Capitolo 6: Tecniche algoritmiche e numeriche di base

6.1 Introduzione

Illustreremo nel seguito alcune tecniche elementari di manipolazione dei dati e di calcolo numerico. Per ogni argomento tratteremo essenzialmente i soli concetti fondamentali e spesso ometteremo discussioni, anche rilevanti, su questioni di efficienza ed ottimizzazione degli algoritmi presentati. Lo scopo è, come al solito, quello di fornire le informazioni di base che possano consentirvi di risolvere semplici problemi e che possano servire da stimolo per uleriori approfondimenti.

6.2 I vettori

Il vettore è l' elemento chiave per la realizzazione di procedure indicizzate, nelle quali cioè ci si trovi a dover manipolare un insieme numerabile di oggetti simile che devono essere elaborati in sequenza o comunque organizzati in modo ordinato. Ad ogni oggetto si assegna un indice, ovvero un numero positivo che corrisponde alla posizione dell' oggetto all'interno del vettore.

6.2.1 Manipolazione dei vettori

La struttura sintattica di base per la manipolazione dei vettori è ovviamente il ciclo `for`; in effetti i vettori hanno lunghezza definita a priori (a livello di compilazione o, nel caso della allocazione dinamica, a livello di esecuzione del programma), e di conseguenza il ciclo `for` consente di manipolare sequenzialmente tutti gli elementi. Consideriamo l' esempio seguente, nel quale leggiamo una lista di nomi, la copiamo in un vettore e rileggiamo il conetnuto:

```
#include <stdio.h>
#include <string.h>

main() {

  int i,n_names;
  char **names;
  char buffer[100];

  /* Legge il numero di nomi e alloca il vettore */
  printf("Quanti sono i nomi ?  ");
  scanf("%d%*c",&n_names);
  names = (char **)malloc(n_names*sizeof(char*));

  /* Carica nel vettore i nomi */
  for (i=0; i<n_names; i++) {
    fgets(buffer, 100, stdin);
    /* Ogni elemento deve essere allocato con la lunghezza giusta */
    names[i] = (char *)malloc((strlen(buffer))*sizeof(char));
    strncpy(names[i], buffer, strlen(buffer)-1);
  }

  /* Stampa il contenuto del vettore */
  for (i=0; i<n_names; i++) {
```

```
    printf("%d = %s\n", i, names[i]);
  }

}
```

È chiaro da questo esempio come il vettore sia uno strumento molto andatto in tutte quelle cercostanze in cui il numero di oggetti da manipolare è noto all' inizio dell' elaborazione e resta costante.

Le matrici, avendo due indici, richiedono due cicli `for` uno dentro l' altro, come si vede nell' esempio seguente dove stampiamo il contenuto di un array a due indici di float:

```
float matrix[10][20];
int i,j;
...

for (i=0; i<10; i++) {
  for (j=0; j<20; j++) {
    printf("%f ", matrix[i][j]);
  }
  printf("\n");
}
```

È importante ricordare, specie nella manipolazione di array di grandi dimensioni (un megabyte o più), di rispettare con la sequenza degli indici la naturale disposizione in memoria degli elementi. In C questo significa che l' indice più a destra deve scorrere per primo, e quindi deve essere nel ciclo `for` più interno.

In alcuni casi è conveniente utilizzare un vettore per implementare un "buffer circolare"; si tratta di una struttura a n elementi nella quale vengono memorizzate informazioni. Nel momento in cui è necessario scrivere l' infromazione $n+1$ si cancella la prima informazione scritta e si mette la $n+1$-ima al suo posto.

Nell' esempio seguente vediamo un modulo che implementa un buffer circolare di 100 numeri interi. La routine `add` consente di aggiungere un numero al buffer, la routine `read` mostra gli ultimi numeri scritti (fino a un massimo di 100).

```
#define BUFLEN 100

int nel = 0; /* numero di elementi presenti nel buffer */
int pointer = 0; /* indice dell'elemento siccessivo */
int buffer[BUFLEN];

void add(int i) {
  buffer[pointer] = i;
  nel++; if (nel > BUFLEN) nel = BUFLEN;
  pointer++; if (pointer >= BUFLEN) pointer = 0;
}

void read() {
  int i;
  int ip = pointer;
```

```
  for (i=0; i<nel; i++) {
    ip--; if (ip <0) ip = BUFLEN - 1;
    printf("%d\n", buffer[ip]);
  }
}
```

