Laboratorio di matematica computazionale
Prima parte del corso: Elaborazione di immagini digitali

D.A. Bini

18 marzo 2015

1 Introduzione

In campo fotografico e cinematografico la tecnologia digitale ha da tempo soppiantato la vecchia tecnologia analogica. Macchine fotografiche digitali vengono comunemente e diffusamente utilizzate a livello amatoriale e professionale.

La disponibilità dell'informazione fotografica data in termini numerici permette di effettuare in modo molto agevole elaborazioni di immagini che una volta erano solo consentite in camera oscura. Inoltre la possibilità di creare e manipolare numericamente immagini digitali permette di rappresentare visivamente oggetti matematici in modo semplice con la possibilità di evidenziarne proprietà non individuabili altrimenti.

In questa parte del laboratorio, a cui si riferisce questa raccolta di appunti, richiamiamo i principali standard di rappresentazione di immagini digitali e li usiamo per svolgere alcune operazioni. Gli appunti sono organizzati nel seguente modo. Nella sezione 1 ricordiamo i principali comandi di Octave per gestire immagini digitali. Infatti useremo questo linguaggio per le nostre sperimentazioni computazionali. Nella sezione 2 andiamo ad evidenziare proprietà dei numeri primi costruendo la spirale di Ulam `http://en.wikipedia.org/wiki/Ulam_spiral`, e come esercizio applichiamo questa metodologia alla costruzione del triangolo di Klauber, del Boustrophedon `http://haslock.com/Boustrophedon.html`, e per tracciare altre figure legate alla numerazione dei punti di un reticolo bidimensionale. Nella sezione 3 ci interesseremo alla costruzioni di insiemi frattali che intervengono nello studio di successioni di numeri complessi definite per ricorrenza. Evidenzieremo bacini di attrazione, la figura di Mandelbrot, gli insiemi di Julia. Nella sezione 4 ci occupiamo di deformazioni di immagini ottenute spostando i singoli pixel con trasformazioni di vario tipo. In particolare esamineremo gli effetti della mappa di Arnold `http://it.wikipedia.org/wiki/Gatto_di_Arnold` ed altre trasformazioni similari dal toro in sé; visualizzeremo alcune trasformazioni ottenute attraverso semplici funzioni di variabile complessa, e trasformazioni legate a semplici proiezioni e riflessioni. Nella sezione 5 ci occupiamo del filtraggio di

immagini mediante la trasformata discreta di Fourier e mediante l'uso di semplici filtri FIR. Infine nella sezione 6 descriviamo il problema del restauro di immagini sfocate e affette da rumore che si può realizzare mediante la risoluzione di sistemi lineari di grosse dimensioni. In ogni sezione vengono proposti alcuni esercizi di cui si richiede la risoluzione.

1.1 Preliminari

Un'immagine digitale in bianco nero può essere descritta da una matrice $m \times n$ di *pixel* (contrazione di picture element). Il generico pixel $p(i, j)$, il cui valore numerico può essere intero o reale, fornisce la luminosità del punto di coordinate (i, j) dell'immagine. Esistono degli standard di rappresentazione di immagini digitali. Forse il più elementare è il *formato PGM (Portable Gray Map)*.

Nella convenzione PGM il file che contiene le informazioni sull'immagine è organizzato nel modo seguente:

```
P2
# eventuali commenti
numero_colonne, numero_righe
massima_luminosita'
p(1,1)
p(1,2)
.
.
p(1,n)
p(2,1)
p(2,2)
.
.
```

dove:
• `P2`, detto numero magico, indica al sistema che visualizzerà il file sullo schermo che si tratta di una immagine bianco nero nel formato PGM;
• `eventuali_commenti` sono commenti arbitrari preceduti dal carattere `#`;
• `numero_colonne` e `numero_righe` sono due interi che determinano le dimensioni dell'immagine (matrice dei pixel);
• `massima_luminosita'` è un intero compreso tra 1 e 65536, che determina il valore di massima luminosità, cioè nel nostro caso il valore numerico del bianco (generalmente si usa 255), mentre il valore numerico del nero è 0;
• `p(i,j)` sono i valori numerici dei pixel (compresi fra 0 e `massima_luminosita'`) relativi alla posizione (i, j). Essi sono ordinati per righe.

Maggiori informazioni su questo e altri formati che descriveremo tra poco si trovano nel sito `http://netpbm.sourceforge.net/doc/pgm.html`.

Esistono diversi programmi per visualizzare un'immagine sullo schermo di un computer, il più semplice è `display`. Un pacchetto che permette di fare elaborazioni varie oltre che visualizzare una immagine è `gimp`. Ad esempio,

se `immagine.pgm` è il nome di un file che contiene un'immagine codificata nel formato PGM, allora il comando

```
display immagine.pgm
```

mostra sullo schermo l'immagine corrispondente al file, mentre il comando

```
gimp immagine.pgm
```

lancia l'applicazione gimp visualizzando sullo schermo il contenuto di `immagine.pgm`.

Questo che segue è un sempio di file PGM di una immagine formata da 4×4 pixel

```
P2
# esempio semplice
4,4
255
0
0
0
0
0
255
255
0
0
255
255
0
0
0
0
0
```

La stessa immagine poteva essere memorizzata come

```
P2
# esempio semplice
4,4
255
0  0   0  0
0 255 255 0
0 255 255 0
0  0   0  0
```

Un file PGM può essere memorizzato nella modalità ASCII o nella modalità RAW. Nella modalità ASCII il valore del generico pixel viene scritto nella sua rappresentazione decimale con caratteri ASCII. Ad esempio. se il valore del pixel è 123, vengono scritte le tre cifre 1,2 e 3, consumando quindi tre byte di memoria, uno per ciascuna cifra, più altri tre byte per gli spazi di separazione. Nella rappresentazione RAW, il pixel il cui valore è 123 viene memorizzato con

il singolo byte il cui valore è 123. In questo modo un solo byte è sufficiente per ciascun pixel. Nella rappresentazione RAW la stringa P2 va sostituita con P5. La rappresentazione ASCII è più comoda perché; permette di essere visualizzata e manipolata più facilmente. Ha lo svantaggio che è più ingombrante. Noi useremo la rappresentazione ASCII. In un file RAW la parte iniziale del file, detta *header*, che contiene il numero magico, i commenti, le dimensioni e la massima intensità luminosa, sono scritti in formato ASCII.

L'applicazione Gimp di Linux permette di aprire un file immagine in un *qualsiasi* formato e salvarla in un altro formato diverso. Ad esempio, una fotografia digitale salvata come file jpeg può essere aperta da Gimp e salvata come file PGM.

Un altro modo di convertire immagini da un formato ad un altro è dato dal comando `convert` presente nelle distribuzioni di linux. Ad esempio,

```
convert foto.jpg foto.pgm
```

trasforma il file foto.jpg, contenente una fotografia nel formato jpeg, nel file foto.pgm contenente la stessa immagine ma in b/n nel formato pgm.

Un altro formato di rappresentazione di immagini è il formato *PBM (Portable Bit Map)*. È un formato analogo al PGM dove il numero magico è P1 in modalità ASCII e P4 in modalità raw, e dove i pixel possono assumere solo due valori 0 e 1 (nero e bianco). Manca quindi il valore della massima intensità del bianco.

Per immagini a colori viene usato il formato *PPM (Portable Pixel Map)*. Infatti, una immagine a colori può essere memorizzata mediante la modalità RGB (Red, Green, Blue). In questo modo ad ogni pixel viene assegnata una terna di numeri r, g, b, dove r fornisce l'intensità del rosso, g l'intensità del verde e b l'intensità del blu. Il file che viene costruito nel formato PPM è organizzato come segue

```
P3
# eventuali commenti
numero_colonne, numero_righe
massima_luminosita'
r(1,1) g(1,1) b(1,1)
r(1,2) g(1,2) b(1,2)
.
.
.
```

Ciò che cambia rispetto ad un file b/n è la prima riga, che contiene P3 a indicare che è un file a colori nella modalità RGB, e la presenza delle terne di pixel (un valore numerico per ogni colore) anziché dei singoli valori. Le terne dei valori rgb possono non stare sulla stessa riga ma essere distribuite su righe consecutive. Nella codifica RAW la stringa P3 viene sostituita da P6. Maggior dettagli si trovano sempre nello stesso sito `http://netpbm.sourceforge.net/doc/ppm.html`.

P2	file PGM, modalità ASCII
P5	file PGM, modalità RAW
P3	file PPM, modalità ASCII
P6	file PPM, modalità RAW

Tabella 1: Lista dei "numeri magici"

Il formato PNM, acronimo di *Portable aNy Map*, è giusto un'astrazione dei formati PBM, PGM, and PPM. Cioè il termine PNM si riferisce collettivamente a PBM, PGM, e PPM.

La tavola dei numeri magici che contraddistingue i vari formati in modalità ASCII e RAW è riportata nella tabella 1.

1.2 Immagini in Octave

Il linguaggio di programmazione Octave permette in modo semplice di trasformare una matrice di numeri in una immagine usando una logica simile a quella dello standard PNM. Se una matrice A ha dimensioni $m \times n \times 3$ allora le tre "fette" $A(:,:,1)$, $A(:,:,2)$, $A(:,:,3)$ rappresentano le intensità dei tre colori RGB di una immagine formata da $m \times n$ pixel. Se la matrice A è invece $m \times n$ i suoi elementi $a_{i,j}$ determinano il colore del pixel in posizione $a_{i,j}$ in base alla *mappa dei colori*.

La mappa dei colori è costituita da una matrice di k righe e tre colonne. Per default $k = 64$. Gli elementi di questa matrice sono numeri compresi tra 0 e 1, ciascuna riga rappresenta un colore composto da quantità di rosso, verde e blu date dai tre valori presenti rispettivamente sulla prima, seconda e terza colonna. Ad esempio una mappa del tipo

```
[1 0 0; 0 1 0; 0 0 1];
```

fornisce tre colori, nell'ordine: il rosso, il verde e il blu. Per default la mappa dei colori contiene 64 tonalità di colore. Per creare una mappa di grigi basta assegnare uguali quantità di rosso, verde e blu a ciascun colore. Una mappa di 64 livelli di grigio è quindi data da

```
[0:1/63:1]'*[1 1 1];
```

Per poter cambiare la mappa dei colori basta dare il comando

```
colormap(nuovamappa);
```

dove `nuovamappa` è la matrice di tre colonne con la nuova mappa dei colori.

Ad esempio per creare una mappa `Mp` che abbia i soli tre colori puri: rosso, verde e blu, basta scrivere

```
Mp = eye(3);
colormap(Mp);
```

oppure più direttamente `colormap(eye(3));` Per tornare alla mappa default di 64 colori basta scrivere

```
colormap('default');
```

Per avere la mappa dei grigi basta scrivere `colormap('gray');` Per assegnare la mappa di colori corrente ad una matrice `Cm` basta scrivere

```
Cm = colormap;
```

Per poter visualizzare un'immagine in octave ci sono diversi modi. Essi possono differire leggermente tra loro a seconda della versione di Octave usata. Sulle macchine dell'aula M c'è la versione 3.8.1. Sulle macchine dell'aula 4 del dipartimento di matematica c'è la versione 3.2.4. Si suggerisce l'uso del comando `help` per avere informazioni sulla sintassi e sul funzionamento di un qualsiasi comando Octave. Qui descriviamo due dei comandi presenti nella versione 3.2.4.

Il comando `image`

Se la variabile reale `A` contiene una matrice di m righe e n colonne con valori numerici `A(i,j)`, il comando

```
image(A);
```

mostra sullo schermo una immagine formata da $m \times n$ pixel dove la luminosità e il colore del pixel di posizione (i, j) sono determinati dal valore di $A(i, j)$ in relazione alla mappa dei colori usata. Più precisamente se la mappa dei colori è formata da k righe, il colore del pixel di posto i, j è dato dal colore della mappa dei colori presente sulla q-esima riga di `mappa` dove q è l'intero più vicino a $A(i, j)$ nell'insieme $\{1, 2, \ldots, k\}$. Quindi, in particolare, valori di $A(i, j)$ fuori dal segmento consentito corrispondono al primo e all'ultimo colore della mappa a seconda della parte del segmento in cui cade $A(i, j)$.

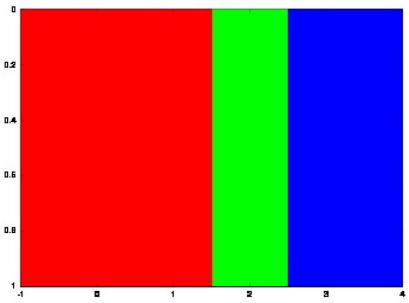
Nello stesso comando si possono specificare gli estremi dell'asse x e dell'asse y in modo che nell'immagine compaiano le coordinate desiderate. Per far questo si usa `image` con la sintassi

```
image([xmin xmax], [ymin ymax], A);
```

Ad esempio, il comando

```
colormap(eye(3));
A = [-1 0 1 2 3 4 ; -1 0 1 2 3 4];
image([-1 4], [0 1], A);
```

genera l'immagine



Si noti che tutti i valori degli elementi di A minori di $1/2$ corrispondono al colore rosso (il primo della mappa dei colori) mentre i valori maggiori o uguali a a 2.5 corrispondono al colore blu (il terzo della mappa dei colori).

Il comando `image` può prendere in input una matrice A di dimensioni $m \times n \times 3$. In questo caso le tre "fette" $A(:,:,1)$, $A(:,:,2)$, $A(:,:,3)$ rappresentano le intensità di rosso, verde e blu, dove il valore 0 indica l'assenza di luminosità mentre il livello di massima luminosità varia a seconda del tipo numerico della matrice A. Più precisamente, se A è costituita da numeri interi senza segno rappresentati con 8 bit (numeri interi tra 0 e 255) allora il livello di massima luminosità è 255. Numeri di questo tipo vengono indicati come `uint8` cioè `unsigned integer`. Se A è costituita da numeri interi senza segno rappresentati con 16 bit (numeri interi tra 0 e 65535, cioè `uint16`) allora il livello di massima luminosità è 65535. Se invece A è una matrice di double (numeri floating point in doppia precisione), allora il livello di massima luminosità è 1. Gli eventuali valori minori di 0 vengono trattati come 0, gli eventuali valori maggiori del livello di massima luminosità vengono trattati come massima luminosità.

Sono utili i comandi `double`, `uint8`, `uint16` che permettono di trasformare una variabile da un tipo all'altro.

Il comando `imagesc`

Il comando `imagesc(A)` ha lo stesso effetto e la stessa sintassi di `image`. L'unica differenza è che se A è una matrice $m \times n$ i suoi elementi vengono scalati in modo che il minimo e il massimo corrispondano al minimo e massimo indice della mappa dei colori.

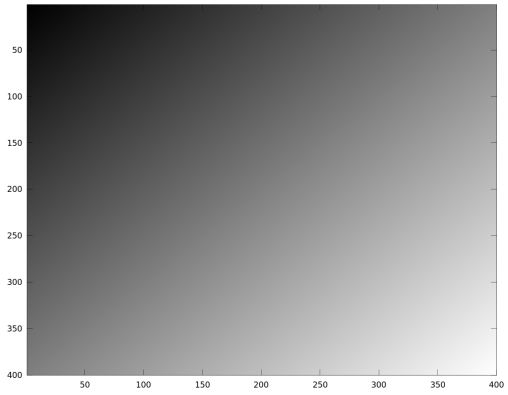
Esempio 1 Il comando `image(rand(20,20,3));` visualizza una immagine a colori 20×20 i cui pixel hanno una quantità casuale di rosso, verde e blu.

Il programma Octave riportato nel listato 1 costruisce un'immagine $n \times n$ con valori di grigio che sfumano da nero a bianco secondo la regola $A(i,j) = i+j-2$, e dove 0 corrisponde a nero e $2n$ a bianco.

L'immagine che si ottiene digitando `grigi(400)` è la seguente:

Listing 1: Immagine con valori di grigio che sfumano da nero a bianco

```
function a = grigi(n)
   a = zeros(n);
   for i = 1 : n
      for j = 1 : n
         a(i,j) = i + j;
      endfor
   endfor
   colormap([0:1/(2*n-1):1]'*[1 1 1]);
   imagesc(a);
endfunction
```



Provate a lanciare il comando `grigi(400)` e poi guardate come cambia la figura col comando

```
mappa = rand(400,3); colormap(mappa);
```

Provate ancora a sostituire l'istruzione `a(i,j) = i + j - 2;` con l'istruzione `a(i,j) = i * j;` e ancora guardate come cambia l'immagine col comando `colormap(mappa)`.

Provate separatamente queste altre scelte e motivate la variazione dell'immagine

```
a(i,j) = (i + j - 2)*rand;
```

```
a(i,j) = i*j*rand;
```

Osserviamo che la costruzione col doppio ciclo `for` per assegnare i valori `i+j`, oppure `i*j` alla matrice `a` può essere sostituita dai comandi più semplici e più efficienti

```
a = [1:n]'*ones(1,n) + ones(n,1)*[1:n] - 2;
```

e, rispettivamente
`a = [1:n]'*[1:n];`.

Osserviamo che la function `grigi` permette di tracciare le linee di livello della funzione che abbiamo assegnato alla matrice `a(i,j)`. Proviamo ad esempio a considerare la funzione $\cos(x)\cos(y)$ per x e y compresi tra zero e π. Poniamo allora

```
a(i,j) = cos(pi*i/n)*cos(pi*j/n);
```

lanciamo `grigi(200);` e poi `mappa=rand(10,3);colormap(mappa);`. In questo modo vengono evidenziate 10 linee di livello.

Se vogliamo creare un file che contiene i dati di questa immagine nel formato pgm basta inserire nella function precedente i seguenti comandi:

```
fid = fopen('grigi.pgm', 'w');
fprintf(fid,'P2\n');
fprintf(fid, '%d , %d \n', n, n);
fprintf(fid, '255\n');
for i = 1 : n
   for j = 1 : n
      fprintf(fid, '%d\n', a(i,j) );
   endfor
endfor
```

Il comando `fopen` di Octave apre un file per la lettura/scrittura la sintassi è `fid=fopen(nomefile, modo)` dove `nomefile` è una stringa col nome del file che si vuole aprire, `modo` è una stringa che contiene la modalità di apertura, in particolare:

`'w'` write (file di scrittura)

`'r'` read (file di lettura)

La variabile `fid` prende come valore un numero intero che servirà a identificare il file di lettura/scrittura. Infatti, l'istruzione

```
fprintf(fid, 'scrivo %d', n);
```

scrive il valore intero della variabile `n` preceduto dalla parola `scrivo`. La sintassi per i formati di scrittura è la stessa del linguaggio C.

Se invece vogliamo salvare l'immagine come file di tipo jpg è sufficiente dare il comando

```
print -djpg nomefile.jpg
```

L'opzione `-d` che significa `device` può essere seguita da altre specifiche. Ad esempio `-dpng` salva come file di tipo png, `-dpdf` salva come file di tipo pdf. Per la lista completa delle specifiche basta dare il comando `help print`.

Listing 2: Test di primalità

```
function v = primo(p)
   v=1;
   for i = 2 : sqrt(p)
      if mod(p,i)==0
         v = 0;
         break;
      endif
   endfor
endfunction
```

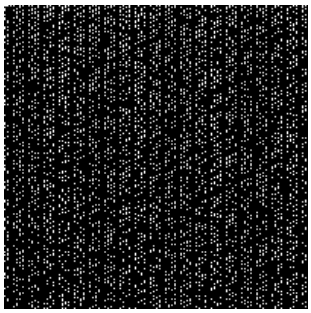
2 Immagini e numeri primi: La spirale di Ulam

Immagini interessanti possono essere costruite per evidenziare proprietà dei numeri primi. Ad esempio, possiamo facilmente costruire una immagine $n \times n$ tale che il pixel di posto (i, j) sia bianco se $n(i-1)+j$ è un numero primo, sia nero altrimenti. Per questo ci occorre una funzione che dato un intero p ci dice se p è primo o no. Questa funzione è riportata nel listato 2

Disponendo di questa funzione possiamo generare l'immagine con la funzione

```
function evidenzia_primi(n)
   imag = zeros(n);
   for i = 1 : n
      for j = 1 : n
         imag(i,j) = primo(n*(i-1)+j);
      endfor
   endfor
   imagesc(imag);
endfunction
```

La figura che si ottiene con $n = 100$ è la seguente



che non è molto informativa. Possiamo scegliere valori diversi di n, ad esempio $n = 119$, $n = 120$, $n = 121$. Si ottengono delle immagini che hanno configurazioni particolari che ci danno informazioni abbastanza ovvie.

Mentre si ottengono risultati interessanti ordinando i numeri interi nel piano con regole diverse. Un pochino più complicato, ma vale la pena farlo, è numerare i pixel lungo una spirale che si dipana dal centro del quadrato, partendo dal numero 1.

```
37—36—35—34—33—32—31
 |                   |
38  17—16—15—14—13  30
 |   |           |   |
39  18   5— 4— 3  12  29
 |   |   |     |   |   |
40  19   6   1— 2  11  28
 |   |   |         |   |
41  20   7— 8— 9—10  27
 |   |                 |
42  21—22—23—24—25—26
 |
43—44—45—46—47—48—49...
```

e accendere i pixel che corrispondono ai numeri primi. In questo modo si ottiene la *spirale di Ulam* `http://it.wikipedia.org/wiki/Spirale_di_Ulam` riportata in figura 1

Osservate che i numeri primi sembrano addensarsi maggiormente lungo delle rette inclinate di un angolo di 45 gradi. È questo un effetto ottico o è qualcosa che evidenzia una qualche proprietà nascosta? Una discussione su questa domanda si può trovare su Wikipedia `http://en.wikipedia.org/wiki/Ulam_spiral`.

Test rigorosi hanno dimostrato che effettivamente su alcune diagonali la concentrazione di numeri primi è maggiore rispetto ad altre. Questo è legato al fatto che esistono molte costanti a, b, c per le quali la funzione $f(n) = an^2+bn+c$ genera, sostituendo n con una serie di numeri consecutivi, una grande quantità di numeri primi. Ad esempio, $f(n) = n^2 + n + 17$ dà numeri primi per $n = 0, 1, 2, 3, \ldots, 15$.

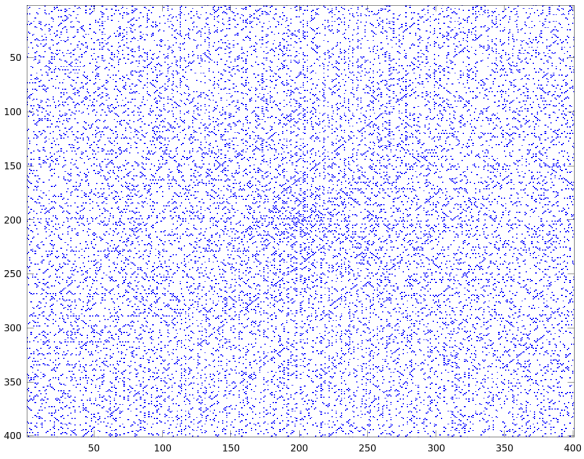
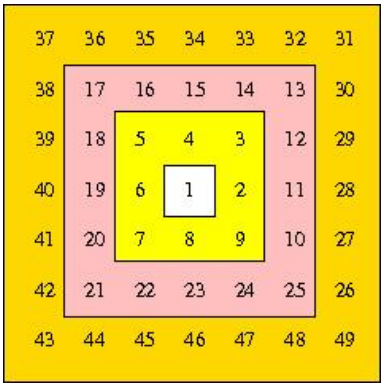


Figura 1: Spirale di Ulam.

