Capitolo 1

Il MATLAB

1.1 Introduzione al MATLAB

Il Matlab (acronimo delle parole inglesi *MATrix LABoratory*) è un software basato sulla manipolazione di matrici molto utilizzato nel campo della ricerca scientifica, non solo matematica, per la sua grande portabilità (infatti è disponibile sia per grandi workstation che per comuni personal computers), unita ad una notevole facilità d'uso e alle potenzialità di calcolo. Inoltre l'uso del Matlab è reso facile dalla presenza di un manuale dei comandi in linea, che può essere invocato tramite il comando `help`, e dalla presenza del comando `demo` che presenta numerosi e significativi esempi di applicazioni di tutte le funzioni Matlab. Nelle seguenti pagine faremo riferimento alle istruzioni presenti nelle ultime versioni del software.
Una volta lanciata l'esecuzione del programma compare il prompt del software

```
>>
```

il che significa che MatLab resta in attesa che venga effettuata una qualsiasi operazione (editare un programma, lanciare l'esecuzione di un file oppure eseguire un'istruzione digitata sulla riga di comando e così via).
Il comando `help`, come già detto, fornisce tutte le informazioni relative ad un particolare comando oppure una lista di tutti gli argomenti per i quali è presente un aiuto. La sintassi del comando è semplice:

```
>> help
```

oppure

```
>> help comando
```

Per esempio per sapere l'uso del comando `load`, che descriveremo in dettaglio nel seguito, è sufficiente scrivere

```
>> help load
```

Anche il comando `demo` ha una sintassi molto semplice:

```
>> demo
```

a questo punto compariranno sullo schermo alcuni menu e basterà scegliere, tramite il mouse, l'argomento del quale si vuole vedere una dimostrazione.
Il Matlab può essere considerato un interprete le cui istruzioni sono del tipo:

```
variabile = espressione
```

oppure

```
variabile
```

In quest'ultimo caso, quando cioè un'istruzione è costituita solo dal nome di una variabile viene interpretata come la visualizzazione del valore di tale variabile. Vediamo i seguenti esempi.

```
>> b=5;
>> b
ans =
   5
>>


>> b=5
b =
   5
>>
```

Nel primo caso il valore di output di `b` è stato attribuito alla variabile di comodo `ans` (abbreviazione per la parola inglese *answer*). Questo modo di procedere viene utilizzato anche quando si chiede di valutare un'espressione di tipo numerico senza l'ausilio di variabili.

```
>> 3+4
ans =
   7
>>
```

Ogni espressione introdotta viene interpretata e calcolata. Ogni istruzione può essere scritta anche su due righe purchè prima di andare a capo vengano scritti 3 punti $''\ldots''$. Più espressioni possono essere scritte sulla stessa riga purchè siano separate da una virgola o dal punto e virgola. Se una riga di un file Matlab inizia con % allora tale riga viene considerata come un commento. Il Matlab fa distinzione tra lettere minuscole e maiuscole, quindi se abbiamo definito una variabile `A` e facciamo riferimento a questa scrivendo `a` essa non viene riconosciuta.
Le frecce della tastiera consentono di richiamare e riutilizzare comandi scritti in precedenza; utilizzando infatti ripetutamente il tasto $\uparrow$ vengono visualizzate le linee di comando precedentemente scritte. Per tornare ad un'istruzione sorpassata basta premere il tasto $\downarrow$. Con i tasti $\longleftarrow$ e $\longrightarrow$ ci si sposta veso sinistra oppure verso destra sulla riga di comando su cui ci si trova.

1.2 Assegnazione di matrici

La prima cosa da imparare del Matlab è come manipolare le matrici che costituiscono la struttura fondamentale dei dati. Una matrice è una tabella di elementi caratterizzata da due dimensioni: il numero delle righe e quello delle colonne. I vettori sono matrici aventi una delle dimensioni uguali a 1. Infatti esistono due tipi di vettori: i *vettori riga* aventi dimensione $1 \times n$, e i *vettori colonna* aventi dimensione $n \times 1$. I dati scalari sono matrici di dimensione 1×1. Le matrici possono essere introdotte in diversi modi, per esempio possono essere assegnate esplicitamente, o caricate da file di dati esterni, o generate utilizzando funzioni predefinite.
Per esempio l'istruzione

```
>> A = [1 2 3; 4 5 6; 7 8 9];
```

assegna alla variabile `A` una matrice di tre righe e tre colonne. Gli elementi di una riga della matrice possono essere separate da virgole o dallo spazio, mentre le diverse righe sono separate da un punto e virgola. Se alla fine dell'assegnazione viene messo il punto e virgola allora la matrice non viene vi-

sualizzata sullo schermo. In generale se vogliamo assegnare ad `A` una matrice ad m righe ed n colonne la sintassi è la seguente:

```
>> A = [riga 1; riga 2; ...; riga m];
```

Per assegnare ad una variabile `x` un vettore riga si ha

```
>> x = [3 -4 5];
```

gli elementi possono anche essere separati da una virgola

```
>> x = [3,-4,5];
```

Per assegnare invece ad una variabile un vettore colonna basta separare gli elementi con un punto e virgola:

```
>> y = [1;-3;6];
```

La stessa matrice `A` dell'esempio visto in precedenza può essere assegnata anche a blocchi:

```
>> A = [ 1 2 3; 4 5 6];
>> b = [ 7 8 9];
>> A = [ A; b];
```

mentre in modo analogo si può anche aggiungere una colonna:

```
>> A = [-1 2 3; 0 5 6; -5 4 3];
>> x = [-7; 0; 9];
>> A = [ A, x];
```

Descriviamo ora alcune funzioni predefinite che forniscono in output determinate matrici.

```
>> A=rand(m,n)
```

costruisce una matrice $m \times n$ di elementi casuali uniformemente distribuiti tra 0 e 1;

```
>> A=zeros(m,n)
```

costruisce una matrice $m \times n$ di elementi nulli;

```
>> A=ones(m,n)
```

costruisce una matrice $m \times n$ di elementi tutti uguali a 1;

```
>> A=eye(m,n)
```

costruisce una matrice $m \times n$ i cui elementi sono uguali a 1 sulla diagonale principale e 0 altrove. Per le funzioni appena viste se uno dei due parametri è omesso allora la matrice costruita viene considerata quadrata.
Il dimensionamento delle matrici è automatico. Per esempio se si pone

```
>> B = [1 2 3; 4 5 6];
```

e successivamente

```
>> B = [1 0; 0 7];
```

il programma riconosce che la matrice `B` ha cambiato le dimensioni da 2×3 a 2×2.
Per identificare l'elemento della matrice che si trova al posto `j` nella riga `i` si usa la notazione `A(i,j)`.
Per esempio `A(4,2)` indica l'elemento che si trova nella quarta riga e in colonna 2. La numerazione delle righe e delle colonne parte sempre da 1 (quindi il primo elemento della matrice è sempre `A(1,1)`).
Per fare riferimento ad un singolo elemento di un vettore (sia riga che colonna) è sufficiente utilizzare un solo indice (quindi la notazione `x(i)` indica l'`i`-esima componente del vettore `x`).
Se si fa riferimento a un elemento di una matrice di dimensione $m \times n$ che non esiste allora il Matlab segnala l'errore con il seguente messaggio:
`Index exceeds matrix dimension`.
Se `C` è una matrice non ancora inizializzata allora l'istruzione

```
>> C(3,2)= 1
```

fornisce come risposta

```
C =
   0   0
   0   0
   0   1
```

cioè il programma assume come dimensioni per `C` dei numeri sufficientemente grandi affinchè l'assegnazione abbia senso. Se ora si pone

```
>> C(1,3)= 2
```

si ha:

```
C =
   0   0   2
   0   0   0
   0   1   0
```

In Matlab gli indici devono essere strettamente positivi mentre se si richiede un elemento di indice negativo oppure uguale a zero si ha sullo schermo il seguente messaggio di errore:
`Index into matrix is negative or zero`

1.2.1 Sottomatrici e notazione :

Vediamo ora alcuni esempi che illustrano l'uso di : per vettori e matrici. Le istruzioni

```
>> x=[1:5];
```

e

```
>> x=1:5;
```

sono equivalenti all'assegnazione diretta del vettore `x`:

```
>> x=[1 2 3 4 5];
```

Ciò vale anche per vettori di elementi reali. Infatti l'istruzione

```
>> x=[0.2:0.2:1.2];
```

equivale a scrivere

```
>> x=[0.2 0.4 0.6 0.8 1.0 1.2];
```

Inoltre è possibile anche l'uso di incrementi negativi:

```
>> x=[5:-1:1];
```

è equivalente a

```
>> x=[5 4 3 2 1];
```

L'istruzione

```
>> x=x(n:-1:1);
```

inverte gli elementi del vettore `x` di dimensione n. La notazione : può essere anche applicata a matrici. Infatti se `A` è una matrice abbiamo:

```
>> y=A(1:4,3);
```

assegna al vettore colonna `y` i primi 4 elementi della terza colonna della matrice `A`;

```
>> y=A(4,2:5);
```

assegna al vettore riga `y` gli elementi della quarta riga di `A` compresi tra il secondo e il quinto;

```
>> y=A(:,3);
```

assegna al vettore colonna `y` la terza colonna di `A`;

```
>> y=A(2,:);
```

assegna al vettore riga `y` la seconda riga di `A`;

```
>> B=A(1:4,:);
```

assegna alla matrice `B` le prime 4 righe di `A`;

```
>> B=A(:,2:6);
```

assegna alla matrice `B` le colonne di `A` il cui indice è compreso tra 2 e 6;

```
>> B=A(:,[2 4]);
```

assegna alla matrice `B` la seconda e la quarta colonna di `A`;

```
>> A(:,[2 4 5])=B(:,1:3);
```

sostituisce alle colonne 2, 4 e 5 della matrice `A` le prime 3 colonne della matrice `B`.

1.3 Operazioni su matrici e vettori

In Matlab sono definite le seguenti operazioni su matrici e vettori:

$+$	addizione
$-$	sottrazione
$*$	moltiplicazione
$\hat{}$	elevazione a potenza
$'$	trasposto
$/$	divisione
$(\ )$	specificano l'ordine di valutazione delle espressioni

Ovviamente queste operazioni possono essere applicate anche a scalari. Se le dimensioni delle matrici coinvolte non sono compatibili allora viene segnalato un errore eccetto nel caso di operazione tra uno scalare e una matrice. Per esempio se A è una matrice di qualsiasi dimensione allora l'istruzione

```
>> C = A+2;
```

assegna alla matrice C gli elementi di A incrementati di 2.
Nel caso del prodotto tra matrici è necessario prestare molta attenzione alle dimensioni delle matrici. Infatti ricordiamo che se $A \in \mathbb{R}^{m\times p}$ e $B \in \mathbb{R}^{p\times n}$ allora la matrice

$$C = A \cdot B, \qquad C \in \mathbb{R}^{m\times n}$$

si definisce nel seguente modo:

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}, \qquad i = 1, \ldots, m \; j = 1, \ldots, n$$

ed è la matrice che viene calcolata scrivendo l'istruzione

```
>> C = A*B
```

In caso contrario se scriviamo

```
>> C = B*A
```

allora il programma segnala errore a meno che non sia $m = n$.
È importante notare che le operazioni $*, \hat{}$, e $/$ operano elemento per elemento se sono precedute da un punto:

`C=A.*B` $\Rightarrow$ $c_{ij} = a_{ij}b_{ij}$

`C=A./B` $\Rightarrow$ $c_{ij} = a_{ij}/b_{ij}$

`C=A.^B` $\Rightarrow$ $c_{ij} = a_{ij}^{b_{ij}}$

1.3.1 Costanti predefinite

Come la maggior parte dei linguaggi di programmazione il Matlab ha alcune costanti predefinite cioè delle variabili che hanno un proprio valore senza che esso venga esplicitamente assegnato:

`eps`	precisione di macchina ($\simeq 2.2 \cdot 10^{-16}$)
`pi`	π (cioe' 3.14159265358979)
`i`	unita' immaginaria ($\sqrt{-1}$)
`j`	unita' immaginaria ($\sqrt{-1}$)
`realmax`	il piu' grande numero floating point ($1.7976e+308$)
`realmin`	il piu' piccolo numero floating point ($2.2251e-308$)
`inf`	infinito (∞)
`NaN`	Not a Number.

La costante `inf` è ottenuta come risultato di una divisione per zero oppure il calcolo del logaritmo di zero o se il risultato è un overflow (per esempio `2*realmax`). La costante `NaN` invece è ottenuta come risultato di operazioni matematicamente non definite come $0/0$ oppure $\infty - \infty$.
Come accade per la maggior parte dei linguaggi di programmazione anche in Matlab è possibile definire variabili il cui nome è una costante predefinita, quindi per esempio è possibile usare la variabile `i` come indice intero.

1.3.2 Operatori relazionali e logici

I seguenti sono gli operatori relazionali

$<$	minore
$>$	maggiore
$<=$	minore o uguale
$>=$	maggiore o uguale
$==$	uguale
$\sim=$	diverso

Una relazione di tipo logico assume valore 0 o 1 a seconda del fatto se essa sia rispettivamente falsa o vera. Per esempio scrivendo

```
>> 3<5
```

otterremo

```
>> 3<5
ans =
   1
```

oppure scrivendo

```
>> 1>5
```

la risposta è

```
>> 1>5
ans =
   0
```

Quando un operatore relazionale è applicato a matrici di dimensioni uguali si ottiene come risultato una matrice i cui elementi sono 1 oppure 0. Vediamo il seguente esempio:

```
>> A=[2 1 ; 0 3];
>> B=[2 -1 ; -2 3];
>> A==B
ans =
   1 0
   0 1

>> A>B
ans =
   0 1
   1 0

>> A>=B
ans =
   1 1
   1 1
```

Gli operatori logici che il Matlab consente di utilizzare sono i seguenti:

&	AND
|	OR
$\sim$	NOT

1.4 Funzioni predefinite

In MatLab è possibile utilizzare un gran numero di funzioni predefinite in grado di semplificare notevolmente la scrittura di programmi. In questo paragafo elenchiamo soltanto quelle più utilizzate nell'ambito di programmi di calcolo. Tali funzioni sono state suddivise in tre gruppi: le funzioni scalari, quelle vettoriali e quelle matriciali.

1.4.1 Funzioni scalari

Le funzioni Matlab riportate di seguito sono praticamente quelle di tipo matematico che accettano come argomento di input un numero reale (o complesso), che sintatticamente deve essere scritto tra parentesi tonde subito dopo il nome della funzione (esempio: `cos(2*pi)`, oppure `log(x+1)`).

`sin`	seno
`cos`	coseno
`tan`	tangente
`asin`	arcoseno
`acos`	arcocoseno
`atan`	arcotangente
`sinh`	seno iperbolico
`cosh`	coseno iperbolico
`tanh`	tangente iperbolica
`asinh`	arcoseno iperbolico
`acosh`	arcocoseno iperbolico
`atanh`	arcotangente iperbolica
`exp`	esponenziale
`log`	logaritmo naturale
`log10`	logaritmo in base 10
`sqrt`	radice quadrata
`abs`	valore assoluto

`rem`	resto della divisione
`sign`	segno
`round`	arrotondamento
`floor`	parte intera inferiore
`ceil`	parte intera superiore

Tra queste funzioni appena nominate le ultime tre meritano un piccolo approfondimento; nella seguente tabella sono riportati i valori di tali funzioni per differenti numeri reali:

`x`	`round(x)`	`floor(x)`	`ceil(x)`
3.7	4	3	4
3.1	3	3	4
-4.7	-5	-5	-4
-4.3	-4	-5	-4

Osserviamo che `floor(x)` è sempre minore di `x` mentre `ceil(x)` è maggiore di `x`.
Le funzioni riportate in precedenza possono essere applicate anche a variabili di tipo vettore (o matrice), in questo si comportano come se fossero applicate a ciascun elemento del vettore (o della matrice). Per esempio se `x` è un vettore riga allora `abs(x)` è anch'esso un vettore riga le cui componenti sono i valori assoluti delle componenti del vettore `x`.

1.4.2 Funzioni vettoriali

Le seguenti funzioni Matlab operano principalmente su vettori (riga o colonna):

`max`	massimo elemento di un vettore
`min`	minimo elemento di un vettore
`sum`	somma degli elementi di un vettore
`prod`	prodotto degli elementi di un vettore
`sort`	ordinamento di un vettore
`length`	numero di elementi di un vettore
`fliplr`	inverte gli elementi di un vettore riga
`flipud`	inverte gli elementi di un vettore colonna

Le funzioni `max` e `min` possono fornire in uscita anche l'indice della componente massima (o minima) del vettore. La sintassi in questo caso è la seguente:

```
>> [massimo,k]=max(x);
>> [minimo,k]=min(x);
```

Le funzioni appena elencate possono essere applicate anche a variabili di tipo matrice, ma producono come risultato un vettore riga contenente i risultati della loro applicazione a ciascuna colonna. Per esempio scrivere

```
>> y=max(A);
```

significa assegnare al vettore (riga) `y` gli elementi massimi delle colonne della matrice `A`. Per ottenere il risultato della loro azione alle righe basta applicare la stessa funzione alla matrice trasposta `A'`. Volendo trovare il massimo elemento della matrice A si dovrebbe scrivere `max(max(A))`. La funzione `max` può essere applicata per trovare anche la posizione dell'elemento massimo (ovvero l'indice della riga e della colonna in cui si trova). Infatti, se `A` è una matrice allora

```
>> [mass,r]=max(A);
```

calcola il vettore dei massimi (ovvero `mass`) e l'indice di riga in cui si trova. Per esempio il massimo della prima colonna è `mass(1)` che si trova sulla riga `r(1)`, il massimo della seconda colonna è `mass(2)` che si trova sulla riga `r(2)`, e così via. Applicando la funzione al vettore `mass`, si trova:

```
>> [massimo,c]=max(mass);
```

in cui `massimo` è il valore cercato che si trova nella colonna `c`, la riga sarà invece `r(c)`.

1.4.3 Funzioni di matrici

Le più utili funzioni di matrici sono le seguenti:

`eig`	autovalori e autovettori
`inv`	inversa
`det`	determinante
`size`	dimensioni
`norm`	norma
`cond`	numero di condizione in norma 2
`rank`	rango
`tril`	parte triangolare inferiore

`triu`	parte triangolare superiore
`diag`	fornisce in output un vettore colonna dove e' memorizzata la parte diagonale di una matrice. Se la funzione e' applicata invece ad un vettore allora in uscita avremo una matrice diagonale i cui elementi principali sono quelli del vettore di input.

Le funzioni Matlab possono avere uno o più argomenti di output. Per esempio `y = eig(A)`, o semplicemente `eig(A)`, produce un vettore colonna contenente gli autovalori di A mentre

```
>> [U,D] = eig(A);
```

produce una matrice U le cui colonne sono gli autovettori di A e una matrice diagonale D con gli autovalori di A sulla sua diagonale. Anche la funzione `size` ha due parametri di output:

```
>> [m,n] = size(A);
```

assegna a `m` ed `n` rispettivamente il numero di righe e di colonne della matrice `A`.
La funzione `norm` se viene applicata ad una matrice calcola la norma 2 della stessa matrice. È tuttavia possibile specificare anche altre norme. Per esempio

```
>> norm(A,'inf');
```

calcola la norma infinito di `A` mentre

```
>> norm(A,1);
```

calcola la norma 1 di `A`.

1.5 Le istruzioni for, while, if e switch

Il Matlab è dotato delle principali istruzioni che servono a renderlo un linguaggio strutturato. La più importante istruzione per la ripetizione in sequenza delle istruzioni è il `for`, che ha la seguente sintassi:

```
for var=val_0:step:val_1
    lista istruzioni
end
```

La variabile *var* assume come valore iniziale *val_0*, viene eseguita la lista di istruzioni che segue, poi è incrementata del valore *step*, vengono rieseguite le istruzioni che seguono e così via, finchè il suo valore non supera *val_1*. Il valore dello *step* può essere negativo, nel qual caso il valore di *val_0* deve essere logicamente superiore a *val_1*.
Nel caso in cui il valore di *step* sia uguale a 1 questo può essere omesso, in tutti gli altri casi va necessariamente specificato. Questo vuol dire che se $step = 1$ la sintassi dell'istruzione `for` è la seguente

```
for var=val_0:val_1
    lista istruzioni
end
```

La sintassi per l'istruzione `while` è la seguente.

```
while espressione logica
    istruzioni
end
```

Le *istruzioni* vengono eseguite fintantochè l'*espressione logica* rimane vera. La sintassi completa dell'istruzione `if` è la seguente:

```
if espressione logica
    istruzioni
elseif espressione logica
    istruzioni
else
    istruzioni
end
```

I rami `elseif` possono essere più di uno come anche essere assenti. Anche il ramo `else` può mancare. Vediamo ora alcuni esempi di come le istruzioni appena descritte possono essere applicate.
Se all'interno delle istruzioni che seguono il `for` o il `while` si verifica la necessità di interrompere il ciclo delle istruzioni allora ciò può essere fatto utilizzando l'istruzione `break`.
Ultima istruzione di questo tipo (e presente solo nell'ultima versione del programma) è l'istruzione `switch` che ha lo stesso ruolo e quasi la stessa sintassi dell'omonima istruzione in linguaggio C:

```
switch variabile
   case valore_0
      istruzioni
   case valore_1
      istruzioni
   case valore_2
      istruzioni
   otherwise
      istruzioni
end
```

che, in funzione del valore assunto dalla variabile, esegue o meno una serie di istruzioni. In particolare se nessuno dei valori previsti è assunto dalla variabile allora viene previsto un caso alternativo (`otherwise`) che li contempla tutti. Vediamo il seguente esempio:

```
switch rem(n,2)
   case 0
      disp('n e'' un numero pari')
   case 1
      disp('n e'' un numero dispari')
   otherwise
      disp('Caso impossibile')
end
```

1.6 Istruzioni per gestire il Workspace

Il comando

```
>> who
```

elenca le variabili presenti nell'area di lavoro, mentre il comando

```
>> whos
```

elenca, oltre al nome delle variabili, anche il tipo e l'occupazione di memoria. Una variabile può essere cancellata da tale area con il comando

```
>> clear nome variabile
```

mentre il comando

```
>> clear
```

cancella tutte le variabili presenti nell'area di lavoro.
Premendo contemporaneamente i tasti *Ctrl* e *c* si interrompe l'esecuzione di un file Matlab. L'istruzione

```
>> save
```

salva il contenuto dell'area di lavoro (cioè le variabili e il loro valore) nel file binario `matlab.mat`. Se invece si scrive

```
>> save nomefile
```

allora tutta l'area di lavoro viene salvata nel file `nomefile.mat`. Se invece si vogliono salvare solo alcune variabili e non tutta l'area di lavoro allora è possibile farlo specificando, oltre al nome del file, anche l'elenco di tali variabili. Per esempio

```
>> save nomefile A B x
```

salva nel file `nomefile.mat` solo il contenuto delle variabili `A, B` e `x`. Scrivendo

```
>> save nomefile A B x -ascii
```

allora il file `nomefile.mat` non ha il formato binario ma ascii, e questo è utile se si vuole verificare il contenuto del file.
Per ripristinare il contenuto dell'area di lavoro dal file `matlab.mat` il comando è

```
>> load
```

mentre è possibile anche in questo caso specificare il file da caricare. Facendo riferimento all'esempio del comando `save` allora scrivendo

```
>> load nomefile
```

ripristina le variabili e il loro valore che erano stati memorizzati nel file `nomefile.mat`.

1.7 M-files

Il Matlab può eseguire una sequenza di istruzioni memorizzate in un file. Questi file prendono il nome di *M-files* perchè la loro estensione è `.m`. Ci sono due tipi di M-files: gli *script files* e i *function files*.

Script files

Uno script file consiste in una sequenza di normali istruzioni Matlab. Se il file ha come nome `prova.m` allora basterà eseguire il comando

```
>> prova
```

per far sì che le istruzioni vengano eseguite. Le variabili di uno script file sono di tipo globale, per questo sono spesso utilizzati anche per assegnare dati a matrici di grosse dimensioni, in modo tale da evitare errori di input. Per esempio se in file `assegna.m` vi è la seguente assegnazione:

```
A=[0 -2 13 4; -5 3 10 -8; 10 -12 14 17; -1 4 5 6];
```

allora l'istruzione `assegna` servirà per definire la matrice `A`.