I buffer circolari possono essere utilizzati per implementare strutture di tipo FIFO (First In First Out) o LIFO (Last In First Out) di dimensione fissa.
La ricerca in un vettore è spesso di tipo euristico: si leggono tutti gli elementi e si confronatano con il valore desiderato; quando si trova l' elemento giusto si può usare l' istruzione `break` per interrompre il ciclo senza scandire gli elementi rimanenti; nel modulo appena visto aggiungiamo ad esempio una function che ritorna la posizione nel buffer di un dato numero o -1 se il numero non è nel buffer:

```
int find(int val) {
  int i;
  int ip = pointer;
  int pos = -1;

  for (i=0; i<nel; i++) {
    ip--; if (ip <0) ip = BUFLEN - 1;
    if (buffer[ip] == val) {
      pos = ip;
      break;
    }
  }
  return pos;
}
```

Nel caso si abbia a che fare con vettori di grnde dimensione, la ricerca euristica risulta inefficiente. Si usa talvolta la ricerca "a chiave". Supponiamo ad esempio di aver caricato un elenco telefonico in un vettore (gli elementi del vettore saranno tipo composti da un nome e un numero di telefono), e di voler ricercare un dato nome. Potremmo fabbricare un vettore ausilirario che memorizzi, per ogni lettera dell' alfabeto, l' indice del primo nome nell' elenco che comincia per quella lettera. Invece di cercare euristicamente in tutto l' elenco potremmo con questa tecnica cercare soltanto in un sottoinsieme. Naturalmente il vettore ausilirario deve essere mantenuto aggiornato ogni volta che si aggiunge o si toglie un elemento dall' elenco.

6.2.2 Ordinamento degli elementi di un vettore

Ci sono molte tecnche per ordinare gli elementi di un vettore; la più intuitiva consiste nel cercare euristicamente nell' intero vettore l' elemento più piccolo, copiarlo nella prima locazione di un vettore ausiliario eliminandolo dal vettore originario e procedere coìfino ad avere esaurito gli elementi. Una tecnica un po' più raffinata, detta "bubble sort" consiste nell' analizzare tutti gli elementi, partendo dall' ultimo, scambiando ogni elemento $x(i)$ con il precedente $x(i-1)$ se $x(i) < x(i-1)$; in questo modo si porta l' elemento più piccolo nella posizione iniziale. La procedura va poi ripetuta nel sotto-vettore ottenuto ignorando la prima componente, ed iterata fino a quando non si esegue più alcuno scambio o si è rimasti cin un sotto-vettore di un elemento. Ecco un esempio di codifica in C del "bubble sort":

```
void sort(int *vect, int n) {
  int i,swap,top,temp;

  if (n < 2) return;

  top = 0;
  do {
    swap = 0;
    for (i=n-1; i>top; i--) {
      if (vect[i] < vect[i-1]) {
        swap = 1;
        temp = vect[i];
        vect[i] = vect[i-1];
        vect[i-1] = temp;
      }
    }
    top++;
  } while(swap != 0 && top < n-1);
}
```

6.2.3 Aggiunta di elementi a un vettore

Siccome un vettore consiste in una sequenza statica e di lunghezza definita di elementi, a rigore non è possibile aggiungere un elemento a un vettore. In pratica si usa spesso il trucco di definire in modo generoso le dimensioni dei vettori, usandone poi solo una parte; in questo caso è possibile aggiungere un elemento anche nel mezzo del vettore, a patto di spostare tutti gli elementi successivi di una posizione. Inoltre, se si usa l' alloazione dinamica, è possibile sostituire il vettore con uno più grande se manca lo spazio. Come esempio, vediamo una routine che aggiunge elementi ad un vettore allocato dinamicamente mantenendo costantemente l' ordinamento crescente degli elementi:

```
int *vect = NULL;
int n_el = 0, max_el = 0;

void increase_size(int siz) {
  int i,*temp;

  /* Crea un nuovo vettore */
  max_el = max_el + siz;
  temp = (int *)malloc(max_el*sizeof(int));

  /* Copia il vecchio vettore nel nuovo */
  for (i=0; i<n_el; i++) temp[i] = vect[i];

  /* Cancella il vecchio vettore e riassegna il puntatore */
  if (vect != NULL) free(vect);
  vect = temp;

}
```

```
int add_element(int val) {
   int i;

  /* Controlla se c'e' ancora spazio */
  if (n_el + 1 > max_el) increase_size(100);

  /* Aggiungi il nuovo elemento al suo posto */
  for (i=n_el++; i>0; i--) {
    if (vect[i-1]>val) {
      vect[i] = vect[i-1];
    } else {
      vect[i] = val;
      return i;
    }
  }
  vect[0] = val;
  return 0;
}
```

