}$. γ invece é un parametro con $0 < \gamma < 1$.

5.1.2 Esempio

La seguente figura mostra, da sinistra verso destra, l'immagine originale di una porzione di suolo marziano e l'immagine deconvoluta usando l'algoritmo di Van Cittert con un numero crescente di iterazioni pari a 5, 15 e 25. Al crescere del numero di iterazioni l'immagine deconvoluta si approssima sempre piú all'immagine reale, come mostrato dall'incremento sempre maggiore di dettagli nell'immagine.



5.2 Algoritmo di Richardson-Lucy

Questo metodo iterativo, indicato talvolta con *metodo RL*, fu descritto per la prima volta da William H. Richardson in un articolo del 1972 e in

maniera indipendente due anni dopo da L.B. Lucy in un articolo per *Astronomical Journal*. Si tratta di un algoritmo molto simile a quello di Van Cittert, ma che fa uso di moltiplicazioni al posto delle addizioni.

Il punto di partenza é lo stesso, si suppone cioé che la stima iniziale dell'immagine reale sia $o^{(0)}(x) = i(x)$. L'algoritmo iterativo é descritto dalla seguente equazione ricorsiva:

$$o^{(k+1)}(x) = o^{(k)}(x)\left[s(-x) * \frac{i(x)}{s(x) * o^{(k)}(x)}\right]\frac{1}{s(x) * u(x)} \tag{5.11}$$

dove $u(x)$ é una funzione identicamente uguale a 1, oppure in forma integrale:

$$o^{(k+1)}(x) = o^{(k)}(x)\frac{\int[\frac{i(x)}{\int s(x)o^{(k)}(x)dx}]dx}{\int s(x)dx} \tag{5.12}$$

Nel caso di PSF invarianti spazialmente, cioé $s(-x) = s(x)$, l'equazione 5.11 diventa

$$o^{(k+1)}(x) = o^{(k)}(x)\frac{\frac{i(x)}{s(x)*o^{(k)}(x)} * s(-x)}{s(-x) * u(x)} \tag{5.13}$$

In questo caso il calcolo puó essere accelerato usando la FFT. Ricordiamo che la trasformata di Fourier $S(-u)$ di $s(-x)$ é la coniugata complessa della trasformata di $s(x)$, indicata con $S^*(u)$. Indicando con $F(\ )$ la FFT e con $F^{-1}(\ )$ la FFT inversa, possiamo ricavare l'iterazione $(k+1)$-esima come

$$o^{(k+1)}(x) = o^{(k)}(x)\frac{F^{-1}\left[F\left(\frac{i(x)}{F^{-1}(S(u)O^{(k)}(u))}\right)S^*(u)\right]}{F^{-1}\Big(S^*(u)U(u)\Big)} \tag{5.14}$$

Ma in generale, dato che non esiste nessuna richiesta che la PSF sia invariante spazialmente, il calcolo dell'algoritmo RL viene eseguito come prodotti tra matrici, risultando quindi piú lento.

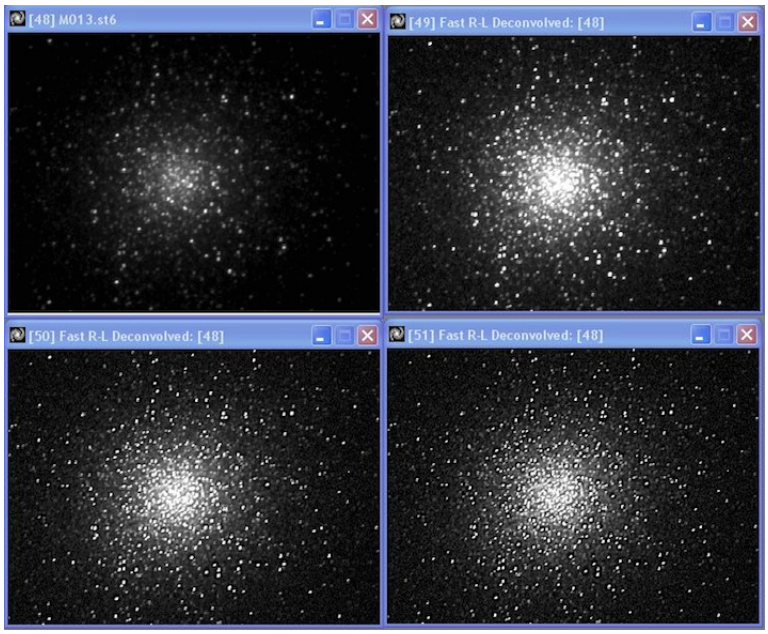
Al contrario dell'algoritmo di Van Cittert che faceva uso di addizioni, l'algoritmo RL facendo uso solamente di moltiplicazioni gode di una buona proprietá. Se infatti $s(x)$, $n(x)$ e $o^{(k)}(x)$ sono non negativi, allora anche $o^{(k+1)}(x)$

lo sará.

La convergenza dell'algoritmo RL é stata verificata empiricamente, e la sua dimostrazione é stata provata nel 1982 da Shepp e Vardi riconducendo il metodo RL a un caso particolare dell'algoritmo EM (*expectation-maximization*) di Dempster *et all*, il quale converge.

5.2.1 Esempio

La figura sottostante mostra l'applicazione dell'algoritmo RL a una immagine di campo stellare. In alto a sinistra l'immagine originale affetta da molto rumore, e successivamente in sequenza le immagini ottenute con un numero crescente di iterazioni pari a 5, 15 e 25:



Bibliografia

[1] Berry, R., Burnell, J., *The handbook of astronomical image processing*, Richmond, Va. 2000.

[2] Chellappa, R., Sawchuk, A., *Digital image processing and analysis*, Silver Spring, MD 1985.

[3] Gonzales, R., Wintz, P., *Digital image processing*, Reading, Mass. 1987.

[4] Jager, C., Nieuwenhuijzen, H., *Image processing techniques in astronomy*, Dordrecht, Holland 1975.

[5] Jansson, P., *Deconvolution of images and spectra*, San Diego 1997.

[6] Sedmak, G. -al., *International workshop on image processing in astronomy*, [Trieste] 1979.

[7] Starck, J., Murtagh, F., *Astronomical image and data analysis*, Berlin [u.a] 2002.

[8] Pantin, E., Starck, J., Murtagh, F., *Deconvolution and blind deconvolution in astronomy*, URL: `http://jstarck.free.fr/Blind07.pdf`

[9] Nicolini, M., Facchini, M., *Algoritmo di Larson-sekanina: applicazioni sulla chioma della Hyakutake*, URL: `http://comete.uai.it/fotometria`