Function files

Permettono all'utente di definire funzioni che non sono standard. Le variabili definite nelle funzioni sono locali, anche se esistono delle istruzioni che permettono di operare su variabili globali. Vediamo il seguente esempio.

```
function a = randint(m,n)
% randint(m,n) Fornisce in output una matrice
% di dimensioni m×n di numeri casuali
% interi compresi tra 0 e 9.
a = floor(10*rand(m,n));
return
```

Tale funzione va scritta in un file chiamato `randint.m` (corrispondente al nome della funzione). La prima linea definisce il nome della funzione e gli argomenti di input e di output. Questa linea serve a distinguere i function files dagli script files. Quindi l'istruzione Matlab

```
>> c=randint(5,4);
```

assegna a c una matrice di elementi interi casuali di 5 righe e 4 colonne. Le variabili m, n e a sono interne alla funzione quindi il loro valore non modifica il valore di eventuali variabili globali aventi lo stesso nome. Vediamo ora il seguente esempio di funzione che ammette un numero di argomenti di output superiore a 1,

```
function [somma, prodotto]= stat(x)
% stat(x) Fornisce in output due numeri
% contenenti la somma ed il prodotto degli
% elementi di un vettore di input x.
somma = sum(x);
prodotto = prod(x);
```

La funzione `stat` deve essere richiamata mediante l'istruzione:

```
>> [p q]=stat(y);
```

così alle variabili `p` e `q` vengono assegnate rispettivamente la somma e il prodotto degli elementi di `y`.
In definitiva se la funzione ammette più di un parametro di output allora la prima riga del function file deve essere modificata nel seguente modo:

```
function [var_0,var_1,var_2]= nomefunzione(inp_0,inp_1)
```

Come ulteriore esempio scriviamo una funzione Matlab per calcolare il valore assunto da un polinomio per un certo valore x. Ricordiamo che assegnato un polinomio di grado n

$$p(x) = a_1 + a_2x + a_3x^2 + \cdots + a_nx^{n-1} + a_{n+1}x^n$$

la seguente *Regola di Horner* permette di valutare il polinomio minimizzando il numero di operazioni necessarie. Tale regola consiste nel riscrivere lo stesso polinomio in questo modo:

$$p(x) = a_1 + x(a_2 + x(a_3 + \cdots + x(a_n + a_{n+1}x)\ldots))).$$

In questo modo il numero di moltiplicazioni necessarie passa all'incirca da $O(n^2/2)$ a $O(n)$. Vediamo ora la funzione Matlab che implementa tale regola.

```
function y= horner(a,x,n)
% horner(a,x,n) Fornisce in output il valore
```

```
% di un polinomio di grado n nel punto x
% a vettore di n+1 elementi contenente i
%  coefficienti del polinomio
% x punto dove si vuol calcolare il polinomio
% n grado del polinomio
p=0;
for i=n+1:-1:1
   p=p*x+a(i);
end
y=p;
```

Per interrompere l'esecuzione di una funzione e tornare al programma chiamante si usa l'istruzione

```
return
```

Tale istruzione può essere anche quella che chiude una funzione (anche se la sua presenza non è obbligatoria).
All'interno di una funzione possono essere utilizzate due variabili, `nargin`, che è uguale al numero di parametri di input che sono passati alla funzione, e `nargout`, uguale al numero di parametri che si vogliono in output. Infatti una caratteristica sintattica è che una funzione può accettare un numero variabile di parametri di input o fornire un numero variabile di parametri di output.

1.8 Messaggi di errore, Istruzioni di Input

Stringhe di testo possono essere visualizzate sullo schermo mediante l'istruzione `disp`. Per esempio

```
disp('Messaggio sul video')
```

Se la stringa tra parentesi contiene un apice allora deve essere raddoppiato. La stessa istruzione può essere utilizzata anche per visualizzare il valore di una variabile: è sufficiente scrivere, al posto della stringa e senza apici, il nome della variabile.
I messaggi di errore possono essere visualizzati con l'istruzione `error`. Consideriamo il seguente esempio:

```
if a==0
   error('Divisione per zero')
else
   b=b/a;
end
```

l'istruzione `error` causa l'interruzione nell'esecuzione del file.
In un M-file è possibile introdurre dati di input in maniera interattiva mediante l'istruzione `input`. Per esempio quando l'istruzione

```
iter = input('Inserire il numero di iterate ')
```

viene eseguita allora il messaggio è visualizzato sullo schermo e il programma rimane in attesa che l'utente digiti un valore da tastiera e tale valore, di qualsiasi tipo esso sia, viene assegnato alla variabile `iter`.
Vediamo ora un modo per poter memorizzare in un file di ascii l'output di un M-file oppure di una sequenza di istruzioni Matlab. Infatti

```
>> diary nomefile
>> istruzioni
>> diary off
```

serve a memorizzare nel file *nomefile* tutte le istruzioni e l'output che è stato prodotto dopo la prima chiamata della funzione e prima della seconda.

1.8.1 Formato di output

In Matlab tutte le elaborazioni vengono effettuate in doppia precisione. Il formato con cui l'output compare sul video può però essere controllato mediante i seguenti comandi.

```
format short
```

È il formato utilizzato per default dal programma ed è di tipo fixed point con 4 cifre decimali;

```
format long
```

Tale formato è di tipo fixed point con 14 cifre decimali;

```
format short e
```

Tale formato è la notazione scientifica (esponenziale) con 4 cifre decimali;

```
format long e
```

Tale formato è la notazione scientifica (esponenziale) con 15 cifre decimali. Vediamo per esempio come i numeri $4/3$ e $1.2345e-6$ sono rappresentati nei formati che abbiamo appena descritto e negli altri disponibili:

```
format short            1.3333
format short e          1.3333e+000
format short g          1.3333
format long             1.33333333333333
format long e           1.333333333333333e+000
format long g           1.333333333333333
format rat              4/3


format short            0.0000
format short e          1.2345e-006
format short g          1.2345e-006
format long             0.00000123450000
format long e           1.234500000000000e-006
format long g           1.2345e-006
format rat              1/810045
```

Oltre ai formati appena visti il comando

```
format compact
```

serve a sopprimere le righe vuote e gli spazi dell'output scrivendo sullo schermo il maggior numero di informazioni possibile, in modo appunto compatto.

1.9 La grafica con il Matlab

Il Matlab dispone di numerose istruzioni per grafici bidimensionali e tridimensionali e anche di alcune funzioni per la creazione di animazioni. Il comando `plot` serve a disegnare curve nel piano xy. Infatti se `x` e `y` sono due vettori di uguale lunghezza allora il comando

```
>> plot(x,y)
```

traccia una curva spezzata che congiunge i punti `(x(i),y(i))`. Per esempio

```
>> x=-4:.01:4;
>> y=sin(x);
>> plot(x,y)
```

traccia il grafico della funzione seno nell'intervallo $[-4, 4]$.
L'istruzione `plot` ammette un parametro opzionale di tipo stringa (racchiuso tra apici) per definire il tipo e il colore del grafico. Infatti è possibile scegliere tra 4 tipi di linee, 5 di punti e 8 colori base. In particolare

`'-'`	linea continua
`'--'`	linea tratteggiata
`'-.'`	linea tratteggiata e a punti
`':'`	linea a punti
`'+'`	piu'
`'o'`	cerchio
`'x'`	croce
`'.'`	punto
`'*'`	asterisco
`'y'`	colore giallo
`'r'`	colore rosso
`'c'`	colore ciano
`'m'`	colore magenta
`'g'`	colore verde
`'w'`	colore bianco
`'b'`	colore blu
`'k'`	colore nero

Volendo tracciare per esempio un grafico con linea a puntini e di colore verde l'istruzione è:

```
>> plot(x,y,':g')
```

L'istruzione `plot` consente di tracciare più grafici contemporaneamente. Per esempio

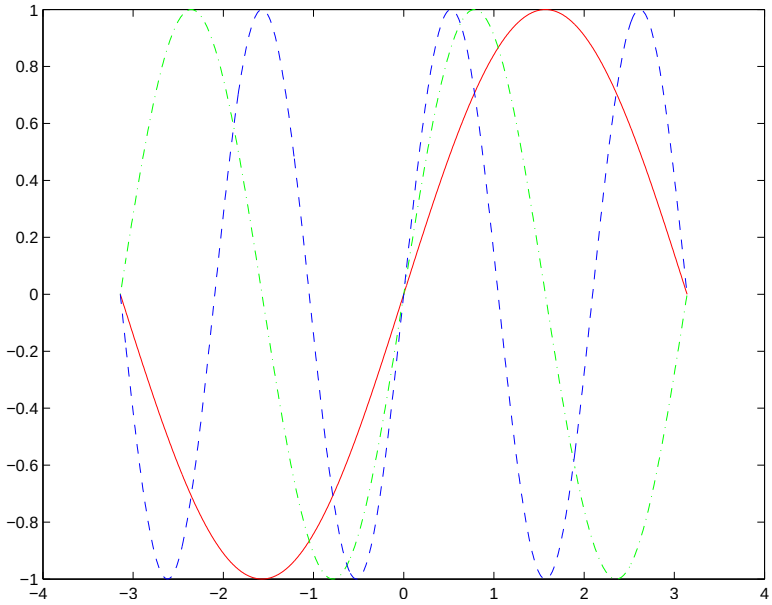


Figura 1.1:

```
>> x=-pi:pi/500:pi;
>> y=sin(x);
>> y1=sin(2*x);
>> y2=sin(3*x);
>> plot(x,y,'r',x,y1,'-.g',x,y2,'--b')
```

traccia tre grafici nella stessa figura, il primo a tratto continuo (tratto di default) rosso, il secondo verde tratteggiato e a punti, il terzo tratteggiato e di colore blu. Nella Figura 1.1 è riportato il risultato di tale istruzione.
Vediamo ora le altre più importanti istruzioni grafiche:

```
>> title(stringa)
```

serve a dare un titolo al grafico che viene visualizzato al centro nella parte superiore della figura;

```
>> xlabel(stringa)
```

stampa una stringa al di sotto dell'asse delle ascisse;

```
>> ylabel(stringa)
```

stampa una stringa a destra dell'asse delle ordinate (orientata verso l'alto). Per inserire un testo in una qualsiasi parte del grafico esiste il comando

```
>> text(x,y,'testo')
```

che posiziona la stringa di caratteri *testo* nel punto di coordinate (x, y) (`x` e `y` non devono essere vettori). Di tale comando ne esiste anche una versione che utilizza il mouse:

```
>> gtext('testo')
```

posiziona il testo nel punto selezionato all'interno del grafico schiacciando il pulsante sinistro del mouse.
Il grafico tracciato con il comando `plot` è scalato automaticamente, questo vuol dire che le coordinate della finestra grafica sono calcolate dal programma, tuttavia l'istruzione

```
>> axis([x_min x_max y_min y_max])
```

consente di ridefinire gli assi, e quindi le dimensioni della finestra del grafico corrente. Le istruzioni grafiche del Matlab permettono di tracciare curve in tre dimensioni, superfici, di creare delle animazioni e così via. Per approfondire le istruzioni che consentono queste operazioni si può richiedere l'`help` per le istruzioni `plot3`, `mesh` e `movie`.
Nel caso di una superficie, per tracciare il grafico si devono definire due vettori, uno per le ascisse, cioè `x`, uno per le ordinate, `y`, e una matrice `A` per memorizzare le quote, cioè tale che

```
A(j,i)=f(x(j),y(i))
```

Nella Figura 1.2 è tracciato come esempio il grafico della funzione

$$f(x, y) = x(10 - x) + y(10 - y), \qquad 0 \leq x, y \leq 10.$$

Le istruzioni per tracciare tale grafico sono le seguenti:

```
x=[0:0.1:10];
y=[0:0.1:10];
n=length(x);
for i=1:n
     for j=1:n
```

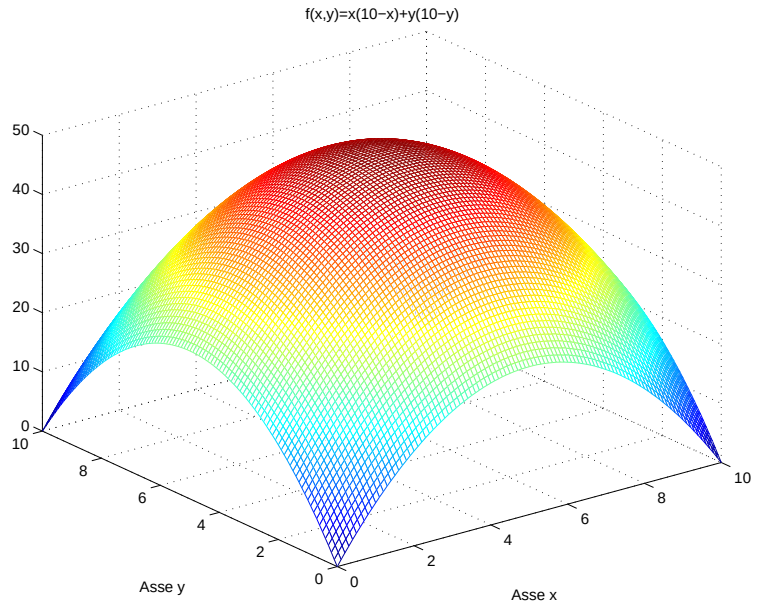


Figura 1.2:

```
            A(j,i)=x(j)*(10-x(j))+y(i)*(10-y(i));
        end
end
mesh(x,y,A);
xlabel('Asse x');
ylabel('Asse y');
title('f(x,y)=x(10-x)+y(10-y)');
```

1.10 Octave

Il programma `Octave` è essenzialmente un emulatore del linguaggio di `Matlab`. `Octave` nacque presso l'Università del Texas nel 1988 come programma di calcolo per l'ingegneria chimica; il nome `Octave` è il nome del docente di uno dei corsi presso i quali John W. Eaton, il principale autore del programma, ne iniziò lo sviluppo nel 1992.
Le caratteristiche fondamentali di `Octave` sono due:

- è completamente gratuito e copiabile dal sito web `http://www.octave.org`;
- è 'software libero': il suo codice sorgente è disponibile a tutti e tutti possono modificarlo secondo le loro esigenze.

Entrambe le caratteristiche lo rendono molto interessante dal punto di vista didattico.

1.10.1 Alcune differenze

Per lanciare il programma `Octave` basta cliccare due volte sull'icona del programma. Per uscire dal programma digitare `quit` o `exit`. Lanciato il programma si entra in una sessione interattiva avente come prompt `>`. Come per il Matlab, il comando `help` fornisce informazioni sulla documentazione di supporto. In particolare, scrivendo il nome di un comando specifico dopo la parola `help` si visualizzano le principali informazioni relative a quel comando. Tutti i comandi Matlab descritti nei paragrafi di questo capitolo (assegnazione variabili, matrici, vettori; variabili predefinite realmin, realmax, eps, pi; comandi rand, triu, ones, ';', ':', plot, plot3, mesh, title, xlabel, ylabel, hold on, hold off, i vari format, size, length, clear, who, whos, inline, feval, input, disp, save, load, diary, ...) funzionano anche in `Octave`. Nel linguaggio `Octave` il commento è preceduto dal carattere %, come in Matlab, o dal simbolo ♯. Analogamente per l'elevamento a potenza si può usare il comando `^`, come in Matlab, o $**$. Ad esempio per calcolare il cubo di 2 si possono usare indistintamente `2^3` e `2**3`.
A differenza del Matlab il programma `Octave` permette di fare assegnazioni multiple contemporaneamente. Ad esempio:
`> x=y=z=0`
assegna simultaneamente alle variabili x, y e z il valore zero.
Il programma `Octave` prevede operatori di *incremento*:

- pre incremento:
  `> ++x`
  incrementa la varibile x di uno e restituisce il nuovo valore. È equivalente a `> x = x+1`
- post incremento:
  `> x++`

incrementa la varibile x di uno e restituisce il valore prima dell'incremento

- pre decremento:
  `> --x`
  decrementa la varibile x di uno e restituisce il nuovo valore. È equivalente a `> x = x-1`

- post decremento:
  `> x--`
  decrementa la varibile x di uno e restituisce il valore prima del decremento.

Per matrici e vettori gli operatori di incremento e decremento lavorano su ogni elemento dell'operando.
Le istruzioni `for`, `if`, `while` e `switch` funzionano come in Matlab tranne per il fatto che le istruzioni si chiudono con `endfor`, `endif`, `endwhile` e `endswitch`, rispettivamente e non con un generico `end`.
Un comando che differenzia l'`Octave` dal Matlab è il `do-until`. Esso è simile al `while` solo che questo esegue rapidamente lo statement fino a che la condizione non è vera e il test della condizione è alla fine del ciclo, in modo da eseguire il corpo del ciclo almeno una volta.
L'espressione di do-until è:

```
do
istruzioni
until (condizione)
```

Ad esempio

```
> fib = ones (1,10);
> do
> i++;
> fib(i)=fib(i-1)+fib(i-2);
> until (i= =10)
```

genera il vettore `fib` che contiene i primi 10 elementi della sequenza di Fibonacci.

1.11 Programmazione

Come per il Matlab anche in `Octave` si distinguono i files *function* e quelli *script*.
Per aprire l'editor di testo si usa il comando
`> edit`.
Una volta creato il file esso viene salvato automaticamente con estensione `.m`; per richiamarlo dal terminale di `Octave` basta digitare il nome del file.
Il programma legge i programmi dalla propria 'cartella attuale'. Per sapere quale sia, si usi il comando `pwd`. In Windows, per cambiare tale cartella nella cartella 'Desktop' si usi il comando

```
cd "C:/Documents and Settings/.../Desktop",
```

dove ... rappresenta il percorso necessario a raggiungere la cartella Desktop relativa alla propria area del disco. In Linux la sintassi è identica, usando il percorso `/home/nomeutente/.../`.

Capitolo 1

L'insieme dei numeri macchina

1.1 Introduzione al Calcolo Numerico

Il Calcolo Numerico è una disciplina che fa parte di un ampio settore della Matematica Applicata che prende il nome di Analisi Numerica. Si tratta di una materia che è al confine tra la Matematica e l'Informatica poichè cerca di risolvere i consueti problemi matematici utilizzando però una via algoritmica. In pratica i problemi vengono risolti indicando un processo che, in un numero finito di passi, fornisca una soluzione numerica e soprattutto che sia implementabile su un elaboratore. I problemi matematici che saranno affrontati nelle pagine seguenti sono problemi di base: risoluzione di sistemi lineari, approssimazione delle radici di funzioni non lineari, approssimazione di funzioni e dati sperimentali, calcolo di integrali definiti. Tali algoritmi di base molto spesso non sono altro se non un piccolo ingranaggio nella risoluzione di problemi ben più complessi.

1.2 Rappresentazione macchina dei numeri reali

Uno dei più importanti aspetti legati all'uso di algoritmi numerici per la soluzione di problemi matematici è l'affidabilità dei risultati ottenuti. I metodi descritti in questo capitolo funzionerebbero sempre bene (cioè fornirebbero risultati numericamente affidabili) se non si dovesse aver a che fare con una serie di problemi legati alla struttura dell'elaboratore che stiamo utilizzando.

Un primo problema che ci troviamo ad affrontare è il modo con cui i numeri reali sono rappresentati nella memoria di un elaboratore. Giusto per rendersi conto delle difficoltà che si incontrano va osservato che i numeri reali sono infiniti mentre la memoria di un calcolatore ha una capacità finita. Una seconda osservazione consiste nel fatto che un numero reale ammette molteplici rappresentazioni. Per esempio il numero 12.47 può essere scritto in diversi modi

$$12.47 = 1.247 \times 10^1 = 0.1247 \times 10^2 = 1247 \times 10^{-2}.$$

Questa prima ambiguità viene risolta convenzionalmente utilizzando come rappresentazione la seguente:

$$x = segno(x)\ q \times 10^p$$

dove segno$(x) = \pm 1$, $q = 0.d_1d_2d_3 \ldots d_k \ldots$ è un numero reale positivo minore di 1 con le cifre d_i comprese tra 0 e 9 se si utilizza la rappresentazione in base 10, e si impone $d_1 \neq 0$, cioè

$$\frac{1}{10} \leq q < 1.$$

Lo stesso discorso si può ripetere scegliendo una qualsiasi base $\beta \in \mathbb{N}$, cioè

$$x = segno(x)\ q \times \beta^p, \qquad \text{dove} \quad q = 0.d_1d_2d_3 \ldots d_k \ldots$$

con le cifre d_i comprese tra 0 e $\beta - 1$, e $d_1 \neq 0$, cioè

$$\frac{1}{\beta} \leq q < 1.$$

Assegnato $x \in \mathbb{R}$, $x \neq 0$, l'espressione

$$x = segno(x)\ \beta^p \times 0.d_1d_2 \ldots d_k \ldots$$

prende il nome di rappresentazione in base β di x. Il numero p viene detto esponente (o caratteristica), i valori d_i sono le cifre della rappresentazione, mentre $0.d_1d_2 \ldots d_k \ldots$ si dice mantissa. Il numero x viene normalmente rappresentato con la cosiddetta notazione posizionale $x = segno(x)(.d_1d_2d_3 \ldots) \times \beta^p$, che viene detta normalizzata. In alcuni casi à ammessa una rappresentazione in notazione posizionale tale che $d_1 = 0$, che viene detta denormalizzata. Quindi un qualunque numero reale $x \neq 0$ può essere rappresentato con infinite cifre nella mantissa. Inoltre come è noto l'insieme dei numeri reali ha cardinalità infinita. Poichè un elaboratore è dotato di memoria finita non è

possibile memorizzare:
a) gli infiniti numeri reali
b) le infinite (in generale) cifre di un numero reale.
Risulta perciò importante stabilire dei criteri che permettano di rappresentare in macchina i numeri reali che è possibile rappresentare in modo da commettere il minimo errore possibile.
Assegnati i numeri $\beta, t, m, M \in \mathbb{N}$ con $\beta \geq 2$, $t \geq 1$, $m, M > 0$, si dice insieme dei numeri di macchina con rappresentazione normalizzata in base β con t cifre significative l'insieme:

$$\mathcal{F}(\beta, t, m, M) = \{ \quad x \in \mathbb{R} : \ x = \pm\beta^p \times 0.d_1 d_2 \dots d_t \text{ con } \ 0 \leq d_i \leq \beta - 1,$$

$$d_1 \neq 0, -m \leq p \leq M\} \cup \{0\}.$$

Infatti poichè zero sfugge alla rappresentazione in base normalizzata viene assegnato per definizione all'insieme $\mathcal{F}$. Normalmente lo zero viene rappresentato con mantissa nulla ed esponente $-m$.
Osserviamo che un elaboratore che abbia le seguenti caratteristiche:

- t campi di memoria per la mantissa, ciascuno dei quali può assumere β differenti configurazioni (e perciò può memorizzare una cifra d_i),
- un campo di memoria che può assumere $m + M + 1$ differenti configurazioni (e perciò può memorizzare i differenti valori p dell'esponente),
- un campo che può assumere due differenti configurazioni (e perciò può memorizzare il segno + o −),

è in grado di rappresentare tutti gli elementi dell'insieme $\mathcal{F}(\beta, t, m, M)$.
Assumiamo ora che $x \in \mathbb{R}$ e che abbia la seguente rappresentazione in base β:

$$x = segno(x) \ \beta^p \times 0.d_1 d_2 \dots d_t$$

con $d_1 \neq 0$ e $p \in [-m, M]$. Allora è evidente che $x \in \mathcal{F}(\beta, t, m, M)$ e pertanto verrà rappresentato esattamente su un qualunque elaboratore che utilizzi $\mathcal{F}(\beta, t, m, M)$ come insieme dei numeri di macchina.
Assumiamo ora che $x \in \mathbb{R}$ ma $x \notin \mathcal{F}(\beta, t, m, M)$. In questo caso si pone il problema di associare ad x un numero di macchina che lo rappresenti in modo da commettere il più piccolo errore possibile. Per risolvere questo problema facciamo innanzitutto, e solo per semplicità di esposizione, le seguenti ipotesi $x \in \mathbb{R}$, $x > 0$ e β numero pari.
Distinguiamo quindi i seguenti casi:

a) $p \notin [-m, M]$. Se $p < -m$ allora x è più piccolo del più piccolo numero di macchina: in questo caso si dice che si è verificato un underflow (l'elaboratore interrompe la sequenza di calcoli e segnala con un messaggio l'underflow). Se $p > M$ allora vuol dire che x è più grande del più grande numero di macchina e in questo caso si dice che si è verificato un overflow (anche in questo caso l'elaboratore si ferma e segnala l'overflow).

b) $p \in [-m, M]$, $x = \beta^p \times 0.d_1 d_2, \ldots d_t \ldots$, ed esiste un $k > t$ tale che $d_k \neq 0$. Anche in questo caso poichè x ha più di t cifre significative $x \notin \mathcal{F}$. È però possibile rappresentare x mediante un numero in $\mathcal{F}$ con un'opportuna operazione di taglio delle cifre decimali che seguono la t-esima. In pratica i criteri di taglio sono i seguenti:

1. troncamento di x alla t-esima cifra significativa

$$\tilde{x} = \mathrm{tr}(x) = \beta^p \times 0.d_1 d_2 \ldots d_t$$

2. arrotondamento di x alla t-esima cifra significativa

$$\tilde{x} = \mathrm{arr}(x) = \beta^p \times 0.d_1 d_2 \ldots \tilde{d}_t$$

dove

$$\tilde{d}_t = \begin{cases} d_t + 1 & \text{se } d_{t+1} \geq \beta/2 \\ \\ d_t & \text{se } d_{t+1} < \beta/2. \end{cases}$$

Se $x \in \mathbb{R}$ e $\tilde{x}$ è la sua rappresentazione di macchina, chiameremo errore assoluto la quantità

$$E_a = |x - \tilde{x}|$$

mentre per $x \neq 0$ chiameremo errore relativo la quantità

$$E_r = \frac{|x - \tilde{x}|}{|x|}.$$

Nel seguito assumeremo $x > 0$ e supporremo anche che la rappresentazione di x in $\mathcal{F}(\beta, t, m, M)$ non dia luogo ad underflow o overflow.

Teorema 1.2.1 *Sia $x = \pm\beta^p \ 0.d_1 d_2 \ldots d_t \ldots$ tale che la sua rappresentazione macchina non dia luogo a fenomeni di underflow o overflow, allora risulta:*

$$|tr(x) - x| < \beta^{p-t}$$

$$|arr(x) - x| \leq \frac{1}{2}\beta^{p-t}$$

dove il segno di uguaglianza vale se e solo se $d_{t+1} = \frac{\beta}{2}$ *e* $d_{t+i} = 0$ *per* $i \geq 2$.