6.3 Tecniche Numeriche

Il calcolo numerico consiste nella soluzione di equazioni o nel calcolo di integrali ottenuta per sostituzione di particolari valori numerici. Si tratta ovviamente di un metodo molto meno elegante rispetto alla soluzione analitica del problema, ma che rischia di essere la sola possibile in molti casi in cui una soluzione analitica non è realizzabile. Malgrado l' apparente rozzezza, non si deve commettere l' errore di sottovalutare la complessità delle tecniche numeriche: un approccio troppo "spensierato" può condurre a risultati totalmente sbagliati che pure hanno una apparenza del tutto ragionevole.

Nel seguito illustreremo i rudimenti del calcolo numerico. I metodi che illustreremo hanno di fatto solo valenza didattica, e sono del tutto indadatti (per precisone, efficienza e dominio di applicabilità) alla soluzione di problemi "seri". Per maggiori approfondimenti si può ricorrere ad un qualunque testo di calcolo numerico; citiamo il seguente:

```
W. H. Press, S. A. Teukolsky, W. T. Wetterling, B. P. Flannering -
Numerical Recipes in C - Cambridge University Press
```

che ha il singolare vantaggio di essere disponibile on–line all' indirizzo `http:www.nr.com`.

6.3.1 Calcolo di integrali definiti

Il metodo intuitivo di calcolare un integrale definito di una funzione $f(x)$ non singolare, consiste nel dividere l' intervallo di integrazione in N sottointervalli uguali. Siano $x_0, x_1, ..., x_N$ i punti che delimitano i sottointervalli e h la lunghezza dei sottointervalli; potremo allora approssimare l' integrale nell' intervallo $[x_i, x_{i+1}]$ con l' area sotto la retta che passa per i punti $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$. In formule scriveremo:

$$\int_{x_0}^{x_n} f(x)dx \simeq \sum_{i=0}^{N-1} \frac{h}{2}(f(x_i) + f(x_{i+1}))$$

È facile convincersi che questa formula è esatta per una retta; si può anche dimostrare che l' errore E, ovvero la differenza tra l' integrale e la somma, è proporzionale ad $h^3 f''(\bar{x})$ dove $\bar{x}$ è un punto non noto interno all' intervallo di integrazione (da cui si ritrova il risultato che per una retta, che ha derivata seconda sempre nulla, l' errore è zero).

Si può altresì dimostrare che, raggruppando due a due gli intervalli si ottiene una formula del tipo

$$\int_{x_i}^{x_{i+2}} f(x)dx = h\Big(\frac{f(x_i)}{3} + \frac{4f(x_{i+1})}{3} + \frac{f(x_{i+2})}{3}\Big) + O(h^5 f^{(4)}(\bar{x}))$$

dove l' errore è proporzionale ad h^5 moltiplicato per la derivata quarta di f in un punto interno all' intervallo (significa che è esatta per la polinomiali fino all' ordine x^3).

Le tecniche di integrazione presentate non possono ovviamente funzionare su intervalli illimitati e neppure nel caso che la unzione presenti singolarità all' interno dell' intervallo di integrazione.

6.3.2 Ricerca del minimo di una funzione

Per trovare il minimo (o il massimo) di una funzione continua e derivabile si sfrutta il fatto che in tale punto la derivata prima della funzione si annulla e cambia segno. partendo da un punto sufficientemente vicino al minimo ricercato (vuol dire che non ci devono essere altri minimi relativi nei paraggi) si calcola la derivata approssimandola con il rapporto incrementale su un intervallo finito di dimensione h:

$$f'(x) \simeq \frac{f(x+h) - f(x)}{h}$$

Ci si sposta quindi di una quantità pari ad h e si continua fino a che il rapporto incrementale non cambia segno. A questo punto si dimezza il passo di campionamento e si ripete l' operazione nella direzione inversa. Ci si può fermare quando l' intervallo di campionamento scende al di sotto di una quantità minima definita come errore tollerato sulla posizione dell' estremo.

Questo metodo funziona in modo soddisfacente solo con funzioni regolari e prive di estremi relativi. Inoltre è di facile applicabilità solo con funzioni di una variabile. Nelle esercitazioni utilizzeremo delle routines delle librerie CERN per la determinazione dei minimi di funzione a molti parametri. Tali routines sono descritte al Par. 7.1.