Per realizzare questa immagine ci costruiamo prima una function che dispone i numeri naturali in una matrice $(2m+1) \times (2m+1)$ partendo dal centro, cioè dall'elemento di posto (m, m) e muovendosi a spirale che si dipana in senso antiorario compiendo m giri.

Per capire come procedere facciamo riferimento alla seguente figura



Si è colorato il primo giro di giallo, il secondo di rosa e il terzo di oro. Ciascun giro è ripartito in quattro tratti: un tratto a salire, uno di movimento a sinistra, uno a scendere ed uno di movimento a destra.

La function che fa questo è descritta nel listato 3:

Listing 3: Function che numera gli elementi di una matrice con ordinamento “a spirale”.

```
function s = spirale(m)
% Costruisce la matrice s di dimensione 2m+1 che contiene i
% numeri interi positivi a partire dal centro a spirale antioraria
    s = zeros(2*m+1);
    i = m+1; j=m+1;
    k = 1;
    s(i,j) = k;
    for giri = 1:m % conta i giri della spirale
        k = k+1;
        j = j+1;
        s(i,j)=k;
        for t = 1:2*giri-1 % sale
            k = k+1;
            i = i-1;
            s(i,j) = k;
        endfor
        for t = 1:2*giri % sinistra
            k = k+1;
            j = j-1;
            s(i,j) = k;
        endfor
        for t = 1:2*giri % basso
            k = k+1;
            i = i+1;
            s(i,j) = k;
        endfor
        for t = 1:2*giri % destra
            k = k+1;
            j = j+1;
            s(i,j) = k;
        endfor
    endfor
endfunction
```

Listing 4: Costruzione della spirale di Ulam

```
function a = spirale_di_ulam(m)
   a = zeros(2*m + 1);
   s = spirale(m);
   for i = 1 : 2*m + 1
      for j = 1 : 2*m + 1
         if primo(s(i,j))
            a(i,j) = 1;
         endif
      endfor
   endfor
endfunction
```

I quattro tratti in cui abbiamo spezzato la numerazione degli interi lungo la spirale sono evidenziati con commenti nel programma. Osservate che la variabile `giri` conta i giri della spirale intorno al suo centro, e per ciascun giro si percorrono i quattro lati. La variabile `k` contiene il valore dell'intero da depositare nella casella corrente della spirale.

Un programma che genera la spirale di Ulam è riportato nel listato 4

Il linguaggio Octave, essendo interpretato, ha tempi di elaborazione molto elevati. In particolar modo questo succede in presenza di cicli `for` annidati. Usando i programmi scritti sopra siamo in grado di trattare dimensioni relativamente basse e quindi possiamo creare solo immagini piccole. Per poter svolgere elaborazioni a dimensione più ampia occorre implementare questi algoritmi con linguaggi più efficienti tipo il linguaggio C o il Fortran 90. Una alternativa possibile è usare ancora Octave evitando però di usare cicli `for` annidati e utilizzando il più possibile istruzioni vettoriali, cioè istruzioni che anziché operare sulle singole componenti di un vettore o di una matrice agiscono simultaneamente su porzioni del vettore o della matrice stessa. Mostriamo un esempio di ciò nell'implementazione del crivello di Eratostene.

2.1 Il crivello di Eratostene

Il crivello di Eratostene è un metodo per selezionare i numeri primi compresi tra 1 ed n, dove n è un intero positivo assegnato. Il metodo funziona così : dalla lista degli interi da 1 a n si toglie il numero 1, poi tutti i multipli interi di 2, escluso naturalmente 2, poi tutti i multipli interi di 3 escludendo 3, e così via fino ad arrivare a togliere i multipli del più grande intero minore o uguale alla radice quadrata di n. Alla fine rimangono i numeri primi. Questo metodo può essere implementato partendo dal vettore di tutti uni e azzerando le componenti che hanno indice non primo in modo che alla fine il vettore ottenuto conterrà il valore 1 in corrispondenza delle componenti di indice primo e zero altrimenti. Di questo metodo proponiamo due implementazioni una di tipo sequenziale che

Listing 5: Function che implementa il Crivello di Eratostene in modalità sequenziale.

```
function primi = crivello0(n)
% Calcola i numeri primi da 1 a n col crivello di Eratostene
% versione sequenziale
    primi = ones(1,n);
    primi(1) = 0;
    m = round(sqrt(n));
    for i = 2 : m
       for j = 2:n/i
          primi(i*j) = 0;
       endfor
    endfor
endfunction
```

Listing 6: Function che implementa il Crivello di Eratostene in modalità vettoriale. Rispetto alla versione riportata nel listato 5 si risparmia un ciclo for.

```
function primi = crivello(n)
% Calcola i numeri primi da 1 a n col crivello di Eratostene
% versione vettoriale
    primi = ones(1,n);
    m = round(sqrt(n));
    primi(1) = 0;
    for i = 2 : m
       primi(2*i:i:n) = 0;
    endfor
endfunction
```

usa cicli `for`, e quindi più lenta, e l'altra di tipo vettoriale e quindi più veloce. Le due function sono riportate rispettivamente nei listati 5 e 6.

Infatti, il ciclo dato dal comando `for j=2:n/i` può essere sostituito dalla sola istruzione `primi(2*i:i:n)=0;` che azzera tutte le componenti di indice multiplo di i a partire da $2i$.

La spirale di Ulam può allora essere generata in modo più efficiente nel modo riportato nel listato 7.

Una caratteristica importante di Octave è data dall'istruzione `z=c(s);` in cui viene fornita in uscita la matrice il cui elemento di posto (i, j) è dato da $c(s(i, j))$. Cioè il vettore c, inteso come funzione definita sull'insieme $\{1, 2, ..., m\}$ a valori in $\{0, 1\}$, viene composto con la matrice s intesa come applicazione definita su $\{1, 2..., 2n + 1\} \times \{1, 2, ..., 2n + 1\}$ a valori in $\{1, 2, ..., m\}$.

Listing 7: Function che costruisce la spirale di Ulam in modo vettoriale.

```
function z = ulam(n)
% Disegna la spirale di Ulam di dimenione 2n+1
% e fornisce in uscita la matrice relativa
    s = spirale(n);
    m = max(max(s));
    c = crivello(m);
    z = c(s);
    imagesc(z);
endfunction
```

2.2 Esercizi

E adesso un po' di esercizi da fare. Ciascuno studente scelga un esercizio di tipo A e uno di tipo B operando nel seguente modo. Prendere l'iniziale del proprio cognome e associargli il numero d'ordine nell'alfabeto internazionale

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

modulo il numero degli esercizi disponibili; scegliere l'esercizio corrispondente. Ad esempio, se ci sono 5 esercizi disponibili numerati da 0 a 4, lo studente R. King, avendo il proprio cognome di iniziale K che ha numero d'ordine 11, deve scegliere l'esercizio il cui numero è dato da 11 modulo 5, cioè il numero 1.

Esercizi del gruppo A (6 esercizi):

Esercizio A0. Evidenziare i numeri primi nel triangolo di Klauber in cui la riga i-esima contiene i numeri da $(i-1)^2+1$ a i^2. Cioè la numerazione avviene nel seguente modo

$$\begin{array}{ccccccccc}
 & & & & 1 & & & & \\
 & & & 2 & 3 & 4 & & & \\
 & & 5 & 6 & 7 & 8 & 9 & & \\
 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Utilizzare una matrice $n \times n$ con $n = 500$ per costruire l'immagine. Due immagini del triangolo di Klauber relativo ai primi 150^2 numeri interi e relativo ai primi 300^2 numeri interi sono riportate nelle figure 2 e 3.

Si possono notare gli allineamenti lungo segmenti verticali. In particolare i numeri del tipo $f(n) = n^2 + n + 17$ per $n = 0, 1, 2, 3, \ldots, 15$

Per maggiori informazioni sul triangolo di Klauber si veda sempre la pagina di Wikipedia sulla spirale di Ulam.

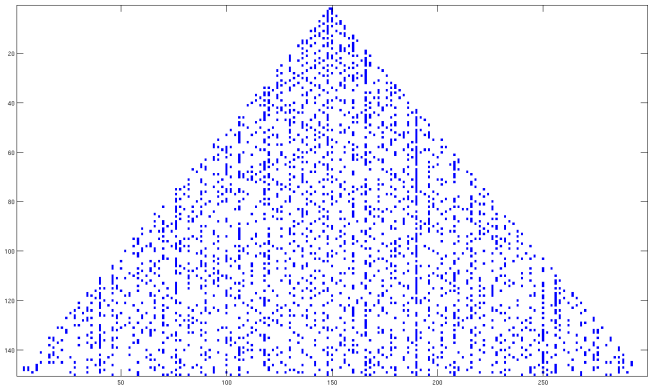


Figura 2: Triangolo di Klauber dei numeri primi da 2 a 150^2.

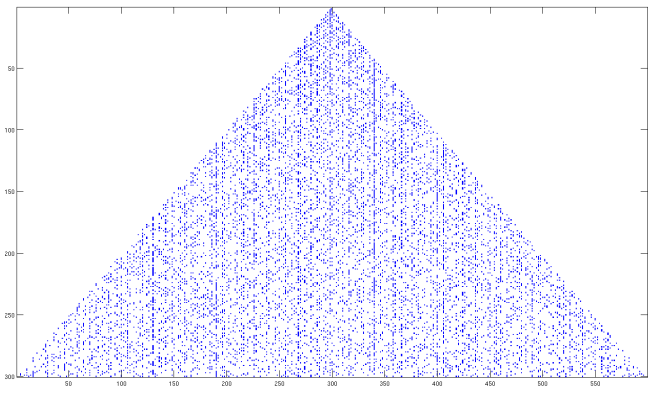


Figura 3: Triangolo di Klauber dei numeri primi da 2 a 300^2.

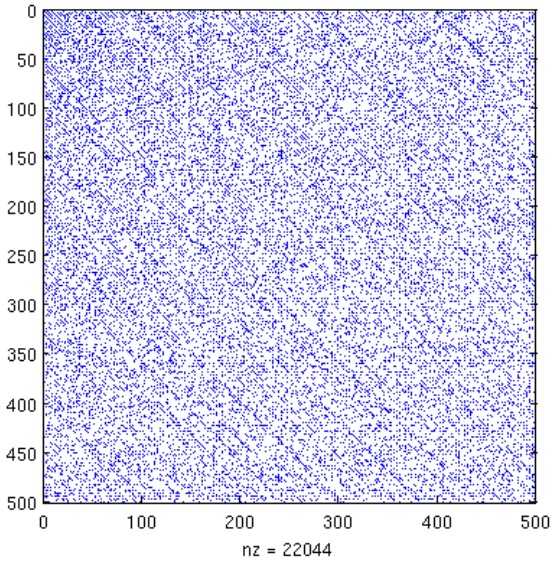


Figura 4: Risultato dell'esercizio A1

Esercizio A1. Evidenziare i numeri primi in una matrice $n \times n$, $n = 500$, in cui la numerazione avviene nel seguente modo

1	2	6	7	15	16
3	5	8	14	17	
4	9	13	18	·	
10	12	19	·		
11	20	·			
21	23				
22					

L'immagine che si ottiene in questo caso è riportata in figura 4.

Esercizio A2. Evidenziare i numeri primi in una matrice $n \times n$, $n = 500$, in cui la numerazione avviene nel seguente modo

1	2	9	10	25	26
4	3	8	11	24	27
5	6	7	12	23	28
16	15	14	13	22	29
17	18	19	20	21	30
36	35	34	33	32	31
37	38	...			

L'immagine che si ottiene in questo caso è riportata in figura 5.

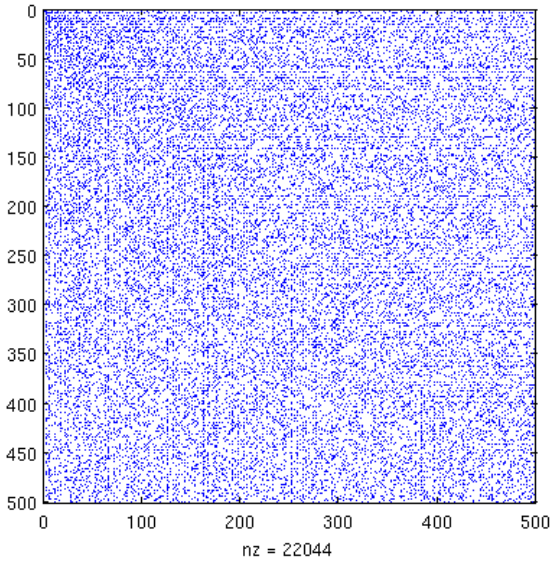


Figura 5: Risultato dell'esercizio A2

Esercizio A3. Evidenziare i numeri primi in una matrice $n \times n$, $n = 500$, in cui la numerazione avviene nel seguente modo

$$\begin{array}{ccccccc} 1 & 2 & 5 & 10 & 17 & 26 & 37 \\ 4 & 3 & 6 & 11 & 18 & 27 & 38 \\ 9 & 8 & 7 & 12 & 19 & 28 & . \\ 16 & 15 & 14 & 13 & 20 & 29 & . \\ 25 & 24 & 23 & 22 & 21 & 30 & . \\ 36 & 35 & 34 & 33 & 32 & 31 & \end{array}$$

L'immagine che si ottiene in questo caso è riportata in figura 6.

Esercizio A4. Evidenziare i numeri primi in una matrice $n \times n$, $n = 500$, in cui la numerazione (dei soli numeri dispari) avviene nel seguente modo (si veda il "Boustrophedon" e le sue proprietà a `http://haslock.com/Boustrophedon.html`).

$$\begin{array}{ccccccccc} & & & & 1 & & & & \\ & & & & 3 & 5 & & & \\ & & & 11 & 9 & 7 & & & \\ & & & 13 & 15 & 17 & 19 & & \\ & & 29 & 27 & 25 & 23 & 21 & & \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

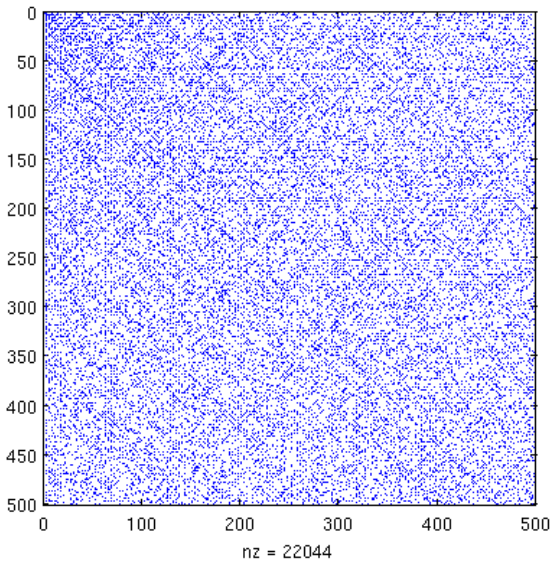


Figura 6: Risultato dell'esercizio A3

Figura 7: Risultato dell'esercizio A4

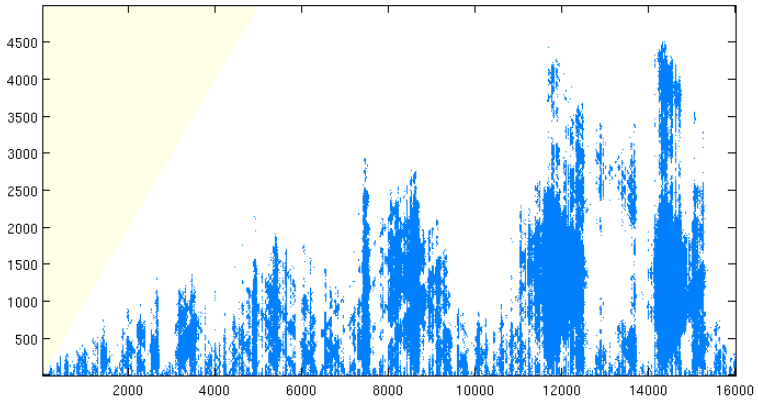


Figura 8: Risultato dell'esercizio A5. Densità dei numeri primi: il pixel $(n-j+1, i)$ è acceso se il numero di primi nell'intervallo $[i-j, i]$ è minore del numero di primi nell'intervallo $[i, i+j]$.

L'immagine che si ottiene in questo caso è riportata in figura 7.

Esercizio A5. Per evidenziare graficamente la maggiore o minore concentrazione di numeri primi in $\mathbb{N}$ si costruisca una immagine $A = (a_{i,j})$ di dimensione $n \times n$ così fatta. Per ogni coppia di interi (i, j) con $j < i$ si considerino gli intervalli $\mathcal{I}_+ = [i, i+j]$ e $\mathcal{I}_- = [i-j, i]$. Se il numero di primi nell'intervallo $\mathcal{I}_+$ è maggiore del numero di primi in $\mathcal{I}_-$ allora si accende il pixel di coordinate $(n-j+1, i)$, cioè si pone $a(n-j+1, i) = 1$ altrimenti si pone $a(n-j+1, i) = 0$.

L'immagine che si ottiene in questo caso con i primi 16000 interi ($n = 16000$), tagliata a dimensione $\frac{n}{2} \times n$ è riportata in figura 8

La figura 9 riporta le immagini ottenute con vari valori di n

Esercizi del gruppo B (5 esercizi)

Esercizio B0. Dare una versione vettoriale, basata sul crivello di Eratostene, del programma che crea un'immagine in cui il pixel di posto (i, j) è bianco se il numero $n(i-1)+j$ è primo, altrimenti è nero.

Aiuto: costruire una matrice S il cui elemento di posto (i, j) è $j + n(i-1)$ e calcolare $c(S)$ dove c è il vettore dato dal crivello di Eratostene. Per calcolare S si scriva S come somma $P + Q$ dove $P =$ `ones(n,1) *[1:n];` e Q è data da ....

Esercizio B1. Evidenziare le coppie di numeri interi (p, q) tali che $p^2 + q^2$ è un numero primo. Per questo create una immagine di dimensione $n \times n$ in cui il pixel di posto (p, q) è bianco se $p^2 + q^2$ è primo, nero altrimenti. Dare anche una versione vettoriale del programma.

Aiuto: per la versione vettoriale costruire una matrice S che abbia nel posto

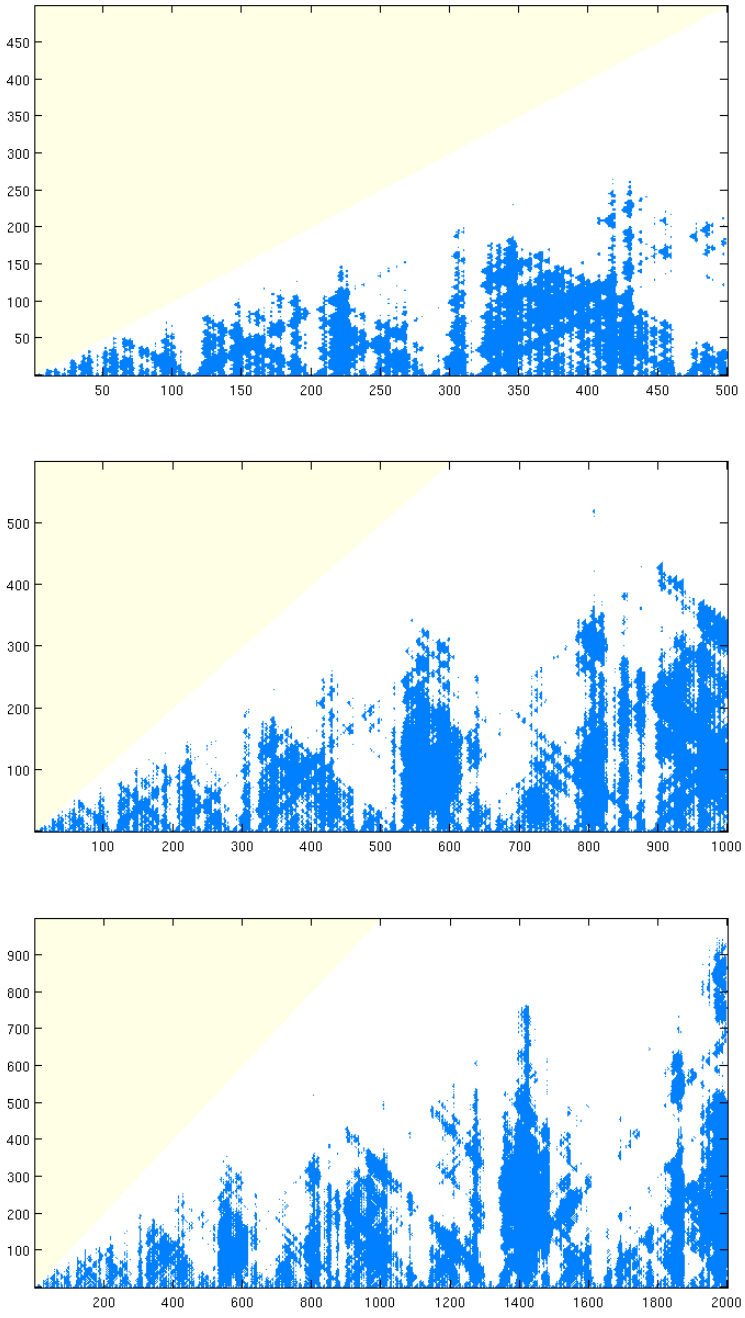


Figura 9: Risultato dell'esercizio A5. Densità dei numeri primi con valori di $n = 500, 1000, 2000$.

(p, q) l'elemento $p^2 + q^2$ e si calcoli $c(S)$. Per il calcolo di S si proceda come nell'esercizio 1.