Teorema 1.2.2 *Sia* $x = \pm\beta^p\ 0.d_1d_2\ldots d_t\ldots$, $x \neq 0$, *se* $\tilde{x}$ *è la sua rappresentazione di macchina cioè* $\tilde{x} \in \mathcal{F}(\beta, t, m, M)$, *allora*

$$\left|\frac{\tilde{x} - x}{x}\right| < u$$

$$\left|\frac{\tilde{x} - x}{\tilde{x}}\right| < u$$

dove

$$u = \begin{cases} \beta^{-t+1} & se \quad \tilde{x} = tr(x) \\ \frac{1}{2}\beta^{-t+1} & se \quad \tilde{x} = arr(x). \end{cases}$$

La quantità u che interviene nel precedente teorema si chiama precisione di macchina o zero macchina. La rappresentazione di $x \in \mathbb{R}$ attraverso $\tilde{x} \in \mathcal{F}(\beta, t, m, M)$ si dice rappresentazione in virgola mobile di x o rappresentazione floating point, con troncamento se $\tilde{x} = \mathrm{tr}(x)$, con arrotondamento se $\tilde{x} = \mathrm{arr}(x)$.
Se $\tilde{x}$ è un'approssimazione di $x \in \mathbb{R}$ con un errore relativo minore di β^{1-t}, si dice che t cifre della rappresentazione in base β sono significative.
Un problema simile si presenta anche quando si effettuano delle operazioni aritmetiche su numeri reali. In fatti non è garantito, in generale, che un'operazione aritmetica su due numeri macchina fornisca come risultato un numero di macchina. Per esempio considerati $x = (.11)10^0$ e $y = (.11)10^{-2}$, $x, y \in \mathcal{F}(10, 2, m, M)$, si ha:

$$x + y = (.1111)10^0 \notin \mathcal{F}(10, 2, m, M)$$

Per poter realizzare la naturale ed importante Proprietà di chiusura di una operazione in un certo insieme risulta importante definire delle operazioni di macchina che permettano appunto di realizzare tale proprietà. Un requisito essenziale che si richiede nel costruire un'aritmetica di macchina è il seguente. Indicata con $\cdot$ una delle quattro operazioni aritmetiche $+, -, \times, \div$ e con $\odot$ la corrispondente operazione di macchina dev'essere:

$$x \odot y = (x \cdot y)(1 + \varepsilon), \qquad |\varepsilon| < u \tag{1.1}$$

per ogni $x, y \in \mathcal{F}(\beta, t, m, M)$ tali che $x \odot y$ non dia luogo ad overflow o underflow. Si può dimostrare che

$$x \odot y = \mathrm{tr}(x \cdot y)$$

e

$$x \odot y = \mathrm{arr}(x \cdot y)$$

soddisfano la (1.1) e dunque danno luogo ad operazioni di macchina. Le quattro operazioni così definite danno luogo alla aritmetica di macchina o aritmetica finita.
Si può dimostrare che per le operazioni di macchina non valgono alcune proprietà, come per esempio l'associatività dell'addizione e della moltiplicazione o la distributività della moltiplicazione rispetto all'addizione, che invece sono valide per le operazioni tra numeri reali.
Supponiamo ora di voler valutare la somma tra due numeri reali x e y. Siano $fl(x)$ e $fl(y)$ rispettivamente le loro rappresentazioni di macchina. Vogliamo vedere quale è l'errore relativo che viene commesso dall'elaboratore quando calcola $x + y$.

$$\begin{aligned} fl(x) \oplus fl(y) &= [fl(x) + fl(y)](1+\varepsilon) = \\ &= [x(1+\varepsilon_x) + y(1+\varepsilon_y)](1+\varepsilon) = \\ &= (x + x\varepsilon_x + y - y\varepsilon_y)(1+\varepsilon) = \\ &= (x+y) + (x+y)\varepsilon + x\varepsilon_x + y\varepsilon_y + x\varepsilon\varepsilon_x - y\varepsilon\varepsilon_y. \end{aligned}$$

Una maggiorazione per l'errore relativo è la seguente

$$\begin{aligned} \frac{|(fl(x) \oplus fl(y)) - (x+y)|}{|x+y|} \leq \; & |\varepsilon| + \frac{|x|}{|x+y|}(|\varepsilon_x| + |\varepsilon||\varepsilon_x|) + \\ & + \frac{|y|}{|x+y|}(|\varepsilon_y| + |\varepsilon||\varepsilon_y|). \end{aligned} \tag{1.2}$$

Se x e y hanno lo stesso segno risulta

$$\max(|x|, |y|) \leq |x+y|$$

e dalla (1.2) segue la maggiorazione

$$\frac{|(fl(x) \oplus fl(y)) - (x+y)|}{|x+y|} \leq 3u + O(u^2)$$

dove u è la precisione di macchina ed $O(u^2)$ indica la maggiorazione per i cosiddetti termini quadratici dell'errore, che sono trascurabili in quanto ogni singolo fattore è maggiorato dalla precisione di macchina.

Capitolo 2

Equazioni non Lineari

2.1 Introduzione

Le radici di un'equazione non lineare $f(x) = 0$ non possono, in generale, essere espresse esplicitamente e anche se ciò è possibile spesso l'espressione si presenta in forma talmente complicata da essere praticamente inutilizzabile. Di conseguenza per poter risolvere equazioni di questo tipo siamo obbligati ad utilizzare metodi numerici che sono, in generale, di tipo iterativo, cioè partendo da una (o in alcuni casi più) approssimazioni della radice, producono una successione x_0, x_1, $x_2, \ldots$, convergente alla radice. Per alcuni di questi metodi per ottenere la convergenza è sufficiente la conoscenza di un intervallo $[a, b]$ che contiene la soluzione, altri metodi richiedono invece la conoscenza di una buona approssimazione iniziale. Talvolta è opportuno utilizzare in maniera combinata due metodi, uno del primo tipo e uno del secondo.
Prima di analizzare alcuni metodi per l'approssimazione delle radici dell'equazione $f(x) = 0$ diamo la definizione di molteplicità di una radice.

Definizione 2.1.1 *Sia $f \in \mathcal{C}^r([a, b])$ per un intero $r > 0$. Una radice α di $f(x)$ si dice di molteplicità r se*

$$\lim_{x \to \alpha} \frac{f(x)}{(x - \alpha)^r} = \gamma, \qquad \gamma \neq 0, c \neq \pm\infty. \tag{2.1}$$

Se α è una radice della funzione $f(x)$ di molteplicità r allora risulta

$$f(\alpha) = f'(\alpha) = \cdots = f^{(r-1)}(\alpha) = 0, \qquad f^{(r)}(\alpha) = \gamma \neq 0.$$

2.1.1 Localizzazione delle radici

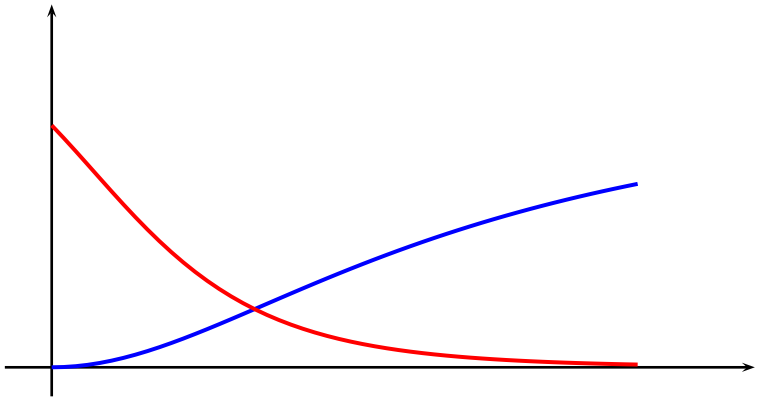
Nei successivi paragrafi saranno descritti alcuni metodi numerici per il calcolo approssimato delle radici di un'equazione non lineare. Tali metodi numerici sono di tipo iterativo, ovvero consistono nel definire una successione (o più successioni), che, a partire da un'assegnata approssimazione iniziale (nota), converga alla radice α in un processo al limite. Infatti poichè non esistono tecniche generali che consentano di trovare l'espressione esplicita di α in un numero finito di operazioni, allora questa può essere calcolata in modo approssimato solo in modo iterativo. Questa perculiarità tuttavia richiede che sia nota appunto un'approssimazione iniziale o, almeno, un intervallo di appartenenza. Il problema preliminare è quello di localizzare la radice di una funzione, problema che viene affrontato in modo grafico. Per esempio considerando la funzione

$$f(x) = \sin\big(\log(x^2+1)\big) - \frac{e^{-x}}{x^2+1}$$

risulta immediato verificare che il valore dell'ascissa in cui si annulla è quello in cui si intersecano i grafici delle funzioni

$$g(x) = \sin\big(\log(x^2+1)\big) \qquad\qquad h(x) = \frac{e^{-x}}{x^2+1}.$$

Un modo semplice per stimare tale valore è quello di tracciare i grafici delle due funzioni, come riportato nella seguente figura in cui il grafico di $h(x)$ è in rosso, mentre quello di $g(x)$ è blu, e l'intervallo di variabilità di x è $[0, 2.5]$.



Calcolando le funzioni in valori compresi in tale intervallo di variabilità si può restringere lo stesso intervallo, infatti risulta

$$g(0.5) = 0.2213 < h(0.5) = 0.48522$$

e

$$g(1) = 0.63896 > h(1) = 0.18394,$$

da cui si deduce che $\alpha \in]0.5, 1[$.

2.1.2 Il Metodo di Bisezione

Sia $f : [a, b] \to \mathbb{R}$, $f \in \mathcal{C}([a, b])$, e sia $f(a)f(b) < 0$. Sotto tali ipotesi esiste sicuramente almeno un punto nell'intervallo $[a, b]$ in cui la funzione si annulla. L'idea alla base del Metodo di Bisezione (o metodo delle bisezioni) consiste nel costruire una successione di intervalli $\{I_k\}_{k=0}^{\infty}$, con $I_0 = [a_0, b_0] \equiv [a, b]$, tali che:

1. $I_{k+1} \subset I_k$;
2. $\alpha \in I_k$, $\forall k \geq 0$;
3. l'ampiezza di I_k tende a zero per $k \to +\infty$.

La successione degli I_k viene costruita nel seguente modo. Innanzitutto si pone

$$I_0 = [a_0, b_0] = [a, b]$$

e si calcola il punto medio

$$c_1 = \frac{a_0 + b_0}{2}.$$

Se $f(c_1) = 0$ allora $\alpha = c_1$, altrimenti si pone:

$$I_1 = [a_1, b_1] \equiv \begin{cases} a_1 = a_0 \quad b_1 = c_1 & \text{se } f(a_0)f(c_1) < 0 \\ a_1 = c_1 \quad b_1 = b_0 & \text{se } f(a_0)f(c_1) > 0. \end{cases}$$

Ora, a partire da $I_1 = [a_1, b_1]$, si ripete la stessa procedura. In generale al passo k si calcola

$$c_{k+1} = \frac{a_k + b_k}{2}.$$

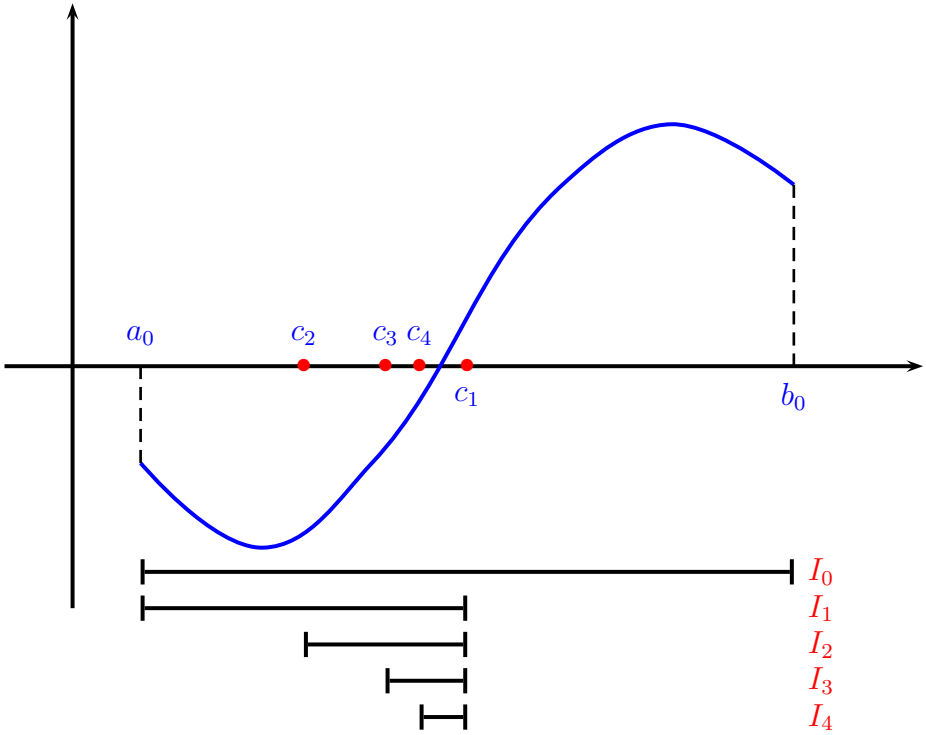
Se $f(c_{k+1}) = 0$ allora $\alpha = c_{k+1}$, altrimenti si pone:

$$I_{k+1} = [a_{k+1}, b_{k+1}] \equiv \begin{cases} a_{k+1} = a_k \quad b_{k+1} = c_k \quad \text{se } f(a_k)f(c_{k+1}) < 0 \\ \\ a_{k+1} = c_{k+1} \quad b_{k+1} = b_k \quad \text{se } f(a_k)f(c_{k+1}) > 0. \end{cases}$$

La successione di intervalli I_k così costruita soddisfa automaticamente le condizioni 1) e 2). Per quanto riguarda la 3) abbiamo:

$$b_k - a_k = \frac{b_{k-1} - a_{k-1}}{2} = \frac{b_0 - a_0}{2^k}$$

e dunque l'ampiezza di I_k tende a zero quando $k \to +\infty$.



Generalmente costruendo le successioni $\{a_k\}$ e $\{b_k\}$ accade che la condizione $f(c_k) = 0$, per un certo valore k, non si verifica mai a causa degli errori di arrotondamento. Quindi è necessario stabilire un opportuno criterio di stop che ci permetta di fermare la procedura quando riteniamo di aver raggiunto una precisione soddisfacente. Per esempio si può imporre:

$$b_k - a_k \leq \varepsilon \tag{2.2}$$

dove ε è una prefissata tolleranza. La (2.2) determina anche un limite per il numero di iterate infatti:

$$\frac{b_0 - a_0}{2^k} \leq \varepsilon \qquad \Rightarrow \qquad k > \log_2 \left(\frac{b_0 - a_0}{\varepsilon} \right).$$

Poichè $b_k - \alpha \leq b_k - a_k$, il criterio (2.2) garantisce che α è approssimata da c_{k+1} con un errore assoluto minore di ε. Se $0 \notin [a, b]$ si può usare come criterio di stop

$$\frac{b_k - a_k}{\min(|a_k|, |b_k|)} \leq \varepsilon \tag{2.3}$$

che garantisce che α è approssimata da c_{k+1} con un errore relativo minore di ε. Un ulteriore criterio di stop è fornito dal test:

$$|f(c_k)| \leq \varepsilon. \tag{2.4}$$

È comunque buona norma utilizzare due criteri di stop insieme, per esempio (2.2) e (2.4) oppure (2.3) e (2.4).

2.1.3 Il metodo della falsa posizione

Una variante del metodo delle bisezioni è appunto il metodo della falsa posizione. Partendo sempre da una funzione $f(x)$ continua in un intervallo $[a, b]$ tale che $f(a)f(b) < 0$, in questo caso si approssima la radice considerando l'intersezione della retta passante per i punti $(a, f(a))$ e $(b.f(b))$ con l'asse x. L'equazione della retta è

$$y = f(a) + \frac{f(b) - f(a)}{b - a}(x - a)$$

pertanto il punto c_1, sua intersezione con l'asse x, è:

$$c_1 = a - f(a)\frac{b - a}{f(b) - f(a)}.$$

Si testa a questo punto l'appartenenza della radice α ad uno dei due intervalli $[a, c_1]$ e $[c_1, b]$ e si procede esattamente come nel caso del metodo delle bisezioni, ponendo

$$[a_1, b_1] \equiv \begin{cases} a_1 = a, \quad b_1 = c_1 & \text{se } f(a)f(c_1) < 0 \\ \\ a_1 = c_1, \quad b_1 = b & \text{se } f(a)f(c_1) > 0. \end{cases}$$

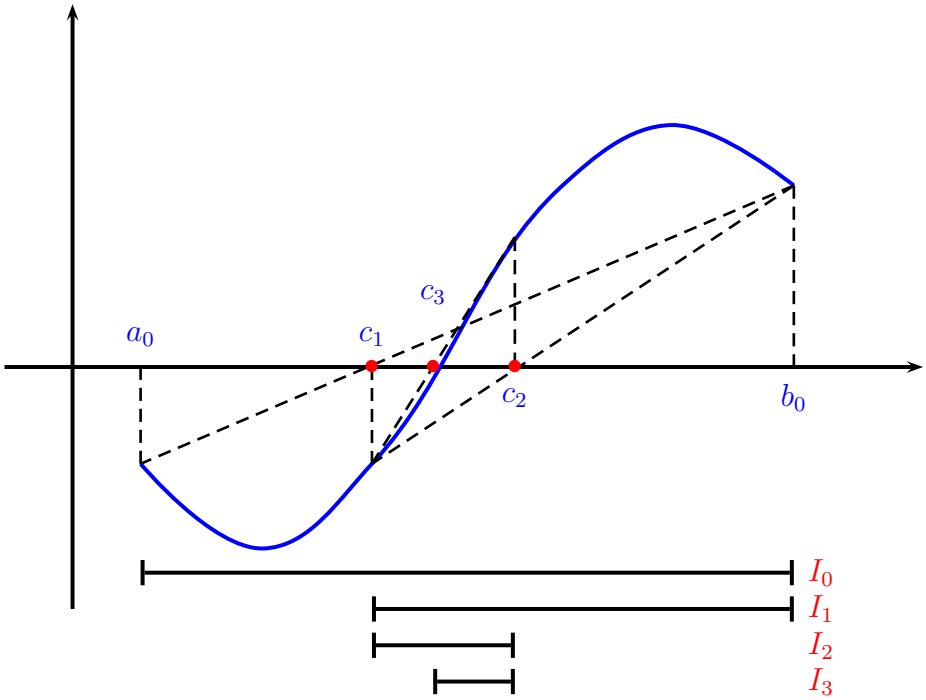
Ad un generico passo k si calcola

$$c_k = a_{k-1} - f(a_{k-1})\frac{b_{k-1} - a_{k-1}}{f(b_{k-1}) - f(a_{k-1})}$$

e si pone

$$[a_k, b_k] \equiv \begin{cases} a_k = a_{k-1} \quad b_k = c_k & \text{se } f(a_{k-1})f(c_k) < 0 \\ a_k = c_k \quad b_k = b_{k-1} & \text{se } f(a_{k-1})f(c_k) > 0. \end{cases}$$

Anche per questo metodo è possibile dimostrare la convergenza nella sola ipotesi di continuità della funzione $f(x)$. Nella seguente figura è rappresentato graficamente il metodo della falsa posizione.




```
function [alfa,k]=bisezione(f,a,b,tol)
%
% La funzione approssima la radice con il metodo di bisezione
%
% Parametri di input
```

```
% f = funzione della quale calcolare la radice
% a = estremo sinistro dell'intervallo
% b = estremo destro dell'intervallo
% tol = precisione fissata
%
% Parametri di output
% alfa = approssimazione della radice
% k = numero di iterazioni
%
if nargin==3
    tol = 1e-8; % Tolleranza di default
end
fa = feval(f,a);
fb = feval(f,b);
if fa*fb>0
    error('Il metodo non e'' applicabile')
end
c = (a+b)/2;
fc = feval(f,c);
k = 0;
while (b-a)>tol | abs(fc)>tol
    if fa*fc<0
        b = c;
        fb = fc;
    else
        a = c;
        fa = fc;
    end
    c = (a+b)/2;
    fc = feval(f,c);
    if nargout==2
        k = k+1;
    end
end
alfa = c;
return
```

2.2 Metodi di Iterazione Funzionale

Il metodo di bisezione può essere applicato ad una vastissima classe di funzioni, in quanto per poter essere applicato si richiede solo la continuità della funzione. Tuttavia ha lo svantaggio di risultare piuttosto lento, infatti ad ogni passo si guadagna in precisione una cifra binaria. Per ridurre l'errore di un decimo sono mediamente necessarie 3.3 iterazioni. Inoltre la velocità di convergenza non dipende dalla funzione $f(x)$ poichè il metodo utilizza esclusivamente il segno assunto dalla funzione in determinati punti e non il suo valore. Il metodo delle bisezioni può essere comunque utilizzato con profitto per determinare delle buone approssimazioni della radice α che possono essere utilizzate dai metodi iterativi che stiamo per descrivere.
Infatti richiedendo alla f supplementari condizioni di regolarità è possibile individuare una vasta classe di metodi che forniscono le stesse approssimazioni del metodo di bisezione utilizzando però un numero di iterate molto minore. In generale questi metodi sono del tipo:

$$x_{k+1} = g(x_k) \qquad k = 0, 1, 2, \ldots \tag{2.5}$$

dove x_0 è un'assegnato valore iniziale e forniscono un'approssimazione delle soluzioni dell'equazione

$$x = g(x). \tag{2.6}$$

Ogni punto α tale che $\alpha = g(\alpha)$ si dice punto fisso o punto unito di g.
Per poter applicare uno schema del tipo (2.5) all'equazione $f(x) = 0$, bisogna prima trasformare questa nella forma (2.6). Ad esempio se $[a, b]$ è l'intervallo di definizione di f ed $h(x)$ è una qualunque funzione tale che $h(x) \neq 0$, per ogni $x \in [a, b]$, si può porre:

$$g(x) = x - \frac{f(x)}{h(x)}. \tag{2.7}$$

Ovviamente ogni punto fisso di g è uno zero di f e viceversa.

Teorema 2.2.1 *Sia $g \in \mathcal{C}([a, b])$ e assumiamo che la successione $\{x_k\}$ generata da (2.5) sia contenuta in $[a, b]$. Allora se tale successione converge, il limite è il punto fisso di g.*

Dimostrazione.

$$\alpha = \lim_{k\to+\infty} x_{k+1} = \lim_{k\to+\infty} g(x_k) = g(\lim_{k\to+\infty} x_k) = g(\alpha). \ \square$$

Teorema 2.2.2 *Sia α punto fisso di g e $g \in \mathcal{C}^1([\alpha-\rho, \alpha+\rho])$, per qualche $\rho > 0$. Scelto x_0 tale che*

$$|x_0 - \alpha| \leq \rho$$

per la successione $\{x_k\}_{k=0}^{\infty}$ generata da (2.5) si ha che se $|g'(x)| < 1$, per $|x-\alpha| \leq \rho$, allora $|x_k - \alpha| \leq \rho$, per ogni k, e la successione $\{x_k\}$ converge a α.

Dimostrazione. Sia

$$\lambda = \max_{|x-\alpha|\leq\rho} |g'(x)| < 1.$$

Proviamo per induzione che tutti gli elementi della successione $\{x_k\}$ sono contenuti nell'intervallo di centro α e ampiezza 2ρ. Per $k = 0$ si ha banalmente $x_0 \in [\alpha-\rho, \alpha+\rho]$. Assumiamo che $|x_k - \alpha| \leq \rho$ e dimostriamolo per $k+1$.

$$|x_{k+1} - \alpha| = |g(x_k) - g(\alpha)| = |g'(\xi_k)||x_k - \alpha|$$

dove $|\xi_k - \alpha| < |x_k - \alpha| \leq \rho$. Pertanto

$$|x_{k+1} - \alpha| \leq \lambda|x_k - \alpha| < |x_k - \alpha| \leq \rho.$$

Proviamo ora che:

$$\lim_{k\to+\infty} x_k = \alpha.$$

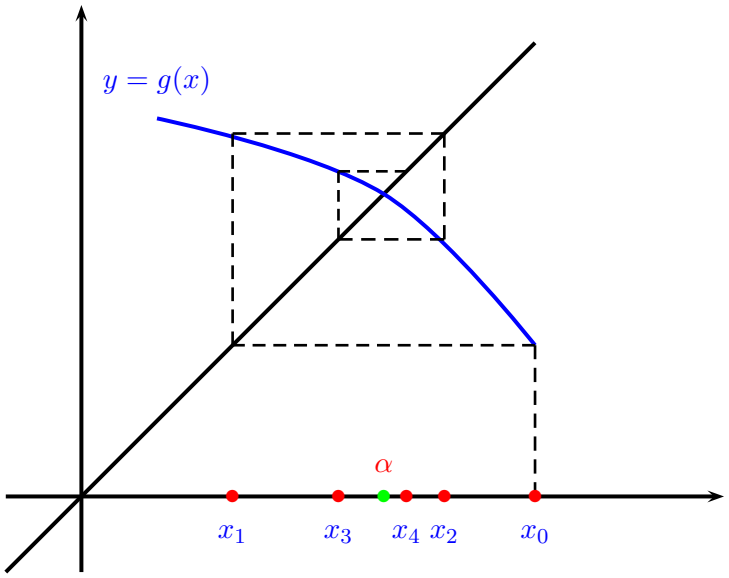
Da $|x_{k+1} - \alpha| \leq \lambda|x_k - \alpha|$ segue

$$|x_{k+1} - \alpha| \leq \lambda^{k+1}|x_0 - \alpha|.$$

Conseguentemente qualunque sia x_0 si ha:

$$\lim_{k\to+\infty} |x_k - \alpha| = 0 \Leftrightarrow \lim_{k\to+\infty} x_k = \alpha. \ \square$$

Nella seguente figura viene rappresentata l'interpretazione geometrica di un metodo di iterazione funzionale in ipotesi di convergenza.