6.3.3 Il metodo dei minimi quadrati

Il metodo dei minimi quadrati viene utilizzato per determinare quale tra le funzioni appartenenti ad una data famiglia descriva al meglio un insieme di punti sperimentali affetti (ovviamente) da errore. La situazione tipica è quella in cui si dispone di un set di misure y_i con $1 < i < N$ ciascuna affetta da un errore σ_i e ciascuna corrispondente ad una particolare scelta di variabili indipendenti $\vec{x}_i$ (assumeremo che non ci siano errori sugli $\vec{x}_i$, ipotesi di solito non verificata. Nel corso di ESP1 vedrete come trattare gli errori sulle variabili indipendenti). Si dispone poi di una teoria che prevede una relazione funzionale tra y e $\vec{x}$ espressa sotto forma di una famiglia di funzioni del tipo $y = f(\vec{x}, \vec{p})$, dove $\vec{p}$ rappresenta un insieme di parametri fisici che determinano l' andamento delle funzioni f ma che non sono a priori note. Il problema è quello di determinare quale scelta di $\vec{p}$ fornisce il migliore accordo tra i dati sperimentali e la funzione teorica.

Il metodo dei minimi quadrati consiste nell' affermare che la migliore scelta di $\vec{p}$ è quella che minimizza la funzione $\chi^2(\vec{p})$ definita come

$$\chi^2(\vec{p}) = \sum_{i=0}^{N} \Big(\frac{(y_i - f(\vec{x}_i, \vec{p}))^2}{\sigma_i^2} \Big)$$

Nel caso la funzione f sia una retta, cioè

$$f(x, \vec{p}) = ax + b; \quad \vec{p} = (a, b)$$

il problema della minimizzazione del χ^2 ha una soluzione analitica che avete visto a ESP1. In tutti gli altri casi ci si deve accontentare di una soluzione numerica, che può essere ottenuta con i metodi illustrati nel paragrafo precedente.

Se la miglior funzione teorica si adatta in modo accettabile ai dati sperimentali, ovvero se il valore del minimo χ^2 è sufficientemente piccolo (imparerete nel corso del secondo semestre a dare un significato quantitativo, di tipo statistico, a queste affermazioni) il metodo dei minimi quadrati fornisce in sostanza una misura di $\vec{p}$ derivata dalla misura degli y_i. Questo significa che è necessario calcolare un errore sul valore di $\vec{p}$ ricavato: se ripetessimo la misura degli y_i otterremmo valori diversi (sebbene entro gli errori), e quindi una diversa funzione χ^2 da minimizzare ed un diverso valore di $\vec{p}$ che la minimizza. Quindi in qualche modo l' entità delle fluttuazioni degli y_i determinano una più o meno grande fluttuazione dei valori di $\vec{p}$. Si può dimostrare che l' incertezza su $\vec{p}$ è data dall' insieme dei punti tali che:

$$\chi^2(\vec{p}) - \chi^2(\vec{p}_{min}) < 1$$

dove $\vec{p}_{min}$ è il valore di $\vec{p}$ che minimizza la funzione χ^2. Pur senza dimostrare l' affermazione precedente, possiamo osservare che se abbiamo una sola misura y_0 ed un singolo parametro p potremo scegliere il valore della funzione teorica f in modo che sia identico a y_0, per cui il valore minimo del χ^2 sarà zero in corrispondenza di un dato p_0. Supponiamo ora di ripetere la misura e di trovare un valore $y_0 + \sigma_y$; il nuovo valore di p che annulla χ^2 sarà tale per cui

$$f(p_o + \Delta p) = y_0 + \sigma_i$$

Se calcoliamo la prima funzione χ^2 ottenuta in questo nuovo punto otteniamo:

$$\chi^2(p_0 + \Delta p) = \frac{(y_0 - f(p_0 + \Delta p))^2}{\sigma_y^2} =$$

$$\frac{(y_0 - y_0 - \sigma_y)^2}{\sigma_y^2} = 1$$

Cioè la variazione di p in corrispondenza della variazione di un sigma della misura y_0 corrisponde ad una variazione di χ^2 pari a 1.

Osserviamo che, nel caso di molti parametri, l' insieme dei $\vec{p}$ per cui χ^2 differisce dal minimo per meno di uno può avere una forma complicata, ed eventualmente potrebbe non trattarsi di una regione connessa. Nel caso la funzione χ^2 intorno al minimo sia assimilabile ad un paraboloide in M dimensioni (dove M è il numero di parametri ovvero la dimensione di $\vec{p}$) tale regione sarà un ellissoide.