**SINA** **Dicembre 2003**

# Corso di programmazione – test finale

Si supponga di voler stimare il valore di una funzione *f* in un punto arbitrario del suo insieme di definizione a partire da un insieme di *N* misure. Formalmente il problema equivale alla stima di $y = f(x)$ per qualunque punto $x \neq x_i$ a partire dai valori $y_i$ della funzione in un insieme di punti $x_i$ $i = 1, 2..N$. Tale problema è detto *interpolazione*. Esistono molti metodi per interpolare una funzione a partire dai campioni più vicini. Per semplicità si suppongano i campioni a distanza 1. Si considerino i seguenti casi:

1) Nearest neighbor (interpolazione di ordine zero). E' il caso più semplice, il valore della funzione in $x$ è preso uguale a quello del campione più vicino. Ossia: $y = y_i = f(x_i)$ dove $x_i = \min_{x_i} \|x - x_i\|$
2) Interpolazione lineare: $y \stackrel{x_i}{=} y_i(1-a) + y_{i+1}(a)$ dove $a \leq 1$ è la distanza tra $x$ e il campione $x_i$ che lo precede (figura 1).

Si consideri ora l'interpolazione di una funzione bidimensionale $F(p,q)$. Il metodo più semplice è nuovamente quello nearest neighbor (immediatamente estendibile nel caso di due dimensioni). Per ottenere risultati migliori è comunemente utilizzato il metodo di interpolazione bilineare (*bilinear interpolation*), che consiste nell'interpolare linearmente prima lungo la dimensione *p* e successivamente lungo *q.* La formula risultante è la seguente (con riferimento alla figura 2):

$$\widehat{F}(p',q') = (1-a)\left[(1-b)F(p,q) + bF(p,q+1)\right] + a\left[(1-b)F(p+1,q) + bF(p+1,q+1)\right]$$

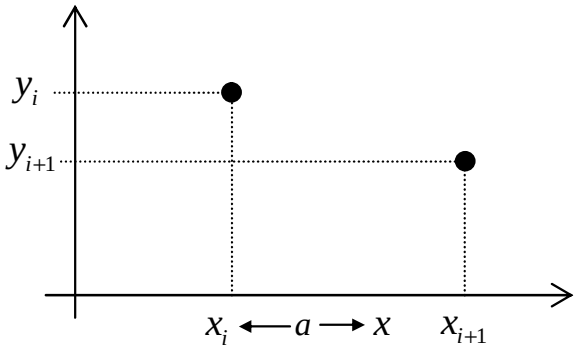


*Figura 1. Interpolazione linere*

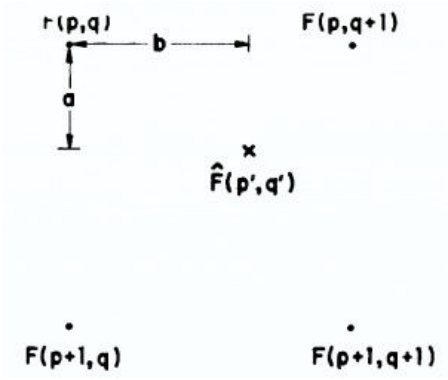


*Figura 2. Interpolazione bilinere*

## Test 1.

Viene fornito il file *f.txt* contenente alcuni campioni di una funzione monodimensionale. Il formato del file è il seguente:
num_elementi (int)
campione1 (double)
campione2 (double)
….
campioneN (double)

Tali valori costituiscono i campioni di un'ipotetica funzione *f*.

a) Eseguire il plot dei singoli campioni con un programma quale Excel o Matlab (non e' necessario scrivere codice C in questo caso).
b) Scrivere un programma C che legga i campioni di *f* dal file fornito e ne interpoli i valori utilizzando entrambi i metodi descritti sopra, in modo da ricavare:
   b1) il doppio dei campioni
   b2) il triplo dei campioni
   Eseguire il plot delle funzioni così ottenute e confrontarli con la funzione originale.

## Test 2.

L'ingrandimento di un'immagine è un caso particolare di interpolazione (resampling). Un'immagine infatti non è altro che una funzione bidimensionale che per ogni valore di *i,j* restituisce il valore della componente rossa, verde e blu. Viene fornito il codice di una classe che permette la lettura/scrittura di un immagine in formato PPM (una spiegazione dettagliata del formato viene fornita qui sotto, allegato 3).

a) Aggiungere alla classe *immagine* due metodi *zoomNN()* e *zoomBilinear()* che eseguono l'ingrandimento dell'immagine utilizzando rispettivamente l'interpolazione nearest neighbor e bilineare. Per semplicità si consideri solo il caso particolare dell'ingrandimento per un fattore due. Confrontare i risultati ottenuti nei due casi sull'immagine *test.ppm* fornita.

*Suggerimenti*:
Non vi è richiesta l'implementazione dei metodi di lettura/scrittura del formato PPM; tuttavia avere un'idea di come funziona il formato può esservi utile, insieme al codice, per capire come sono organizzati i pixel dell'immagine all'interno della classe.
Eseguire l'interpolazione sui canali rosso, verde e blu in maniera indipendente. La classe immagine non fornisce metodi per accedere direttamente al pixel di coordinate (i,j); scrivere opportuni metodi (*setPixel(i,j, pixel)* e *getPixel(i,j,pixel)*). I bordi possono essere gestiti prolungando per continuità l'immagine stessa.