Definizione 2.2.1 *Un metodo iterativo del tipo (2.5) si dice localmente convergente ad una soluzione α del problema $f(x) = 0$ se esiste un intervallo $[a, b]$ contenente α tale che, per ogni $x_0 \in [a, b]$, la successione generata da (2.5) converge a α.*

Una volta determinata una condizione sufficiente per la convergenza della successione $\{x_k\}$ ad un punto fisso di $g(x)$ si deve essere sicuri che tale punto fisso è unico. Infatti se, altre ad α esistesse anche $\beta \in [a, b]$ tale che $\beta = g(\beta)$, con $\alpha \neq \beta$, allora

$$|\alpha - \beta| = |g(\alpha) - g(\beta)| = |g'(\xi)||\alpha - \beta|$$

con $\xi \in [a, b]$. Poichè $|g'(\xi)| < 1$ si ha:

$$|\alpha - \beta| < |\alpha - \beta|$$

e ciò è assurdo.
Come abbiamo già visto nel caso del metodo delle bisezioni anche per metodi di iterazione funzionale è necessario definire dei criteri di arresto per il calcolo delle iterazioni. Teoricamente, una volta stabilita la precisione voluta, ε, si dovrebbe arrestare il processo iterativo quando l'errore al passo k

$$e_k = |\alpha - x_k|$$

risulta minore della tolleranza prefissata ε. In pratica l'errore non può essere noto quindi è necessario utilizzare qualche stima. Per esempio si potrebbe considerare la differenza tra due iterate consecutive e fermare il calcolo degli elementi della successione quando

$$|x_{k+1} - x_k| \leq \varepsilon,$$

oppure

$$\frac{|x_{k+1} - x_k|}{\min(|x_{k+1}|, |x_k|)} \leq \varepsilon \qquad |x_{k+1}|, |x_k| \neq 0$$

se i valori hanno un ordine di grandezza particolarmente elevato. Una stima alternativa valuta il residuo della funzione rispetto al valore in α, cioè

$$|f(x_k)| \leq \varepsilon.$$

2.2.1 Ordine di Convergenza

Per confrontare differenti metodi iterativi che approssimano la stessa radice α di $f(x) = 0$, si può considerare la velocità con cui tali successioni convergono verso α. Lo studio della velocità di convergenza passa attraverso il concetto di ordine del metodo.

Definizione 2.2.2 *Sia $\{x_k\}_{k=0}^{\infty}$ una successione convergente ad α e tale che $x_k \neq \alpha$, per ogni k. Se esiste un numero reale $p \geq 1$ tale che*

$$\lim_{k \to +\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^p} = \gamma \qquad con \quad \begin{cases} 0 < \gamma \leq 1 & se\ p = 1 \\ \gamma > 0 & se\ p > 1 \end{cases} \tag{2.8}$$

allora si dice che la successione ha ordine di convergenza p. La costante γ prende il nome di costante asintotica di convergenza.

In particolare se $p = 1$ e $0 < \gamma < 1$ allora la convergenza si dice *lineare*, se $p = 1$ e $\gamma = 1$ allora la convergenza si dice *sublineare*, mentre se $p > 1$ allora la convergenza si dice superlineare.
Osservazione. La relazione (2.8) implica che esiste una costante positiva β ($\beta \simeq \gamma$) tale che, per k sufficientemente grande:

$$|x_{k+1} - \alpha| \leq \beta |x_k - \alpha|^p \tag{2.9}$$

ed anche

$$\frac{|x_{k+1}-\alpha|}{|\alpha|} \leq \beta|\alpha|^{p-1}\left|\frac{x_k-\alpha}{\alpha}\right|^p. \tag{2.10}$$

Le (2.9) e (2.10) indicano che la riduzione di errore (assoluto o relativo) ad ogni passo è tanto maggiore quanto più alto è l'ordine di convergenza e, a parità di ordine, quanto più piccola è la costante asintotica di convergenza.

Teorema 2.2.3 *Sia $\{x_k\}_{k=0}^{\infty}$ una successione generata dallo schema (2.5) convergente ad α punto fisso di $g \in \mathcal{C}^1([\alpha-\rho,\alpha+\rho])$, $\rho > 0$. Se la convergenza della successione $\{x_k\}_{k=0}^{\infty}$ è lineare (risp. sublineare) allora:*

$$0 < |g'(\alpha)| < 1 \qquad (\textit{risp. } |g'(\alpha)| = 1)$$

Dimostrazione.

$$x_{k+1} - \alpha = g(x_k) - g(\alpha)$$

da cui:

$$\frac{|x_{k+1}-\alpha|}{|x_k-\alpha|} = g'(\xi_k) \qquad \text{con } |\xi_k-\alpha| < |x_k-\alpha|.$$

Poichè

$$\lim_{k\to+\infty} |g'(\xi_k)| = |g'(\alpha)| = \gamma$$

la tesi segue direttamente dalla definizione di convergenza lineare (risp. sublineare). □

Teorema 2.2.4 *(Enunciato) Sia $\alpha \in [a,b]$ punto fisso di $g \in \mathcal{C}^1([a,b])$.*
1) Se $0 < |g'(\alpha)| < 1$ esiste $\rho > 0$ tale che per ogni x_0, $|x_0-\alpha| < \rho$, la successione $\{x_k\}$ generata da (2.5) è convergente ed ha convergenza lineare.
2) Se $|g'(\alpha)| = 1$ ed esiste $\rho > 0$ tale che $0 < |g'(x)| < 1$, se $|x-\alpha| < \rho$, allora per ogni x_0, $|x_0-\alpha| < \rho$, la successione $\{x_k\}$ generata da (2.5) è convergente ed ha convergenza sublineare. □

Teorema 2.2.5 *Sia α punto fisso di $g \in \mathcal{C}^p([a,b])$, $p \geq 2$, intero. Se per $x_0 \in [a,b]$ la successione $\{x_k\}$ generata dal metodo (2.5) converge a α con ordine p allora:*

$$g'(\alpha) = g''(\alpha) = \cdots = g^{(p-1)}(\alpha) = 0, \qquad g^{(p)}(\alpha) \neq 0.$$

Dimostrazione. Poichè la successione converge con ordine p, risulta:

$$\lim_{k\to+\infty} \frac{|x_{k+1}-\alpha|}{|x_k-\alpha|^r} = 0 \qquad \text{se } r < p. \tag{2.11}$$

Dalla formula di Taylor

$$x_{k+1}-\alpha = g(x_k)-g(\alpha) = g'(\alpha)(x_k-\alpha) + \frac{1}{2}g''(\xi_k)(x_k-\alpha)^2$$

con $|\xi_k-\alpha| < |x_k-\alpha|$, ovvero

$$\frac{x_{k+1}-\alpha}{x_k-\alpha} = g'(\alpha) + \frac{1}{2}g''(\xi_k)(x_k-\alpha).$$

Passando al limite e tenendo conto della (2.11) e del fatto che $g''(x)$ è limitata segue

$$g'(\alpha) = 0.$$

Assumiamo ora che $g^{(i)}(\alpha) = 0$, per $i = 1, \ldots, r-1$, con $r < p$, e proviamolo per $i = r < p$. Dalla formula di Taylor

$$x_{k+1}-\alpha = g(x_k)-g(\alpha) = \sum_{i=1}^{r} \frac{g^{(i)}(\alpha)}{i!}(x_k-\alpha)^i + \frac{g^{(r+1)}(\sigma_k)}{(r+1)!}(x_k-\alpha)^{r+1}.$$

con $|\sigma_k-\alpha| < |x_k-\alpha|$. Per l'ipotesi induttiva:

$$x_{k+1}-\alpha = \frac{g^{(r)}(\alpha)}{r!}(x_k-\alpha)^r + \frac{g^{(r+1)}(\sigma_k)}{(r+1)!}(x_k-\alpha)^{r+1}.$$

In virtù della (2.11) e della limitatezza di $g^{(r+1)}(x)$ si ha

$$g^{(r)}(\alpha) = 0.$$

Infine, poichè:

$$\frac{x_{k+1}-\alpha}{(x_k-\alpha)^p} = \frac{1}{p!}g^{(p)}(\eta_k) \qquad |\eta_k-\alpha| < |x_k-\alpha|$$

si ha

$$\lim_{k\to+\infty} \frac{|x_{k+1}-\alpha|}{|x_k-\alpha|^p} = \frac{1}{p!}|g^{(p)}(\alpha)| \neq 0. \ \square$$

Vale anche il viceversa.

Teorema 2.2.6 *Sia $\alpha \in [a,b]$ punto fisso di $g \in \mathcal{C}^p([a,b])$, $p \geq 2$, intero. Se*

$$g'(\alpha) = g''(\alpha) = \cdots = g^{(p-1)}(\alpha) = 0, \qquad g^{(p)}(\alpha) \neq 0$$

allora esiste $\rho > 0$ tale che per ogni $x \in [\alpha - \rho, \alpha + \rho]$ la successione $\{x_k\}$ generata da (2.5) è convergente con ordine di convergenza p.

Dimostrazione. Poichè $g'(\alpha) = 0$ esiste $\rho > 0$ tale che $|g'(x)| < 1$ per $|x - \alpha| < \rho$, e la convergenza segue dal teorema (**??**). Inoltre per ogni successione $\{x_k\}$ ottenuta da (2.5) si ha

$$\lim_{k \to +\infty} \frac{|x_{k+1} - \alpha|}{|x_k - \alpha|^r} = \frac{1}{r!}|g^{(r)}(\alpha)| = 0 \qquad \text{per } r < p.$$

e

$$\gamma = \frac{1}{p!}|g^{(p)}(\alpha)| > 0,$$

e quindi la successione ha ordine di convergenza p. □

Osservazione. L'ordine di convergenza p può essere anche un numero non intero. In questo caso, posto $q = [p]$, se $g \in \mathcal{C}^q([a,b])$ si ha anche

$$g'(\alpha) = g''(\alpha) = \cdots = g^{(q)}(\alpha) = 0,$$

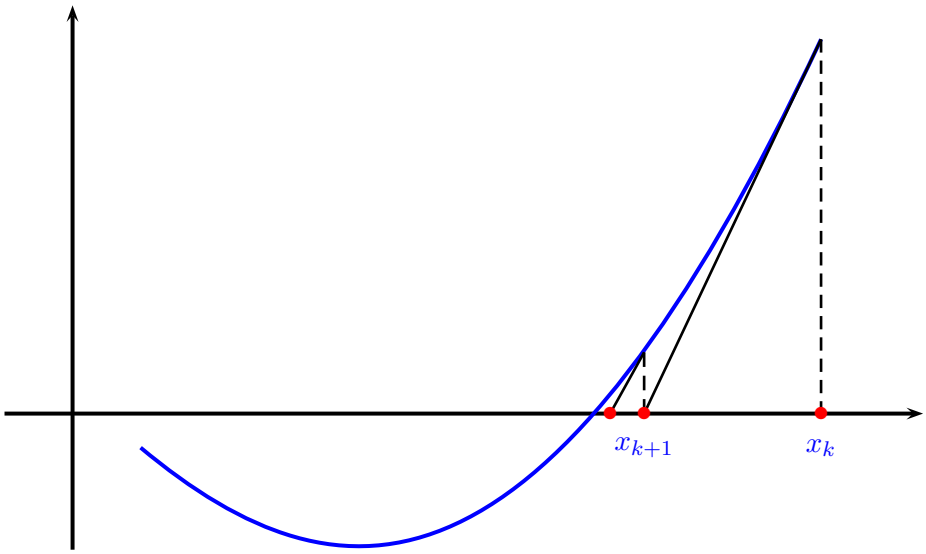
e che g non ha derivata di ordine $q+1$ altrimenti per il precedente teorema tutte le successioni ottenute da (2.5) a partire da $x_0 \in [\alpha - \rho, \alpha + \rho]$ avrebbero ordine almeno $q + 1$.

Definizione 2.2.3 *Un metodo iterativo convergente ad α si dice di ordine p (di ordine almeno p) se tutte le successioni ottenute al variare del punto iniziale in un opportuno intorno di α convergono con ordine di convergenza p (almeno p).*

2.2.2 Metodo di Newton-Raphson

Nell'ipotesi che f sia derivabile ed ammetta derivata prima continua allora un altro procedimento per l'approssimazione dello zero della funzione $f(x)$ è il metodo di Newton-Raphson, noto anche come metodo delle tangenti. Nella figura seguente è riportata l'interpretazione geometrica di tale metodo. A partire dall'approssimazione x_0 si considera la retta tangente la funzione

f passante per il punto P_0 di coordinate $(x_0, f(x_0))$. Si calcola l'ascissa x_1 del punto di intersezione tra tale retta tangente e l'asse delle x e si ripete il procedimento a partire dal punto P_1 di coordinate $(x_1, f(x_1))$. Nella seguente figura è rappresentato graficamente il metodo di Newton-Raphson.



È facile vedere che il metodo definisce il seguente processo iterativo:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \qquad k = 0, 1, 2, \ldots \tag{2.12}$$

che equivale, scegliendo in (2.7) $h(x) = f'(x)$, al metodo di iterazione funzionale in cui la funzione $g(x)$ è

$$g(x) = x - \frac{f(x)}{f'(x)}. \tag{2.13}$$

Per esaminare la convergenza del metodo consideriamo che per ipotesi $f'(x) \neq 0$, per $x \in [a, b]$, dove $[a, b]$ è un opportuno intervallo contenente α. Calcoliamo quindi la derivata prima di $g(x)$:

$$g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}. \tag{2.14}$$

Poichè α è semplice risulta $f'(\alpha) \neq 0$ e quindi:

$$g'(\alpha) = \frac{f(\alpha)f''(\alpha)}{[f'(\alpha)]^2} = 0$$

esiste quindi un intorno di α nel quale $|g'(x)| < 1$, per ogni x, e per il teorema (2.2.2) comunque si sceglie un punto iniziale appartenente a tale intorno il metodo di Newton-Raphson risulta convergente. Se la radice α ha molteplicità $r > 1$ l'ordine di convergenza del metodo non è più 2. Se x_0 è sufficientemente vicino ad α è $|g'(x)| < 1$ e quindi per il teorema 2.2.2 il metodo è ancora convergente ma l'ordine di convergenza è 1.

```
function [alfa,k]=newton(f,f1,x0,tol,Nmax)
%
% La funzione approssima la radice con il metodo di Newton-Raphson
%
% Parametri di input
% f = funzione della quale calcolare la radice
% f1 = derivata prima della funzione f
% x0 = approssimazione iniziale della radice
% tol = precisione fissata
% Nmax = numero massimo di iterazioni fissate
%
% Parametri di output
% alfa = approssimazione della radice
% k = numero di iterazioni
%
if nargin==3
    tol=1e-8;
    Nmax=1000;
end
k=0;
x1=x0-feval(f,x0)/feval(f1,x0);
fx1 = feval(f,x1);
while abs(x1-x0)>tol | abs(fx1)>tol
    x0 = x1;
    x1 = x0-feval(f,x0)/feval(f1,x0);
    fx1 = feval(f,x1);
    k=k+1;
    if k>Nmax
        disp('Il metodo non converge');
        alfa = inf;
        break
```

```
    end
end
alfa=x1;
return
```

Il metodo della direzione costante

Se applicando ripetutamente la formula di Newton-Raphson accade che la derivata prima della funzione $f(x)$ si mantiene sensibilmente costante allora si può porre

$$M = f'(x)$$

e applicare la formula

$$x_{k+1} = x_k - \frac{f(x_k)}{M} \tag{2.15}$$

anzichè la (2.12). La (2.15) definisce un metodo che viene detto metodo di Newton semplificato oppure metodo della direzione costante in quanto geometricamente equivale all'applicazione del metodo di Newton in cui anzichè prendere la retta tangente la curva f si considera la retta avente coefficiente angolare uguale a M. La funzione iteratrice del metodo è

$$g(x) = x - \frac{f(x)}{M}$$

ed il metodo è convergente se

$$|g'(x)| = \left|1 - \frac{f'(x)}{M}\right| < 1$$

da cui si deduce che è necessario che $f'(x)$ ed M abbiano lo stesso segno.

2.2.3 Il Metodo della Secante

Il metodo della secante è definito dalla relazione

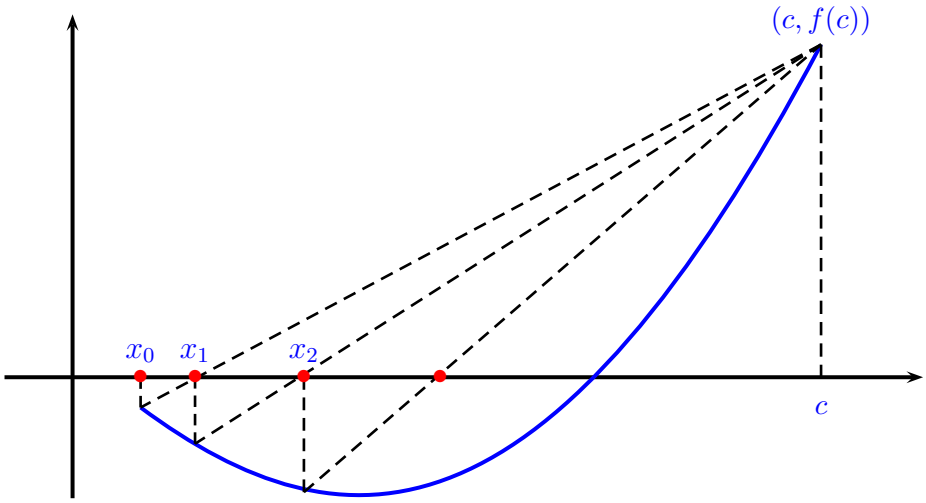
$$x_{k+1} = x_k - f(x_k)\frac{x_k - c}{f(x_k) - f(c)}$$

dove $c \in [a, b]$. Il significato geometrico di tale metodo è il seguente: ad un generico passo k si considera la retta congiungente i punti di coordinate

$(x_k, f(x_k))$ e $(c, f(c))$ e si pone x_{k+1} pari al punto di intersezione di tale retta con l'asse x. Dalla formula si evince che la funzione iteratrice del metodo è

$$g(x) = x - f(x)\frac{x-c}{f(x)-f(c)}.$$

Il metodo è rappresentato graficamente nella seguente figura.



In base alla teoria vista nei paragrafi precedenti il metodo ha ordine di convergenze 1 se $g'(\alpha) \neq 0$. Può avere ordine di convergenza almeno 1 se $g'(\alpha) = 0$. Tale eventualità si verifica se la tangente alla curva in α ha lo stesso coefficiente angolare della retta congiungente i punti $(\alpha, 0)$ e $(c, f(c))$.
Poichè il metodo delle secanti ha lo svantaggio di avere, solitamente, convergenza lineare mentre il metodo di Newton-Raphson, pur avendo convergenza quadratica, ha lo svantaggio di richiedere, ad ogni passo, due valutazioni di funzioni: $f(x_k)$ ed $f'(x_k)$, quindi se il costo computazionale di $f'(x_k)$ è molto più elevato rispetto a quello di $f(x_k)$ può essere più conveniente l'uso di metodi che necessitano solo del calcolo del valore della funzione $f(x)$.
Un metodo basato sullo stesso principio, ma appartenente ad una diversa classe di algoritmi è **il metodo delle secanti a due punti**, che consiste nel considerare, alla k-esima iterata, la retta passante per i punti di coordinate $(x_{k-1}, f(x_{k-1}))$ e $(x_k, f(x_k))$ e definire x_{k+1} come l'ascissa del punto d'intersezione di tale retta con l'asse x. Ne risulta il seguente metodo:

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}. \tag{2.16}$$

Poichè il metodo delle secanti a due punti non rientra tra quelli di iterazione funzionale, per ricavare l'ordine del metodo non si possono applicare i teoremi enunciati in precedenza, pertanto si deve procedere in un modo più che sperimentale. Infatti in questo caso si può vedere che l'ordine non è neanche un numero intero ma irrazionale:

$$p = \frac{1+\sqrt{5}}{2}.$$

Il metodo delle secanti a due punti ha comunque una convergenza di tipo superlineare, come il metodo di Newton-Raphson, ma ha il vantaggio di richiedere solo una valutazione funzionale ad ogni passo (tranne al primo passo) e di non richiedere il calcolo della derivata prima.

2.2.4 Il Metodo di Newton a Doppio Passo

Analizziamo in questa sezione una particolare applicazione del metodo di Newton per il calcolo degli zeri di un polinomio. In particolare supponiamo che il polinomio a coefficienti reali

$$p(x) = \sum_{i=0}^{n} a_i x^{n-i} \tag{2.17}$$

ammetta solo le radici reali e distinte

$$\alpha_1 > \alpha_2 > \alpha_3 > \cdots > \alpha_n.$$

In questo caso il metodo di Newton fornisce il seguente schema iterativo:

$$x_{k+1} = x_k - \frac{p(x_k)}{p'(x_k)} \qquad k = 0, 1, 2, \ldots \tag{2.18}$$

In questo caso vale il seguente teorema.

Teorema 2.2.7 *Sia $p(x)$ un polinomio reale di grado $n \geq 1$, che possiede solo radici reali α_i,*

$$\alpha_1 > \alpha_2 > \alpha_3 > \cdots > \alpha_n$$

allora il metodo di Newton, per ogni valore iniziale $x_0 > \alpha_1$ fornisce una successione monotona decrescente $\{x_k\}$ di approssimazioni, convergente verso α_1. Per $n \geq 2$ la successione delle x_k è strettamente monotona decrescente.
□

Tuttavia nonostante la convergenza quadratica del metodo di Newton, se x_k non si trova abbastanza vicino ad uno zero semplice di $p(x)$, la successione delle x_k può avvicinarsi molto lentamente alla radice (la derivata prima può essere infatti molto grande) pertanto si preferisce applicare una variante del metodo di Newton, noto appunto come Metodo di Newton a Doppio Passo:

$$x_{k+1} = x_k - 2\frac{p(x_k)}{p'(x_k)} \qquad k = 0, 1, 2, \ldots \tag{2.19}$$

Questo metodo, che non converge alla radice α_1, viene utilizzato per far sì che la successione $\{x_k\}$ si avvicini più rapidamente ad α_1. Tuttavia per un certo indice m accade che:

$$\begin{array}{ll} p(x_j)p(x_{j-1}) & > 0 \qquad j = 1, 2, \ldots, m-1 \\ p(x_m)p(x_{m-1}) & < 0 \end{array}$$

e questo vuol dire:

$$x_m < \alpha_1 < x_{m-1} < x_{m-2} < \cdots < x_1 < x_0$$

cioè la successione delle x_k scavalca la radice α_1. A questo punto si può applicare il metodo di Newton semplice prendendo come punto iniziale proprio x_m. Infatti si può dimostrare che il punto x_m pur scavalcando la radice α_1 non scavalca il punto ξ_1, compreso tra α_2 e α_1, che annulla la derivata prima del polinomio $p(x)$. La successione $\{x_k\}$ così costruita converge alla radice α_1. Un altro problema da risolvere a questo punto è la scelta del valore iniziale x_0. Per quello possiamo sfruttare una delle maggiorazioni che vengono fornite dal seguente teorema.