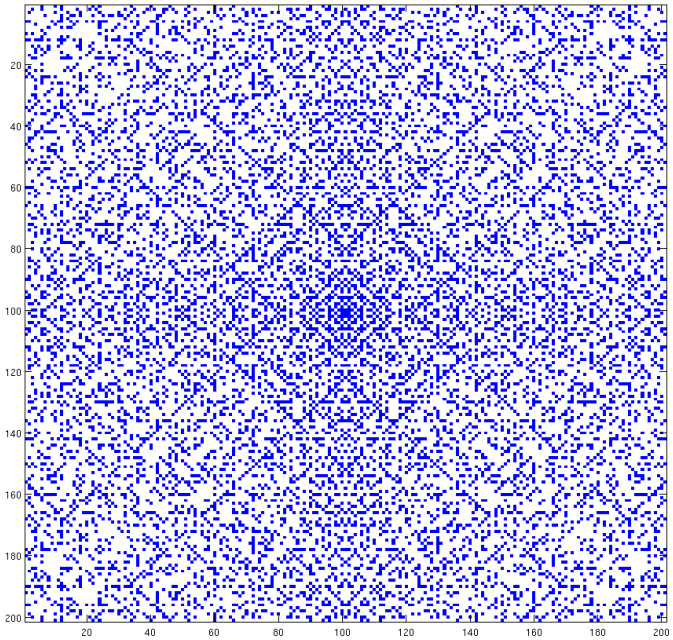
Esercizio B2. Evidenziare le coppie di numeri interi (p, q) tali che $|p^2 - q^2| + k$ è un numero primo. Per questo create una immagine di dimensione $n \times n$ in cui il pixel di posto (p, q) è bianco se $|p^2 - q^2| + k$ è primo, nero altrimenti, dove $k = 1$, $k = 2$, $k = 3$. Dare anche una versione vettoriale del programma.
Aiuto: per la versione vettoriale costruire una matrice S che abbia nel posto (p, q) l'elemento $p^2 - q^2$ e si calcoli $c(S)$. Per il calcolo di S si proceda come nell'esercizio 1.

Esercizio B3. Evidenziare le coppie di numeri interi (p, q) tali che $|p^2 - q^2| + k$ è un numero primo sia per $k = 1$ che per $k = 3$. Per questo create una immagine di dimensione $n \times n$ in cui il pixel di posto (p, q) è bianco se $|p^2 - q^2| + k$ è primo, sia per $k = 1$ che per $k = 3$, è nero altrimenti. Dare anche una versione vettoriale del programma.
Aiuto: per la versione vettoriale costruire una matrice S che abbia nel posto (p, q) l'elemento $|p^2 - q^2|$ e si calcoli $c(S + k)$. Per il calcolo di S si proceda come nell'esercizio 1.

Esercizio B4. Evidenziare le coppie di numeri interi (p, q) tali che $pq + 1$ è un numero primo. Per questo create una immagine di dimensione $(2m + 1) \times (2m + 1)$, dove m è un intero assegnato ad esempio $m = 100$, e accendere il pixel in posizione (p, q) se $|(p - m)(q - m)| + 1$ è primo. Dare anche una versione vettoriale del programma.
Aiuto: per la versione vettoriale costruire una matrice S che abbia nel posto (p, q) l'elemento $|(p - m)(q - m)|$ e si calcoli $c(S + 1)$. Per il calcolo di S si proceda come nell'esercizio 1.

In quest'ultimo esercizio si ottiene l'immagine riportata in figura 10. In questa immagine l'occhio umano riesce ad individuare particolari allineamenti anche evidenziati dalla simmetria centrale.

Due numeri primi vengono detti primi gemelli se la loro differenza è 2. Nella spirale di Ulam possiamo accendere solamente quei pixel che corrispondono ai numeri pari che separano due primi gemelli. In questo modo otteniamo l'immagine riportata nella figura 11.



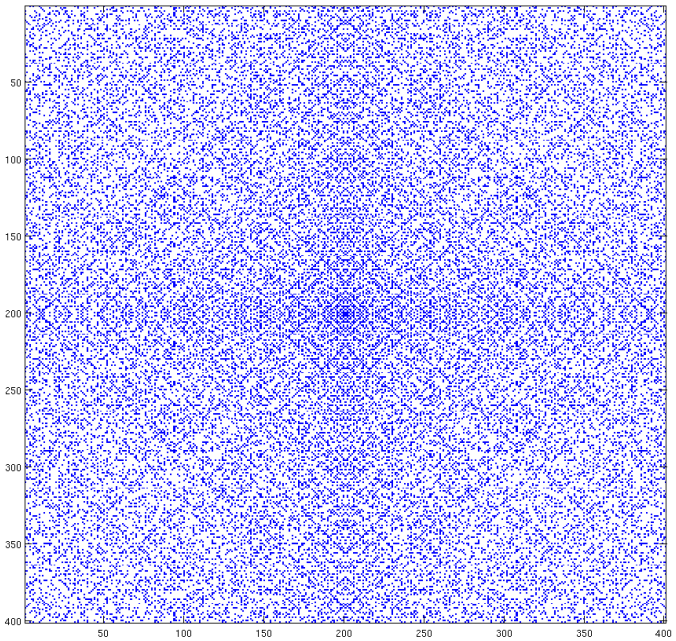


Figura 10: Risultato dell'esercizio B4 con dimensioni 201,401.

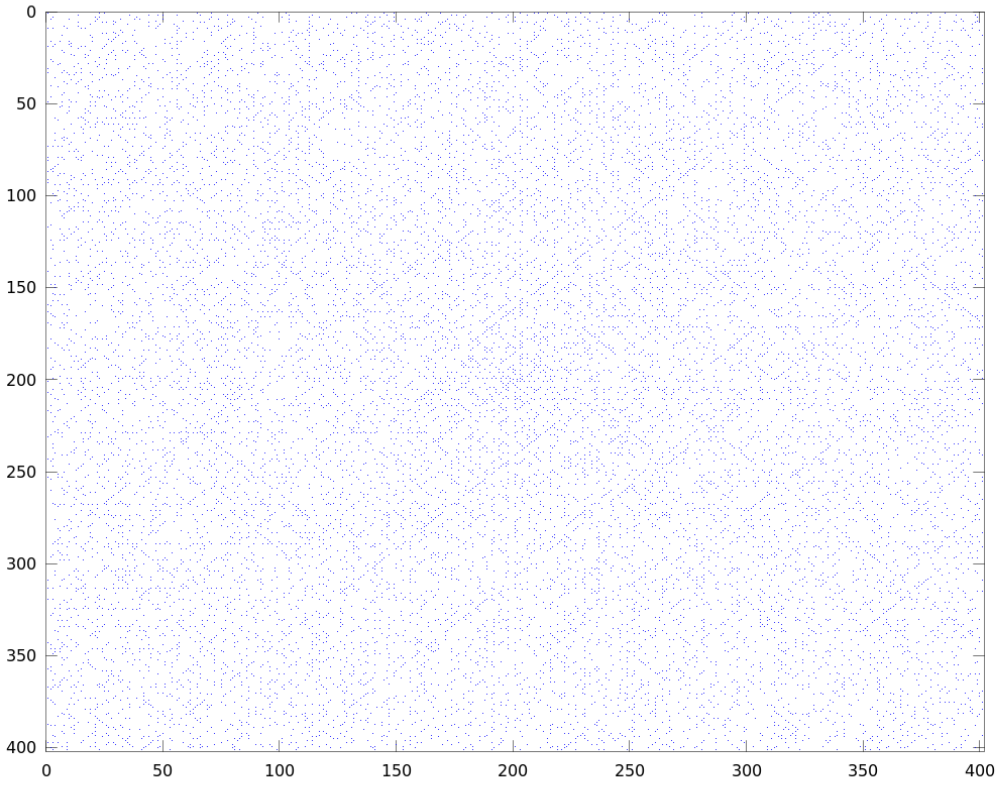


Figura 11: Primi gemelli nella spirale di Ulam.

3 Bacini di attrazione

Talvolta è utile studiare la dinamica delle successioni generate da espressioni del tipo

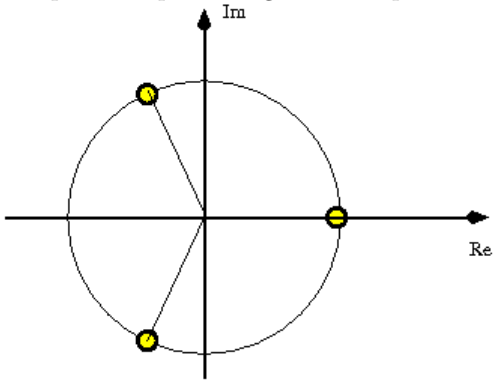
$$z_{k+1} = g(z_k)$$

al variare del valore iniziale z_0. Questo capita ad esempio quando la successione viene costruita per calcolare i punti fissi della funzione $g(x)$ definita sul piano complesso. Un esempio classico è dato dal metodo di Newton applicato all'equazione $f(x) = 0$ che fornisce l'iterazione ottenuta con $g(x) = x - f(x)/f'(x)$. Nel caso della funzione $f(x) = x^3 - 1$ si ottiene l'iterazione

$$z_{k+1} = z_k - \frac{z_k^3 - 1}{3z_k^2}.$$

Le successioni generate in questo modo possono convergere a ciascuno dei tre zeri del polinomio $x^3 - 1$ o possono non convergere. È allora interessante individuare per quali valori di z_0 la successione converge a 1, per quali valori la successione converge a $-1/2 + \mathbf{i}\sqrt{3}/2$ e per quali valori la successione converge a $-1/2 - \mathbf{i}\sqrt{3}/2$, dove $\mathbf{i}$ è l'unità immaginaria.

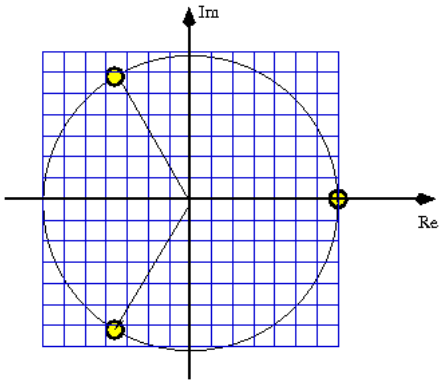
Si riportano qui sotto gli zeri del polinomio $x^3 - 1$.



Proviamo a creare una immagine che rappresenti una porzione del piano complesso, ad esempio di centro 0 e semilato 1, in cui il generico pixel viene colorato di rosso se il corrispondente numero complesso assegnato a z_0 genera una successione che converge a 1, di verde se la successione converge a $-1/2+\mathbf{i}\sqrt{3}/2$, di blu se converge a $-1/2-\mathbf{i}\sqrt{3}/2$ e di nero se la successione non converge. Naturalmente dobbiamo dare un numero massimo di passi oltre il quale si assume che non ci sia convergenza se il valore corrente x_k non si trova in un intorno sufficientemente piccolo di uno degli zeri.

Procediamo nel modo seguente per creare una immagine di $(2m+1)\times(2m+1)$ punti, dove m è un intero assegnato.

Poniamo $h = 1/m$ e consideriamo una griglia di $(2m + 1) \times (2m + 1)$ punti $w_{i,j}$ nel quadrato $[-1, 1]\times[-1, 1]$ del piano complesso definiti da $w_{i,j} = (j+\mathbf{i}i)h$, $i = -m, m$, $j = -m, m$.



Associamo ad ogni $w_{i,j}$ il pixel di posto (i, j) di una immagine $(2m + 1) \times (2m + 1)$ dove $w_{0,0}$ è l'elemento centrale. Posto $z_0 = w_{i,j}$ consideriamo la successione z_k generata dal metodo di Newton e coloriamo rispettivamente di rosso verde e blu il pixel di posto (i, j) che corrisponde al punto $w_{i,j}$ se la successione z_k converge a $x_1 = 1$, $x_2 = -1/2 + \mathbf{i}\sqrt{3}/2$, $x_3 = -1/2 - \mathbf{i}\sqrt{3}/2$ che sono le tre radici del polinomio $z^3 - 1$. Per fare questo eseguiamo 20 passi del metodo di Newton e controlliamo se z_k dista meno di $1/10000$ rispettivamente da x_1, x_2, x_3.

Un programma che realizza questo è riportato nel listato 8.

Naturalmente il programma scritto in questo modo è particolarmente lento a causa dei vari cicli `for` annidati. Un modo per vettorizzare il programma di sopra è riportato nel listato 9

Per tracciare l'immagine occorre poi dare il comando `image(a)` seguito, o preceduto dal comando `colormap(eye(3))` che sceglie una mappa di colori formata dai soli tre colori rosso, verde e blu.

Si osservi l'uso dell'operatore .* che, applicato a due matrici, calcola il prodotto componente a componente. Analogamente l'operatore ./ applicato a due matrici svolge la divisione componente a componente.

Si osservi ancora che per estrarre l'informazione sulla avvenuta convergenza alle soluzioni si usa una matrice a tre indici `a(:,:,:)` costituita da tre fette: la prima contiene la matrice `x-x1`, la seconda `x-x2`, la terza `x-x3`.

Il comando `[valori, posizioni]=min(abs(b));` dà nella variabile `posizioni` una matrice $1 \times n \times n$ tale che `posizione(1,i,j)` vale $1, 2$ o 3 a seconda che il minimo su k di `abs(b(k,i,j))` sia preso per $k = 1, 2$ o 3. Cioè vale $1, 2$ o 3 a seconda che $z(i, j)$ sia più vicino a x1, x2 o x3. Il comando `reshape(posizioni, [n n])` riorganizza la matrice a tre indici `posizioni` come una matrice a due indici $n \times n$. La variabile `valori` contiene i valori corrispondenti del minimo.

La function scritta in questa forma assegna il colore rosso, verde o blu al pixel (i, j) a seconda che dopo `maxit` iterazioni il valore di z_k con $k =$ `maxit`, sia più vicino rispettivamente a x_1, x_2 o x_3. Non viene richiesto che z_k si trovi in un intorno abbastanza piccolo di una di queste tre radici. Per dare un colore diverso, ad esempio nero, al pixel (i, j) nel caso in cui z_k non si trovi in un intorno piccolo di una delle tre radici, si può operare come segue.

Listing 8: Function che costruisce i bacini di attrazione per il metodo di Newton applicato al polinomio $x^3 - 1$.

```
function a = newton(m)
   maxit = 20;
% zeri del polinomio
   x1 = 1;
   x2 = -0.5+I*sqrt(3)/2;
   x3 = -0.5-I*sqrt(3)/2;
   a = 4*ones(2*m+1);
   h = 1/m;
   for i = -m : m
      for j = -m:m
         z = (j-i*I)*h;
% itera
         for k = 1 : maxit
            z = g(z);
         endfor
% colora
         if abs(z-x1)<1.0e-4
            a(m+i+1,m+j+1) = 1;
         endif
         if abs(z-x2)<1.0e-4
            a(m+i+1,m+j+1) = 2;
         endif
         if abs(z-x3)<1.0e-4
            a(m+i+1,m+j+1) = 3;
         endif
      endfor
   endfor
   mappa = [1 0 0; 0 1 0; 0 0 1; 0 0 0];
   colormap(mappa);
   image(a);
endfunction

function y = g(z)
   y = 0;
   if z!=0
      y = z - (z^3 - 1)/(3*z^2);
   endif
endfunction
```

Listing 9: Function che costruisce i bacini di attrazione per il metodo di Newton applicato al polinomio $x^3 - 1$. Il programma agisce in modo vettoriale

```
function a = newton1(m)
   maxit = 20;
% zeri del polinomio
   x1 = 1;
   x2 = -0.5+I*sqrt(3)/2;
   x3 = -0.5-I*sqrt(3)/2;
   range = -2:2/m:2;
   n = 2*m+1;
   z = ones(n,1)*range - I*range'*ones(1,n);
% itera simultaneamente su tutti i punti
   z = z + 1.e-30; % per evitare lo zero
   for k = 1 : maxit
      y = z.*z;
      z = z - (z.*y - 1)./(3*y);
   endfor
   b = zeros(3,n,n);
   b(1,:,:) = z-x1; % sottrae x1 a tutte le componenti di z
   b(2,:,:) = z-x2; % sottrae x2 a tutte le componenti di z
   b(3,:,:) = z-x3; % sottrae x3 a tutte le componenti di z
   [valori, posizioni] = min(abs(b));
   a = reshape(posizioni, [n n]); %
endfunction
```

```
v = reshape(valori, [n,n]);
a = a.*(v < 1.e-7) + 1;
colormap([0 0 0; 1 0 0; 0 1 0; 0 0 1]);
image(a);
```

In questo modo si moltiplica, elemento per elemento, i valori della matrice `a` per 1 se la distanza da uno degli zeri è minore di 10^{-7}, si moltiplica per 0 altrimenti, infine si aggiunge 1 agli elementi di `a`. In questo modo il colore 1 corrisponde a "non convergenza" i colori 2,3 e 4 corrispondono a convergenza alle radici x_1, x_2, x_3.

Adesso un po' di esercizi:

Esercizio C0. Si modifichi la function `newton1` in modo da colorare i pixel in base al minimo numero di passi sufficienti affinché $|z_k - z_{k-1}| < 1/10000$, indipendentemente dal valore del limite. Cioè si assegni il colore k della mappa dei colori al generico pixel in posizione (i, j) se k è il più piccolo intero per cui $|z_k - z_{k-1}| < 1/10000$, dove z_0 è il numero complesso associato al pixel (i, j). Adattare questa function al metodo di Newton applicato al polinomio $z^5 + (10z + 1)^2$ sul quadrato del piano complesso di centro 0 e semilato 5. Si usi `imagesc` per tracciare l'immagine.
Aiuto: siano `Z` e `W` le matrici col valore corrente e il valore successivo al generico passo dell'iterazione; si consideri la matrice `H=abs(W-Z) > 1.0e-4` che in posizione (i, j) contiene 1 se la componente di posto (i, j) di `abs(W-Z)` è maggiore di `1.0e-4`, contiene zero altrimenti; si accumuli in una variabile `S` la somma delle `H`.

Esercizio C1. Procedere come nell'esercizio C0, ma adattare la function al metodo di Newton applicato alla funzione $z^3 + (10 + 1/z)^2$, che ha gli stessi zeri del polinomio $z^5 + (10z + 1)^2$, sul quadrato del piano complesso di centro 0 e semilato 5. Si usi `imagesc` per tracciare l'immagine. Le immagini che si ottengono nei due casi sono riportate in figura 12 dove si riporta anche la rappresentazione con una mappa di colori casuale.

Uno zoom nel punto -0.15, con semilato $\ell = 1$, dei bacini di attrazione del metodo di Newton applicato alla funzione razionale $f(z)$ dell'esercizio C1 è riportato in figura 13. L'immagine è ottenuta con la mappa dei colori default.

Esercizio C2. Modificare la function `newton1` in modo da creare una immagine con colori rosso, verde, blu e giallo di diversa intensità in base al numero di iterazioni impiegate, dove il colore è scelto a seconda del valore del limite della successione $\{z_k\}$ generata dalla funzione $g(z)$. Applicare alla funzione $g(z)$ ottenuta dal metodo di Newton per le funzioni $z^4 - 1$, $z^3 - z^{-1}$, $z - z^{-3}$. Tracciare le immagini relative al quadrato centrato in 0 e di semilato $\ell = 2$. Nella figura 14 sono riportati i due risultati ottenuti con le due funzioni razionali. Nella figura 15 sono riportati i casi di $z^3 - 1$, $z^2 - z^{-1}$, $z - z^{-2}$, $1 - z^{-3}$ con mappa dei colori casuale.
Aiuto: per valutare il numero di passi si costruisca la matrice `S` come nell'esercizio C1, per valutare il colore si costruisca la matrice `a` come nella function

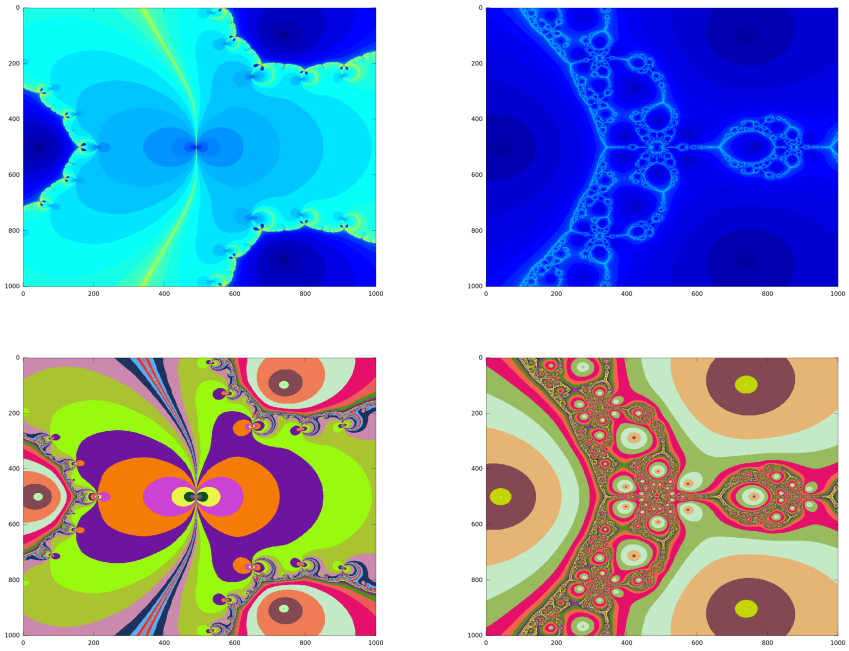


Figura 12: Rappresentazione dei bacini di attrazione per il metodo di Newton applicato al polinomio $p(z) = z^5 + (10z + 1)^2$ (prima immagine) e alla funzione razionale$f(z) = p(z)/z^2$ (seconda immagine), relativi al quadrato del piano complesso di centro 0 e semilato 5. Sono riportate le immagini ottenute con la mappa default (prime due immagini) e con una mappa casuale (seconde due immagini) usando il comando `imagesc`

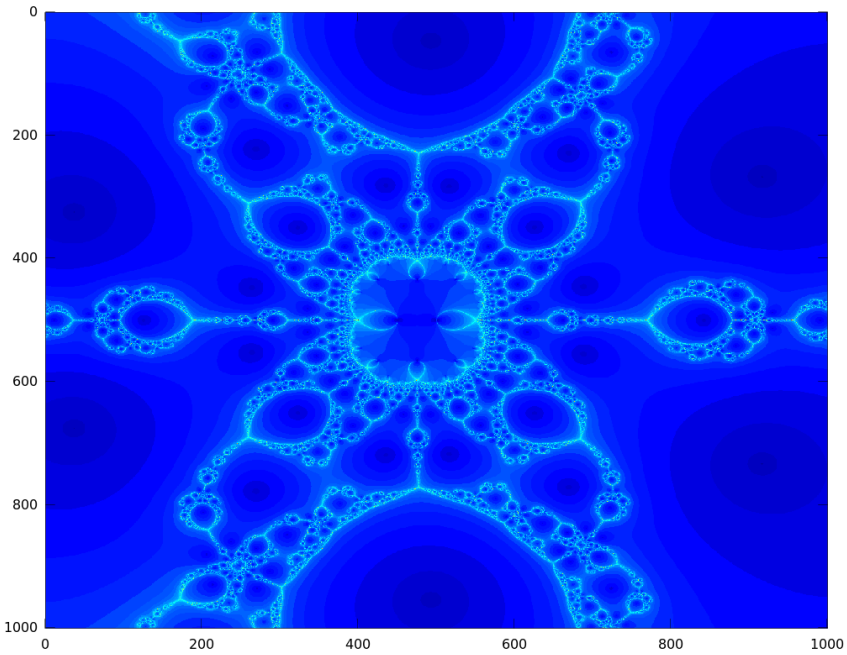


Figura 13: Porzione relativa al quadrato di centro -0.15 e semilato $\ell = 1$ dei bacini di attrazione del metodo di Newton applicato alla funzione razionale $f(z) = z^3 + (10 + 1/z)^2$. L'immagine è ottenuta con la mappa di colori default

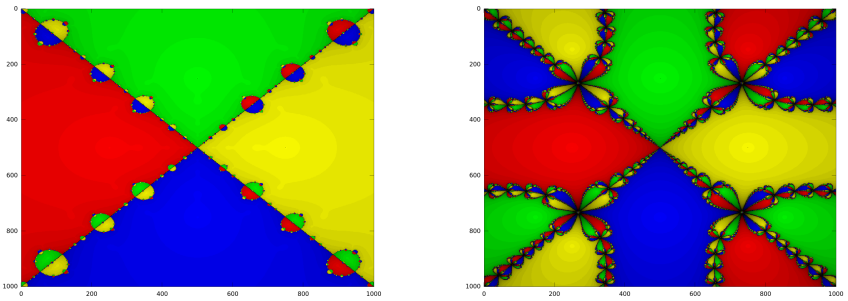


Figura 14: Bacini di attrazione per il metodo di Newton applicato alle funzioni $x^3 - x^{-1}$, $x - x^{-3}$

`newton1`. Si determini la matrice `C` che dà il colore ponendo `C = (a-1)*maxit + S`. Si costruisca poi una mappa dei colori `MP` che nei primi maxit colori contiene i rossi da intensità 1 a intensità 0, nei secondi maxit colori contiene i verdi da intensità 1 a intensità 0, nei terzi maxit colori contiene i blu da intensità 1 a intensità 0, nei quarti maxit colori contiene i gialli da intensità 1 a intensità 0. Si tracci la figura con `image(C); colormap(MP)`.