## Estensione al test 2.

L'interpolazione può anche essere eseguita per convoluzione. Nel caso di ingrandimento per un fattore due, la procedura è la seguente. Si costruisce una nuova immagine intervallando i pixel dell'immagine originaria con zeri, nel seguente modo:

| | |
|---|---|
| A B<br><br>C D | A 0 B<br>0 0 0<br>C 0 D |
| immagine originale | immagine intervallata da zeri |

*Figura 3.*

L'immagine così ottenuta si convoluisce con un kernel opportuno in modo da ottenere l'interpolazione voluta. Alcuni esempi di kernel sono riportati in figura 4.

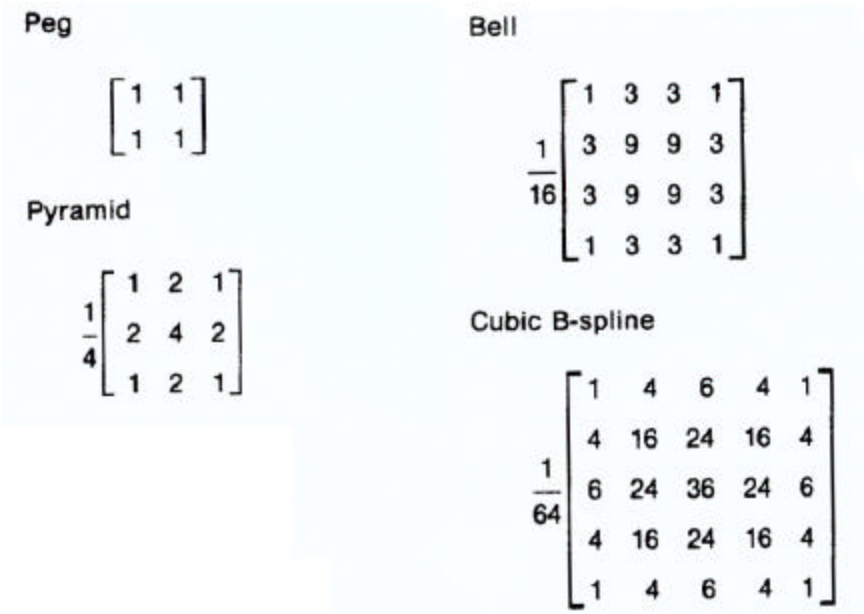


*Figura 4. Alcuni kernel per l'interpolazione con convoluzione.*

Si richiede:

1) aggiungere alla classe immagine un metodo per eseguire la convoluzione con un kernel generico di dimensione NxM.

2) aggiungere un metodo per raddoppiare le dimensioni dell'immagine utilizzando tutti i kernel riportati in figura 4.

## Allegati.

**Allegato 1:** formato ppm, su questo file.

**Allegato 2:** file "f.txt", contenente i dati necessari per il test 1.

**Allegato 3:** file immagine.zip, classe immagine, header file e implementazione (immagine.h e immagine.cpp).

**Allegato 4:** file test.ppm, immagine di prova sulla quale eseguire il test2.

## Allegato1: formato PPM

A PPM file consists of two parts, a header and the image data. The header consists of at least three parts normally terminated by carriage returns and/or linefeeds but the PPM specification only requires white space. The first "line" is a magic PPM identifier, it can be "P3" or "P6" (not including the double quotes!). The next line consists of the width and height of the image as ascii numbers. The last part of the header gives the maximum value of the color components for the pixels, this allows the format to describe more than single byte (0..255) color values. In addition to the above required lines, a comment can be placed anywhere with a "#" character, the comment extends to the end of the line.

The following are all valid PPM headers.

**Header example 1**
P6 1024 788 255
**Header example 2**
P6
1024 788
# A comment
255
**Header example 3**
P3
1024 # the image width
788 # the image height
# A comment
1023

The format of the image data itself depends on the magic PPM identifier. If it is "P3" then the image is given as ascii text, the numerical value of each pixel ranges from 0 to the maximum value given in the header. The lines should not be longer than 70 characters.

**PPM example 4**
P3
# example from the man page
4 4
15
 0  0  0    0  0  0    0  0  0   15  0 15
 0  0  0    0 15  7    0  0  0    0  0  0
 0  0  0    0  0  0    0 15  7    0  0  0
15  0 15    0  0  0    0  0  0    0  0  0

If the PPM magic identifier is "P6" then the image data is stored in byte format, one byte per color component (r,g,b). Comments can only occur before the last field of the header and only one byte may appear after the last header field, normally a carriage return or line feed. "P6" image files are obviously smaller than "P3" and much faster to read. Note that "P6" PPM files can only be used for single byte colors.

While not required by the format specification it is a standard convention to store the image in top to bottom, left to right order. Each pixel is stored as a byte, value 0 == black, value 255 == white. The components are stored in the "usual" order, red - green - blue.