Teorema 2.2.8 *Per ogni zero α_i del polinomio (2.17) risulta:*

$$
\begin{aligned}
|\alpha_i| &\leq \max\left\{\left|\frac{a_n}{a_0}\right|, 1+\left|\frac{a_{n-1}}{a_0}\right|, \ldots, 1+\left|\frac{a_1}{a_0}\right|\right\} \\
|\alpha_i| &\leq \max\left\{1, \sum_{i=1}^{n}\left|\frac{a_i}{a_0}\right|\right\} \\
|\alpha_i| &\leq \max\left\{\left|\frac{a_n}{a_{n-1}}\right|, 2\left|\frac{a_{n-1}}{a_{n-2}}\right|, \ldots, 2\left|\frac{a_1}{a_0}\right|\right\} \\
|\alpha_i| &\leq \sum_{i=0}^{n-1}\left|\frac{a_{i+1}}{a_i}\right| \\
|\alpha_i| &\leq 2\max\left\{\left|\frac{a_1}{a_0}\right|, \sqrt{\left|\frac{a_2}{a_0}\right|}, \sqrt[3]{\left|\frac{a_3}{a_0}\right|}, \ldots, \sqrt[n]{\left|\frac{a_n}{a_0}\right|}\right\}.
\end{aligned}
$$

Dopo aver determinato una buona approssimazione $\hat{\alpha}_1$ della radice più grande di $p(x)$ si pone il problema di determinare gli altri zeri. Un metodo può essere quello di calcolare, utilizzando per esempio la regola di Ruffini, il polinomio di grado $n-1$:

$$p_1(x) = \frac{p(x)}{x-\hat{\alpha}_1} \tag{2.20}$$

il cui massimo zero è ora α_2 e applicare il metodo di Newton. Tuttavia l'applicazione della (2.20) per il calcolo di $p_1(x)$ fa sì che l'errore con cui la radice α_1 è approssimata da $\hat{\alpha}_1$ si propaghi al calcolo dei coefficienti di $p_1(x)$ con l'effetto di ottenere un polinomio che sicuramente non ammette come zeri α_2, $\alpha_3, \ldots, \alpha_n$. Per evitare questo problema conviene non calcolare direttamente il polinomio $p_1(x)$ ma ricorrere alla cosiddetta Variante di Maehly. Infatti per il polinomio $p_1(x)$ abbiamo:

$$p_1'(x) = \frac{p'(x)}{x-\hat{\alpha}_1} - \frac{p(x)}{(x-\hat{\alpha}_1)^2}$$

e quindi il metodo di Newton a doppio passo è definito dalla formula:

$$x_{k+1} = x_k - 2\frac{p_1(x_k)}{p_1'(x_k)} = x_k - 2\frac{p(x_k)}{p'(x_k) - \dfrac{p(x_k)}{x_k-\hat{\alpha}_1}}. \tag{2.21}$$

Quindi per ottenere un'approssimazione di α_2 si considera nuovamente il metodo di Newton a doppio passo (2.21) prendendo come punto iniziale proprio il valore x_m trovato dopo lo scavalcamento di α_1. In generale per la derivata prima del polinomio

$$p_j(x) = \frac{p(x)}{(x-\hat{\alpha}_1)\dots(x-\hat{\alpha}_j)}$$

vale la formula

$$p_j'(x) = \frac{p'(x)}{(x-\hat{\alpha}_1)\dots(x-\hat{\alpha}_j)} - \frac{p(x)}{(x-\hat{\alpha}_1)\dots(x-\hat{\alpha}_j)} \sum_{i=1}^{j} \frac{1}{x-\hat{\alpha}_i}.$$

Il metodo di Newton per la determinazione di α_{j+1} nella variante di Maehly assume la forma:

$$x_{k+1} = x_k - \frac{p(x_k)}{p'(x_k) - \sum_{i=1}^{j} \frac{p(x_k)}{x_k - \hat{\alpha}_i}}.$$

Il vantaggio di questa formula sta nel fatto che la successione delle x_k converge localmente in maniera quadratica verso la radice α_{j+1} indipendentemente dall'errore del quale i numeri $\hat{\alpha}_1, \dots, \hat{\alpha}_j$ sono affetti in quanto approssimazioni delle radici del polinomio $p(x)$.

```
function [alfa, iter]=doppio(p,p1,epsilon)
%
% La funzione doppio(p,p1,epsilon) calcola gli zeri di un
% polinomio a coefficienti e radici reali utilizzando il
% metodo di Newton a doppio passo e la variante di Maehly.
% Parametri di input
% p=vettore dei coefficienti del polinomio
% p1=vettore dei coefficienti della derivata prima del polinomio
% epsilon=tolleranza prefissata
%
n=length(p)-1;
j=0;
alfa=[];
x0=inizio(p);
```

```
%
% La funzione inizio calcola l'approssimazione iniziale
%
while j<n
   x1=x0-2*newton(p,p1,x0,j,alfa);
   while polyval(p,x1)*polyval(p,x0)>0
        x0=x1;
        x1=x0-2*newton(p,p1,x0,j,alfa)
   end
   x2=x1;
%
% Scavalcamento avvenuto
% Si memorizza in x2 il valore assunto della successione
% dopo lo scavalcamento e che sara' il punto iniziale per
% il calcolo della successiva radice
%
   while abs(polyval(p,x1))>epsilon | abs(x1-x0)>epsilon
       x0=x1;
       x1=x0-newton(p,p1,x0,j,alfa)
   end
   alfa=[alfa x1];
   j=j+1;
   x0=x2;
end
return
```

La funzione `newton` calcola i rapporti $p(x_k)/p'(x_k)$ al primo passo e ai passi successivi quelli della variante di Maehly.

```
function y=newton(p,p1,x,j,alfa)
px=polyval(p,x);
p1x=polyval(p1,x);
somma=0;
for i=1:j
  somma=somma+1/(x-alfa(i));
end
y=px/(p1x-px*somma);
return
```

Per calcolare l'approssimazione iniziale si sfrutta il Teorema 2.2.8 calcolando la migliore delle 5 approssimazioni proposte.

```
function y=inizio(p)
%
% inizio(p) calcola l'approssimazione iniziale per il calcolo
% della radice piu' grande
%
n=length(p);
pp(1)=abs(p(n)/p(1));
for i=2:n
  pp(i)=1+abs(p(i)/p(1));
end
mx(1)=max(pp);
clear pp
pp(1)=1;
pp(2)=sum(abs(p(2:n)/p(1)));
mx(2)=max(pp);
clear pp
pp(1)=abs(p(n)/p(n-1));
for i=2:n
  pp(i)=2*abs(p(i)/p(i-1));
end
mx(3)=max(pp);
clear pp
mx(4)=0;
for i=1:n-1
  mx(4)=mx(4)+abs(p(i+1)/p(i));
end
pp(1)=abs(p(2)/p(1));
for i=2:n-1
  pp(i)=abs(p(i+1)/p(1))^(1/i);
end
mx(5)=2*max(pp);
y=min(mx);
return
```

Capitolo 3

Metodi diretti per sistemi lineari

3.1 Elementi di Algebra Lineare

Assegnati due numeri interi positivi $m, n \in \mathbb{N}$, $m, n \geq 1$, si definisce matrice una tabella avente m righe ed n colonne. Se gli elementi della matrice sono numeri reali allora l'insieme delle matrici aventi m righe ed n colonne si indica con $\mathbb{R}^{m \times n}$. Per convenzione le matrici si indicano con lettere maiuscole dell'alfabeto latino. Se $A \in \mathbb{R}^{m \times n}$ allora m ed n si dicono dimensioni della matrice. Se consideriamo due numeri interi $i, j \in \mathbb{N}$, tali che $1 \leq i \leq m$ e $1 \leq j \leq n$, questi identificano, rispettivamente, una riga ed una colonna della matrice A, e l'elemento che si trova nella posizione (i, j) viene indicato con a_{ij}. Una matrice di dimensione $m \times n$ può essere rappresentata nel seguente modo:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}.$$

Se $m = n$ allora la matrice A si dice quadrata di dimensione n o di ordine n altrimenti si dice rettangolare. Gli elementi a_{ij} di una matrice quadrata A di ordine n tali che $i = j$ sono detti elementi principali o diagonali e formano la cosiddetta diagonale principale di A.
Assegnata una matrice $A \in \mathbb{R}^{m \times n}$ si definisce matrice trasposta di A la

matrice $B = A^T \in \mathbb{R}^{n\times m}$ tale che

$$b_{ij} = a_{ji}, \quad i = 1, \dots, n, \ j = 1, \dots, m.$$

Se accade che $A = A^T$ allora la matrice è detta simmetrica. Una matrice simmetrica deve essere ovviamente quadrata. Gli elementi di una matrice che si trovano al di sopra della diagonale principale sono detti sopradiagonali, mentre quelli che si trovano al di sotto della stessa diagonale principale sono detti sottodiagonali. Se una matrice ha tutti gli elementi sopradiagonali e sottodiagonali uguali a zero viene detta matrice diagonale. Se invece ha solo gli elementi sopradiagonali nulli allora viene detta triangolare inferiore. Se ha gli elementi sottodiagonali nulli allora è detta triangolare superiore.
Assegnate due matrici $A, B \in \mathbb{R}^{m\times n}$ si definisce somma di A e B, e si denota con $C = A + B$, la matrice $C \in \mathbb{R}^{m\times n}$ i cui elementi sono:

$$c_{ij} = a_{ij} + b_{ij} \qquad i = 1, \dots, m, \ j = 1, \dots, n.$$

In modo analogo si definisce la differenza tra matrici, infatti $D = A - B$ è la matrice avente elementi:

$$d_{ij} = a_{ij} - b_{ij} \qquad i = 1, \dots, m, \ j = 1, \dots, n.$$

Se $\alpha \in \mathbb{R}$ ed $A \in \mathbb{R}^{m\times n}$ allora la matrice $C = \alpha A$ è definita da:

$$c_{ij} = \alpha a_{ij}.$$

Se $A \in \mathbb{R}^{m\times p}$ e $B \in \mathbb{R}^{p\times n}$ si definisce *prodotto* di A per B la matrice $C \in \mathbb{R}^{m\times n}$ i cui elementi sono

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} \qquad i = 1, \dots, m, \ j = 1, \dots, n.$$

Si noti che affinchè tale prodotto abbia senso è necessario che il numero delle colonne di A coincida con il numero delle righe di B. Quando ciò accade le matrici si dicono conformabili, altrimenti si dicono non conformabili. Ad esempio nel nostro caso se $m \neq n$ allora il prodotto BA non ha senso. Ha sempre significato considerare i prodotti AB e BA se A e B sono matrici quadrate dello stesso ordine ($m = n$).
È facile verificare che il prodotto tra matrici gode della proprietà *associativa* ma in generale non di quella commutativa. Vale invece la seguente proprietà:

$$(AB)^T = B^T A^T.$$

Il prodotto tra matrici non gode della proprietà commutativa, ovvero se A, B sono due matrici, ammesso che sia possibile calcolare le matrici prodotto AB e BA, queste risultano essere diverse. Si definisce matrice identità di ordine n la matrice quadrata diagonale I_n avente tutti gli elementi principali uguali a 1:

$$I_n = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & 0 & 1 \end{bmatrix}.$$

La matrice identità è l'elemento neutro per il prodotto, cioè se $A \in \mathbb{R}^{n\times n}$ si ha

$$AI_n = I_n A = A.$$

Siano $A, B \in \mathbb{R}^{n\times n}$ le seguenti matrici

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

dove $A_{11}, B_{11} \in \mathbb{R}^{p\times p}$, $A_{12}, B_{12} \in \mathbb{R}^{p\times(n-p)}$, $A_{21}, B_{21} \in \mathbb{R}^{(n-p)\times p}$ e infine $A_{22}, B_{22} \in \mathbb{R}^{(n-p)\times(n-p)}$, con $p < n$, rappresentano a loro volta matrici e non semplici elementi. Si dice cioè che A e B sono state suddivise a blocchi. Il prodotto AB può essere calcolato utilizzando tale decomposizione delle matrici:

$$AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}.$$

Tale modo di effettuare il prodotto tra matrici viene detto prodotto a blocchi ed è molto utile quando le matrici hanno una particolare struttura (per esempio se uno dei blocchi è identicamente nullo oppure è uguale alla matrice identità). Per esempio se A e B sono matrici triangolari inferiori il blocco in posizione $(1, 2)$ è nullo perchè composto da elementi sopradiagonali, quindi le matrici possono essere divise nel seguente modo:

$$A = \begin{bmatrix} A_{11} & O \\ A_{21} & A_{22} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{11} & O \\ B_{21} & B_{22} \end{bmatrix}.$$

Il loro prodotto a blocchi ha come risultato:

$$AB = \begin{bmatrix} A_{11}B_{11} & O \\ A_{21}B_{11} + A_{22}B_{21} & A_{22}B_{22} \end{bmatrix},$$

evidenziando che il blocco $(1,2)$ non va calcolato perchè identicamente nullo.

Definizione 3.1.1 *Una matrice che si ottiene da I_n scambiando alcune righe (o colonne) viene detta matrice di permutazione.*

Esempio 3.1.1 *Sia P la seguente matrice di permutazione:*

$$P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

che è stata ottenuta da I_3 scambiando la prima riga con la terza. Consideriamo la seguente matrice A

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

e calcoliamo il prodotto PA:

$$PA = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 7 & 8 & 9 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}.$$

La moltiplicazione a sinistra di una matrice di permutazione per A ha l'effetto di scambiare le righe di A esattamente nello stesso modo con cui erano state scambiate le righe dell'identità per ottenere P. Calcoliamo ora il prodotto AP:

$$AP = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \\ 9 & 8 & 7 \end{bmatrix}.$$

Invece la moltiplicazione a destra di una matrice di permutazione per A ha l'effetto di scambiare le colonne di A.

Data una matrice $A \in \mathbb{R}^{m\times n}$, una matrice $B \in \mathbb{R}^{h\times k}$, $0 < h \leq m$, $0 < k \leq n$, è detta sottomatrice di A se è ottenuta da A eliminando $m-h$ righe ed $n-k$ colonne. Data una matrice $A \in \mathbb{R}^{m\times n}$, una sottomatrice quadrata B di ordine $k \leq n$ di A è detta principale se gli elementi principali di B sono anche gli elementi principali di A. Una sottomatrice B principale di ordine k di A è detta principale di testa se è formata dagli elementi a_{ij}, $i,j = 1,\ldots,k$. Si definisce inoltre minore principale di testa di ordine k il determinante della sottomatrice principale di testa di ordine k.

Definizione 3.1.2 *Se $A \in \mathbb{R}^{n\times n}$ è una matrice di ordine 1, si definisce determinante di A il numero*

$$\det A = a_{11}.$$

Se la matrice A è quadrata di ordine n allora fissata una qualsiasi riga (colonna) di A, diciamo la i-esima (j-esima) allora applicando la cosiddetta regola di Laplace il determinante di A è:

$$\det A = \sum_{j=1}^{n} a_{ij}(-1)^{i+j} \det A_{ij}$$

dove A_{ij} è la matrice che si ottiene da A cancellando la i-esima riga e la j-esima colonna.

Il determinante è pure uguale a

$$\det A = \sum_{i=1}^{n} a_{ij}(-1)^{i+j} \det A_{ij},$$

cioè il determinante è indipendente dall'indice di riga (o di colonna) fissato. Se A è la matrice di ordine 2

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}.$$

allora

$$\det A = a_{11}a_{22} - a_{21}a_{12}.$$

Il determinante ha le seguenti proprietà:
1. Se A è una matrice triangolare o diagonale allora

$$\det A = \prod_{i=1}^{n} a_{ii};$$

2. $\det I = 1$;
3. $\det A^T = \det A$;
4. $\det AB = \det A \det B$ (Regola di Binet);
5. se $\alpha \in \mathbb{R}$ allora $\det \alpha A = \alpha^n \det A$;
6. $\det A = 0$ se una riga (o una colonna) è nulla, oppure una riga (o una

colonna) è proporzionale ad un'altra riga (o colonna) oppure è combinazione lineare di due (o più) righe (o colonne) di A.
7. Se A è una matrice triangolare a blocchi

$$A = \begin{bmatrix} B & C \\ O & D \end{bmatrix}$$

con B e D matrici quadrate, allora

$$\det A = \det B \det D. \tag{3.1}$$

Una matrice A di ordine n si dice non singolare se il suo determinante è diverso da zero, in caso contrario viene detta *singolare*. Si definisce inversa di A la matrice A^{-1} tale che:

$$AA^{-1} = A^{-1}A = I_n$$

Per quello che riguarda il determinante della matrice inversa vale la seguente proprietà:

$$\det A^{-1} = \frac{1}{\det A}.$$

Ricordiamo che se A, B e C sono matrici invertibili, in base alla regola di Binet, anche il loro prodotto è una matrice invertibile e risulta

$$(ABC)^{-1} = C^{-1}B^{-1}A^{-1}.$$

Se $A \in \mathbb{R}^{m\times 1}$ (o $A \in \mathbb{R}^{1\times m}$), la matrice si riduce ad una sola colonna (o una sola riga) e viene detta vettore colonna (o riga) ad m elementi o componenti. Solitamente il termine vettore viene associato a vettori colonna e l'insieme dei vettori ad m componenti viene indicato con $\mathbb{R}^m$. Per le operazioni tra vettori valgono le stesse regole viste per le matrici, cioè la somma e la differenza sono possibili tra vettori dello stesso tipo e con lo stesso numero di componenti. Se $\mathbf{x}$ è un vettore colonna di m elementi allora $\mathbf{x}^T$ è un vettore riga sempre di m elementi. Se $A \in \mathbb{R}^{m\times n}$ e $\mathbf{x} \in \mathbb{R}^n$ è possibile definire il prodotto matrice per vettore nel seguente modo:

$$\mathbf{y} = A\mathbf{x}, \qquad y_i = \sum_{j=1}^{n} a_{ij}x_j, \quad i = 1, \ldots, m$$

quindi $\mathbf{y} \in \mathbb{R}^m$. Non è possibile effettuare il prodotto $A\mathbf{x}^T$ perchè le dimensioni non sono compatibili.

Esempio 3.1.2 *Sia*

$$A = \begin{bmatrix} 5 & 1 & 0 \\ -1 & 1 & 2 \\ 5 & -5 & 1 \end{bmatrix}$$

e sia $\mathbf{x}$ *il vettore*

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Calcoliamo il vettore prodotto $\mathbf{y} = A\mathbf{x}$*:*

$$\begin{aligned} y_1 &= 5 \cdot 1 + 1 \cdot 2 + 0 \cdot 3 = 7 \\ y_2 &= -1 \cdot 1 + 1 \cdot 2 + 2 \cdot 3 = 7 \\ y_3 &= 5 \cdot 1 - 5 \cdot 2 + 1 \cdot 3 = -2. \end{aligned}$$

Tra vettori sono consentite due tipi di prodotto:
1. prodotto interno;
2. prodotto esterno.

Il prodotto interno (o scalare), che viene spesso indicato come $(\cdot, \cdot)$, è definito nel seguente modo: siano $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, allora

$$(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y} = \sum_{i=1}^{n} x_i y_i = \alpha$$

e il risultato è un numero reale. Il prodotto scalare soddisfa le seguenti proprietà:

1. $\mathbf{x}^T\mathbf{x} \geq 0$ per ogni $\mathbf{x} \in \mathbb{R}^n$ e $(\mathbf{x}, \mathbf{x}) = 0$ se e solo se $\mathbf{x} = 0$;
2. $\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x}$ per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$;
3. $(\alpha\mathbf{x})^T\mathbf{y} = \alpha(\mathbf{x}^T\mathbf{y})$ per ogni $\alpha \in \mathbb{R}$ e per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$;
4. $(\mathbf{x} + \mathbf{y})^T\mathbf{z} = \mathbf{x}^T\mathbf{z} + \mathbf{y}^T\mathbf{z}$ per ogni $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{R}^n$.
5. se $\mathbf{x}^T\mathbf{y} = 0$ allora i due vettori si dicono ortogonali.

Se $\mathbf{x} \in \mathbb{R}^n$ e $\mathbf{y} \in \mathbb{R}^m$ allora il prodotto esterno viene definito nel seguente modo:

$$A = \mathbf{x}\mathbf{y}^T$$

e il risultato è una matrice di dimensione $n \times m$ i cui elementi sono:

$$a_{ij} = x_i y_j, \qquad i = 1, \ldots, n, \ j = 1, \ldots, m.$$

Esempio 3.1.3 *Siano* $\mathbf{x}$ *e* $\mathbf{y}$ *i seguenti vettori:*

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

e

$$\mathbf{y} = \begin{bmatrix} -1 \\ -2 \\ 4 \end{bmatrix}.$$

Calcoliamo prima il prodotto interno:

$$\mathbf{x}^T\mathbf{y} = 1 \cdot (-1) + 2 \cdot (-2) + 3 \cdot 4 = 7.$$

Osserviamo che tale operazione gode della proprietà commutativa, poichè $\mathbf{y}^T\mathbf{x} = 7$.
Per quello che riguarda il prodotto esterno, il risultato è la matrice

$$A = \mathbf{x}\mathbf{y}^T = \begin{bmatrix} -1 & -2 & 4 \\ -2 & -4 & 8 \\ -3 & -6 & 12 \end{bmatrix}.$$

Tale prodotto non gode della proprietà commutativa, infatti:

$$B = \mathbf{y}\mathbf{x}^T = \begin{bmatrix} -1 & -2 & -3 \\ -2 & -4 & -6 \\ 4 & 8 & 12 \end{bmatrix}.$$

Infatti $B \neq A$, *anche se va osservato che* $B = A^T$.

Norme su Vettori e Matrici

La funzione $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ si dice norma se per ogni vettore $\mathbf{x} \in \mathbb{R}^n$ $\|\mathbf{x}\|$ soddisfa:

1. $\|\mathbf{x}\| \geq 0$ e $\|\mathbf{x}\| = 0$ se e solo se $\mathbf{x} = 0$;
2. $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ per ogni $\alpha \in \mathbb{R}$;
3. $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ per ogni $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ (disuguaglianza triangolare).

Le norme più utilizzate sono le seguenti:

$$\|\mathbf{x}\|_1 = \sum_{j=1}^{n} |x_j| \qquad \text{norma 1}$$

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{n} |x_j|^2} \qquad \text{norma 2 o norma euclidea}$$

$$\|\mathbf{x}\|_\infty = \max_{1\le j\le n} |x_j| \qquad \text{norma infinito.}$$

Definizione 3.1.3 *Una funzione* $\|\cdot\| : \mathbb{R}^{n\times n} \to \mathbb{R}$ *tale che per ogni matrice* $A \in \mathbb{R}^{n\times n}$, $\|A\|$ *soddisfa:*

1. $\|A\| \ge 0$ *e* $\|A\| = 0$ *se e solo se* $A = 0$*;*
2. $\|\alpha A\| = |\alpha|\|A\|$ *per ogni* $\alpha \in \mathbb{R}$*;*
3. $\|A + B\| \le \|A\| + \|B\|$ *per ogni* $A, B \in \mathbb{R}^{n\times n}$*;*
4. $\|A \cdot B\| \le \|A\| \cdot \|B\|$ *per ogni* $A, B \in \mathbb{R}^{n\times n}$*;*

si dice norma di matrice.

Definizione 3.1.4 *Si dice che una norma di matrice è compatibile con una norma di vettore se per ogni matrice* A *e per ogni vettore* $\mathbf{x}$ *risulta*

$$\|A\mathbf{x}\| \le \|A\|\|\mathbf{x}\|.$$

Un modo per definire le norme di matrici compatibili con norme di vettori è il seguente. Sia $\mathbf{x} \neq 0$ con norma $\|\mathbf{x}\|$. Considerata la norma del vettore $A\mathbf{x}$, $\|A\mathbf{x}\|$, definiamo come norma di A il numero $\|A\|$ dato da

$$\|A\| = \sup_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{3.2}$$

$\|A\|$ è detta norma naturale di A oppure norma di A indotta dalla norma di vettore $\|\mathbf{x}\|$. Le norme matriciali indotte dalle norme su vettori $\|\mathbf{x}\|_1$ e $\|\mathbf{x}\|_\infty$

sono

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$$

$$\|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|.$$

3.2 Sistemi Lineari

Siano assegnati una matrice non singolare $A \in \mathbb{R}^{n\times n}$ ed un vettore $\mathbf{b} \in \mathbb{R}^n$. Risolvere un sistema lineare avente A come matrice dei coefficienti e $\mathbf{b}$ come vettore dei termini noti significa trovare un vettore $\mathbf{x} \in \mathbb{R}^n$ tale che

$$A\mathbf{x} = \mathbf{b}. \tag{3.3}$$

Esplicitare la relazione (3.3) significa imporre le uguaglianze tra le componenti dei vettori a primo e secondo membro:

$$\begin{array}{rcccl} a_{11}x_1+ & a_{12}x_2+ & \cdots+ & a_{1n}x_n = & b_1 \\ a_{21}x_1+ & a_{22}x_2+ & \cdots+ & a_{2n}x_n = & b_2 \\ \vdots & & & & \vdots \\ a_{n1}x_1+ & a_{n2}x_2+ & \cdots+ & a_{nn}x_n = & b_n. \end{array} \tag{3.4}$$

Le (3.4) definiscono un sistema di n equazioni algebriche lineari nelle n incognite $x_1, x_2, \ldots, x_n$. Il vettore $\mathbf{x}$ viene detto vettore soluzione. Un metodo universalmente noto per risolvere il problema (3.3) è l'applicazione della cosiddetta Regola di Cramer la quale fornisce:

$$x_i = \frac{\det A_i}{\det A} \qquad i = 1, \ldots, n, \tag{3.5}$$

dove A_i è la matrice ottenuta da A sostituendo la sua i-esima colonna con il termine noto $\mathbf{b}$. Dalla (3.5) è evidente che per ottenere tutte le componenti del vettore soluzione è necessario il calcolo di $n+1$ determinanti di ordine n. Si può facilmente dedurre che il numero di operazioni necessarie per il calcolo del determinate di una matrice di ordine n applicando la regola di Laplace è circa $n!$, quindi questa strada non permette di poter determinare velocemente la soluzione del nostro sistema. Basti pensare che se $n = 100$ il il numero di operazioni per il calcolo di un solo determinante sarebbe all'incirca dell'ordine di 10^{157}.

3.2.1 Risoluzione di sistemi triangolari

Prima di affrontare la soluzione algoritmica di un sistema lineare vediamo qualche particolare sistema che può essere agevolmente risolto. Assumiamo che il sistema da risolvere abbia la seguente forma:

$$\begin{array}{ccccccccc} a_{11}x_1 & +a_{12}x_2 & \dots & +a_{1i}x_i & \dots & +a_{1n}x_n & = b_1 \\ & a_{22}x_2 & \dots & +a_{2i}x_i & \dots & +a_{2n}x_n & = b_2 \\ & & \ddots & \vdots & & \vdots & \vdots \\ & & & a_{ii}x_i & \dots & +a_{in}x_n & = b_i \\ & & & & \ddots & \vdots & \vdots \\ & & & & & a_{nn}x_n & = b_n \end{array} \tag{3.6}$$

con $a_{ii} \neq 0$ per ogni i. In questo caso la matrice A è detta *triangolare superiore*. È evidente che in questo caso, la soluzione è immediatamente calcolabile. Infatti:

$$\begin{cases} x_n = \dfrac{b_n}{a_{nn}} \\ \\ x_i = \dfrac{b_i - \displaystyle\sum_{j=i+1}^{n} a_{ij}x_j}{a_{ii}} \qquad i = n-1, \dots, 1. \end{cases} \tag{3.7}$$

Il metodo (3.7) prende il nome di metodo di sostituzione all'indietro, poichè il vettore $\mathbf{x}$ viene calcolato partendo dall'ultima componente. Anche per il seguente sistema il vettore soluzione è calcolabile in modo analogo.

$$\begin{array}{ccccccccc} a_{11}x_1 & & & & & & & = b_1 \\ a_{21}x_1 & +a_{22}x_2 & & & & & & = b_2 \\ \vdots & \vdots & \ddots & & & & & \vdots \\ a_{i1}x_1 & +a_{i2}x_2 & \dots & +a_{ii}x_i & & & & = b_i \\ \vdots & \vdots & & & \ddots & & & \vdots \\ a_{n1}x_1 & +a_{n2}x_2 & \dots & +a_{ni}x_i & \dots & +a_{nn}x_n & & = b_n \end{array} \tag{3.8}$$

In questo caso la matrice dei coefficienti è triangolare inferiore e la soluzione viene calcolata con il metodo di sostituzione in avanti:

$$\begin{cases} x_1 = \dfrac{b_1}{a_{11}} \\ \\ x_i = \dfrac{b_i - \displaystyle\sum_{j=1}^{i-1} a_{ij}x_j}{a_{ii}} \qquad i = 2, \ldots, n. \end{cases}$$

Concludiamo questo paragrafo facendo alcune considerazioni sul costo computazionale dei metodi di sostituzione. Per costo computazionale di un algoritmo si intende il numero di operazioni che esso richiede per fornire la soluzione di un determinato problema. Nel caso di algoritmi numerici le operazioni che si contano sono quelle aritmetiche che operano su dati reali. Considerano per esempio il metodo di sostituzione in avanti osserviamo che per calcolare x_1 è necessaria una sola operazione (una divisione), per calcolare x_2 le operazioni sono tre (un prodotto, una differenza e una divisione), mentre il generico x_i richiede $2i-1$ operazioni ($i-1$ prodotti, $i-1$ differenze e una divisione), quindi indicato con $C(n)$ il numero totale di operazioni necessarie è:

$$C(n) = \sum_{i=1}^{n}(2i-1) = 2\sum_{i=1}^{n} i - \sum_{i=1}^{n} 1 = 2\frac{n(n+1)}{2} - n = n^2,$$

sfruttando la proprietà che

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}.$$

3.2.2 Metodo di Eliminazione di Gauss

L'idea di base del metodo di Gauss è appunto quella di operare delle opportune trasformazioni sul sistema originale $A\mathbf{x} = \mathbf{b}$, che non costino eccessivamente, e consentano di ottenere un sistema equivalente, che ammetta cioè la stessa soluzione, ma che sia di facile risoluzione, per esempio si può pensare di trasformare il sistema in uno triangolare superiore. Prima di descrivere il metodo di eliminazione di Gauss vediamo un esempio di come funziona.