Esercizio C3. Modificare la function `newton1` in modo che prenda in input i coefficienti di un polinomio $p(x)$ di grado d dati come vettore riga $(p_d, p_{d-1}, \ldots, p_0)$, un numero intero $m > 0$ e un numero reale $\ell > 0$, e costruisca i bacini di attrazione per $p(x)$ relativi ad una porzione del piano complesso di centro 0 e semilato ℓ, tracciando sullo schermo una immagine $(2m+1) \times (2m+1)$ in cui il bacino di attrazione corrispondente alla i-esima radice di $p(x)$ abbia il colore i-esimo nella mappa dei colori. Si costruisca l'immagine per un polinomio `p=rand(1,11)` con $m = 200$. Le immagini di 4 casi con $m = 500$ sono riportati in figura 16.
Aiuto: si calcolino le radici di $p(x)$ col comando `r=roots(p);` si usi il comando `polyval(p,z)` per calcolare il valore del polinomio $p(x)$ nel punto (o matrice) z. Infine si proceda come nella function `newton1`.

3.1 L'insieme di Mandelbrot e gli insiemi di Julia

L'insieme dei numeri complessi c per cui la successione generata da

$$\begin{aligned} z_{k+1} &= z_k^2 + c \\ z_0 &= 0 \end{aligned}$$

rimane limitata è chiamato insieme di Mandelbrot. Per avere un'idea di come questo insieme è fatto possiamo costruire una immagine corrispondente ai punti del piano complesso che stanno nel quadrato di centro 0 e semilato 2, in cui il

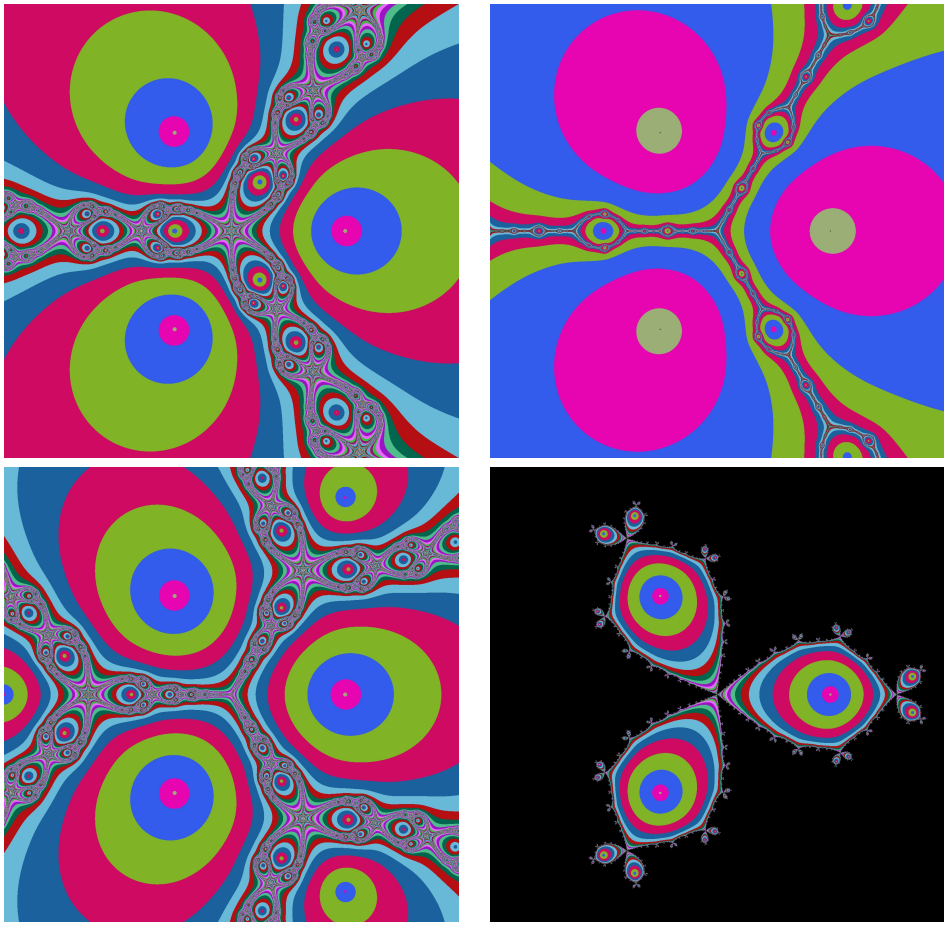


Figura 15: Bacini di attrazione per le funzioni x^3-1, x^2-x^{-1}, $x-x^{-2}$, $1-x^{-3}$, nell'ordine antiorario partendo dalla figura in alto a sinistra.

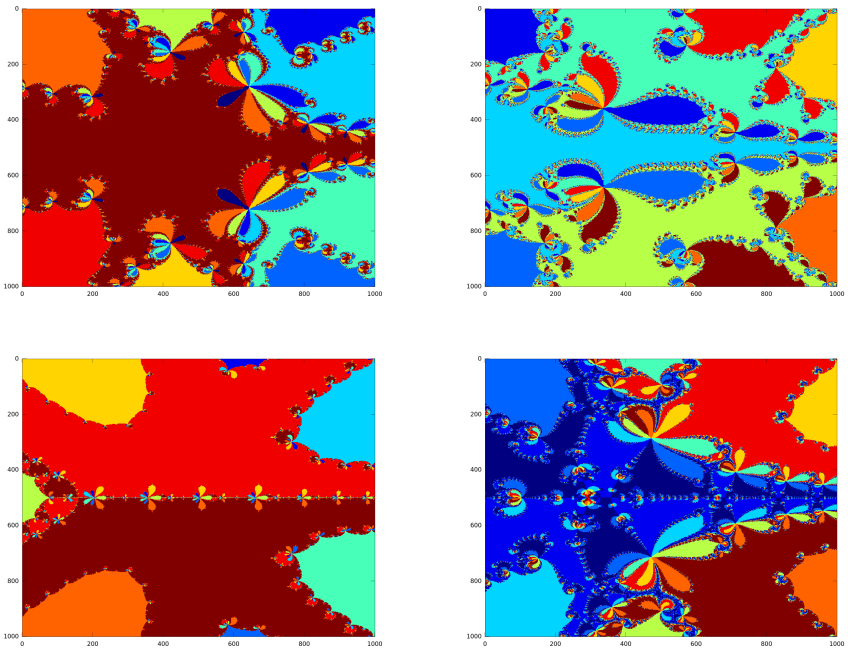


Figura 16: Bacini di attrazione per quattro polinomi di grado 11 con coefficienti casuali tra 0 e 1 relativi al quadrato $[-1, 1] \times [-1, 1]$.

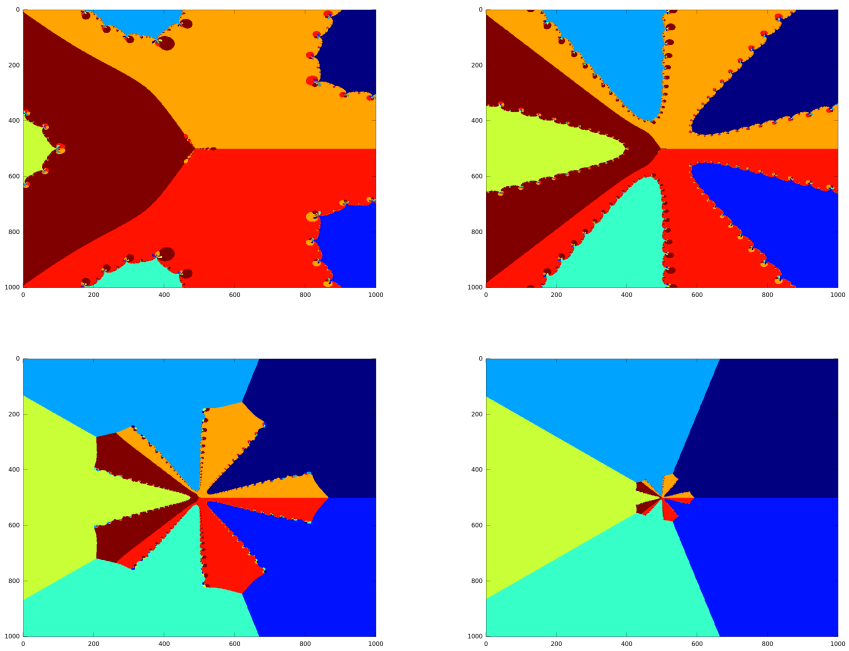
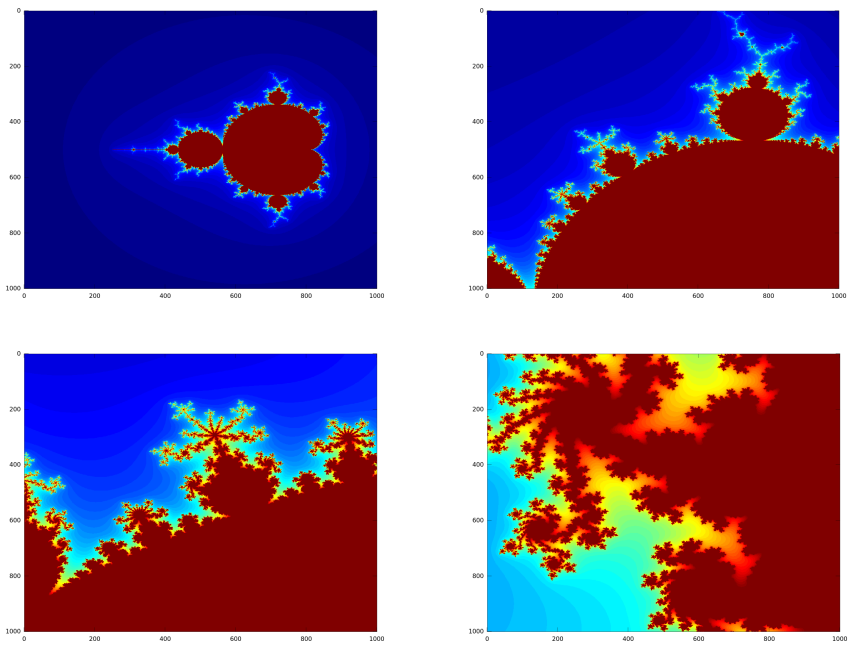


Figura 17: Bacini di attrazione per il polinomio $p(x) = x^8 + (ax+1)^3$, con $a = 3$ relativi al quadrato $[-\ell, \ell] \times [-\ell, \ell]$ per valori $\ell = 4, 16, 64, 256$.

pixel in posizione (i, j) è colorato di nero se il corrispondente numero complesso c sta nell'insieme di Mandelbrot, e viene invece colorato con il q-esimo colore della tavola dei colori se z_q è il primo valore della successione il cui modulo è maggiore di 2.

Esercizio D0. Si scriva una function in Octave che, dati $p \in \mathbb{C}$, $\ell > 0$, $m \in \mathbb{N}$, disegna la porzione della figura di Mandelbrot in un quadrato di centro p e semilato ℓ, come immagine di $(2m+1)\times(2m+1)$ pixel, nel seguente modo: si usa la mappa dei colori `default`, si svolgono $q = 63$ iterazioni e per ogni coppia (i, j) si calcola il numero di passi k che occorrono affinché z_k abbia modulo maggiore di 2; si colora il pixel di posto (i, j), corrispondente al valore di c col k-esimo colore della tavola dei colori; se dopo q iterazioni il modulo di x_q è minore di 2 si colora il pixel (i, j) col colore q-esimo della tavola dei colori. Dare una versione vettorizzata della function. Applicare la function con i valori $p = -1.5$, $\ell = 2$, e con $p = \varphi - 2 + \mathbf{i}(\varphi - 1)$ dove $\varphi = (1 + \sqrt{5})/2$ è la sezione aurea, $\ell = 0.5, 0.1, 0.01$. Qui sotto si riportano le immagini ottenute.

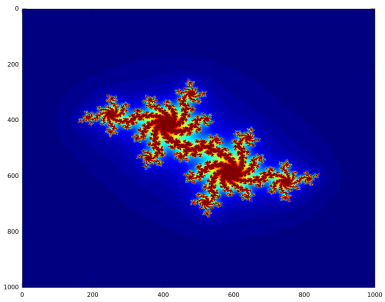


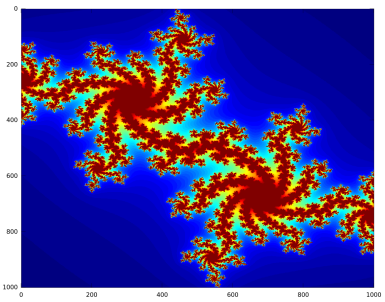
Definiamo insieme di Julia relativo a una funzione $f(x)$ l'insieme dei punti z_0 del piano complesso per cui la successione $\{z_k\}$ definita da $x_{k+1} = f(z_k)$, $k = 0, 1, 2, \ldots$, ha modulo limitato superiormente da una costante. Si veda `http://en.wikipedia.org/wiki/Julia_set` per maggiori informazioni a riguardo.

È interessante esaminare come sono fatti gli insiemi di Julia per la funzione $f(z) = z^2 + c$ per diversi valori di c.

Esercizio D1. Per poter evidenziare un insieme di Julia, si scriva una function in Octave che, dati $c, p \in \mathbb{C}$, $\ell > 0$, $m \in \mathbb{N}$, costruisce un'immagine $(2m+1) \times (2m+1)$ relativa alla porzione del piano complesso di centro p e semilato ℓ procedendo in modo analogo all'esercizio D0. La differenza è che il valore di c è fissato e dato in input alla function, mentre varia in funzione della coppia di indici (i, j) il valore iniziale di z_0 e la colorazione del pixel in posizione (i, j) avviene in relazione al primo valore k per cui $|z_k| > 2$. Si applichi la function con i valori $c = \varphi - 2 + \mathbf{i}(\varphi - 1)$ dove $\varphi = (1+\sqrt{5})/2$ è la sezione aurea, con centro 0 e semilato $\ell = 2,1$. Per la colorazione, si usi la mappa dei colori default.

Qui sotto si riportano le figure ottenute.





4 Manipolazioni geometriche

In questa sezione ci occupiamo di manipolare immagini digitali, ad esempio fotografie, agendo unicamente sulle coordinate dei pixel. È utile prima introdurre dei comandi nel linguaggio Octave che permettono di leggere e scrivere immagini digitali da e in ogni formato.

- `a = imread('nomefile')` permette di leggere una immagine digitale e tradurla in una matrice. Ad esempio se il file è di tipo pgm allora la variabile `a` conterrà una matrice $m \times n$ con i valori numerici dei pixel corrispondenti, dove m e n sono le dimensioni dell'immagine. Se il file è di tipo ppm o pnm a colori in formato ascii o raw (numero magico P3 o P6) allora la variabile `a` conterrà una matrice $m \times n \times 3$ dove m, n sono le dimensioni dell'immagine e le tre "fette" contengono le componenti di rosso, verde e blu. Il file può essere indifferentemente di tipo ascii o di tipo raw. I principali formati sono consentiti, ad esempio jpeg, png.

- `imwrite(a, 'nomefile')` scrive i dati numerici contenuti nella variabile `a` in un file immagine di nome 'nomefile' il cui formato è specificato dall'estensione usata: ad esempio `imwrite(a, 'pippo.jpg')` salva l'immagine come file di tipo jpeg, `imwrite(a, 'pippo.png')` salva l'immagine come file png.

- `imfinfo('nomefile')` dà informazioni sull'immagine contenuta nel file 'nomefile'.

4.1 Il gatto di Arnold

Una sperimentazione interessante è la seguente. Trasformiamo un'immagine A di dimensioni $m \times m$ in questo modo. Il pixel di posto (i, j) dell'immagine originale viene trasformato nel pixel di posto $(2i + j \mod m, i + j \mod m)$, dove abbiamo assunto per comodità che gli indici scorrano da 0 a $m - 1$.

Questa trasformazione è l'analogo discreto della mappa di Arnold `http://it.wikipedia.org/wiki/Gatto_di_Arnold` che trasforma il toro in sé stesso ed è definita per $(x, y) \in [0, 1)^2$ da

$$\Gamma : (x, y) \to (2x + y, x + y) \mod 1 = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \mod 1$$

L'operazione "modulo 1" serve a identificare i bordi opposti del quadrato su cui è definta Γ in modo da operare sul toro. Similmente, nella mappa definita sul discreto, l'operazione "modulo m" ha lo stesso significato.

La mappa è lineare e invertibile poiché la trasformazione lineare, individuata dalla matrice $M = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 1 \end{smallmatrix}\right]$, ha determinante 1. La mappa ha autovalori irrazionali $\frac{-2 \pm \sqrt{5}}{2}$, in modulo uno maggiore l'altro minore di 1. Gli autovettori corrisponenti definiscono rispettivamente un sottospazio contraente ed uno dilatante.

Listing 10: Function che applica k volte la mappa di Arnold all'immagine `a`

```
function b=arnold_cat(a,k)
s=size(a);
b=a;
for h=1:k
  for i=0:s(1)-1
    for j=0:s(2)-1
      ip = mod(2*i+j,s(1));
      jp = mod(i+j,s(2));
      b(ip+1,jp+1,:) = a(i+1,j+1,:);
    endfor
  endfor
  a = b;
endfor
endfunction
```

Una implementazione della mappa discreta può essere ottenuta in modo semplice. La function nel listato 10 applica k passi della mappa di Arnold all'immagine A. La function può essere applicata sia a immagini a colori, dove la variabile `a` è una array $m \times n \times 3$, che a immagini in bianco nero in cui `a` è una matrice $m \times n$.

È interessante vedere come solo dopo pochi passi l'immagine diventi caotica. I pixel sono così mescolati che la foto originale del gatto non si riconosce. Ma dopo un certo numero di iterazioni il gatto ricompare distinto nell'immagine. Precisamente, nel caso della nostra immagine 354×354 questo avviene dopo 348 passi. Questo fenomeno è riportato in figura 18 dove è mostrata la trasformazione dell'immagine di un gatto dopo 2,7,100,116,348 passi. Dopo 116 passi il gatto ricompare in modo parziale, dopo 348 passi compare identico all'immagine originale. La foto, presa dal sito di Jason Davies `http://www.jasondavies.com/catmap/`, proviene da `https://www.flickr.com/photos/miss_pupik/`.

Come si fa a determinare il numero k di volte in cui la mappa di Arnold va applicata per poter riavere l'immagine originale, ammesso che k esista? Computazionalmente si può dare risposta a questa domanda in modo molto semplice.

Infatti, interpretando la mappa in forma matriciale, la trasformazione che abbiamo dopo k applicazioni della mappa di Arnold è data da

$$\begin{bmatrix} i \\ j \end{bmatrix} \to \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}^k \begin{bmatrix} i \\ j \end{bmatrix} \mod m$$

Per poter avere la trasformazione identica è sufficiente che

$$\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}^k = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mod m.$$

Listing 11: Function che calcola il più piccolo valore di k per cui l'applicazione di k passi della mappa di Arnold su una immagine $m \times m$ non altera l'immagine stessa.

```
function k = trova_esponente(m)
  M = [2 1 ; 1 1];
  P = eye(2);
  for j=1:m^3
    P = P*M;
    P = mod(P,m);
    diff = norm(eye(2)-P);
    if diff==0
      k = j+1;
      disp(k)
      break
    endif
  endfor
  if j>=m^3 && diff>0
    disp(''Non c'e' nessun intero k per cui M^k=I mod m'')
  endif
endfunction
```

Basta quindi controllare qual è il più piccolo valore intero di k per cui è verificata la condizione precedente. Una function che fa questo calcolo è riportata nel listato 11. In questa function si svolgono al più m^3 iterazioni, infatti M^k è una matrice simmetrica, e di matrici simmetriche con elementi nell'insieme $\{0, 1, \ldots, m-1\}$ ce ne sono al più m^3.

La tabella seguente riporta i valori dell'esponente k per alcuni valori di m.

m	200	201	202	203	204	205	206	207	208	209	210
k	150	68	75	56	36	20	312	24	84	45	120

Poiché l'esecuzione della function `arnold_cat` è molto lenta a causa dei cicli for annidati, conviene darne una versione vettoriale.