Supponiamo di dover risolvere il sistema:

$$\begin{array}{rrrrcl} 2x_1 & +x_2 & +x_3 & & = & -1 \\ -6x_1 & -4x_2 & -5x_3 & +x_4 & = & 1 \\ -4x_1 & -6x_2 & -3x_3 & -x_4 & = & 2 \\ 2x_1 & -3x_2 & +7x_3 & -3x_4 & = & 0. \end{array}$$

Il vettore soluzione di un sistema lineare non cambia se ad un'equazione viene sommata la combinazione lineare di un'altra equazione del sistema. L'idea alla base del metodo di Gauss è quella di ottenere un sistema lineare con matrice dei coefficienti triangolare superiore effettuando opportune combinazioni lineari tra le equazioni. Poniamo

$$A^{(1)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ -6 & -4 & -5 & 1 \\ -4 & -6 & -3 & -1 \\ 2 & -3 & 7 & -3 \end{bmatrix}, \qquad \mathbf{b}^{(1)} = \begin{bmatrix} -1 \\ 1 \\ 2 \\ 0 \end{bmatrix}$$

rispettivamente la matrice dei coefficienti e il vettore dei termini noti del sistema di partenza. Calcoliamo un sistema lineare equivalente a quello iniziale ma che abbia gli elementi sottodiagonali della prima colonna uguali a zero. Azzeriamo ora l'elemento $a_{21}(1)$. Lasciamo inalterata la prima equazione. Poniamo

$$l_{21} = -\frac{a_{21}}{a_{11}} = -\frac{-6}{2} = 3$$

e moltiplichiamo la prima equazione per l_{21} ottenendo:

$$6x_1 + 3x_2 + 3x_3 = -3.$$

La nuova seconda equazione sarà la somma tra la seconda equazione e la prima moltiplicata per l_{21}:

$$\begin{array}{rrrrll} -6x_1 & -4x_2 & -5x_3 & +x_4 & = 1 & \\ 6x_1 & +3x_2 & +3x_3 & & = -3 & \\ \hline & -x_2 & -2x_3 & +x_4 & = -2 & [\texttt{Nuova seconda equazione}]. \end{array}$$

Prcediamo nello stesso modo per azzerare gli altri elementi della prima colonna. Poniamo

$$l_{31} = -\frac{a_{31}^{(1)}}{a_{11}^{(1)}} = -\frac{-4}{2} = 2$$

e moltiplichiamo la prima equazione per l_{31} ottenendo:

$$4x_1 + 2x_2 + 2x_3 = -2.$$

La nuova terza equazione sarà la somma tra la terza equazione e la prima moltiplicata per l_{31}:

$$\begin{array}{rrrrll} -4x_1 & -6x_2 & -3x_3 & -x_4 & = 2 & \\ 4x_1 & +2x_2 & +2x_3 & & = -2 & \\ \hline & -4x_2 & -x_3 & -x_4 & = 0 & \texttt{[Nuova terza equazione]}. \end{array}$$

Poniamo ora

$$l_{41} = -\frac{a_{41}^{(1)}}{a_{11}^{(1)}} = -\frac{2}{2} = -1$$

e moltiplichiamo la prima equazione per l_{41} ottenendo:

$$-2x_1 - x_2 - x_3 = 1.$$

La nuova quarta equazione sarà la somma tra la quarta equazione e la prima moltiplicata per l_{41}:

$$\begin{array}{rrrrll} 2x_1 & -3x_2 & +7x_3 & -3x_4 & = 0 & \\ -2x_1 & -x_2 & -x_3 & & = 1 & \\ \hline & -4x_2 & +6x_3 & -3x_4 & = 1 & \texttt{[Nuova quarta equazione]}. \end{array}$$

I numeri $l_{21}, l_{3,1}, \ldots$ sono detti moltiplicatori.
Al secondo passo il sistema lineare è diventato:

$$\begin{array}{rrrrl} 2x_1 & +x_2 & +x_3 & & = -1 \\ & -x_2 & -2x_3 & +x_4 & = -2 \\ & -4x_2 & -x_3 & -x_4 & = 0 \\ & -4x_2 & +6x_3 & -3x_4 & = 1. \end{array}$$

La matrice dei coefficienti e il vettore dei termini noti sono diventati:

$$A^{(2)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & -4 & -1 & -1 \\ 0 & -4 & 6 & -3 \end{bmatrix}, \qquad \mathbf{b}^{(2)} = \begin{bmatrix} -1 \\ -2 \\ 0 \\ 1 \end{bmatrix}.$$

Cerchiamo ora di azzerare gli elementi sottodiagonali della seconda colonna, a partire da a_{32}, usando una tecnica simile. Innanzitutto osserviamo che non

conviene prendere in considerazione una combinazione lineare che coinvolga la prima equazione perchè avendo questa un elemento in prima posizione diverso da zero quando sommata alla terza equazione cancellerà l'elemento uguale a zero in prima posizione. Lasciamo quindi inalterate le prime due equazioni del sistema e prendiamo come equazione di riferimento la seconda. Poichè $a_{22}^{(2)} \neq 0$ poniamo

$$l_{32} = -\frac{a_{32}^{(2)}}{a_{22}^{(2)}} = -\frac{-4}{-1} = -4$$

e moltiplichiamo la seconda equazione per l_{32} ottenendo:

$$4x_2 + 8x_3 - 4x_4 = 8.$$

La nuova terza equazione sarà la somma tra la terza equazione e la seconda appena modificata:

$$\begin{array}{rrrll} -4x_2 & -x_3 & -x_4 & = 0 & \\ 4x_2 & +8x_3 & -4x_4 & = 8 & \\ \hline & 7x_3 & -5x_4 & = 8 & [\texttt{Nuova terza equazione}]. \end{array}$$

Poniamo

$$l_{42} = -\frac{a_{42}^{(2)}}{a_{22}^{(2)}} = -\frac{-4}{-1} = -4$$

e moltiplichiamo la seconda equazione per l_{42} ottenendo:

$$4x_2 + 8x_3 - 4x_4 = 8.$$

La nuova quarta equazione sarà la somma tra la quarta equazione e la seconda appena modificata:

$$\begin{array}{rrrll} -4x_2 & +6x_3 & -3x_4 & = 1 & \\ 4x_2 & +8x_3 & -4x_4 & = 8 & \\ \hline & 14x_3 & -7x_4 & = 9 & [\texttt{Nuova quarta equazione}]. \end{array}$$

Al terzo passo il sistema lineare è diventato:

$$\begin{array}{rrrrl} 2x_1 & +x_2 & +x_3 & & = -1 \\ & -x_2 & -2x_3 & +x_4 & = -2 \\ & & 7x_3 & -5x_4 & = 8 \\ & & 14x_3 & -7x_4 & = 9. \end{array}$$

La matrice dei coefficienti e il vettore dei termini noti sono quindi

$$A^{(3)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 7 & -5 \\ 0 & 0 & 14 & -7 \end{bmatrix}, \qquad \mathbf{b}^{(2)} = \begin{bmatrix} -1 \\ -2 \\ 8 \\ 9 \end{bmatrix}.$$

Resta da azzerare l'unico elemento sottodiagonali della terza colonna. Lasciamo inalterate le prime tre equazioni del sistema. Poniamo

$$l_{43} = -\frac{a_{43}^{(3)}}{a_{33}^{(3)}} = -\frac{14}{7} = -2$$

e moltiplichiamo la terza equazione per l_{43} ottenendo:

$$-14x_3 + 10x_4 = -16.$$

La nuova quarta equazione sarà la somma tra la quarta equazione e la terza appena modificata:

$$\begin{array}{rrll} 14x_3 & -7x_4 & = -16 & \\ -14x_3 & +10x_4 & = 9 & \\ \hline & 3x_4 & = -7 & [\texttt{Nuova quarta equazione}]. \end{array}$$

Abbiamo ottenuto un sistema triangolare superiore:

$$\begin{array}{rrrrl} 2x_1 & +x_2 & +x_3 & & = -1 \\ & -x_2 & -2x_3 & +x_4 & = 4 \\ & & 7x_3 & -5x_4 & = 8 \\ & & & 3x_4 & = -7. \end{array}$$

La matrice dei coefficienti e il vettore dei termini noti sono diventati:

$$A^{(4)} = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 0 & -1 & -2 & 1 \\ 0 & 0 & 7 & -5 \\ 0 & 0 & 0 & 3 \end{bmatrix}, \qquad \mathbf{b}^{(4)} = \begin{bmatrix} -1 \\ 4 \\ 8 \\ -7 \end{bmatrix}.$$

Vediamo ora di calcolare le formule che consentano di calcolare gli elementi della matrice dei coefficienti e del vettore dei termini noti ad ogni passo del metodo di Gauss. Abbiamo detto che $A^{(1)}$ e $\mathbf{b}^{(1)}$ sono assegnati inizialmente, ipotizziamo per il momento che $a_{11}^{(1)} \neq 0$. Calcoliamo ora gli stessi dati al passo 2 tenendo presente che:

1. La prima equazione del sistema resta invariata;
2. Gli elementi sottodiagonali della prima colonna di $A^{(2)}$ sono nulli;
3. La $i-$esima equazione del sistema ($i \geq 2$) è ottenuta sommando alla medesima equazione la prima moltiplicata per $-a_{i1}^{(1)}/a_{11}^{(1)}$.

Fissiamo quindi un'equazione i, $i \geq 2$, e calcoliamone i coefficienti $a_{ij}^{(2)}$ e $b_i^{(2)}$:

$$\begin{array}{cccccccccc} & a_{i1}^{(1)} & a_{i2}^{(1)} & a_{i3}^{(1)} & \dots & a_{ij}^{(1)} & \dots & a_{in}^{(1)} & b_i^{(1)} & + \\ -\dfrac{a_{i1}^{(1)}}{a_{11}^{(1)}}\times & a_{11}^{(1)} & a_{12}^{(1)} & a_{13}^{(1)} & \dots & a_{1j}^{(1)} & \dots & a_{1n}^{(1)} & b_1^{(1)} & = \\ \hline & 0 & a_{i2}^{(2)} & a_{i3}^{(2)} & \dots & a_{ij}^{(2)} & \dots & a_{in}^{(2)} & b_i^{(2)} & \end{array}$$

dove

$$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} a_{1j}^{(1)}, \qquad i,j = 2,\dots,n$$

e

$$b_i^{(2)} = b_i^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} b_1^{(1)}, \qquad i = 2,\dots,n.$$

Se ipotizziamo che $a_{22}^{(2)} \neq 0$ possiamo calcolare gli elementi del sistema al passo 3 tenendo presente che:

1. La prime 2 equazioni del sistema restano invariate;
2. Gli elementi sottodiagonali della prime 2 colonna di $A^{(3)}$ sono nulli;
3. La $i-$esima equazione del sistema ($i \geq 3$) è ottenuta sommando alla medesima equazione la seconda moltiplicata per $-a_{i2}^{(2)}/a_{22}^{(2)}$.

Fissiamo quindi un'equazione i, $i \geq 3$, e calcoliamone i coefficienti $a_{ij}^{(3)}$ e $b_i^{(3)}$:

$$\begin{array}{rccccccccc} & 0 & a_{i2}^{(2)} & a_{i3}^{(2)} & \dots & a_{ij}^{(2)} & \dots & a_{in}^{(2)} & b_i^{(2)} & + \\ -\dfrac{a_{i2}^{(2)}}{a_{22}^{(2)}}\times & 0 & a_{22}^{(2)} & a_{23}^{(2)} & \dots & a_{2j}^{(2)} & \dots & a_{2n}^{(2)} & b_2^{(2)} & = \\ \hline & 0 & 0 & a_{i3}^{(3)} & \dots & a_{ij}^{(3)} & \dots & a_{in}^{(3)} & b_i^{(3)} & \end{array}$$

dove

$$a_{ij}^{(3)} = a_{ij}^{(2)} - \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} a_{2j}^{(2)}, \qquad i,j = 3,\dots,n$$

e

$$b_i^{(3)} = b_i^{(2)} - \frac{a_{i2}^{(2)}}{a_{22}^{(2)}} b_2^{(2)}, \qquad i = 3,\dots,n.$$

Avendo ricavato esplicitamente le formule per i primi due passi del metodo di Gauss è semplice ricavare quelle per un generico passo k. La matrice $A^{(k)}$ ha gli elementi sottodiagonali delle prime $k-1$ colonne uguali a zero, e, supposto $a_{kk}^{(k)} \neq 0$, gli elementi di $A^{(k+1)}$ e di $\mathbf{b}^{(k+1)}$ sono quindi:

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}, \qquad i,j = k+1,\dots,n \tag{3.9}$$

e

$$b_i^{(k+1)} = b_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} b_k^{(k)}, \qquad i = k+1,\dots,n. \tag{3.10}$$

Il valore di k varia da 1 (matrice dei coefficienti e vettori dei termini noti iniziali) fino a $n-1$, infatti la matrice $A^{(n)}$ avrà gli elementi sottodiagonali delle prime $n-1$ colonne uguali a zero.
Si può osservare che il metodo di eliminazione di Gauss ha successo se tutti gli elementi $a_{kk}^{(k)}$ sono diversi da zero, che sono detti elementi pivotali.
Un proprietà importante delle matrici $A^{(k)}$ è il fatto che le operazioni effettuate non alterano il determinante della matrice, quindi

$$\det A^{(k)} = \det A,$$

per ogni k. Poichè la matrice $A^{(n)}$ è triangolare superiore allora il suo determinante può essere calcolato esplicitamente

$$\det A^{(k)} = \prod_{k=1}^{n} a_{kk}^{(k)}.$$

Quello appena descritto è un modo, alternativo alla regola di Laplace per calcolare il determinante della matrice A.

3.2.3 Costo Computazionale del Metodo di Eliminazione di Gauss

Cerchiamo ora di determinare il costo computazionale (cioè il numero di operazioni aritmetiche) richiesto dal metodo di eliminazione di Gauss per risolvere un sistema lineare di ordine n. Dalle relazioni

$$\begin{aligned} b_i^{(k+1)} &= b_i^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} b_k^{(k)}, \qquad i = k+1, \ldots, n, \\ a_{ij}^{(k+1)} &= a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)}, \qquad i, j = k+1, \ldots, n \end{aligned}$$

è evidente che servono 3 operazioni per calcolare $b_i^{(k+1)}$ (noto $b_i^{(k)}$) mentre sono necessarie che solo 2 operazioni per calcolare $a_{ij}^{(k+1)}$ (noto $a_{ij}^{(k)}$), infatti il moltiplicatore viene calcolato solo una volta. Il numero di elementi del vettore dei termini noti che vengono modificati è pari ad $n-k$ mentre gli elementi della matrice cambiati sono $(n-k)^2$ quindi complessivamente il numero di operazioni per calcolare gli elementi al passo $k+1$ è:

$$2(n-k)^2 + 3(n-k)$$

Pertanto per trasformare A in $A^{(n)}$ e $\mathbf{b}$ in $\mathbf{b}^{(n)}$ è necessario un numero di operazioni pari ala somma, rispetto a k, di tale valore

$$f(n) = 2\sum_{k=1}^{n-1}(n-k)^2 + 3\sum_{k=1}^{n-1}(n-k).$$

Sapendo che

$$\sum_{k=1}^{n} n^2 = \frac{n(n+1)(2n+1)}{6}$$

ed effettuando un opportuno cambio di indice nelle sommatorie risulta

$$f(n) = 2\left[\frac{n(n-1)(2n-1)}{6}\right] + 3\frac{n(n-1)}{2} = \frac{2}{3}n^3 + \frac{n^2}{2} - \frac{7}{6}n.$$

A questo valore bisogna aggiungere le n^2 operazioni aritmetiche necessarie per risolvere il sistema triangolare superiore ottenendo

$$\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n$$

che è un valore molto inferiore rispetto alle $n!$ operazioni richieste dalla regola di Cramer, applicata insieme alla regola di Laplace.

3.2.4 Strategie di Pivoting per il metodo di Gauss

Nell'eseguire il metodo di Gauss si è fatta l'implicita ipotesi (vedi formule (3.9) e (3.10)) che gli elementi pivotali $a_{kk}^{(k)}$ siano non nulli per ogni k. Tale situazione si verifica quando i minori principali di testa di ordine di A sono diversi da zero. Infatti vale il seguente risultato.

Teorema 3.2.1 *Se $A \in \mathbb{R}^{n\times n}$, indicata con A_k la matrice principale di testa di ordine k, risulta*

$$a_{kk}^{(k)} = \frac{\det A_k}{\det A_{k-1}}, \qquad k = 1, \ldots, n$$

avendo posto per convenzione $\det A_0 = 1$.

In vero questa non è un'ipotesi limitante in quanto la non singolarità di A permette, con un opportuno scambio di righe in $A^{(k)}$, di ricondursi a questo caso. Infatti scambiare due righe in $A^{(k)}$ significa sostanzialmente scambiare due equazioni nel sistema $A^{(k)}\mathbf{x} = \mathbf{b}^{(k)}$ e ciò non altera la natura del sistema stesso.
Consideriamo la matrice $A^{(k)}$ e supponiamo $a_{kk}^{(k)} = 0$. In questo caso possiamo scegliere un elemento sottodiagonale appartenente alla $k-$esima colonna diverso da zero, supponiamo $a_{ik}^{(k)}$, scambiare le equazioni di indice i e k e continuare il procedimento perchè in questo modo l'elemento pivotale è diverso da zero. In ipotesi di non singolarità della matrice A possiamo dimostrare tale elemento diverso da zero esiste sicuramente. Infatti supponendo che,

oltra all'elemento pivotale, siano nulli tutti gli $a_{ik}^{(k)}$ per $i = k+1, \ldots, n$, allora $A^{(k)}$ ha la seguente struttura:

$$A^{(k)} = \begin{bmatrix} a_{11}^{(1)} & \ldots & a_{1,k-1}^{(1)} & a_{1k}^{(1)} & a_{1,k+1}^{(1)} & \ldots & a_{1n}^{(1)} \\ & \ddots & \vdots & \vdots & \vdots & & \vdots \\ & & a_{k-1,k-1}^{(k-1)} & a_{k-1,k}^{(k-1)} & a_{k-1,k+1}^{(k-1)} & \ldots & a_{k-1,n}^{(k-1)} \\ & & & 0 & a_{k,k+1}^{(k)} & & a_{kn}^{(k)} \\ & 0 & & \vdots & \vdots & & \vdots \\ & & & 0 & a_{n,k+1}^{(k)} & \ldots & a_{nn}^{(k)}. \end{bmatrix}$$

Se partizioniamo $A^{(k)}$ nel seguente modo

$$A^{(k)} = \begin{bmatrix} A_{11}^{(k)} & A_{12}^{(k)} \\ 0 & A_{22}^{(k)} \end{bmatrix}$$

con $A_{11}^{(k)} \in \mathbb{R}^{(k-1)\times(k-1)}$ allora il determinante di $A^{(k)}$ è

$$\det A^{(k)} = \det A_{11}^{(k)} \det A_{22}^{(k)} = 0$$

perchè la matrice $A_{22}^{(k)}$ ha una colonna nulla. Poichè tutte la matrici $A^{(k)}$ hanno lo stesso determinante di A, dovrebbe essere $\det A = 0$ e questo contrasta con l'ipotesi fatta. Quindi possiamo concludere che se $a_{kk}^{(k)} = 0$ e $\det A \neq 0$ deve necessariamente esistere un elemento $a_{ik}^{(k)} \neq 0$, con $i \in \{k+1, k+2, \ldots, n\}$. Per evitare che un elemento pivotale possa essere uguale a zero si applica una delle cosiddette strategie di pivoting. La strategia di Pivoting parziale prevede che prima di fare ciò si ricerchi l'elemento di massimo modulo tra gli elementi $a_{kk}^{(k)}$, $a_{k+1,k}^{(k)}$, $\ldots$, $a_{nk}^{(k)}$ e si scambi la riga in cui si trova questo elemento con la k-esima qualora esso sia diverso da $a_{kk}^{(k)}$. In altri termini il pivoting parziale richiede le seguenti operazioni:

1. determinare l'elemento $a_{rk}^{(k)}$ tale che

$$|a_{rk}^{(k)}| = \max_{k \leq i \leq n} |a_{ik}^{(k)}|;$$

2. effettuare lo scambio tra la r-esima e la k-esima riga.

In alternativa si può adottare la strategia di Pivoting totale che è la seguente:

1. determinare gli indici r, s tali che

$$|a_{rs}^{(k)}| = \max_{k \leq i,j \leq n} |a_{ij}^{(k)}|;$$

2. effettuare lo scambio tra la r-esima e la k-esima riga e tra la s-esima e la k-esima colonna.

La strategia di pivoting totale è senz'altro migliore perchè garantisce maggiormente che un elemento pivotale non sia un numero piccolo (in questa eventualità potrebbe accadere che un moltiplicatore sia un numero molto grande) ma richiede che tutti gli eventuali scambi tra le colonne della matrice siano memorizzati. Infatti scambiare due colonne significa scambiare due incognite del vettore soluzione pertanto dopo la risoluzione del sistema triangolare per ottenere il vettore soluzione del sistema di partenza è opportuno permutare le componenti che sono state scambiate. Nelle pagine seguenti sono riportati i codici MatLab che implementano il metodo di Gauss con entrambe le strategie di pivoting descritte.

```
function x=Gauss(A,b)
%
% Metodo di eliminazione di Gauss
%
% Parametri di input:
% A = Matrice dei coefficienti del sistema
% b = Vettore dei termini noti del sistema
%
% Parametri di input:
% x = Vettore soluzione del sistema lineare
%
n = length(b);
x = zeros(n,1);
for k=1:n-1
  if abs(A(k,k))<eps
    error('Elemento pivotale nullo ')
  end
  for i=k+1:n
    A(i,k) = A(i,k)/A(k,k);
    b(i) = b(i)-A(i,k)*b(k);
```

```
    for j=k+1:n
      A(i,j) = A(i,j)-A(i,k)*A(k,j);
    end
 end
end
x(n) = b(n)/A(n,n);
for i=n-1:-1:1
  x(i) = (b(i)-A(i,i+1:n)*x(i+1:n))/A(i,i);
end
return

function x=Gauss_pp(A,b)
%
% Metodo di Gauss con pivot parziale
%
% Parametri di input:
% A = Matrice dei coefficienti del sistema
% b = Vettore dei termini noti del sistema
%
% Parametri di input:
% x = Vettore soluzione del sistema lineare
%
n = length(b);
x = zeros(n,1);
for k=1:n-1
  [a,i] = max(abs(A(k:n,k)));
  i = i+k-1;
  if i~=k
    A([i k],:) = A([k i],:);
    b([i k]) = b([k i]);
  end
  for i=k+1:n
    A(i,k) = A(i,k)/A(k,k);
    b(i) = b(i)-A(i,k)*b(k);
    for j=k+1:n
      A(i,j) = A(i,j)-A(i,k)*A(k,j);
    end
 end
```

```
end
x(n) = b(n)/A(n,n);
for i=n-1:-1:1
  x(i) = (b(i)-A(i,i+1:n)*x(i+1:n))/A(i,i);
end
return

function x=Gauss_pt(A,b)
%
% Metodo di Gauss con pivot totale
%
% Parametri di input:
% A = Matrice dei coefficienti del sistema
% b = Vettore dei termini noti del sistema
%
% Parametri di input:
% x = Vettore soluzione del sistema lineare
%
n = length(b);
x = zeros(n,1);
x1 = x;
indice = [1:n];
for k=1:n-1
  [a,riga] = max(abs(A(k:n,k:n)));
  [mass,col] = max(a);
  j = col+k-1;
  i = riga(col)+k-1;
  if i~=k
  A([i k],:) = A([k i],:);
  b([i k]) = b([k i]);
  end
  if j~=k
  A(:,[j k]) = A(:,[k j]);
    indice([j k]) = indice([k j]);
  end
  for i=k+1:n
    A(i,k) = A(i,k)/A(k,k);
    b(i) = b(i)-A(i,k)*b(k);
```

```
    for j=k+1:n
      A(i,j) = A(i,j)-A(i,k)*A(k,j);
    end
  end
end
%
% Risoluzione del sistema triangolare superiore
%
x1(n) = b(n)/A(n,n);
for i=n-1:-1:1
  x1(i) = (b(i)-A(i,i+1:n)*x1(i+1:n))/A(i,i);
end
%
% Ripermutazione del vettore
%
for i=1:n
  x(indice(i))=x1(i);
end
return
```

3.3 Condizionamento di sistemi lineari

Nel Capitolo 1 è stato introdotto il concetto di rappresentazione in base ed è stata motivata la sostanziale inaffidabilità dei risultati dovuti ad elaborazioni numeriche, a causa dell'artimetica finita dell'elaboratore. Appare chiaro come la bassa precisione nel calcolo potrebbe fornire dei risultati numerici molto lontani da quelli reali. In alcuni casi tale proprietà è insita nel problema. Consideriamo il sistema lineare

$$A\mathbf{x} = \mathbf{b} \tag{3.11}$$

dove $A \in \mathbb{R}^{n\times n}$ è la cosiddetta matrice di Hilbert, i cui elementi sono

$$a_{ij} = \frac{1}{i+j-1}, \qquad i,j = 1,\ldots,n$$

mentre il vettore $\mathbf{b}$ è scelto in modo tale che il vettore soluzione abbia tutte componenti uguali a 1, cosicchè si possa conoscere con esattezza l'errore commesso nel suo calcolo. Risolvendo il sistema di ordine 20 con il metodo di

Gauss senza pivoting si osserva che la soluzione è, in realtà, molto lontana da quella teorica. Questa situazione peggiora prendendo matrici di dimensioni crescenti ed è legata ad un fenomeno che viene detto malcondizionamento. Bisogna infatti ricordare che, a causa degli errori legati alla rappresentazione dei numeri reali, il sistema che l'elaboratore risolve non coincide con quello teorico, poichè alla matrice A ed al vettore $\mathbf{b}$ e necessario aggiungere la matrice δA ed il vettore $\delta\mathbf{b}$ (che contengono le perturbazioni legate a tali errori), e che la soluzione ovviamente non è la stessa, pertanto la indichiamo con $\mathbf{x} + \delta\mathbf{x}$:

$$(A + \delta A)(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}. \tag{3.12}$$

Si può dimostrare che l'ordine di grandezza della perturbazione sulla soluzione è

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \|A\|\|A^{-1}\| \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} \right).$$

Il numero $K(A) = \|A\|\|A^{-1}\|$, detto indice di condizionamento del sistema, misura le amplificazioni degli errori sui dati del problema (ovvero la misura di quanto aumentano gli errori sulla soluzione). Il caso della matrice di Hilbert è appunto uno di quelli per cui l'indice di condizionamento assume valori molto grandi (di ordine esponenziale) all'aumentare della dimensione, si parla infatti di matrici malcondizionate. Quando ciò non accade si parla invece di matrici bencondizionate. A volte tale caratteristica può dipendere anche dalla scelta dell'algoritmo di risoluzione, ovvero vi sono algoritmi che forniscono risultati meno influenzati dal condizionamento dei dati (eseguendo il metodo di Gauss con pivoting parziale, per esempio, i risultati sono affetti comunque da errori, ma di meno rispetto al metodo di Gauss senza alcuna strategia di pivoting).

3.4 La Fattorizzazione LU

Supponiamo di dover risolvere un problema che richieda, ad un determinato passo, la risoluzione del sistema lineare $A\mathbf{x} = \mathbf{b}$ e di utilizzare il metodo di Gauss. La matrice viene resa triangolare superiore e viene risolto il sistema triangolare

$$A^{(n)}\mathbf{x} = \mathbf{b}^{(n)}. \tag{3.13}$$

Ipotizziamo che, nell'ambito dello stesso problema, dopo un certo tempo sia necessario risolvere il sistema

$$A\mathbf{x} = \mathbf{c}$$

i cui la matrice dei coefficienti è la stessa mentre è cambiato il termine noto. Appare chiaro che non è possibile sfruttare i calcoli gia fatti in quanto il calcolo del vettore dei termini noti al passo n dipende dalle matrici ai passi precedenti all'ultimo, quindi la conoscenza della matrice $A^{(n)}$ è del tutto inutile. È necessario pertanto applicare nuovamente il metodo di Gauss e risolvere il sistema triangolare

$$A^{(n)}\mathbf{x} = \mathbf{c}^{(n)}. \tag{3.14}$$

L'algoritmo che sarà descritto in questo paragrafo consentirà di evitare l'eventualità di dover rifare tutti i calcoli (o una parte di questi). La Fattorizzazione *LU* di una matrice stabilisce, sotto determinate ipotesi, l'esistenza di una matrice L triangolare inferiore con elementi diagonali uguali a 1 e di una matrice triangolare superiore U tali che $A = LU$.
Vediamo ora di determinare le formule esplicite per gli elementi delle due matrici. Fissata la matrice A, quadrata di ordine n, imponiamo quindi che risulti

$$A = LU.$$

Una volta note tali matrici il sistema di partenza $A\mathbf{x} = \mathbf{b}$ viene scritto come

$$LU\mathbf{x} = \mathbf{b}$$

e, posto $U\mathbf{x} = \mathbf{y}$, il vettore $\mathbf{x}$ viene trovato prima risolvendo il sistema triangolare inferiore

$$L\mathbf{y} = \mathbf{b}$$

e poi quello triangolare superiore

$$U\mathbf{x} = \mathbf{y}.$$

Imponiamo quindi che la matrice A ammetta fattorizzazione LU:

$$\begin{bmatrix} a_{11} & \dots & a_{1j} & \dots & a_{1n} \\ \vdots & & \vdots & & \vdots \\ a_{i1} & \dots & a_{ij} & \dots & a_{in} \\ \vdots & & \vdots & & \vdots \\ a_{n1} & \dots & a_{nj} & \dots & a_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} 1 & 0 & \dots & \dots & \dots & 0 \\ l_{21} & 1 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & 0 & & \vdots \\ l_{i1} & \dots & l_{i,i-1} & 1 & \ddots & \vdots \\ \vdots & & \vdots & & \ddots & 0 \\ l_{n1} & \dots & l_{n,i-1} & l_{n,i} & \dots & 1 \end{bmatrix} \begin{bmatrix} u_{11} & \dots & \dots & u_{1j} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2j} & \dots & u_{2n} \\ \vdots & \ddots & \ddots & \vdots & & \vdots \\ \vdots & & \ddots & u_{jj} & \dots & u_{jn} \\ \vdots & & & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & \dots & 0 & u_{nn} \end{bmatrix}.$$

Deve essere

$$a_{ij} = \sum_{k=1}^{n} l_{ik}u_{kj} = \sum_{k=1}^{\min(i,j)} l_{ik}u_{kj} \qquad i,j = 1,\dots,n. \tag{3.15}$$

Considerando prima il caso $i \leq j$, uguagliando quindi la parte triangolare superiore delle matrici abbiamo

$$a_{ij} = \sum_{k=1}^{i} l_{ik}u_{kj} \qquad j \geq i \tag{3.16}$$

ovvero

$$a_{ij} = \sum_{k=1}^{i-1} l_{ik}u_{kj} + l_{ii}u_{ij} = \sum_{k=1}^{i-1} l_{ik}u_{kj} + u_{ij} \qquad j \geq i$$

infine risulta

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik}u_{kj} \qquad j \geq i \tag{3.17}$$

e ovviamente $u_{1j} = a_{1j}$, per $j = 1,\dots,n$. Considerando ora il caso $j < i$, uguagliando cioè le parti strettamente triangolari inferiori delle matrici risulta:

$$a_{ij} = \sum_{k=1}^{j} l_{ik}u_{kj} \qquad i > j \tag{3.18}$$

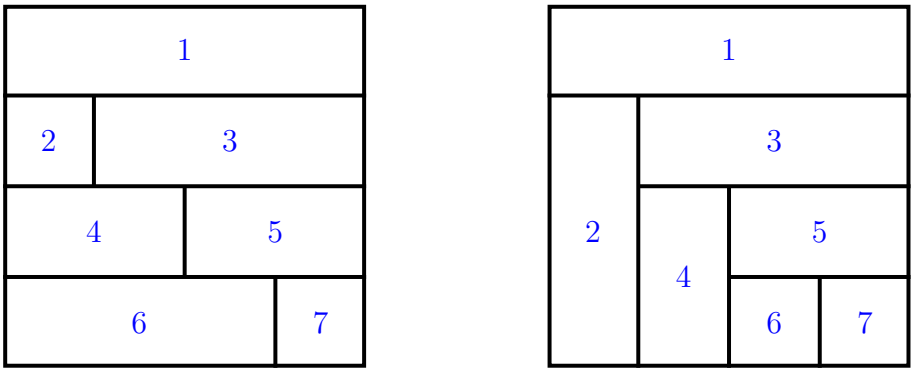
ovvero

$$a_{ij} = \sum_{k=1}^{j-1} l_{ik}u_{kj} + l_{ij}u_{jj} \qquad i > j$$

da cui

$$l_{ij} = \frac{1}{u_{jj}} \left(a_{ij} - \sum_{k=1}^{j-1} l_{ik}u_{kj} \right) \qquad i > j. \tag{3.19}$$

Si osservi che le formule (3.17) e (3.19) vanno implementate secondo uno degli schemi riportati nella seguente figura.



Tecnica di Crout **Tecnica di Doolittle**

Ogni schema rappresenta in modo stilizzato una matrice la cui parte triangolare superiore indica la matrice U mentre quella triangolare inferiore la matrice L mentre i numeri indicano l'ordine con cui gli elementi saranno calcolati. Per esempio applicando la tecnica di Crout si segue il seguente ordine:

- 1^o Passo: Calcolo della prima riga di U;
- 2^o Passo: Calcolo della seconda riga di L;
- 3^o Passo: Calcolo della seconda riga di U;
- 4^o Passo: Calcolo della terza riga di L;
- 5^o Passo: Calcolo della terza riga di U;
- 6^o Passo: Calcolo della quarta riga di L;
- 7^o Passo: Calcolo della quarta riga di U;

e così via procedendo per righe in modo alternato.

```
function [L,U]=crout(A);
%
% La funzione calcola la fattorizzazione LU della
% matrice A applicando la tecnica di Crout
%
% L = matrice triang. inferiore con elementi diagonali
%     uguali a 1
% U = matrice triangolare superiore
%
[m n] = size(A);
U = zeros(n);
L = eye(n);
U(1,:) = A(1,:);
for i=2:n
    for j=1:i-1
        L(i,j) = (A(i,j) - L(i,1:j-1)*U(1:j-1,j))/U(j,j);
    end
    for j=i:n
        U(i,j) = A(i,j) - L(i,1:i-1)*U(1:i-1,j);
    end
end
return
```

Nel caso della tecnica di Doolittle si seguono i seguenti passi:

- 1^o Passo: Calcolo della prima riga di U;
- 2^o Passo: Calcolo della prima colonna di L;
- 3^o Passo: Calcolo della seconda riga di U;
- 4^o Passo: Calcolo della seconda colonna di L;
- 5^o Passo: Calcolo della terza riga di U;
- 6^o Passo: Calcolo della terza colonna di L;
- 7^o Passo: Calcolo della quarta riga di U.

La fattorizzazione LU è un metodo sostanzialmente equivalente al metodo di Gauss, infatti la matrice U che viene calcolata coincide con la matrice $A^{(n)}$. Lo svantaggio del metodo di fattorizzazione diretto risiede essenzialmente nella maggiore difficoltà, rispetto al metodo di Gauss, di poter programmare una strategia di pivot. Infatti se un elemento diagonale della matrice U è uguale a zero non è possibile applicare l'algoritmo.

```
function [L,U]=doolittle(A);
%
% La funzione calcola la fattorizzazione LU della
% matrice A applicando la tecnica di Doolittle
%
% L = matrice triang. inferiore con elementi diagonali
%     uguali a 1
% U = matrice triangolare superiore
%
[m n] = size(A);
L = eye(n);
U = zeros(n);
U(1,:) = A(1,:);
for i=1:n-1
    for riga=i+1:n
        L(riga,i)=(A(riga,i)-L(riga,1:i-1)*U(1:i-1,i))/U(i,i);
    end
    for col=i+1:n
        U(i+1,col) = A(i+1,col)-L(i+1,1:i)*U(1:i,col);
    end
end
return
```

Capitolo 4

Interpolazione e Quadratura

4.1 Introduzione

Nel campo del Calcolo Numerico si possono incontrare diversi casi nei quali è richiesta l'approssimazione di una funzione (o di una grandezza incognita):
1) non è nota l'espressione analitica della funzione $f(x)$ ma si conosce il valore che assume in un insieme finito di punti $x_1, x_2, \ldots, x_n$. Si potrebbe pensare anche che tali valori siano delle misure di una grandezza fisica incognita valutate in differenti istanti di tempo.
2) Si conosce l'espressione analitica della funzione $f(x)$ ma è così complicata dal punto di vista computazionale che è più conveniente cercare un'espressione semplice partendo dal valore che essa assume in un insieme finito di punti. In questo capitolo analizzeremo un particolare tipo di approssimazione di funzioni cioè la cosiddetta interpolazione che richiede che la funzione approssimante assume in determinate ascisse esattamente lo stesso valore di $f(x)$. In entrambi i casi appena citati è noto, date certe informazioni supplementari, che la funzione approssimante va ricercata della forma:

$$f(x) \simeq g(x; a_0, a_1, \ldots, a_n). \tag{4.1}$$

Se i parametri $a_0, a_1, \ldots, a_n$ sono definiti dalla condizione di coincidenza di f e g nei punti $x_0, x_1, \ldots, x_n$, allora tale procedimento di approssimazione si chiama appunto Interpolazione. Invece se $x \notin [\min_i x_i, \max_i x_i]$ allora si parla di *Estrapolazione*. Tra i procedimenti di interpolazione il più usato è

quello in cui si cerca la funzione g in (4.1) nella forma

$$g(x; a_0, a_1, \ldots, a_n) = \sum_{i=0}^{n} a_i \Phi_i(x)$$

dove $\Phi_i(x)$, per $i = 0, \ldots, n$, sono funzioni fissate e i valori di a_i, $i = 0, \ldots, n$, sono determinati in base alle condizioni di coincidenza di f con la funzione approssimante nei punti di interpolazione (detti anche nodi), x_j, cioè si pone

$$f(x_j) = \sum_{i=0}^{n} a_i \Phi_i(x_j) \qquad j = 0, \ldots, n. \tag{4.2}$$

Il processo di determinazione degli a_i attraverso la risoluzione del sistema (4.2) si chiama metodo dei coefficienti indeterminati. Il caso più studiato è quello dell'interpolazione polinomiale, in cui si pone:

$$\Phi_i(x) = x^i \qquad i = 0, \ldots, n$$

e perciò la funzione approssimante g assume la forma

$$\sum_{i=0}^{n} a_i x^i;$$

mentre le condizioni di coincidenza diventano

$$f(x_j) = \sum_{i=0}^{n} a_i x_j^i \qquad j = 0, \ldots, n. \tag{4.3}$$

Se i nodi x_j sono distinti allora la matrice dei coefficienti del sistema (4.3), detta matrice di Vandermonde, è non singolare e pertanto quindi il problema dell'interpolazione ammette sempre un'unica soluzione. Descriviamo ora un modo alternativo di risolvere il problema di interpolazione in grado di fornire l'espressione esplicita del polinomio cercato.

4.2 Il Polinomio Interpolante di Lagrange

Al fine di dare una forma esplicita al polinomio interpolante, scriviamo il candidato polinomio nella seguente forma:

$$L_n(x) = \sum_{k=0}^{n} l_{nk}(x) f(x_k) \tag{4.4}$$

dove gli $l_{nk}(x)$ sono per il momento generici polinomi di grado n. Imponendo le condizioni di interpolazione

$$L_n(x_i) = f(x_i) \qquad i = 0, \ldots, n$$

deve essere, per ogni i:

$$L_n(x_i) = \sum_{k=0}^{n} l_{nk}(x_i) f(x_k) = f(x_i)$$

ed è evidente che se

$$l_{nk}(x_i) = \begin{cases} 0 & \text{se } k \neq i \\ \\ 1 & \text{se } k = i \end{cases} \tag{4.5}$$

allora esse sono soddisfatte. In particolare la prima condizione di (4.5) indica che $l_{nk}(x)$ si annulla negli n nodi $x_0, x_1, \ldots, x_{k-1}, x_{k+1}, \ldots, x_n$ e quindi deve avere la seguente struttura:

$$l_{nk}(x) = c_k \prod_{i=0, i\neq k}^{n} (x - x_i)$$

mentre impondendo la seconda condizione di (4.5)

$$l_{nk}(x_k) = c_k \prod_{i=0, i\neq k}^{n} (x_k - x_i) = 1$$

si trova immediatamente:

$$c_k = \frac{1}{\displaystyle\prod_{i=0, i\neq k}^{n} (x_k - x_i)}.$$

In definitiva il polinomio interpolante ha la seguente forma:

$$L_n(x) = \sum_{k=0}^{n} \left(\prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i} \right) f(x_k). \tag{4.6}$$

Il polinomio (4.6) prende il nome di Polinomio di Lagrange mentre i polinomi:

$$l_{nk}(x) = \prod_{i=0, i\neq k}^{n} \frac{x - x_i}{x_k - x_i}; \qquad k = 0, 1, \ldots, n$$

si chiamano Polinomi Fondamentali di Lagrange.

4.2.1 Il Resto del Polinomio di Lagrange

Assumiamo che la funzione interpolata $f(x)$ sia di classe $\mathcal{C}^{n+1}([a,b])$ e valutiamo l'errore che si commette nel sostituire $f(x)$ con $L_n(x)$ in un punto $x \neq x_i$. Supponiamo che l'intervallo $[a,b]$ sia tale da contenere sia i nodi x_i che l'ulteriore punto x. Sia dunque

$$e(x) = f(x) - L_n(x)$$

l'errore (o resto) commesso nell'interpolazione della funzione $f(x)$. Poichè

$$e(x_i) = f(x_i) - L_n(x_i) = 0 \qquad i = 0, \ldots, n$$

è facile congetturare per $e(x)$ la seguente espressione:

$$e(x) = c(x)\omega_{n+1}(x)$$

dove

$$\omega_{n+1}(x) = \prod_{i=0}^{n}(x - x_i)$$

è il cosiddetto polinomio nodale mentre $c(x)$ è una funzione da determinare. Definiamo ora la funzione

$$\Phi(t;x) = f(t) - L_n(t) - c(x)\omega_{n+1}(t)$$

dove t è una variabile ed x è un valore fissato. Calcoliamo la funzione $\Phi(t;x)$ nei nodi x_i:

$$\Phi(x_i;x) = f(x_i) - L_n(x_i) - c(x)\omega_{n+1}(x_i) = 0$$

e anche nel punto x:

$$\Phi(x;x) = f(x) - L_n(x) - c(x)\omega_{n+1}(x) = e(x) - c(x)\omega_{n+1}(x) = 0$$

pertanto la funzione $\Phi(t;x)$ (che è derivabile con continuità $n+1$ volte poichè $f(x)$ è di classe $\mathcal{C}^{n+1}$) ammette almeno $n+2$ zeri distinti. Applicando il teorema di Rolle segue che $\Phi'(t;x)$ ammette almeno $n+1$ zeri distinti. Riapplicando lo stesso teorema segue che $\Phi''(t;x)$ ammette almeno n zeri distinti. Così proseguendo segue che

$$\exists \xi_x \in [a,b] \;\; \ni' \;\; \Phi^{(n+1)}(\xi_x;x) = 0.$$

Calcoliamo ora la derivata di ordine $n+1$ della funzione $\Phi(t;x)$, osservando innanzitutto che la derivata di tale ordine del polinomio $L_n(x)$ è identicamente nulla. Pertanto

$$\Phi^{(n+1)}(t;x) = f^{(n+1)}(t) - c(x)\frac{d^{n+1}}{dt^{n+1}}\omega_{n+1}(t).$$

Calcoliamo la derivata di ordine $n+1$ del polinomio nodale. Osserviamo innanzitutto che

$$\omega_{n+1}(t) = \prod_{i=0}^{n}(t-x_i) = t^{n+1} + p_n(t)$$

dove $p_n(t)$ è un polinomio di grado al più n. Quindi

$$\frac{d^{n+1}}{dt^{n+1}}\omega_{n+1}(t) = \frac{d^{n+1}}{dt^{n+1}}t^{n+1}.$$

Poichè

$$\frac{d}{dt}t^{n+1} = (n+1)t^n$$

e

$$\frac{d^2}{dt^2}t^{n+1} = (n+1)nt^{n-1}$$

è facile dedurre che

$$\frac{d^{n+1}}{dt^{n+1}}t^{n+1} = \frac{d^{n+1}}{dt^{n+1}}\omega_{n+1}(t) = (n+1)!.$$

Pertanto

$$\Phi^{(n+1)}(t;x) = f^{(n+1)}(t) - c(x)(n+1)!$$

e quindi

$$\Phi^{(n+1)}(\xi_x;x) = f^{(n+1)}(\xi_x) - c(x)(n+1)! = 0$$

cioè

$$c(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}$$

e in definitiva

$$e(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\omega_{n+1}(x). \tag{4.7}$$

Esempio 4.2.1 *Supponiamo di voler calcolare il polinomio interpolante di Lagrange passante per i punti* $(-1,-1)$, $(0,1)$, $(1,-1)$, $(3,2)$ *e* $(5,6)$. *Il grado di tale polinomio è* 4, *quindi definiamo i nodi*

$$x_0 = -1, \quad x_1 = 0, \quad x_2 = 1, \quad x_3 = 3, \quad x_4 = 5,$$

cui corrispondono le ordinate che indichiamo con y_i, $i = 0, \ldots, 4$:

$$y_0 = -1, \quad y_1 = 1, \quad y_2 = -1, \quad y_3 = 2, \quad y_4 = 6.$$

Scriviamo ora l'espressione del polinomio $L_4(x)$:

$$L_4(x) = l_{4,0}(x)y_0 + l_{4,1}(x)y_1 + l_{4,2}(x)y_2 + l_{4,3}(x)y_3 + l_{4,4}(x)y_4 \tag{4.8}$$

e calcoliamo i 5 *polinomi fondamentali di Lagrange:*

$$\begin{aligned}
l_{4,0}(x) &= \frac{(x-0)(x-1)(x-3)(x-5)}{(-1-0)(-1-1)(-1-3)(-1-5)} = \\
&= \frac{1}{48}\, x(x-1)(x-3)(x-5) \\
l_{4,1}(x) &= \frac{(x+1)(x-1)(x-3)(x-5)}{(0+1)(0-1)(0-3)(0-5)} = \\
&= -\frac{1}{15}(x+1)(x-1)(x-3)(x-5) \\
l_{4,2}(x) &= \frac{(x+1)(x-0)(x-3)(x-5)}{(1+1)(1-0)(1-3)(1-5)} = \\
&= \frac{1}{16}x(x+1)(x-3)(x-5) \\
l_{4,3}(x) &= \frac{(x+1)(x-0)(x-1)(x-5)}{(3+1)(3-0)(3-1)(3-5)} = \\
&= -\frac{1}{48}x(x+1)(x-1)(x-5) \\
l_{4,4}(x) &= \frac{(x+1)(x-0)(x-1)(x-3)}{(5+1)(5-0)(5-1)(5-3)} = \\
&= \frac{1}{240}x(x+1)(x-1)(x-3)
\end{aligned}$$

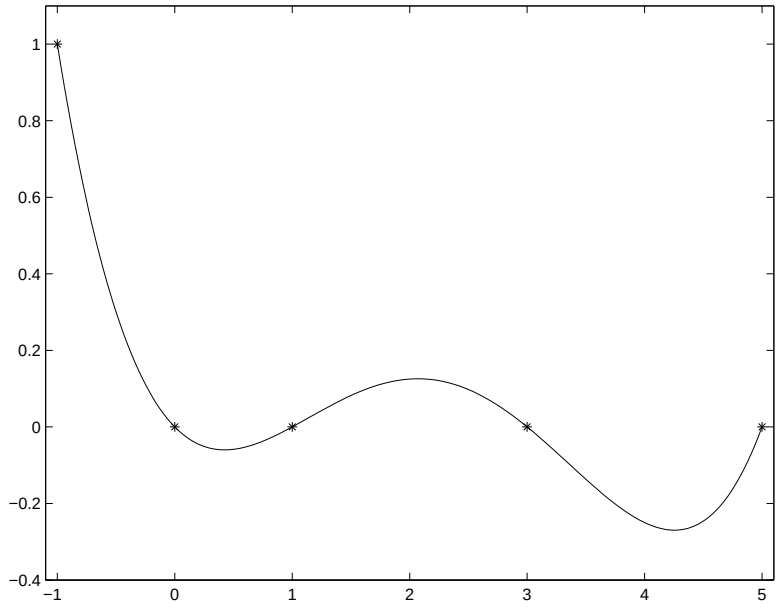


Figura 4.1: Grafico del polinomio $l_{40}(x)$.

Sostituendo in (4.8) il valore della funzione nei nodi si ottiene l'espressione finale del polinomio interpolante:

$$L_4(x) = -l_{4,0}(x) + l_{4,1}(x) - l_{4,2}(x) + 2l_{4,3}(x) + 6l_{4,4}(x).$$

Se vogliamo calcolare il valore approssimato della funzione $f(x)$ in un'ascissa diversa dai nodi, per esempio $x = 2$ allora dobbiamo calcolare il valore del polinomio interpolante $L_4(2)$.
Nelle figure 4.1-4.5 sono riportati i grafici dei cinque polinomi fondamentali di Lagrange: gli asterischi evidenziano il valore assunto da tali polinomi nei nodi di interpolazione. Nella figura 4.6 è tracciato il grafico del polinomio interpolante di Lagrange, i cerchi evidenziano ancora una volta i punti di interpolazione.

4.2.2 Il fenomeno di Runge

Nell'espressione dell'errore è presente, al denominatore, il fattore $(n+1)!$, che potrebbe indurre a ritenere che, interpolando la funzione con un elevato numero di nodi, l'errore tenda a zero e quindi che il polinomio interpolante tenda

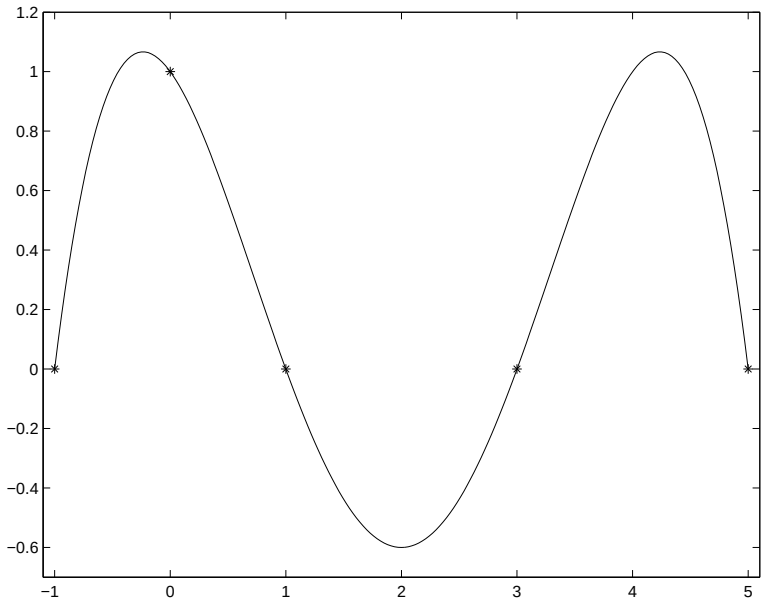


Figura 4.2: Grafico del polinomio $l_{41}(x)$.