Ricordiamo che se (i, j) sono le coordinate del generico pixel dell'immagine originale, sempre considerate con origine 0, allora le coordinate del pixel trasformato sono

$$\begin{bmatrix} i' \\ j' \end{bmatrix} = M^k \begin{bmatrix} i \\ j \end{bmatrix} \mod m, \quad M = \begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix}.$$

Basta allora costruire una matrice S di dimensione $2 \times m^2$ che contenga sulla generica colonna la generica coppia di coordinate (i, j) per $i, j = 0, \ldots, m-1$. Per calcolare le coordinate di tutti i punti ottenuti dopo la trasformazione basta dunque calcolare $T = M^k S$. In questo modo non facciamo cicli `for` e il calcolo viene ottenuto mediante il prodotto di una matrice 2×2 e di una matrice $2 \times m^2$. Dobbiamo quindi costruirci prima la matrice S che ha per colonne le coppie di

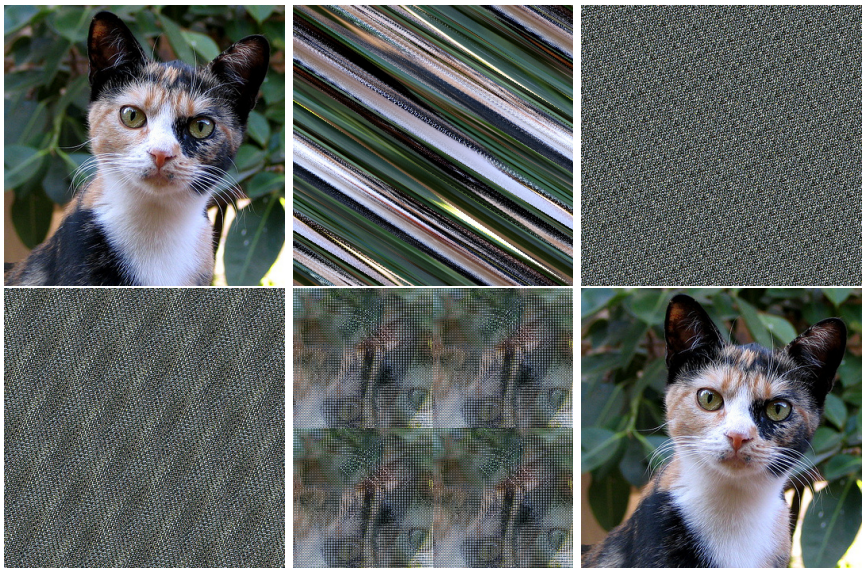


Figura 18: Immagini ottenute applicando la mappa di Arnold alla fotografia di un gatto con valori di $k = 2, 7, 100, 116, 348$.

indici. Ad esempio, per $m = 4$ si ha

$$S = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \end{array}\right].$$

Per costruire S ci sono almeno due modi equivalenti. Il primo consiste nell'usare il prodotto di Kronecker. Ricordiamo che il prodotto di Kronecker $w = u \otimes v$ di due vettori $u \in \mathbb{R}^m$ e $v \in \mathbb{R}^n$ è il vettore $w \in \mathbb{R}^{mn}$ di elementi

$$w = [u_1 v, u_2 v, \ldots, u_m v].$$

Analogamente, il prodotto di Kronecker di due matrici A e B di dimensioni rispettivamente $m \times n$ e $p \times q$ è la matrice $C = A \otimes B$ di dimensioni $(mp) \times (nq)$ definita da

$$C = A \otimes B = \begin{bmatrix} a_{1,1}B & \ldots & a_{1,m}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \ldots & a_{m,n}B \end{bmatrix}.$$

Il prodotto di Kronecker è legato alla funzione vec che applicata ad una matrice A di dimensioni $m \times n$ fornisce il vettore

$$\mathrm{vec}(A) = (a_{1,1}, a_{2,1}, \ldots, a_{m,1}, \ldots, a_{1,n}, a_{2,n}, \ldots, a_{m,n})^T$$

ottenuto giustapponendo una sull'altra le colonne della matrice A. Si può verificare direttamente che $\mathrm{vec}(uv^T) = u \otimes v$.

È evidente che la prima riga di S si può scrivere come $[1,1,1,1] \otimes [0,1,2,3]$, la seconda riga come $[0,1,2,3] \otimes [1,1,1,1]$. Questa proprietà vale in generale e nella sintassi di Octave la matrice S si può scrivere come:

```
S=[kron(ones(1,m),[1:m]) ; kron([1:m],ones(1,m))];
```

Infatti il comando `w = kron(u,v):` fornisce il prodotto di Kronecker degli array `u` e `v`.

Un secondo modo per costruire S consiste nell'usare il comando `vec` che applicato ad una matrice A fornisce il vettore colonna ottenuto giustapponendo le colonne di A. In questo modo la prima riga di S si può scrivere come `vec([0:m-1]'*ones(1,n))'` mentre la seconda riga si può scrivere come `vec(ones(m,1)*[0:n-1])'`

Tenendo conto che operiamo modulo m e quindi che gli indici stanno nel range $\{0,1,\ldots,n-1\}$, le istruzioni che effettuano la trasformazione di coordinate con la mappa di Arnold diventano allora

```
S = [kron(ones(0,m-1),[0:m-1]) ; kron([0:m-1],ones(0,m-1))];
M = [2 1;1 1];
T = mod(M*S,m);
```

dove la prima istruzione può essere sostituita da

```
S =[ vec([0:m-1]'*ones(1,n))' ; vec(ones(m,1)*[0:n-1])' ];
```

A questo punto la matrice T ha per colonne gli indici delle coordinate, con origine in 0, dei trasformati con la mappa di Arnold dei corrispondenti indici. Ci rimane allora il compito di trasformare i pixel dell'immagine usando i valori trasformati degli indici.

Una possibilità per trasformare l'immagine `a` in `b` consiste nello scorrere tutti gli indici e copiare i corrispondenti elementi da `a` a `b`. Cioè è sufficiente scrivere

```
for h=1:m^2
  b(T(1,h),T(2,h),:) = a(S(1,h),S(2,h),:);
endfor
```

dove abbiamo assunto che le immagini siano a colori e quindi le matrici e `b` abbiano 3 indici. In questo modo introduciamo nuovamente un ciclo `for` che consuma tempo di CPU, ma si può ovviare a questo inconveniente. L'idea si basa su questa proprietà del linguaggio Octave:

Se `u` è un vettore di `m` componenti e se `v` è un vettore di n componenti intere comprese tra 1 e `m`, allora `w=u(v)` è il vettore di `n` componenti date da `w(i)=u(v(i))`. Ad esempio, se `u=[4 5 6 7]` e `v=[1 1 2 2 2 4]` allora `u(v)` è il vettore

$$[4\ 4\ 5\ 5\ 5\ 7].$$

Quindi se si tratta di permutare gli elementi di un vettore u con una permutazione data dalle componenti di un vettore v è sufficiente scrivere `u(v)` per avere il vettore con le componenti permutate.

La proprietà vale anche per matrici `U` e `V` se queste le interpretiamo come vettori ottenuti giustapponendo le colonne una dopo l'altra. Considerando la matrice $U = (u_{i,j})$ come vettore $a = (a_i)$, ottenuto come descritto sopra, allora vale $a_{(j-1)*m+i} = u_{i,j}$. Cioè le coordinate bidimensionali (i, j) di U si trasformano nelle coordinate monodimensionali $s = (j-1) * m + i$ del vettore a.

Quindi, se `U` è $m \times n$ e `V` è $p \times q$ con elementi compresi tra 1 e mn, allora `W=U(V)` è la matrice $p \times q$ tale che `W(h,k)=U(i,j)`, dove `i` e `j` si ottengono dalla decomposizione di `V(h,k)=m(j-1)+i`.

Ad esempio, vale

$$\texttt{U} = \begin{bmatrix} a & d & g \\ b & e & h \\ c & f & i \end{bmatrix}, \quad \texttt{V} = \begin{bmatrix} 4 & 9 \\ 9 & 1 \\ 5 & 6 \end{bmatrix}, \quad \texttt{U(V)} = \begin{bmatrix} d & i \\ i & a \\ e & f \end{bmatrix}$$

In Octave la trasformazione di una matrice in un vettore si esegue col comando `vec`. Infatti `v=vec(A)` dà un vettore `v` tale che `v(q)=A(i,j)` con `q=m(j-1)+i`, dove `A` è matrice $m \times m$. Il vettore `v` si ottiene giustapponendo le colonne di `A`. L'operazione inversa la svolge il comando `A=reshape(v,m,n)`. Ad esempio, data la matrice `U` di dimensioni `m`×`n`, se poniamo `V=reshape([1:m*n],m,n)` e calcoliamo `W=U(V)` otteniamo ancora `U`.

Osserviamo che `V=reshape([1:m*n],m,n);` fornisce una matrice `V` che nell'elemento di posto (i, j) ha il valore $(j-1) * m + i$, cioè l'indice ottenuto con l'ordinamento per colonne. Ad esempio, con $m = n = 3$ si ha

$$V = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

In questo modo, se `A` è una matrice $m \times n$ allora `A(V)` dà nuovamente `A`.

Data la matrice S che per colonne ha le coppie (i, j), con ordinamento degli indici "per colonne" con origine degli indici in 0, ad esempio per $n = 4$

$$S = \left[\begin{array}{cccc|cccc|cccc|cccc} 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 & 0 & 1 & 2 & 3 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 3 & 3 & 3 & 3 \end{array}\right],$$

allora `vs = S(1,:)+m*S(2,:)+1` è il vettore `[1:m*n]` che è associato alla trasformazione identica mentre `vt = T(1,:)+m*T(2,:)+1` è il vettore associato alla trasformazione di Arnold. Quindi per realizzare in forma vettoriale la costruzione dell'immagine ottenuta dopo la trasformazione di Arnold avendo disponibile la matrice `T` basta scrivere

```
vt = T(1,:)+m*T(2,:)+1;
B(reshape(vt,m,m)) = A;
```

La function che implementa la trasformazione di Arnold in modo vettoriale è riportata nel listato 12

Listing 12: Function che applica k volte la mappa di Arnold all'immagine `a`: versione vettoriale

```
function b = arnold_cat1(a,k)
% versione vettoriale della trasformazione di Arnold
  s = size(a); m = s(1);
  b = a;
% calcola la matrice P = M^k mod m
  M = [2 1;1 1]; P=eye(2);
  for j=1:k
     P = mod(M*P, m);
  endfor
% crea una matrice S le cui colonne sono tutte le coppie di indici (i,j)
% con l'ordinamento "per colonne" e con origine degli indici in 0
  S = [kron(ones(1,m),[0:m-1]);kron([0:m-1],ones(1,m))];
% trasforma simultaneamente le colonne con la matrice P
  T = mod(P*S,m);
  vt = trasf(2,:)*m+trasf(1,:) + 1;
  vt = reshape(vt,[m,m]);
  if length(s)==3 % immagine a colori RGB
    for j=1:3
      F = a(:,:,j);
      F(vt) = F;
      b(:,:,j) = F;
    endfor
  else % immagine in bianco nero
    b(vt) = a;
  endif
endfunction
```

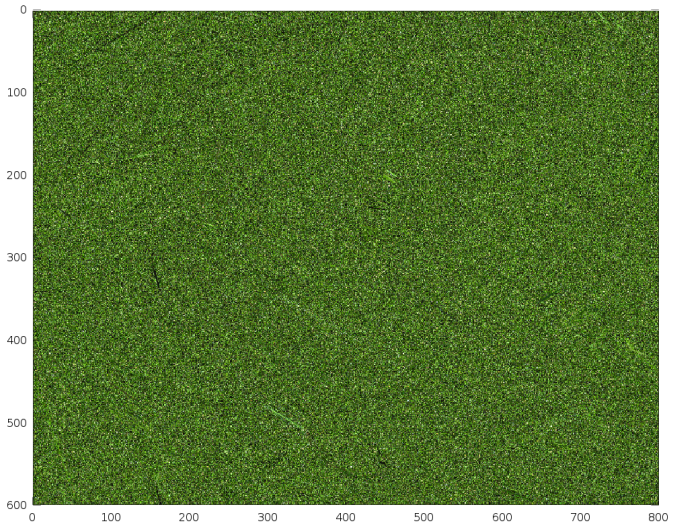
4.2 Un'altra trasformazione

Possiamo costruire trasformazioni analoghe. Ad esempio, lasciamo inalterata la prima colonna di pixel di una immagine mentre la colonna di indice j la solleviamo di $j-1$ posizioni per $j = 1, \ldots, [n/2]$, dove $[n/2]$ è la parte intera di $n/2$, mentre la alziamo di $2[n/2] - j - 1$ posizioni per $j = [n/2]+1, \ldots, n$. Il triangolo che fuoriesce dalla parte superiore dell'immagine lo facciamo rientrare dalla parte inferiore. È come se avessimo applicato la trasformazione ad una fotografia arrotolata in modo che il bordo superiore si incolli al bordo inferiore. Formalmente spostiamo il pixel di coordinate (i, j) in (i', j) dove $i' = (i - j \mod m) + 1$ per $j = 1, \ldots, [n/2]$, mentre $i' = (2 * [n/2] - j \mod m) + 1$ se $j = [n/2]+1, \ldots, n$.

Qui sotto mostriamo come agisce questa trasformazione su una foto scattata ad una mantide religiosa.

Ora vogliamo applicare questa trasformazione anche sulle righe traslando i pixel a sinistra con la stessa regola. Quindi vogliamo applicare un certo numero di queste trasformazioni per vedere se i pixel dell'immagine così ottenuta si mescolano abbastanza in modo da non fare più riconoscere l'immagine.

Ecco cosa succede dopo 10 trasformazioni e dopo 50 trasformazioni con la foto della mantide religiosa.



L'immagine ottenuta dopo 50 passi è completamente irriconoscibile. Essa potrebbe essere ricostruita applicando 50 volte le trasformazioni inverse.

Vediamo ora alcune semplici manipolazioni. È possibile ruotare una immagine attorno al pixel di coordinate (i_0, j_0) di un angolo α in senso antiorario semplicemente applicando una rotazione alle coordinate di ciascun pixel. Prima di vedere questo, osserviamo che nella nostra notazione matriciale la coppia (i, j) denota l'elemento della riga i e della colonna j. In questo modo l'origine degli assi è posta nell'angolo in alto a sinistra dell'immagine col primo asse che punta verso il basso e il secondo che punta a destra. Detto questo, ricordiamo che le coordinate del punto $P' = (x', y')$ ottenuto ruotando attorno a (x_0, y_0) di un angolo α in senso antiorario il punto P di coordinate (x, y) sono

$$\begin{aligned} x' - x_0 &= (x - x_0)\cos\alpha - (y - y_0)\sin\alpha \\ y' - y_0 &= (x - x_0)\sin\alpha + (y - y_0)\cos\alpha \end{aligned}$$

Quindi, si ottiene

$$\begin{aligned} i' - i_0 &= (i - i_0)\cos\alpha - (j - j_0)\sin\alpha \\ j' - j_0 &= (i - i_0)\sin\alpha + (j - j_0)\cos\alpha \end{aligned}$$

Osserviamo che se (i, j) è una coppia di interi non è detto che (i', j') sia ancora una coppia di interi.

Per meglio descrivere la function di rotazione denotiamo con $A = (a_{i,j})$ la matrice corrispondente all'immagine di partenza, e con $B = (b_{i,j})$ la matrice corrispondente all'immagine ruotata.

Allora, per eseguire la rotazione in modo più efficace, scandiamo tutte le coppie intere (i', j') corrispondenti ai pixel del supporto dell'immagine ruotata,

Listing 13: Function che ruota una immagine attorno al pixel `(p(1),p(2))` di un angolo `ang`.

```
function y = ruota(x, p, ang)
% ruota un'immagine data dalla matrice x attorno al pixel p
% di un angolo ang in senso antiorario
   m = size(x)(1);
   n = size(x)(2);
   i0 = p(1); j0 = p(2);
   for ip = 1 : m % seleziona il pixel di posizione (ip,jp)
      for jp = 1 : n % dell'immagine ruotata
% calcola il pixel di posizione (i,j) dell'immagine di provenienza
         i = round(i0+(ip-i0)*cos(ang)+(jp-j0)*sin(ang));
         j = round(j0-(ip-i0)*sin(ang)+(jp-j0)*cos(ang));
         if(i>0 && i<=m && j>0 && j<=n)
% se il pixel di provenienza sta nel supporto dell'immagine
% prende il valore dell'immagine
           y(ip,jp) = x(i,j);
         else
% altrimenti prende il valore nullo (nero)
           y(ip,jp) = 0;
         endif
      endfor
   endfor
endfunction
```

applichiamo la trasformazione inversa ottenendo la coppia (i, j), non necessariamente intera, e andiamo a riempire l'elemento $b_{i',j'}$ col valore di $a_{[i],[j]}$, dove $[i]$ e $[j]$ denotano gli arrotondamenti di i e j alla parte intera. Può accadere che la coppia $([i], [j])$ non stia nel supporto dell'immagine, cioè che $[i]$ non sia compreso tra 1 e n, e $[j]$ non sia compreso tra 1 e m. In tal caso assegnamo a $b_{i',j'}$ il valore 0.

Scriviamoci qui sotto le formule inverse della rotazione cioè

$$\begin{aligned} i - i_0 &= (i' - i_0)\cos\alpha + (j' - j_0)\sin\alpha \\ j - j_0 &= -(i' - i_0)\sin\alpha + (j' - j_0)\cos\alpha \end{aligned}$$

Un programma che realizza questa trasformazione è riportato nel listato 13 dove abbiamo usato le variabili `ip,jp` per denotare i' e j'.

Ecco il risultato che si ottiene con `angolo = 1` radiante a partire da un'immagine originale di cammelli.

Sapreste deformare l'immagine in modo che l'angolo di rotazione sia inversamente proporzionale alla distanza del pixel dal centro dell'immagine di coordinate (i_0, j_0)? Per questo basta cambiare la parte in cui si calcolano i e j ad esempio con

```
s = sqrt((i0-ip)^2 + (j0-jp)^2))/10 + 0.01;
a = 6.28/s;
i = i0 + (ip-i0)*cos(a) + (jp-j0)*sin(a);
j = j0 - (ip-i0)*sin(a) + (jp-j0)*cos(a);
```

Il numero 0.01 che è stato aggiunto alla variabile s, serve per evitare situazioni di singolarità nel centro di rotazione. Ecco il risultato che si ottiene

Naturalmente la presenza dei due cicli for annidati nella function `ruota` rallenta molto l'esecuzione del programma. è possibile però dare una versione vettorizzata di questa function.

Un possibile modo per svolgere la rotazione di una immagine senza usare cicli for procede con i seguenti passi. L'obiettivo è descrivere in modo compatto la trasformazione $(i', j') \to (i, j)$ dove (i, j) è la generica coppia di indici del generico pixel dell'immagine di partenza e (i', j') è la corrispondente coppia di indici del pixel dell'immagine trasformata.

1. si costruisce una matrice S di dimensione $2 \times mn$ che ha per colonne tutte le coppie (i, j) con $i = 1, \ldots, m$, $j = 1, \ldots, n$. Come ordinamento, fissiamo prima l'indice di colonna j e poi facciamo scorrere l'indice di riga i, cioè procediamo per colonne nello scandire le coordinate dell'immagine.

2. Costruiamo la matrice T di dimensione $2 \times mn$ che nella generica colonna k-esima ha gli indici (i, j) dove la k-esima colonna di S ha gli indici (i', j'). Nel caso della rotazione attorno a `(i0,j0)` di un angolo alfa, basta costruire la matrice `R=[cos(alfa), sin(alfa); -sin(alfa), cos(alfa)]` e calcolare

   ```
   T = [i0;j0]*ones(1,m*n) + R*(S - [i0;j0]*ones(1,m*n))
   ```

   Questo implementa in modo vettoriale le formule di rotazione, cioè abbiamo descritto la rotazione come trasformazione di (i, j) in (i', j') in modo compatto come $S \to T$. Naturalmente poiché i nostri pixel hanno coordinate intere dobbiamo arrotondare il risultato di T alla parte intera.

3. Per realizzare la trasformazione dei pixel dell'immagine avendo a disposizione quella delle coordinate basta procedere così .

   (a) si trasformano le coordinate bidimensionali racchiuse nelle colonne di T in coordinate monodimensionali con l'ordinamento per colonne, cioè si applica la trasformazione $(i, j) \to i + (j-1)m$. Per questo si costruisce il vettore `Z=(T(2,:)-1)*m + T(1,:)`.

   (b) si dimensiona `Z` a matrice $m \times n$ come `Z=reshape(T,m,n)`. La matrice `Z` avrà come elemento di posto (i', j') il valore $i + (j-1)m$ dove ricordiamo che (i', j') è il trasformato di (i, j)

   (c) si calcola infine `Y=X(Z)`

Se applichiamo il metodo così come l'abbiamo descritto è molto probabile che Octave ci segnali degli errori in esecuzione. Il problema è che dopo la rotazione alcune coordinate di pixel possono cadere fuori dal supporto dell'immagine, cioè i loro valori numerici possono non appartenere ai segmenti $[1 : m]$ per le righe e $[1 : n]$ per le colonne. In questo caso Octave ci segnalerebbe un errore. Occorre allora rimediare all'inconveniente modificando in modo opportuno il punto 3. Allora si procede così

3.00 individuiamo i valori degli indici che cadono fuori del supporto $[1, m] \times [1, n]$; per questo trasformiamo in 0 tutti i valori di `T` minori o uguali a 0, trasformiamo in $m + 1$ gli indici della prima riga di `T` maggiori di m, e trasformiamo in $n + 1$ gli indici della seconda riga di `T` maggiori di n;

3.01 l'immagine `X` di dimensione $m \times n$ la bordiamo di zeri copiandola in una matrice `XX` di dimensioni $(m+2) \times (n+2)$, allo stesso tempo incrementiamo i valori di `T` di uno. In tal modo gli indici di riga o di colonna uguali a 1 individuano punti fuori del supporto e corrisponderanno a elementi (pixel) della cornice aggiunta all'immagine. Analogamente, indici di riga uguali a $m + 2$ o di colonna uguali a $n + 2$ individuano punti fuori dal supporto che corrisponderanno a elementi della cornice aggiunta all'immagine. Si pone `Y=XX`.

Il programma che si ottiene è riportato nel listato 14

La rotazione è una semplice trasformazione di coordinate. è possibile costruire trasformazioni arbitrarie, basta dare spazio alla fantasia. Ad esempio provate a spostare un pixel (i, j) in un altro pixel (i', j') scelto a caso con la condizione $\max\{|i - i'|, |j - j'|\} \leq 3$.

Un insieme di trasformazioni interessanti si ottiene usando funzioni di variabile complessa. Questo lo vediamo tra poco. Un'altro esempio interessante che richiede un po' di modellazione matematica è il caso di immagini anamorfiche ottenute mediante riflessioni su specchi non piani.

Immagini anamorfiche. Un modo interessante per applicare deformazioni ad una immagine assegnata consiste nel fare riflettere l'immagine originale su uno specchio cilindrico o su una sfera. Ad esempio, supponete di avere uno

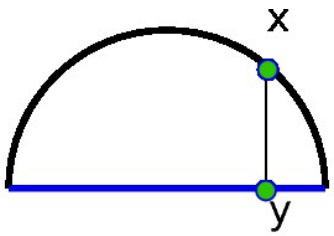
Listing 14: Function che ruota una immagine attorno al pixel `(p(1),p(2))` di un angolo `ang`. Versione vettoriale.

```
function Y = ruota(X, p, ang)
% function Y = ruota(X, p, ang)
% ruota l'immagine X attorno a p di un angolo ang
% in modo antiorario
   m = size(X,1); n=size(X,2);
   i0 = p(1); j0=p(2);
   vi = kron(ones(1,m),[1:m]);
   vj = kron([1:m],ones(1,m));
% trasforma
   S = [vi-i0;vj-j0];
   R = [cos(ang) sin(ang);-sin(ang) cos(ang)];
   T = R*S + [i0;j0]*ones(1,n*m);
   T = round(T);
% controllo delle coordinate fuori dal supporto
   T = max(T,0);
   T(1,:) = min(T(1,:),m+1);
   T(2,:) = min(T(2,:),n+1);
   T = T + 1;
   XX = zeros(m + 2, n + 2);
   XX(2:m+1, 2:n+1) = X;
   Z = (m + 2)*(T(2,:)-1) + T(1,:);
   Z = reshape(Z,m,n);
   Y = XX(Z);
endfunction
```

specchio cilindrico posto su un piano orizzontale P e con l'asse verticale. Considerate un generico raggio che parte dal vostro occhio, colpisce la superficie del cilindro, viene riflesso e interseca il piano orizzontale P in un punto p. Ora prolungate lo stesso raggio oltre il cilindro, senza essere riflesso, fino a intersecare in un punto q un piano obliquo Q che avete posto dietro il cilindro. L'immagine costituita dai punti p del piano orizzontale riflessi dal cilindro viene percepita come l'immagine costituita dai punti q posti nel piano obliquo dietro il cilindro. Sapreste scrivere le relazioni che legano le coordinate di p e di q rispettivamente nel piano P e nel piano Q? Se siete in grado di fare questo, potete modificare il programma ruota in modo che, data un'immagine posta nel piano orizzontale, il programma generi l'immagine riflessa dal cilindro. Viceversa, data una fotografia, potete costruire una immagine opportunamente deformata da collocare nel piano orizzontale in modo che guardando la versione riflessa dal cilindro si veda la fotografia originale.

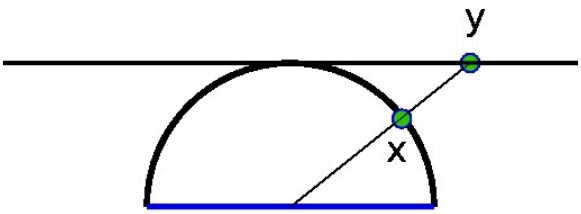
Esercizi un po' più semplici sono i seguenti:

Esercizio E0. Simulare come viene trasformata una fotografia se viene arrotolata attorno ad un semicilindro di altezza pari ad una delle sue dimensioni e i suoi punti vengono proiettati ortogonalmente su AB come in figura dove x è il generico punto dell'immagine e y quello dell'immagine trasformata. Cioè data la foto nera costruire la foto blu.



Esercizio E1. Simulare la trasformazione inversa dell'esercizio E1. Cioè , data la foto blu nel piano generare la foto nera sul cilindro.

Esercizio E2. Con riferimento all'esercizio E1 simulare come viene trasformata una fotografia se il generico punto x dell'immagine originale viene trasformato in y come in figura.



Esercizio E3. Simulare la trasformazione inversa dell'esercizio E3. Cioè , data la foto posta nel piano tangente generare la foto sul cilindro.

4.3 Manipolazioni geometriche con funzioni di variabile complessa

Interessanti trasformazioni si ottengono utilizzando funzioni di variabile complessa. Data un'immagine costituita dall'insieme dei pixel $a(i,j)$, per $i = 1,\ldots,m$, $j = 1,\ldots,n$ associamo alla coppia (i,j) un numero complesso $z = z(i,j)$, ad esempio $z = (j-j_0)h - \mathbf{i}(i-i_0)h$ dove abbiamo denotato con $\mathbf{i}$ l'unità immaginaria tale che $\mathbf{i}^2 = -1$, e dove abbiamo scelto i_0, j_0 e h a nostro piacere. Oltre all'applicazione $(i,j) \to z(i,j)$ consideriamo l'applicazione "inversa" $z(i,j) \to (i,j)$ tale che j è la parte intera di $\mathrm{Re}(z)/h + j_0$, mentre i è la parte intera di $i_0 - \mathrm{Im}(z)/h$, dove abbiamo indicato con Re(.) e Im(.) la parte reale e la parte immaginaria di un numero complesso. Consideriamo poi una funzione di variabile complessa $y = f(z)$, ad esempio $f(z) = z^2/|z|$, e costruiamo questa trasformazione tra coppie di interi (i',j') e (i,j) $(i,j) \to z(i,j)$, $w = f(z)$, $w \to (i',j')$ Costruiamo l'immagine che nel pixel di coordinate (i,j) ha il valore $a(i',j')$ se (i',j') appartiene a $[1,m] \times [1,n]$, ha il valore 0 altrimenti. Come è fatta l'immagine deformata in questo modo? Cosa accade con semplici funzioni quali $f(z) = zt$, se t è un numero complesso di modulo 1, oppure se t è un numero reale positivo? Cosa accade con funzioni più complesse quali $f(z) = z(z/|z|)^k$, per k intero? Oppure con $f(z) = z - (z^3-1)/(2z^2)$?

Questa e una traccia di programma Octave per deformare immagini

```
function Y = complextransform(X)
   m = size(X)(1);
   n = size(X)(2);
   h = 2/max(m,n);
   Y = zeros(m,n);
   i0 = round(m/2); j0 = round(n/2);
% scandisco supporto della foto trasformata
   for ip = 1 : m
```

```
      for jp = 1 : n
% calcolo il numero complesso corrispondente
         z = (jp-j0)*h - I*(ip-i0)*h;
% trasformo all'indietro
         w=f(z);
% calcolo le coordinate del pixel corrispondente a w
         i = round(i0-imag(w)/h);
         j = round(j0+real(w)/h);
% assegno il colore
         if(1<=i && i<=m && 1<=j && j<=n)
            Y(ip,jp) = X(i,j);
         else
            Y(ip,jp)=0;
         endif
      endfor
   endfor
endfunction

function w = f(z)
   if z!=0
      w = z*(z/abs(z))^2;
   else
      w = 0;
   endif
endfunction
```

Ecco alcuni esempi ottenuti con varie funzioni

Provare a modificare la function `complextransform` in modo che prenda in input, oltre alla matrice X di dimensioni $m \times n$, i seguenti valori:
zc: numero complesso, centro dell'immagine originale
h: numero reale, dimensione del pixel (l'immagine sul piano complesso ha dimensioni $hm \times hn$)
wc: numero complesso, centro dell'immagine deformata
k: numero reale, dimensione del pixel dell'immagine trasformata (l'immagine trasformata sul piano complesso ha dimensioni $km \times kn$)

e colloca l'immagine X di dimensioni $m \times n$ che vogliamo deformare, nel piano complesso centrata in zc con dimensioni $mh \times nh$ e poi costruisce l'immagine deformata come risulta nella porzione di piano complesso di centro wc e dimensioni $mk \times nk$. Quindi in uscita dà ancora una matrice $m \times n$ che corrisponde all'immagine così costruita.

Di seguito si riporta una function per costruire immagini ottenute con trasformazioni di variabile complessa evitando cicli for. La function è ottenuta modificando leggermente la function ruota nella versione vettoriale. È stata aggiunta anche la lunghezza di un lato della foto nel piano complesso che determina la dimensione h di ciascun pixel.

```
function Y = complextransform2(X, p, lato)
% trasforma l'immagine X mediante funzione di variabile
% complessa
% X: immagine
% p=[i0,j0]: coordinate del pixel in cui e' messa l'origine
% dimensione di un lato dell'immagine nel piano complesso
% nel piano complesso
% il programma e' un adattamento della function ruota2
   m = size(X,1); n=size(X,2);
   h = lato/m;  % dimensione del pixel
   i0 = p(1); j0=p(2);
   vi = kron(ones(1,m),[1:m]);
   vj = kron([1:m],ones(1,m));
% trasforma
   S = [vi-i0;vj-j0];
   z = S(2,:)+I*S(1,:);
   z = z*h;
   z = z.*(z./(abs(z)+1.0e-16)).^2;  % funzione
   T = zeros(2,m*n);
   T(2,:) = real(z);  T(1,:) = imag(z);
   T = round(T/h) + [i0;j0]*ones(1,n*m);
%
% controllo delle coordinate fuori dal supporto
   T = max(T,0);
   T(1,:) = min(T(1,:),m+1);
   T(2,:) = min(T(2,:),n+1);
```

```
  T = T + 1;
  XX = zeros(m + 2, n + 2);
  XX(2:m+1, 2:n+1) = X;
  Z = (m + 2)*(T(2,:)-1) + T(1,:);
  Z = reshape(Z,m,n);
  Y = XX(Z);
endfunction
```

Esercizio F0. Simulare l'azione di una lente di ingrandimento trasformando una immagine in modo da ingrandirne una parte con continuità , e quindi senza strappi, lasciando la parte rimanente inalterata.

Le seguenti immagini sono state create da studenti del corso dell'aa. 2008-2009.



Esercizio F1. Realizzare la rotazione di una immagine usando una opportuna funzione di variabile complessa.

Esercizio F2. Realizzare la trasformazione di una immagine usando la funzione esponenziale e la funzione logaritmo.

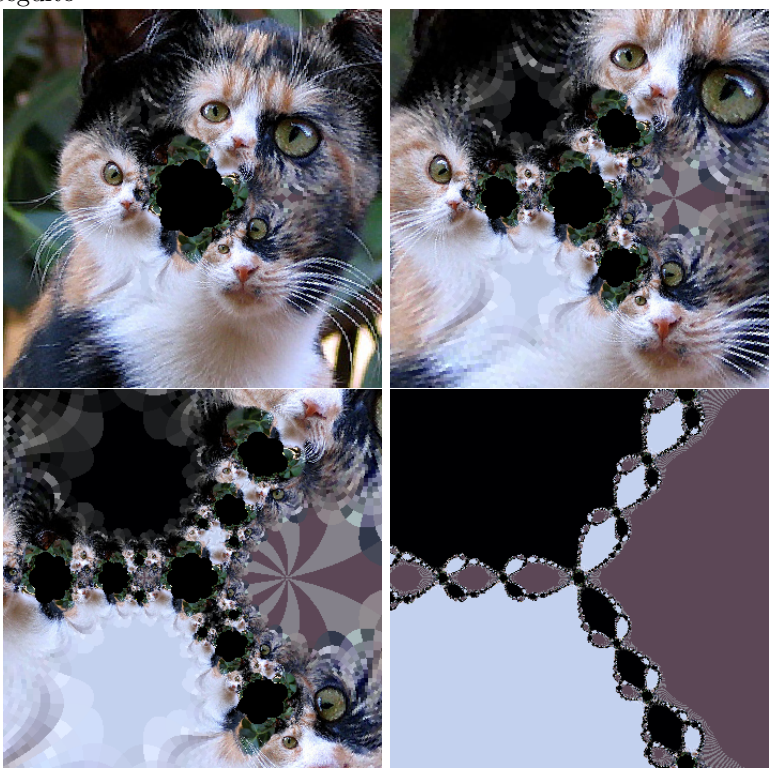
Esercizio F3. Realizzare la trasformazione di una immagine usando la funzione z^k e $z^{1/k}$

Esercizio F4. Realizzare la trasformazione di una immagine usando la trasformata di Cayley $z \to (1-z)/(1+z)$.

Esercizio F5. Realizzare la trasformazione di una immagine usando la funzione di Joukowski $z \to (z + a/z)/2$, con a numero reale.

Esercizio F6. Realizzare la trasformazione di una immagine usando come trasformazione inversa separatamente una, due e tre iterazioni del metodo di Newton applicato a $x^3 - 1$.

Le immagini ottenute in questo caso con 1,2,3, e 10 iterazioni sono riportate di seguito



Esercizio F7. Realizzare la trasformazione di una immagine usando come trasformazione inversa separatamente una, due e tre iterazioni del metodo di Newton applicato a $x^2 - 1/x$.

Esercizio F8. Realizzare la trasformazione di una immagine usando come trasformazione inversa separatamente una, due e tre iterazioni del metodo di Newton applicato a $1 - 1/x^3$.

Esercizio F9. Realizzare la trasformazione di una immagine usando come trasformazione inversa separatamente una, due e tre iterazioni del metodo di Newton applicato a $x - 1/x^2$.

5 Decomposizione spettrale, filtraggio e compressione

In questa parte del laboratorio ci occupiamo di manipolazioni di immagini ottenute mediante la trasformata discreta di Fourier (DFT). In letteratura le notazioni relative alla DFT cambiano a seconda del contesto in cui viene applicata. Nel nostro caso conviene usare le notazioni e le definizioni che sono adottate nella implementazione della DFT nel linguaggio Octave. Questo ci permette di mantenere coerenza tra teoria e implementazione.

Ricordiamo che in Octave, dato un vettore `v` di n componenti, il comando

```
u = fft(v);
```

fornisce la trasformata discreta di Fourier, calcolata con gli algoritmi veloci FFT (Fast Fourier Transform), definita da:

$$u_k = \sum_{j=0}^{n-1} (\cos \frac{2\pi}{n} kj - \mathbf{i} \sin \frac{2\pi}{n} kj) v_j, \quad k = 0, \ldots, n-1,$$

dove al solito $\mathbf{i}$ denota l'unità immaginaria. Mentre il comando

```
v = ifft(u);
```

fornisce la trasformata inversa discreta di Fourier definita da

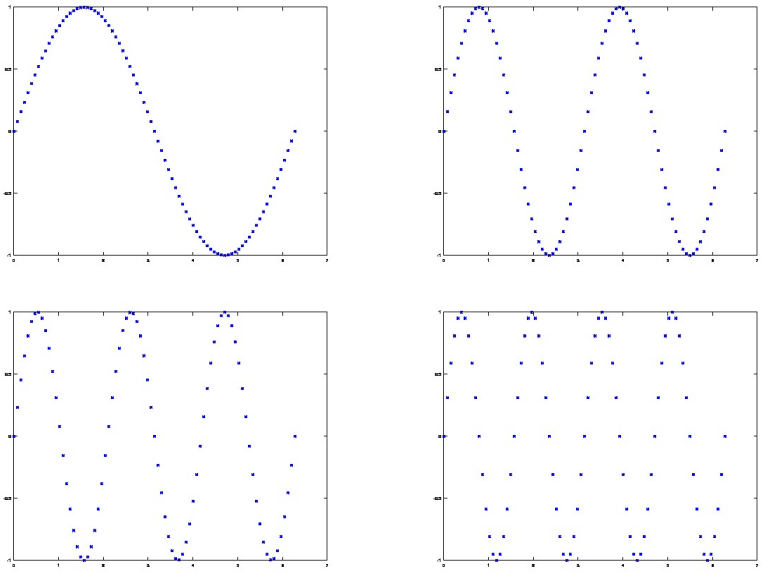
$$v_k = \frac{1}{n} \sum_{j=0}^{n-1} (\cos \frac{2\pi}{n} kj + \mathbf{i} \sin \frac{2\pi}{n} kj) u_j, \quad k = 0, \ldots, n-1.$$

Le due trasformate sono l'una l'inversa dell'altra per cui vale `ifft(fft(u))=u` a meno di errori di roundoff dovuti all'aritmetica floating point.

5.1 Decomposizione spettrale

Sia n un intero positivo che supponiamo per semplicità pari, cioè $n = 2m$. Consideriamo i vettori $c^{(j)}$, $j = 0, 1, \ldots, m$, $s^{(j)}$, $j = 1, \ldots, m-1$, di n componenti definiti da $c_i^{(j)} = \cos(2ij\pi/n)$, $s_i^{(j)} = \sin(2ij\pi/n)$, $i = 0, \ldots, n-1$. Ciascuno di questi vettori si ottiene calcolando rispettivamente le funzioni $\cos(jx)$ e $\sin(jx)$ nei punti $x_i = 2i\pi/n$, $i = 0, 1, \ldots, n-1$. Come dicono gli ingegneri i vettori $s^{(j)}$ e $c^{(j)}$ sono i campionamenti delle funzioni $\cos(jx)$ e $\sin(jx)$ nei nodi x_i. Quindi i vettori $c^{(j)}$ e $s^{(j)}$ rappresentano in termini discreti le funzioni $\cos(jx)$ e $\sin(jx)$ che hanno periodo $2\pi/j$ e quindi frequenza $j/2\pi$.

Qui sotto si riportano graficamente i vettori $s^{(1)}$, $s^{(2)}$, $s^{(3)}$, $s^{(4)}$ per $n = 80$.



Si può verificare che gli n vettori così costruiti sono linearmente indipendenti per cui ogni vettore $v = (v_i)$ di n componenti può essere rappresentato nella base costituita da questi vettori speciali. Vale cioè

$$v = \frac{1}{2}a_0c^{(0)}+(a_1c^{(1)}+b_1s^{(1)})+(a_2c^{(2)}+b_2s^{(2)})+\cdots+(a_{m-1}c^{(m-1)}+b_{m-1}s^{(m-1)})+\frac{1}{2}a_mc^{(m)}$$

Nel caso di n dispari, cioè $n = 2m - 1$ i vettori $c^{(j)}$, $j = 0, ..., m - 1$, $s^{(j)}$, $j = 1, \ldots, m - 1$, sono ancora linearmente indipendenti e vale una espressione analoga alla precedente, cioè

$$v = \frac{1}{2}a_0c^{(0)}+(a_1c^{(1)}+b_1s^{(1)})+(a_2c^{(2)}+b_2s^{(2)})+\cdots+(a_{m-1}c^{(m-1)}+b_{m-1}s^{(m-1)})$$

Questa rappresentazione del vettore v, sia che n sia par o dispari, la chiamiamo "rappresentazione spettrale".

Possiamo dare una formulazione più utile della rappresentazione spettrale nel modo seguente. Si consideri l'espressione

$$a\cos x + b\sin x = \sqrt{a^2+b^2}(\frac{a}{\sqrt{a^2+b^2}}\cos x + \frac{b}{\sqrt{a^2+b^2}}\sin x).$$

Ponendo $A = \sqrt{a^2+b^2}$ e $B \in [-\pi, \pi]$ tale che $\cos B = \frac{a}{\sqrt{a^2+b^2}}$, $\sin B = -\frac{b}{\sqrt{a^2+b^2}}$, si ha

$$a\cos x + b\sin x = A\cos(x+B).$$

Per cui la rappresentazione di un vettore nella base speciale di seni e di coseni permette di considerare un vettore generico come somma di "vettori coseni"

oscillanti con frequenze multiple intere della frequenza $1/(2\pi)$, di ampiezze date da $A_j = (a_j^2 + b_j^2)^{1/2}$ e da fasi date da $B_j = \arctan(-b/a)$. Infatti, la rappresentazione data si può scrivere come (nel caso pari in cui $n = 2m$)

$$\begin{aligned} v_k =& \frac{1}{2}a_0 + A_1\cos(k\frac{2\pi}{n} + B_1) + A_2\cos(2k\frac{2\pi}{n} + B_2) + A_3\cos(3k\frac{2\pi}{n} + B_3) + \cdots \\ &+ A_{m-1}\cos((m-1)k\frac{2\pi}{n} + B_{m-1}) + \frac{1}{2}a_m\cos(mk\frac{2\pi}{n}) \end{aligned}$$

Il calcolo a basso costo dei coefficienti a_j, b_j e quindi delle ampiezze A_j e le fasi B_j della rappresentazione spettrale di un vettore v si può effettuare mediante la FFT.

Infatti, se v è reale e $u =$fft(v), cioè

$$u_k = \sum_{j=0}^{n-1}(\cos(2\pi kj/n) - \mathbf{i}\sin(2\pi kj/n))v_j, \quad k = 0, 1, \ldots, n-1$$

vale la seguente proprietà (si veda R. Beilacqua et al. "Metodi Numerici", Zanichelli 1992):

$$u = \begin{cases} [u_0, u_2, \ldots, u_m, u_{m+1}, \bar{u}_m, \ldots, \bar{u}_1], \quad u_0, u_{m+1} \in \mathbb{R} & \text{se } n = 2m \\ [u_0, u_1, \ldots, u_m, \bar{u}_m, \ldots, \bar{u}_1], \quad u_0 \in \mathbb{R} & \text{se } n = 2m-1 \end{cases}$$

$$a_j = \frac{2}{n}\mathrm{Re}(u_j), \quad b_j = -\frac{2}{n}\mathrm{Im}(u_j),$$

dove $\bar{u}_j$ indica il coniugato del numero complesso u_j.

La rappresentazione spettrale è molto usata dagli ingegneri per filtrare segnali. Infatti ci permette di conoscere le ampiezze A_j e le fasi B_j con cui ogni singola frequenza elementare compare nel segnale. Con questa informazione possiamo fare manipolazioni molto efficaci di segnali, quali amplificare o ridurre le tonalità acute o gravi di un suono rappresentato in forma digitale mediante il vettore v o rimuovere rumore presente nel segnale.

Si veda il sito `http://www.westga.edu/~jhasbun/osp/Fourier.htm` per un grazioso strumento legato a queste rappresentazioni.

Ad esempio si può costruire un nuovo vettore w ottenuto amplificando le componenti in "bassa frequenza" sostituendo ad esempio i valori A_j con $2A_j$ se j è minore di una soglia fissata, ad esempio $n/4$. Analogamente possiamo amplificare o attenuare le alte o le medie frequenze e trasformare il vettore originale in diversi modi. Se il vettore rappresenta un segnale acustico, ad esempio un brano di musica, le operazioni descritte hanno l'effetto analogo a quello che si ottiene regolando i toni in un impianto hifi. Questo tipo di manipolazione numerica è esattamente quella che svolgono certe applicazioni che riproducono musica digitale su di un pc.

Questo filtraggio digitale si realizza col seguente schema:

– è dato il vettore v che rappresenta un segnale;

– si calcola $u = \text{fft}(v)$ e si ottiene la rappresentazione spettrale;

– si alterano come desiderato le ampiezze e le fasi delle componenti in frequenza trasformando quindi il vettore v in un vettore w;

– si calcola il nuovo segnale $y = \text{ifft}(w)$.

Si osserva che le quantità a_j e b_j sono relative alla componente in frequenza j-esima del segnale. Inoltre esse sono determinate dalla parte reale ed immaginaria delle componenti u_j del vettore u =fft(v). Evidenziamo questa relazione riportando un vettore di interi in cui si mette in luce la dipendenza della frequenza dalla componente del vettore trasformato. Si descrive il caso $n = 16$ e $n = 17$, in grassetto le componenti reali

Caso $n = 16$ $\quad [\mathbf{0}\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ \mathbf{8}\ \bar{7}\ \bar{6}\ \bar{5}\ \bar{4}\ \bar{3}\ \bar{2}\ \bar{1}]$

Caso $n = 17$ $\quad [\mathbf{0}\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ \bar{8}\ \bar{7}\ \bar{6}\ \bar{5}\ \bar{4}\ \bar{3}\ \bar{2}\ \bar{1}]$

Se ad esempio volessimo rimuovere da un vettore v di 16 componenti le componenti in frequenza relative alle frequenze 5, 6, 7, 8, dovremmo operare nel seguente modo:
-1- si calcola u =fft(v)
-2- si considera il vettore filtro $f = [\mathbf{1}\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ \mathbf{0}\ 0\ 0\ 0\ 1\ 1\ 1\ 1]$
-3- si moltiplica componente a componente u con f e si ottiene z (nella sintassi di Octave `z = u.*f;`)
-4- si antitrasforma z e si ottiene w =ifft(z)

dove ifft è la trasformata discreta inversa di Fourier. Il vettore w fornisce il segnale filtrato. Se ad esempio volessimo ridurre di un fattore 1/2 l'ampiezza delle componenti in frequenza 5,6, e annullare le componenti in frequenza 7 e 8 il vettore filtro sarebbe

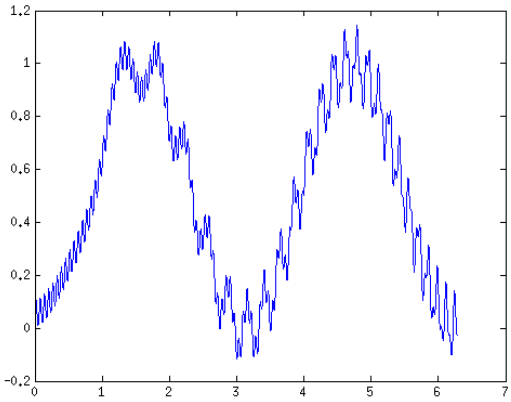
$$f = [\mathbf{1}\ 1\ 1\ 1\ 1\ \frac{1}{2}\ \frac{1}{2}\ 0\ \mathbf{0}\ 0\ \frac{1}{2}\ \frac{1}{2}\ 1\ 1\ 1\ 1]$$

Diamo ora un esempio di filtraggio di un segnale. Sia $n = 512$ il numero di punti di campionamento in cui si divide l'intervallo $[0, 2\pi]$. Poniamo `h=2pi/n` e `x=[0:h:2*pi-h]`. Scegliamo come funzione "segnale" $s = \sin(x)^2 + 0.1\cos(4x^2 - x + 1)$ e come rumore $r = 0.05\sin(100x)$ e consideriamo la funzione segnale più rumore $y = s + r$.

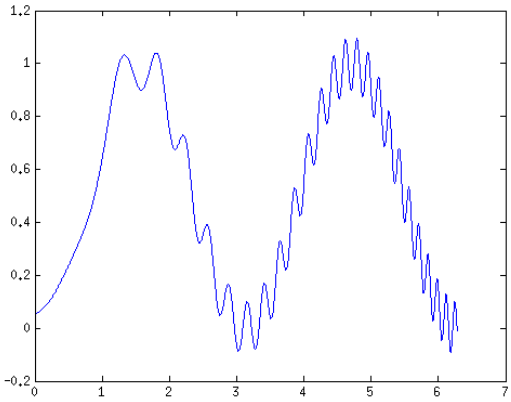
Quindi, usando Octave scriviamo

```
n = 512; h = 2*pi/n; x = [0:h:2*pi-h];
s = sin(x).^2  +  0.1*cos(4*x.^2 - x + 1);
y = s + 0.05*sin(100*x);
plot(y);
```

Il grafico che otteniamo ha delle oscillazioni molto fitte in alta frequenza come mostra la figura seguente



mentre il grafico del segnale originale ottenuto mediante il comando `plot(s)` è dato da



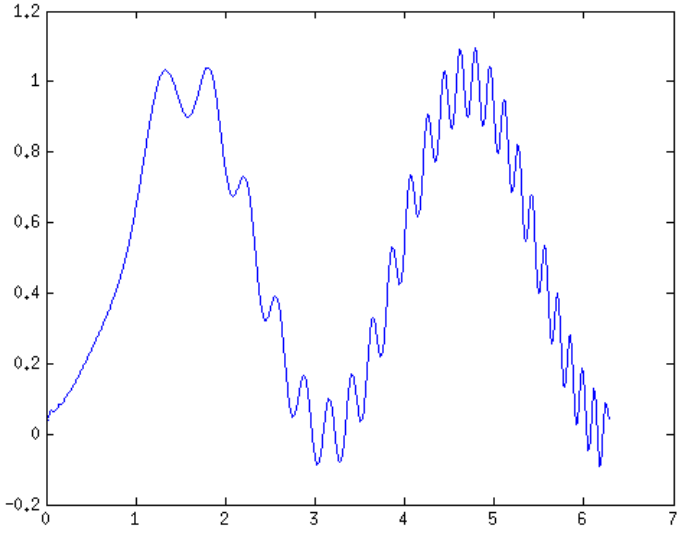
Per rinuovere il rumore procediamo nel modo seguente. Si crea un filtro che azzera le componenti nelle frequenze più alte come segue

```
f = ones(1,n);
m  =n/2+1;
f(m-200:m+200) = 0;
```

e adesso filtriamo

```
w = ifft(fft(y).*f);
w = real(w);
plot(w);
```

In questo modo otteniamo il grafico



5.2 Filtraggio di immagini digitali

Nel caso in cui il vettore v sia sostituito da una matrice V di dimensioni $m \times n$ è possibile creare una base dello spazio delle matrici $m \times n$ costituita da mn elementi dotati di "frequenze di oscillazione". Infatti, se denotiamo con $w^{(1)}, \ldots, w^{(m)}$ una base di $\mathbb{R}^m$ formata da vettori seni e coseni come si è fatto nel caso di vettori, e se denotiamo analogamente $z^{(1)}, \ldots, z^{(n)}$ una base di vettori seno e coseno per $\mathbb{R}^n$, allora le mn matrici ottenute scrivendo $z^{(i)}w^{(j)T}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$ sono linearmente indipendenti e formano una base dello spazio delle matrici $m \times n$. In questo modo ciascun elemento della base è dotato di una frequenza per le righe e una per le colonne.

Nel caso in cui la matrice V rappresenti una immagine digitale allora le componenti in alta frequenza rappresentano variazioni di luminosità molto ravvicinate. Come esempio, in figura 19 si riportano le immagini corrispondenti ad alcune delle matrici della base $z^{(i)}w^{(i)T}$ con $n = m = 400$.

Nel caso delle immagini, e quindi di matrici, si possono fare filtraggi analoghi mediante FFT agendo separatamente sui vettori colonna e sui vettori riga. Si supponga di avere una matrice A $m \times n$ che contiene una immagine. Per rimuovere le alte frequenze (dettagli minuscoli) in A basta calcolare la trasformata discreta di Fourier delle righe e la trasformata discreta delle colonne di A, moltiplicare ciascuna colonna del risultato componente a componente con un vettore filtro, fare la stessa cosa sulle righe e antitrasformare righ e colonne.

Octave ha le funzioni `fft2` e `ifft2` che svolgono la trasformata discreta diretta e inversa delle righe e delle colonne di una matrice.

Una function di filtraggio di immagini in Octave, che rimuove le frequenze alte dall'immagine è riportata nel listato 15.

Listing 15: Function che filtra una immagine mediante FFT rimuovendo le alte frequenze.

```
function B = filtra(A)
   m = size(A)(1); n = size(A)(2);
% calcolo fft di righe e colonne
   B = fft2(A);
% filtro le righe
   k = floor((m-1)/2);
   k1 = round(k/2);
   v = ones(1,k);
   v(k1:k) = 0;
   if(mod(m,2) == 0) % caso m pari: [0 1 2 3 4 3 2 1]
      f1 = [1,v,0,v(k:-1:1)];
   else % caso m dispari: [0 1 2 3 3 2 1]
      f1 = [1,v,v(k:-1:1)];
   endif
% filtro per le colonne
   k = floor((n-1)/2);
   k1=round(k/2);
   v = ones(1,k);
   v(k1:k) = 0;
   if(mod(n,2) == 0) % caso n pari: [0 1 2 3 4 3 2 1]
      f2 = [1,v,0,v(k:-1:1)];
   else % caso n dispari: [0 1 2 3 3 2 1]
      f2 = [1,v,v(k:-1:1)];
   endif
% filtro e antitrasformo
    B = ifft2(diag(f1)*B*diag(f2));
    B = real(B); % tolgo eventuale roundoff immaginario
    B = round(B); % arrotondo alla parte intera
    B = max(B,0); % riporto i valori nel range 0--255
    B = min(B,255);
    B = uint8(B); % trasformo variabile double in intera
endfunction
```

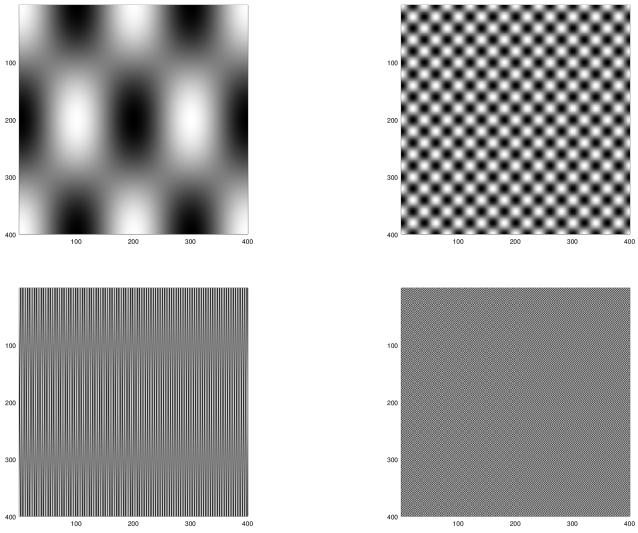


Figura 19: Immagini corrispondenti alle matrici della base di Fourier bidimensionale con frequenze $(1, 2)$, $(10, 10)$, $(1, 100)$ e $(100, 100)$.

Si noti la simmetria di un filtro data dal fatto che il vettore ottenuto rimuovendo la prima componente di f è simmetrico rispetto al centro.

Si ricorda che se v è reale allora il vettore u è tale che u_0 e $u_{n/2+1}$ sono reali mentre u_j è il complesso coniugato di u_{n-j}. Per cui, affinché un filtro f mantenga la realtà del segnale filtrato, occorre che f sia "simmetrico" nel senso che $f_j = f_{n-j}$, $j = 1, \ldots, n/2 - 1$. Basta quindi definire il filtro in $f_0, \ldots, f_m$.

Le seguenti tre immagini riportano il caso di una foto originale a cui è stato aggiunto del rumore dato da una grana sottile. L'immagine rumorosa è stata filtrata con la function filtra rimuovendo le componenti in alta frequenza. Il rumore aggiunto R è stato calcolato con la formula

```
S = 2*pi*[0:m-1]'*(250*ones(1,n)+5*rand(1,n))/m;
T = [0:n-1]'*(250*ones(1,m)+5*rand(1,m))*2*pi/m;
R = 10*(sin(S) + sin(T)');
```

che genera frequenze a caso comprese tra $250/(2\pi)$ e $255/(2\pi)$ sia sulle righe che sulle colonne di ampiezza minore o uguale a 20.

Foto originale

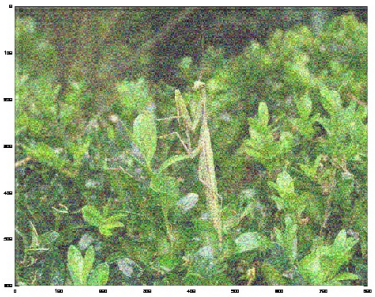


Foto con rumore

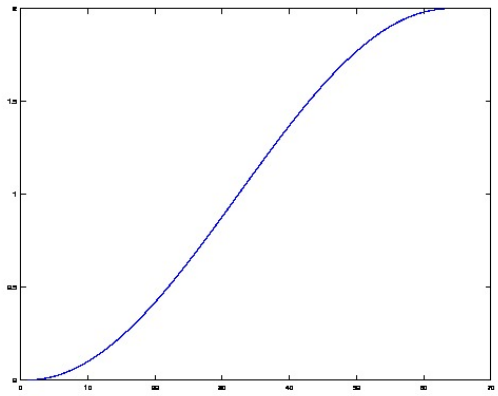
Foto filtrata

Ecco un esempio di immagine ottenuta a partire dalla fotografia dei cammelli amplificando le alte frequenze. A sinistra l'immagine filtrata, a destra quella originale. In questo caso, essendo l'immagine a colori, si è fatto il filtraggio sulle tre componenti R,G,B. Si può vedere il maggior dettaglio nei particolari delle foglie e nei tronchi delle palme.

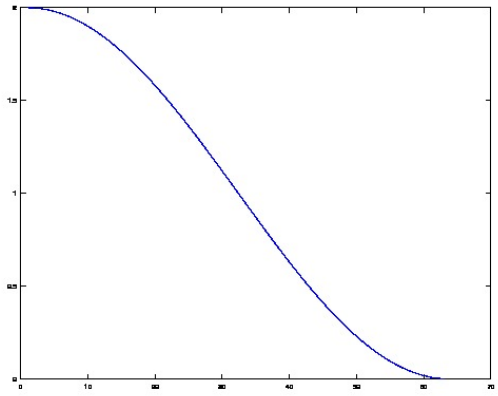
L'uso di filtri discontinui come quelli che rimuovono alcune componenti in frequenza e ne lasciano inalterate delle altre possono creare effetti fastidiosi

come echi ripetuti lungo le linee di maggior contrasto. è preferibile usare filtri ottenuti da funzioni continue.

Un filtro "passa alto" è dato da $f_j = 1 - \cos(2\pi j/n)$, $j = 0, \ldots, n/2$. Infatti, dal grafico della funzione $1 - \cos x$



si vede che il filtro riduce l'ampiezza delle componenti in bassa frequenza. Mentre un filtro passa basso è $f_j = 1+\cos(2\pi j/n)$, $j = 0, \ldots, n/2$. Infatti dal grafico della funzione $1 + \cos x$

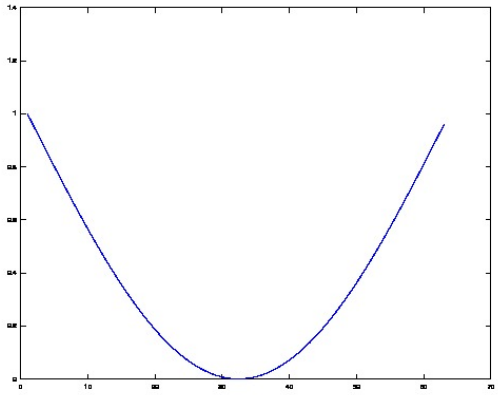


si vede che il filtro riduce l'ampiezza delle componenti in alta frequenza.

Come agisce invece il filtro seguente?

$$f_j = 1 - \cos(3\pi/2 + 2\pi j/n), \quad , j = 0, \ldots, n/2.$$

Il grafico della funzione associata $1 - \cos(3\pi/2 + x)$ è



Si osservi che il grafico della funzione associata al filtro l'abbiamo tracciato sull'intervallo $[0, \pi]$. Infatti il numero di componenti di f è $n/2$ se n è pari e $(n-1)/2$ se n è dispari.

La rappresentazione spettrale può essere usata per comprimere un'immagine. Questo si realizza memorizzando solamente i valori delle ampiezze che corrispondono alle frequenze più basse e si basa sul fatto che generalmente le "alte frequenze", cioè la ricchezza di dettagli minuscoli, non sono presenti in tutte le parti di una immagine. Quindi decomponendo un'immagine in porzioni più piccole ad esempio di 8×8 pixel, e calcolando la rappresentazione spettrale di queste matrici 8×8, è possibile rappresentarle in modo abbastanza fedele memorizzando solo le basse frequenze nel caso in cui le alte abbiano valori in ampiezza trascurabili. Questa è l'idea alla base della compressione JPG. Nella compressione JPG le immagini vengono rappresentate mediante il valore della luminanza e mediante i due valori di crominanza relative al blu e al rosso. Al posto della trasformata discreta di Fourier viene usata la trasformata discreta dei coseni per cui le matrici associate alle immagini vengono rappresentate da soli coseni. Ogni matrice 8×8 viene rappresentata come combinazione lineare delle 64 matrici riportate in figura 20 (immagine presa da Wikipedia). Maggiori dettagli si trovano in `http://en.wikipedia.org/wiki/JPEG`

Si implementino e si applichino i seguenti filtri ad una foto a colori mediante FFT. Se l'immagine risulta troppo scura si provveda a schiarirla numericamente.

Esercizio G0. Filtro passa alto $f_j = 1 - \cos(2\pi j/n)$, $j = 0, \ldots, n/2$.

Esercizio G1. Filtro passa basso $f_j = 1 + \cos(2\pi j/n)$, $j = 0, \ldots, n/2$.

Esercizio G2. Filtro dato da $1 - \cos(3\pi/2 + 2\pi j/n)$.

Esercizio G3. Si implementi il filtro dato da $2 - \cos(2\pi j/n)$.

Esercizio G4. Si implementi il filtro ottenuto prendendo la potenza k-esima, con $k = 3$, del filtro dell'esercizio 0.

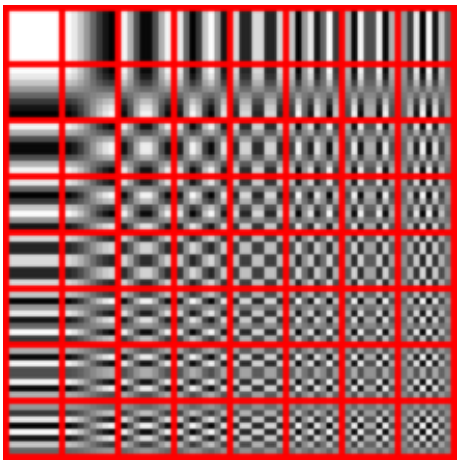


Figura 20: Base di 64 matrici associate alla trasformata dei Coseni bidimensionale con cui viene decomposta una generica matrice 8×8 nell'implementazione del formato JPG. La foto è ripresa dal sito di Wikipedia.

Esercizio G5. Si implementi il filtro ottenuto prendendo la potenza k-esima, con $k = 3$, del filtro dell'esercizio 1.

Esercizio G6. Si implementi il filtro ottenuto prendendo la potenza k-esima, con $k = 3$, del filtro dell'esercizio 2.

Esercizio G7. Si implementi il filtro ottenuto prendendo la potenza k-esima, con $k = 3$, del filtro dell'esercizio 3.

5.3 Filtri FIR

Filtri bassa basso e passa alto si possono più semplicemente costruire mediante i filtri FIR (Finite Impulse Response) `http://en.wikipedia.org/wiki/Finite_impulse_response` che operano combinazioni lineari di pixel evitando l'uso della FFT.

Ad esempio, una trasformazione che realizza un filtro passa basso e trasforma l'immagine A nell'immagine B in cui le "alte frequenze" vengono attenuate è dato da $b_{i,j} = (a_{i,j} + a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})/5$.

Questo filtro opera prendendo la media dei valori del pixel di coordinate (i, j) e dei 4 pixel contigui, e pone il risultato in posizione (i, j) dell'immagine trasformata. Filtri passa basso che attenuano maggiormente le alte frequenze si ottengono allargando il supporto dei valori di cui si calcola la media.

Un filtro passa alto è $b_{i,j} = 4a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1}$. Si osserva che una mmagine costante, cioè tale che $a_{i,j} = \alpha$ costante, viene trasformata in 0. Anche una immagine con andamento bilineare, cioè tale che $a_{i,j} = \alpha +$

$\beta i + \gamma j + \delta ij$ è trasformata in 0. Si verifichi cosa succede ad una immagine che dipende quadraticamente da i e da j.

Un filtro FIR può essere implementato facilmente con un doppio ciclo for, ma è altrettanto facile darne una implementazione vettoriale con la sintassi di Octave. Ad esempio, il filtro $b_{i,j} = (a_{i,j} + a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1})/5$, si scrive come

```
b = a;
b(1:m-1,:) = b(1:m-1,:) + a(2:m,:);   % b(i,j)=b(i,j)+b(i+1,j)
b(:,1:n-1) = b(:,1:n-1) + a(:,2:n);   % b(i,j)=b(i,j)+b(i,j+1)
b(2:m,:) = b(2:m,:) + a(1:m-1,:);     % b(i,j)=b(i,j)+b(i-1,j)
b(:,1:n-1) = b(:,1:n-1) + a(:,1:n-1); % b(i,j)=b(i,j)+b(i,j-1)
b = b/5;
```

dove abbiamo riportato come commento la trasformazione vista sul singolo pixel.

Si implementino in modo vettoriale e si applichino ad una immagine a colori i seguenti filtri FIR dove si assume $a_{i,j} = 0$ se almeno uno dei due indici esce fuori dai valori consentiti. Nel caso le immagini dovessero risultare molto scure, si aumenti la loro luminosità numericamente.

Esercizio H0. Filtro $b_{i,j} = 4a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1}$

Esercizio H1. Filtro $b_{i,j} = a_{i,j} + a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1}$

Esercizio H2. Filtro $b_{i,j} = a_{i,j} + a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} + a_{i+1,j+1} + a_{i+1,j-1} + a_{i-1,j+1} + a_{i-1,j-1}$

Esercizio H3. Filtro $b_{i,j} = 8a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1} - a_{i+1,j+1} - a_{i+1,j-1} - a_{i-1,j+1} - a_{i-1,j-1}$

Esercizio H4. Filtro $b_{i,j} = 9a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1} - a_{i+1,j+1} - a_{i+1,j-1} - a_{i-1,j+1} - a_{i-1,j-1}$

Esercizio H5. Filtro $b_{i,j} = 5a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1}$

Esercizio H6. Filtro $b_{i,j} = 9a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1} + a_{i+1,j+1} + a_{i+1,j-1} + a_{i-1,j+1} + a_{i-1,j-1}$

Esercizio H7. Filtro $b_{i,j} = 8a_{i,j} - a_{i+1,j} - a_{i-1,j} - a_{i,j+1} - a_{i,j-1} + a_{i+1,j+1} + a_{i+1,j-1} + a_{i-1,j+1} + a_{i-1,j-1}$

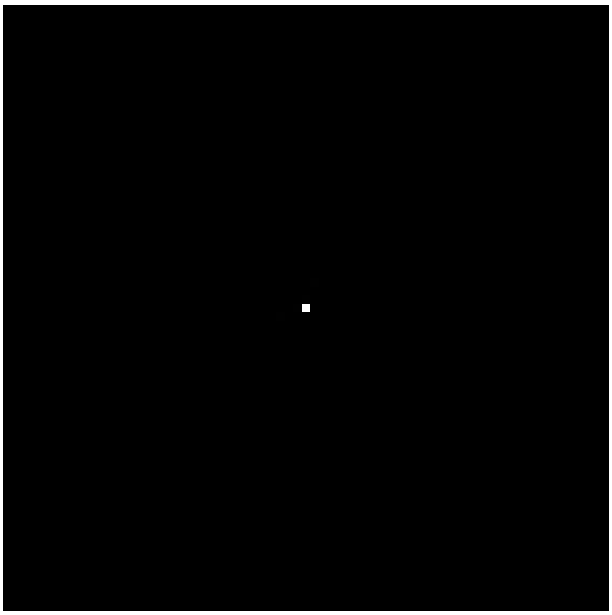


Figura 21: Immagine di un punto luminoso

6 Il modello di sfocatura (cenno)

Una delle applicazioni più interessanti della tecnologia digitale è il restauro di immagini alterate da difetti di messa a fuoco o da difetti dovuti a movimenti della macchina fotografica all'atto dello scatto effettuato con tempi di otturazione troppo lunghi.

Fotografie che sono venute sfocate per un diffetto di messa a fuoco dell'obiettivo o per movimenti della fotocamera possono essere in qualche modo rimesse a fuoco se si conosce il tipo di sfocatura. Supponiamo di avere scattato una foto ad una immagine costituita da un solo punto luminoso di intensità unitaria come in figura 21. Con l'obiettivo regolato male si otterrà qualcosa tipo l'immagine in figura 22 dove abbiamo volutamente esagerato l'effetto della sfocatura. Questa nuova immagine sarà definita da dei pixel $f_{i,j}$, dove per comodità facciamo scorrere gli indici i e j da $-k$ a k dove $2k+1$ è l'ampiezza del supporto dell'immagine del puntolino sfocato. La tabella di numeri $f_{i,j}$ per $i, j = -k, k$ descrive quindi il tipo di sfocatura ed è chiamata in gergo *Point-Spread Function*, o più semplicemente PSF. Essa descrive il modo in cui il punto luminoso si spande in una macchiolina. I valori $f_{i,j}$ sono non negativi e devono sommarsi ad 1 poiché la quantità di luminosità presente nell'immagine sfocata deve essere uguale a quella presente nell'immagine originale.

Allora, se $A = a_{i,j}$ per $i, j \in \mathbb{Z}$, sono i pixel di una immagine originale A costituita da infiniti pixel collocati su un piano, possiamo identificare una fotografia di A come una matrice $B = (b_{i,j})$ che cattura una porzione rettangolare di

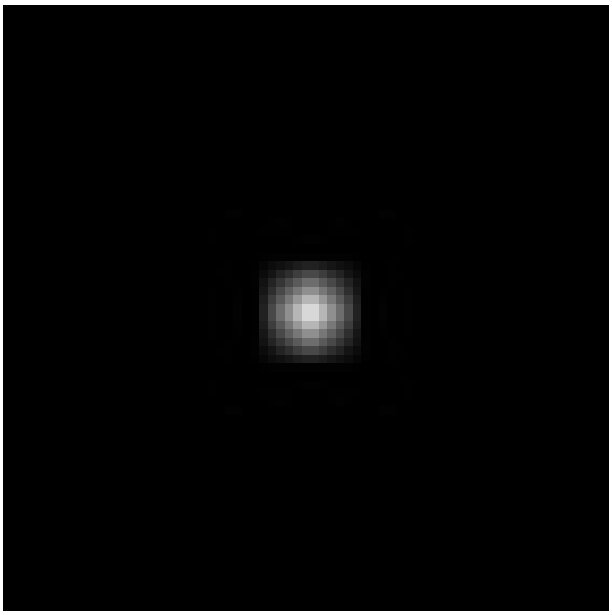


Figura 22: Immagine di un punto luminoso fotografato con l'obiettivo fuori fuoco

A, precisamente quella i cui indici sono $i = 1, \ldots, m$, $j = 1, \ldots, n$, cioè $b_{i,j} = a_{i,j}$ per $i = 1, \ldots, m$, $j = 1, \ldots, n$. Ora supponiamo che tutti i pixel di A siano nulli tranne $a_{p,q} = g \neq 0$, cioè l'immagine è costituita da un unico punto luminoso di intensità g posto nella posizione (p, q). Se la foto viene scattata con l'obiettivo fuori fuoco l'immagine sfocata $b_{i,j}$ avrà pixel uguali a

$$b_{i,j} = \begin{cases} g f_{i-p,j-q}, & \text{se } |i-p|, |j-q| \leq k \\ b_{i,j} = 0, & \text{altrove.} \end{cases}$$

In questo modello il tipo di sfocatura (di PSF) è invariante spazialmente, cioè non dipende dalla posizione del punto (p, q) ed è definito dalla stessa tabella di numeri $(f_{i,j})_{i,j=-k,k}$ qualunque sia il punto (p, q). Il modello fornisce una approssimazione della situazione reale. Infatti, nella realtà il comportamento di un obiettivo fotografico è diverso a seconda che si consideri il centro o il bordo dell'immagine.

Un'altra considerazione riguarda il supporto dell'immagine che per A è dato da $\mathbb{Z} \times \mathbb{Z}$. Però a incidere sui pixel dell'immagine sfocata intervengono solo i pixel $a_{i,j}$ di A tali che $i = -k+1, \ldots, m+k$, $j = -k+1, \ldots, n+k$. Cioè l'immagine sfocata viene originata non solo dai pixel che hanno indici $1 \leq p \leq m$, $1 \leq j \leq n$, ma anche dai pixel che sono fuori da questo supporto purché rimangano dentro una cornice di ampiezza k. Infatti, se il pixel luminoso di coordinate (p, q) si trova a distanza al più k (semiampiezza del supporto della PSF) dal supporto

$[1:m]\times[1:n]$, allora l'effetto della sfocatura si ritrova anche dentro il supporto di B.

In questo modello l'immagine sfocata di una immagine generica costituita da più pixel non nulli sarà la somma delle immagini che si ottengono sfocando i singoli pixel. Vale cioè

$$b_{i,j} = \sum_{p,q} f_{i-p,j-q} a_{p,q}, \quad i = 1,\dots,m,\ j = 1,\dots,n, \tag{1}$$

dove la somma è fatta per $p = 1-k,\dots,m+k$, $q = 1-k,\dots,n-k$, su tutti gli indici per cui $i-p$ e $j-q$ sono compresi tra $-k$ e k. Ponendo $r = i-p$ e $s = j-q$, la (1) può essere riscritta come

$$b_{i,j} = \sum_{r,s=-k}^{k} f_{r,s} a_{i-r,j-s}, \quad i = 1,\dots,m,\ j = 1,\dots,n. \tag{2}$$

Si riportano alcuni esempi di PSF.

- PSF uniforme: in questo caso la matrice F di dimensioni $(2k+1)\times(2k+1)$ ha tutti elementi uguali tra di loro che valgono quindi $f_{i,j} = 1/(2k+1)^2$. Un punto viene quindi trasformato in un rettangolo di $(2k+1)\times(2k+1)$ pixel. Maggiore è k e maggiore è l'effetto di sfocatura.

- PSF esponenziale: i valori della PSF sono dati da $f_{i,j} = \theta e^{-\alpha(i^2+j^2)}$, per $i,j = -k,\dots,k$, dove θ è un parametro di normalizzazione scelto in modo che la somma degli $f_{i,j}$ faccia 1, e α è un numero positivo che determina l'ampiezza dell'effetto di sfocatura. In questo caso il punto centrale della PSF, quello corrispondente agli indici $i = j = 0$, è quello di luminosità massima, inoltre la luminosità decade esponenzialmente a zero man mano che ci si allontana dal punto centrale. Il decadimento dipende dalla grandezza di α. Per $\alpha = 0$ si ottiene la PSF uniforme. La figura 23 mostra il caso in cui $k = 10$ e $\alpha = 0.05$.

- PSF sinc: i valori della PSF di tipo sinc sono dati da $f_{i,j} = \theta(\sigma+\mathrm{sinc}(\alpha(i^2+j^2)^{1/2}))$ dove la funzione sinc è data da $\mathrm{sinc}(x) = \sin x/x$. Questo tipo di PSF è quello più vicino alla situazione reale delle macchine fotografiche digitali. Il parametro α determina l'intensità della sfocatura: più piccolo è α e maggiore è la sfocatura. Il parametro σ è scelto in modo da avere valori non negativi della PSF, il valore θ viene scelto in modo che gli elementi della PSF si sommino a 1. Si riporta nella figura 24 il grafico della funzione $\mathrm{sinc}(2x)$.

  Nella figura 25 si riporta il grafico della PSF bidimensionale di tipo sinc con $\alpha = 1$.

La trasformazione che trasforma una immagine F in una immagine B sfocata può essere vista come un filtro FIR dove la PSF non è altro che la Finite Impulse Response, cioè l'immagine sfocata di una sorgente luminosa puntiforme di valore unitario.

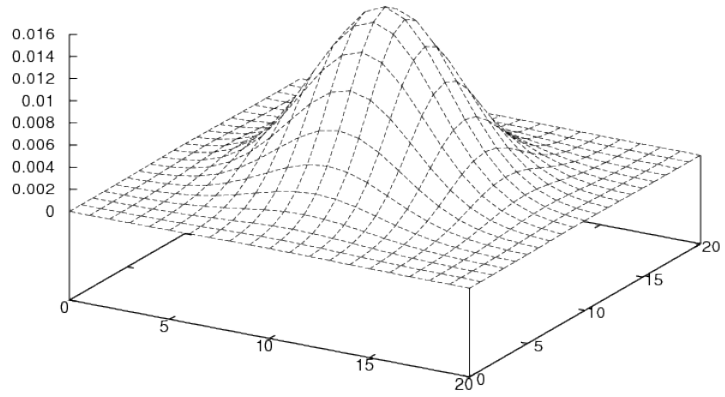


Figura 23: Point-Spread Function di tipo esponenziale

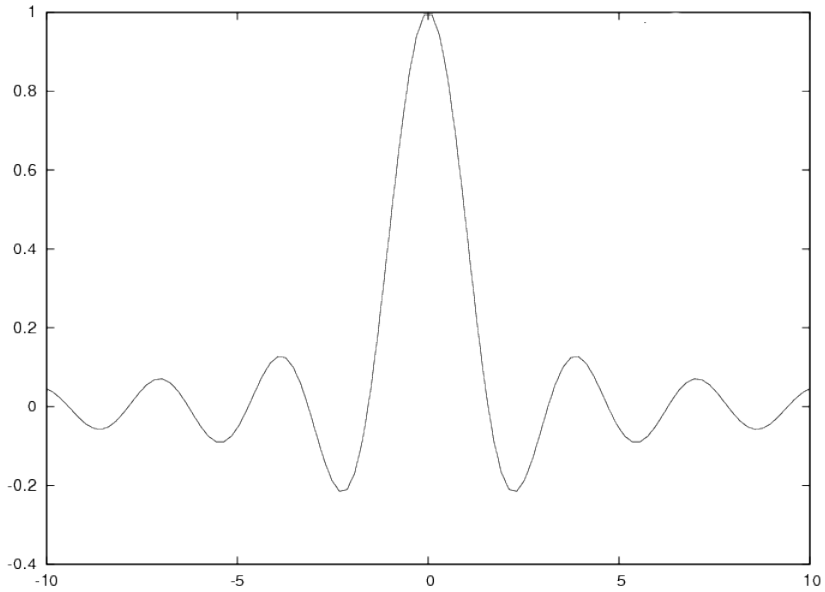


Figura 24: Point-Spread Function di tipo sinc: $\sin(2x)/(2x)$

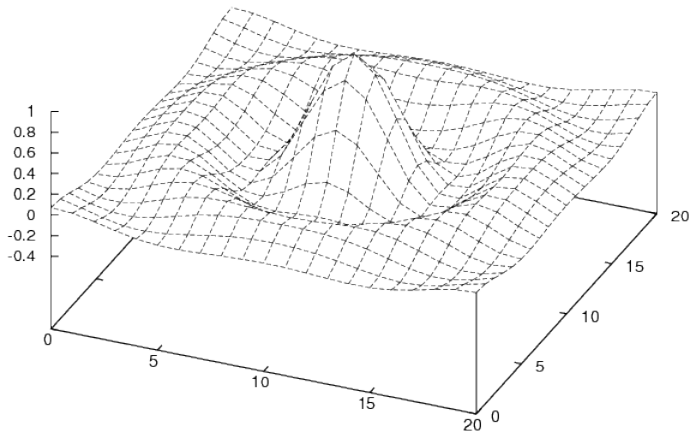


Figura 25: Point-Spread Function di tipo sinc: $\sin((x^2+y^2)^{\frac{1}{2}})/((x^2+y^2)^{\frac{1}{2}})$

Dal punto di vista computazionale per sfocare una immagine in bianco e nero basta calcolare una doppia sommatoria: data $A = (a_{i,j})$ e $F = (f_{i,j})$, si calcola $B = (b_{i,j})$ mediante la (1) o la (2). Per rimettere a fuoco una immagine basta "risolvere" un sistema lineare di mn equazioni in $(m+2k)(n+2k)$ incognite: data l'immagine sfocata B e la PSF F si calcola l'immagine originale A risolvendo il sistema (2).

Per immagini a colori la manipolazione va compiuta sui tre canali del rosso, verde e blu. Per una foto scattata con un macchina fotografica da 5 megapixel si devono risolvere tre sistemi di circa 5 milioni di equazioni e di incognite per poter rimettere a fuoco la foto.

I sistemi che si ottengono in questo modo sono sottodeterminati, hanno cioè meno equazioni che incognite. La risoluzione di questi sistemi va allora intesa in termini di "risoluzione ai minimi quadrati". Il sistema (2) può essere scritto in forma matriciale come $Ha = b$ dove a e b sono i vettori ottenuti giustapponendo una sull'altra rispettivamente le colonne di A e di B. Mentre H è la matrice del sistema lineare che si ottiene con questa organizzazione in vettori.

Si osserva che, usando un linguaggio di programmazione Octave il procedimento di sfocatura si realizza in modo abbastanza semplice. La function riportata nel listato 16 realizza giusto questo.

Un modo più efficiente di realizzare la sfocatura in Octave consiste nell'usare il comando `conv2`. Infatti, `b = conv2(a, psf, shape);` calcola la convoluzione (sfocatura) della matrice (immagine) `a` di dimensioni $m \times n$ con la point-spread function memorizzata nella variabile `psf` di dimensioni $(2k+1)\times(2h+1)$.

Listing 16: Function che sfoca una immagine

```
function b = sfoca(k,psf,a)
  % Sfoca l'immagine a e la mette in b usando la psf assegnata
  % Calcola cioe' il prodotto matrice vettore b=Ha
  [m,n] = size(a);
  k = (size(psf)-1)/2;
  b = zeros(m,n);
  for i=1:m
     for j=1:n
        for p=-k:k
           for q=-k:k
              b(i,j) = b(i,j) + psf(p+k+1,q+k+1)*a(i-p+k,j-q+k);
           endfor
        endfor
     endfor
  endfor
endfunction
```

La variabile shape può prendere i seguenti valori

'full' la trasformata `b` ha dimensioni $(m + 2h) \times (n + 2k)$

'same' la trasformata `b` è tagliata alle stesse dimensioni di `a`

'valid' la trasformata `b` contiene la parte di dimensioni $m - 2h \times (n - 2h)$

In questo modo lo stesso effetto della function `sfoca` si ottiene semplicemente col comando `b = conv2(a, psf, 'valid');`

Attenzione: Il comando `conv2` ha un comportamento indesiserato se le variabili a cui è applicato non sono `double`, cioè numeri floating point. Questo può capitare con le immagini quando le variabili sono spesso `uint8` cioè interi senza segno rappresentati con 8 bit. Per questo motivo, all'interno delle function che effettueranno la rimessa a fuoco di una immagine memorizzata nella variabile `A` si raccomanda di convertire `A` ad un `double` scrivendo `A=double(A);`

In figura 27 si riporta come appare l'immagine che si ottiene sfocando con una PSF uniforme di semi ampiezza $k = 4$ la fotografia di figura 26 scattata ad una pianta di peperoni habanero. La fotografia in origine era di 2288×1712 pixel, scattata da una macchina fotografica da 4 megapixel, ed è qui stata ridotta a 800×600 pixels.

6.1 Matrice del sistema

Vediamo di descrivere la struttura della matrice H del sistema $Ha = b$ che lega l'immagine originale A e l'immagine sfocata B organizzate come vettori rispettivamente a e b.

Per far questo esaminiamo prima il caso più elementare in cui $n = 1$, cioè le immagini A, B e la PSF F sono costituite da una sola colonna, cioè $A =$

Figura 26: Fotografia originale

Figura 27: Fotografia sfocata

$(a_{k-1}, \ldots, a_{m+k})^T$, $B = (b_1, \ldots, b_m)^T$, $F = (f_{-k}, \ldots, f_k)^T$. In questo caso il sistema (2) prende la forma

$$b_i = \sum_{r=-k}^{k} f_r a_{i-r}, \quad i = 1, \ldots, m.$$

Si può osservare che in ciascuna equazione di questo sistema compaiono $2k+1$ incognite consecutive, ad esempio nella i-esima equazione sono $a_{i-k}, \ldots, a_{i+k}$ e che i coefficienti di queste incognite in ciascuna equazione sono sempre gli stessi, cioè gli elementi della PSF. Questo fatto comporta che la matrice del sistema ha una struttura a banda di ampiezza $2k+1$ e che gli elementi della matrice sulla banda sono uguali lungo le diagonali. Cioè, in notazione matriciale, il sistema ha la forma

$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} f_k & f_{k-1} & \cdots & f_{-k+1} & f_{-k} & & & \\ & f_k & f_{k-1} & \ddots & f_{-k+1} & f_{-k} & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & f_k & f_{k-1} & \cdots & f_{-k+1} & f_{-k} \end{bmatrix} \begin{bmatrix} a_{1-k} \\ \vdots \\ a_0 \\ a_1 \\ \vdots \\ a_m \\ a_{m+1} \\ \vdots \\ a_{m+k} \end{bmatrix}$$

Matrici i cui elementi sono uguali lungo le diagonali, o più precisamente, i cui elementi dipendono unicamente dalla differenza degli indici di riga e colonna vengono dette *matrici di Toeplitz* Queste matrici hanno speciali proprietà computazionali e spettrali.

Nel caso generale in cui $m, n > 1$, se scriviamo la matrici A e B rispettivamente come vettori a e b ottenuti incolonnando una sull'altra le colonne rispettivamente di A e di B, la matrice del sistema H prende la forma di matrice $(n+2k) \times n$ a blocchi

$$\begin{bmatrix} F_k & F_{k-1} & \cdots & F_{-k+1} & F_{-k} & & & \\ & F_k & F_{k-1} & \ddots & F_{-k+1} & F_{-k} & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & F_k & F_{k-1} & \cdots & F_{-k+1} & F_{-k} \end{bmatrix}$$

i cui blocchi F_r sono matrici $(m+2k) \times m$ dati da

$$F_r = \begin{bmatrix} f_{r,k} & f_{r,k-1} & \cdots & f_{r,-k+1} & f_{r,-k} & & & \\ & f_{r,k} & f_{r,k-1} & \ddots & f_{r,-k+1} & f_{r,-k} & & \\ & & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & f_{r,k} & f_{r,k-1} & \cdots & f_{r,-k+1} & f_{r,-k} \end{bmatrix}$$

Cioè la matrice del sistema ha due livelli di struttura: in termini di blocchi e in termini di elementi dei blocchi. Infatti, la matrice è di Toeplitz a blocchi con blocchi di Toeplitz; è inoltre a banda a blocchi con blocchi a banda.

Esistono metodi iterativi per risolvere il problema di minimi quadrati che sfruttano la particolare struttura del sistema. È importante osservare che se si usassero metodi classici generali per risolvere questo problema avremmo un costo che cresce col cubo della dimensione del sistema, cioè il cubo del numero di pixel dell'immagine. Una immagine di 10 megapixel richiederebbe un numero di operazioni dell'ordine di 10^{18}. Un calcolatore in grado di svolgere 10 miliardi di operazioni al secondo impiegherebbe 10^8 secondi cioè oltre 3 anni per mettere a fuoco l'immagine. I metodi iterativi disponibili attualmente sul mercato rimettono a fuoco l'immagine in pochi secondi.

La soluzione ai minimi quadrati del sistema $Ha = b$ si può scrivere come

$$a = H^+ b$$

dove

$$H^+ = H^T (HH^T)^{-1}$$

è l'inversa generalizzata di Moore-Penrose della matrice H del sistema. Per cui possiamo prima calcolare la soluzione del sistema

$$HH^T x = b \tag{3}$$

e poi calcolare il prodotto $a = H^T x$.

Il sistema (3), che ha matrice definita positiva, si risolve applicando un metodo iterativo.

Come metodo iterativo può essere applicato il metodo di Richardson `http://en.wikipedia.org/wiki/Modified_Richardson_iteration`

$$a^{(\nu+1)} = a^{(\nu)} - \theta(HH^T a^{(\nu)} - b) \tag{4}$$

dove θ è un parametro che rende la matrice di iterazione $P = I - \theta HH^T$ di raggio spettrale minore di 1, ad esempio $\theta = 1/\|HH^T\|$ per una qualsiasi norma indotta $\|\cdot\|$. Il metodo di Richardson in ambiente di "image restoration" è più noto come "metodo di Landweber" `http://en.wikipedia.org/wiki/Landweber_iteration`. Una alternativa più efficace è il metodo del gradiente coniugato `http://en.wikipedia.org/wiki/Conjugate_gradient_method`.

Metodi di questo tipo hanno come operazione più costosa, il calcolo del prodotto tra la matrice HH^T ed un vettore. Questo prodotto, $z = HH^T x$

può essere calcolato valutando separatamente i prodotti $y = H^T x$ e $z = Hy$. Quest'ultimi due prodotti possono essere calcolati a basso costo sfruttando la particolare struttura della matrice H.

Riportiamo nelle figure 28 – 32, le immagini rimesse a fuoco usando il metodo di Richardson con un numero di iterazioni pari rispettivamente a 10, 20, 40, 80, 160.

Esercizio L0. Trasformare in bianco/nero la foto del gatto usata nella mappa di Arnold. Sfocare la foto del gatto con la psf 7×7 con tutti i valori uguali a $1/49$. Successivamente applicare `maxit` iterazioni del metodo di Richardson con $\theta = 1$. Per far questo si scriva una function che implementa 4. Si usi il fatto che per questa PSF il prodotto $a = H^T x$ in Octave si calcola con `a = conv2(x,psf,'full');` mentre il prodotto $y = Ha$ si calcola con `y = conv2(a, psf, 'valid');`. Applicare `maxit=500` iterazioni alla immagine sfocata. Inviare la function e le due immagini sfocata e rifocata.

Esercizio L1. Fare la sperimentazione dell'esercizio L0 con la PSF data dalla matrice tridiagonale 21×21 che ha elementi diagonali, sopradiagonali e sottodiagonali uguali a $1/61$.

L'immagine sfocata (a colori) ottenuta con la PSF dell'esercizio L1 e quella ottenuto applicando 100 passi del metodo di Richardson sono riportate qui sotto.

Esercizio L2. Fare la sperimentazione dell'esercizio L0 con la PSF data dalla matrice 51×51 che ha elementi diagonali e antidiagonali (cioè tali che $i + j = 52$) uguali a $1/101$ e nulli i rimanenti.

Esercizio L3. Fare la sperimentazione dell'esercizio L0 con la PSF data dalla matrice 51×51 che ha elementi uguali a $1/200$ sulla prima ed ultima riga e colonna e nulli altrove.

Esercizio L4. Fare la sperimentazione dell'esercizio L0 con la PSF data dalla matrice 51×51 che ha elementi uguali a $1/101$ sulla riga e sulla colonna di indice 26 e nulli altrove.

Figura 28: Metodo di Richardson con 10 iterazioni

Figura 29: Metodo di Richardson con 20 iterazioni

Figura 30: Metodo di Richardson con 40 iterazioni

Figura 31: Metodo di Richardson con 80 iterazioni

Figura 32: Metodo di Richardson con 160 iterazioni

Si può notare che nelle immagini che approssimano la soluzione, fornite dal metodo iterativo di Richardson, il valore approssimato della soluzione presenta degli "echi" in prossimità delle linee in cui si trova una brusca variazione di luminosità. Questo fenomeno è riconducibile all'effetto Gibbs che produce i cosiddetti Ringing Artifacts.