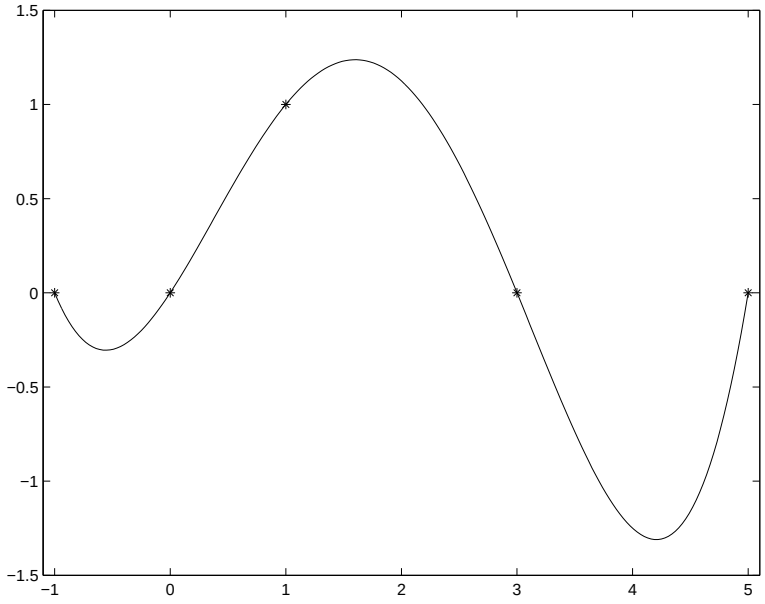


Figura 4.3: Grafico del polinomio $l_{42}(x)$.

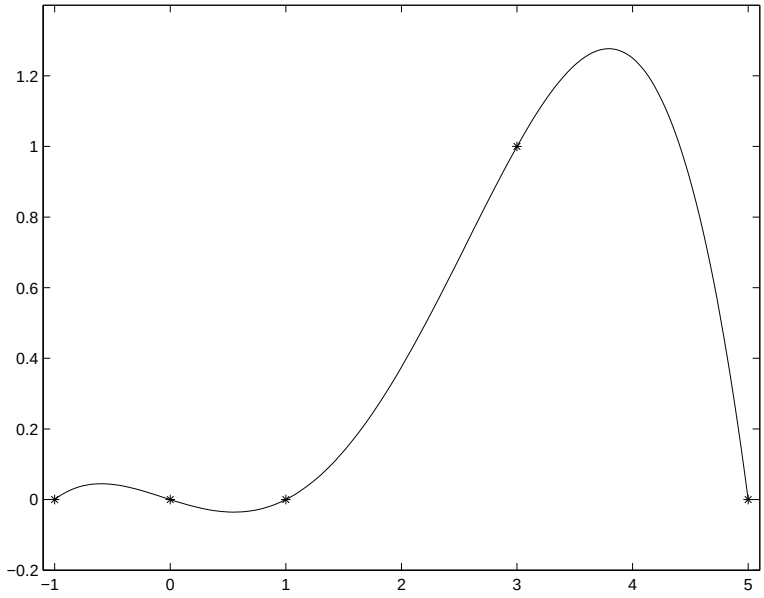


Figura 4.4: Grafico del polinomio $l_{43}(x)$.

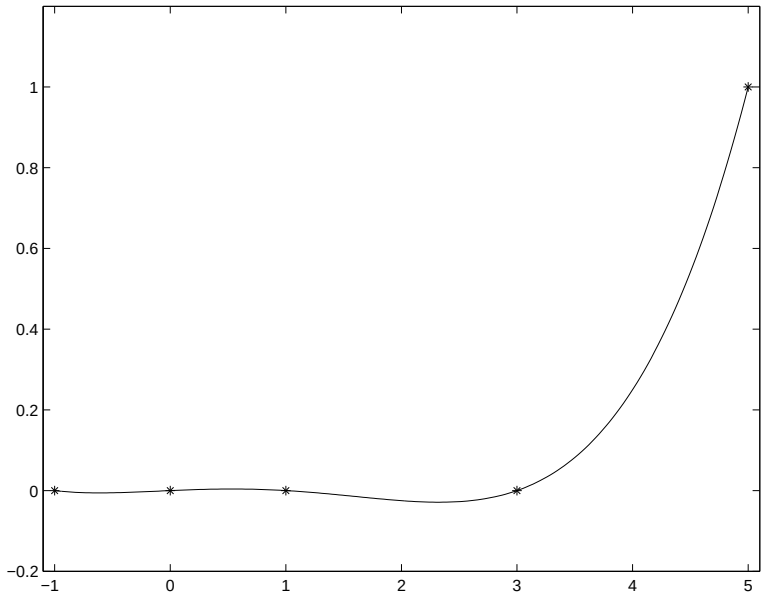


Figura 4.5: Grafico del polinomio $l_{44}(x)$.

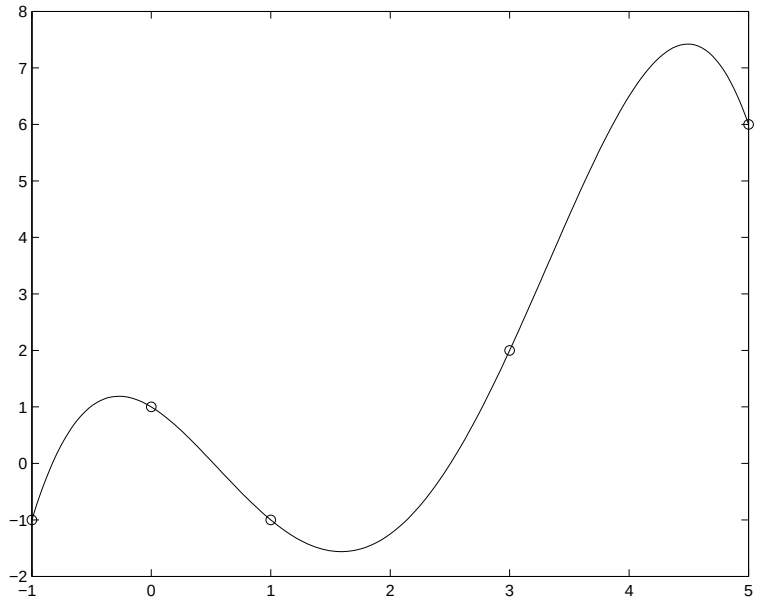


Figura 4.6: Grafico del polinomio interpolante di Lagrange $L_4(x)$.

alla funzione $f(x)$. Questa ipotesi è confutata se si costruisce il polinomio che interpola la funzione

$$f(x) = \frac{1}{1+x^2}$$

nell'intervallo $[-5, 5]$ e prendendo 11 nodi equidistanti $-5, -4, -3, \ldots, 3, 4, 5$. Nella successiva figura viene appunto visualizzata la funzione (in blu) ed il relativo polinomio interpolante (in rosso).
Il polinomio interpolante presenta infatti notevoli oscillazioni, soprattutto verso gli estremi dell'intervallo di interpolazione, che diventano ancora più evidenti all'aumentare di n. Tale fenomeno, detto appunto fenomeno di Runge, è dovuto ad una serie di situazioni concomitanti:

1. il polinomio nodale, al crescere di n, assume un'andamento fortemente oscillante, soprattutto quando i nodi sono equidistanti;

2. alcune funzioni, come quella definita nell'esempio, hanno le derivate il cui valore tende a crescere con un ordine di grandezza talmente elevato da neutralizzare di fatto la presenza del fattoriale al denominatore dell'espressione dell'errore.

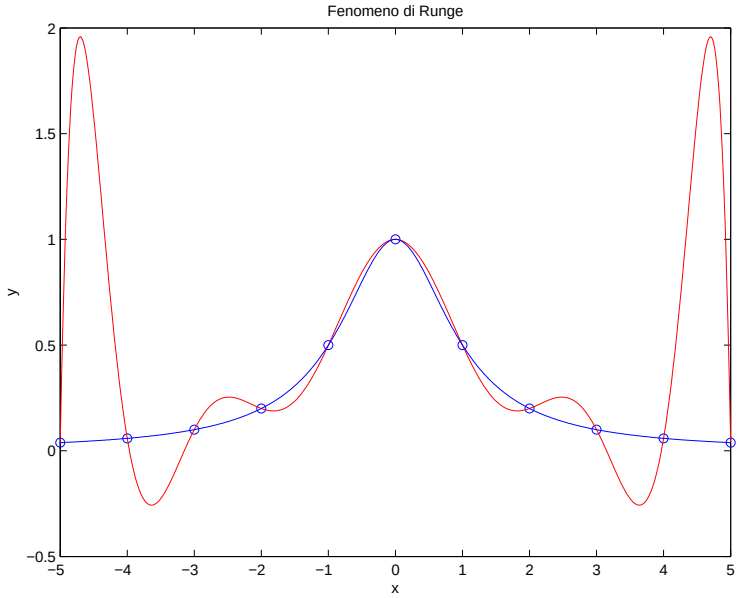


Figura 4.7: Il fenomeno di Runge.

Per ovviare al fenomeno di Runge si possono utilizzare insiemi di nodi non equidistanti oppure utilizzare funzioni interpolanti polinomiali a tratti (interpolando di fatto su intervalli più piccoli e imponendo le condizioni di continuità fino ad un ordine opportuno).

```
function  yy=lagrange(x,y,xx);
%
% La funzione calcola il polinomio interpolante di Lagrange
% in un vettore assegnato di ascisse
%
% Parametri di input
% x = vettore dei nodi
% y = vettore delle ordinate nei nodi
% xx = vettore delle ascisse in cui calcolare il polinomio
% Parametri di output
% yy = vettore delle ordinate del polinomio
%
n = length(x);
m = length(xx);
```

```
yy = zeros(size(xx));
for i=1:m
    yy(i)=0;
    for k=1:n
        yy(i)=yy(i)+prod((xx(i)-x([1:k-1,k+1:n]))./...
            (x(k)-x([1:k-1,k+1:n])))*y(k);
    end
end
return
```

4.3 Formule di Quadratura di Tipo Interpolatorio

Siano assegnati due valori a, b, con $a < b$, ed una funzione f integrabile sull'intervallo (a, b). Il problema che ci poniamo è quello di costruire degli algoritmi numerici che ci permettano di valutare, con errore misurabile, il numero

$$I(f) = \int_a^b f(x)dx.$$

Diversi sono i motivi che possono portare alla richiesta di un algoritmo numerico per questi problemi.
Per esempio pur essendo in grado di calcolare una primitiva della funzione f, questa risulta così complicata da preferire un approccio di tipo numerico. Non è da trascurare poi il fatto che il coinvolgimento di funzioni, elementari e non, nella primitiva e la loro valutazione negli estremi a e b comporta comunque un'approssimazione dei risultati. Un'altra eventualità è che f sia nota solo in un numero finito di punti o comunque può essere valutata in ogni valore dell'argomento solo attraverso una routine. In questi casi l'approccio analitico non è neanche da prendere in considerazione.
Supponiamo dunque di conoscere la funzione $f(x)$ nei punti distinti $x_0, x_1, \ldots,$ x_n prefissati o scelti da noi, ed esaminiamo la costruzione di formule del tipo

$$\sum_{k=0}^{n} w_k f(x_k) \tag{4.9}$$

che approssimi realizzare $I(f)$.
Formule di tipo (4.9) si dicono di quadratura, i numeri reali $x_0, x_1, \ldots, x_n$ e

$w_0, \ldots, w_n$ si chiamano rispettivamente nodi e pesi della formula di quadratura.
Il modo più semplice ed immediato per costruire formule di tipo (4.9) è quello di sostituire la funzione integranda $f(x)$ con il polinomio di Lagrange $L_n(x)$ interpolante $f(x)$ nei nodi x_i, $i = 0, \ldots, n$. Posto infatti

$$f(x) = L_n(x) + e(x)$$

dove $e(x)$ è la funzione errore, abbiamo:

$$\begin{aligned}\int_a^b f(x)dx &= \int_a^b [L_n(x) + e(x)]dx = \int_a^b L_n(x)dx + \int_a^b e(x)dx = \\ &= \int_a^b \sum_{k=0}^{n} l_{nk}(x) f(x_k)dx + \int_a^b e(x)dx = \\ &= \sum_{k=0}^{n} \left(\int_a^b l_{nk}(x)dx \right) f(x_k) + \int_a^b e(x)dx.\end{aligned}$$

Ponendo

$$w_k = \int_a^b l_{nk}(x)dx \qquad k = 0, 1, \ldots, n \tag{4.10}$$

e

$$R_{n+1}(f) = \int_a^b e(x)dx \tag{4.11}$$

otteniamo

$$I(f) \simeq \sum_{k=0}^{n} w_k f(x_k)$$

con un errore stabilito dalla relazione (4.11). Le formule di quadratura con pesi definiti dalle formule (4.10) si dicono interpolatorie. La quantità $R_{n+1}(f)$ prende il nome di Resto della formula di quadratura. Un utile concetto per misurare il grado di accuratezza con cui una formula di quadratura, interpolatoria o meno, approssima un integrale è il seguente.

Definizione 4.3.1 *Una formula di quadratura ha grado di precisione q se fornisce il valore esatto dell'integrale quando la funzione integranda è un qualunque polinomio di grado al più q ed inoltre esiste un polinomio di grado $q+1$ tale che l'errore è diverso da zero.*

È evidente da questa definizione che ogni formula di tipo interpolatorio con nodi $x_0, x_1, \ldots, x_n$ ha grado di precisione almeno n.

4.4 Formule di Newton-Cotes

Suddividiamo l'intervallo $[a, b]$ in n sottointervalli di ampiezza h, con

$$h = \frac{b-a}{n}$$

e definiamo i nodi

$$x_i = a + ih \qquad\qquad i = 0, 1, \ldots, n.$$

La formula di quadratura interpolatoria costruita su tali nodi, cioè

$$\int_a^b f(x)dx = \sum_{i=0}^{n} w_i f(x_i) + R_{n+1}(f)$$

è detta Formula di Newton-Cotes.
Una proprietà di cui godono i pesi delle formule di Newton-Cotes è la cosiddetta proprietà di simmetria. Infatti poichè i nodi sono a due a due simmetrici rispetto al punto medio c dell'intervallo $[a, b]$, cioè $c = (x_i + x_{n-i})/2$, per ogni i, tale proprietà si ripercuote sui pesi che infatti sono a due a due uguali, cioè $w_i = w_{n-i}$, per ogni i. Descriviamo ora due esempi di formule di Newton-Cotes.

4.4.1 Formula dei Trapezi

Siano $x_0 = a$, $x_1 = b$ e $h = b - a$.

$$T_2 = w_0 f(x_0) + w_1 f(x_1)$$

$$w_0 = \int_a^b l_{1,0}(x)dx = \int_a^b \frac{x - x_1}{x_0 - x_1}dx = \int_a^b \frac{x-b}{a-b}dx =$$

$$= \frac{1}{a-b}\left[(x-b)^2\right]_{x=a}^{x=b} = \frac{h}{2}.$$

Poichè i nodi scelti sono simmetrici rispetto al punto medio $c = (a + b)/2$ è

$$w_1 = w_0 = \frac{h}{2}.$$

Otteniamo dunque la formula

$$T_2 = \frac{h}{2}\left[f(a) + f(b)\right].$$

che viene detta Formula dei Trapezi. Per quanto concerne il resto abbiamo

$$R_2(f) = \frac{1}{2}\int_a^b (x-a)(x-b)f''(\xi_x)dx.$$

Prima di vedere come tale espressione può essere manipolata dimostriamo il seguente teorema che è noto come teorema della media generalizzato.

Teorema 4.4.1 *Siano $f, g : [a,b] \to \mathbb{R}$, funzioni continue con $g(x)$ a segno costante e $g(x) \neq 0$ per ogni $x \in]a,b[$. Allora*

$$\int_a^b f(x)g(x)dx = f(\xi)\int_a^b g(x)dx \qquad \xi \in [a,b]. \ \square$$

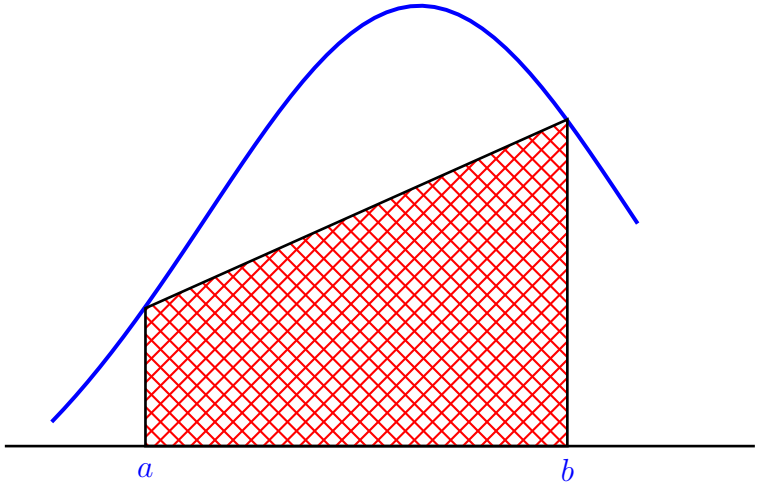
Poichè la funzione $(x-a)(x-b)$ è a segno costante segue:

$$R_2(f) = \frac{1}{2}f''(\eta)\int_a^b (x-a)(x-b)dx$$

posto $x = a + ht$ otteniamo

$$R_2(f) = \frac{1}{2}f''(\eta)h^3\int_0^1 t(t-1)dt = -\frac{1}{12}h^3 f''(\eta).$$

L'interpretazione geometrica della formula del trapezio è riassunta nella seguente figura, l'area tratteggiata (ovvero l'integrale della funzione viene approssimato attraverso l'area del trapezio che ha come basi i valori della funzione in a e b e come altezza l'intervallo $[a,b]$).



4.4.2 Formula di Simpson

Siano $x_0 = a$, $x_2 = b$ mentre poniamo $x_1 = c$, punto medio dell'intervallo $[a, b]$. Allora

$$S_3 = w_0 f(a) + w_1 f(c) + w_2 f(b).$$

Posto

$$h = \frac{b-a}{2}$$

abbiamo

$$w_0 = \int_a^b l_{2,0}(x)dx = \int_a^b \frac{(x-c)(x-b)}{(a-c)(a-b)}dx.$$

Effettuando il cambio di variabile $x = c + ht$ è facile calcolare quest'ultimo integrale, infatti

$$x = a \;\Rightarrow\; a = c + ht \;\Rightarrow\; a - c = ht \;\Rightarrow\; -h = ht \;\Rightarrow\; t = -1$$

e

$$x = b \;\Rightarrow\; b = c + ht \;\Rightarrow\; b - c = ht \;\Rightarrow\; h = ht \;\Rightarrow\; t = 1.$$

Inoltre $a - c = -h$ e $a - b = -2h$ mentre

$$x-c = c+ht-c = ht, \qquad x-b = c+ht-b = c-b+ht = -h+ht = h(t-1),$$

ed il differenziale $dx = hdt$ cosicchè

$$w_0 = \int_a^b \frac{(x-c)(x-b)}{(a-c)(a-b)}dx = \int_{-1}^1 \frac{hth(t-1)}{(-h)(-2h)}hdt =$$

$$= \frac{h}{2}\int_{-1}^1 (t^2-t)dt = \frac{h}{2}\int_{-1}^1 t^2 dt = \frac{h}{2}\left[\frac{t^3}{3}\right]_{-1}^1 = \frac{h}{3}.$$

Per la simmetria è anche

$$w_2 = w_0 = \frac{h}{3}$$

mentre possiamo calcolare w_1 senza ricorrere alla definizione. Infatti possiamo notare che la formula deve fornire il valore esatto dell'integrale quando la funzione è costante nell'intervallo $[a, b]$, quindi possiamo imporre che, prendendo $f(x) = 1$ in $[a, b]$, sia

$$\int_a^b dx = b - a = \frac{h}{3}(f(a) + f(b)) + w_1 f(c)$$

da cui segue

$$w_1 = b - a - \frac{2}{3}h = 2h - \frac{2}{3}h = \frac{4}{3}h.$$

Dunque

$$S_3 = \frac{h}{3}\left[f(a) + 4f(c) + f(b)\right].$$

Questa formula prende il nome di Formula di Simpson. Per quanto riguarda l'errore si può dimostrare, e qui ne omettiamo la prova, che vale la seguente relazione

$$R_3(f) = -h^5 \frac{f^{(4)}(\sigma)}{90} \qquad \eta, \sigma \in (a, b),$$

che assicura che la formula ha grado di precisione 3.

4.5 Formule di Quadratura Composte

Come abbiamo già avuto modo di vedere le formule di quadratura interpolatorie vengono costruite approssimando su tutto l'intervallo di integrazione la funzione integranda con un unico polinomio, quello interpolante la funzione sui nodi scelti. Per formule convergenti la precisione desiderata si ottiene

prendendo n sufficientemente grande. In tal modo comunque, per ogni fissato n, bisogna costruire la corrispondente formula di quadratura. Una strategia alternativa che ha il pregio di evitare la costruzione di una nuova formula di quadratura, e che spesso produce risultati più apprezzabili, è quella delle formule composte. Infatti scelta una formula di quadratura l'intervallo di integrazione (a, b) viene suddiviso in N sottointervalli di ampiezza h,

$$h = \frac{b-a}{N} \tag{4.12}$$

sicchè

$$\int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx$$

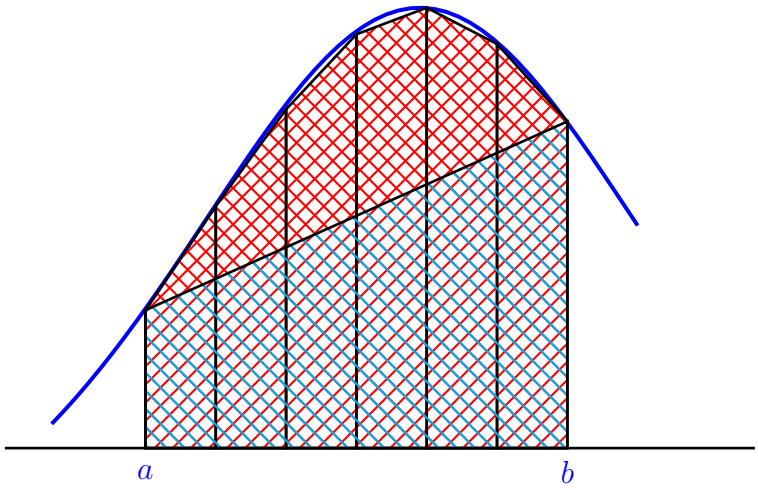
dove i punti x_i sono:

$$x_i \quad = a + ih \qquad i = 0, \ldots, N \tag{4.13}$$

quindi la formula di quadratura viene applicata ad ognuno degli intervalli $[x_i, x_{i+1}]$. Il grado di precisione della formula di quadratura composta coincide con il grado di precisione della formula da cui deriva. Descriviamo ora la Formula dei Trapezi Composta.

4.5.1 Formula dei Trapezi Composta

Per quanto visto in precedenza suddividiamo l'intervallo $[a, b]$ in N sottointervalli, ognuno di ampiezza data da h, come in (4.12), e con i nodi x_i definiti in (4.13). Applichiamo quindi in ciascuno degli N intervalli $[x_i, x_{i+1}]$ la formula dei trapezi. Nella seguente figura sono evidenziate le aree che approssimano l'integrale utilizzando la formula dei trapezi semplice e quella composta.



Applicando la formula dei trapezi a ciascun sottointervallo si ottiene

$$\int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx = \sum_{i=0}^{N-1} \left[\frac{h}{2} \left(f(x_i) + f(x_{i+1}) \right) - \frac{1}{12} h^3 f''(\eta_i) \right]$$

con $\eta_i \in (x_i, x_{i+1})$. Scrivendo diversamente la stessa espressione

$$\begin{aligned} \int_a^b f(x)dx &= \frac{h}{2} \left(f(x_0) + f(x_N) \right) + h \sum_{i=1}^{N-1} f(x_i) - \frac{1}{12} h^3 \sum_{i=0}^{N-1} f''(\eta_i) = \\ &= \frac{h}{2} \left(f(x_0) + f(x_N) \right) + h \sum_{i=1}^{N-1} f(x_i) - \frac{1}{12} h^3 N f''(\eta) \end{aligned}$$

dove $\eta \in (a, b)$. L'esistenza di tale punto η è garantito dal cosiddetto Teorema della media nel discreto applicato a $f''(x)$, che stabilisce che se $g(x)$ è una funzione continua in un intervallo $[a, b]$ e $\eta_i \in [a, b]$ $i = 1, N$, sono N punti distinti, allora esiste un punto $\eta \in (a, b)$ tale che

$$\sum_{i=1}^{N} g(\eta_i) = N g(\eta).$$

Dunque la formula dei trapezi composta è data da:

$$T_C(h) = \frac{h}{2} \left(f(x_0) + f(x_N) \right) + h \sum_{i=1}^{N-1} f(x_i)$$

con resto

$$R_T = -\frac{1}{12}h^3 N f''(\eta) = -\frac{1}{12}\frac{(b-a)^3}{N^3}Nf''(\eta) = -\frac{1}{12}\frac{(b-a)^3}{N^2}f''(\eta).$$

Quest'ultima formula talvolta può essere utile per ottenere a priori una suddivisione dell'intervallo $[a, b]$ in un numero di intervalli che permetta un errore non superiore ad una prefissata tolleranza. Infatti

$$|R_T| \leq \frac{1}{12}\frac{(b-a)^3}{N^2}M, \qquad M = \max_{x\in[a,b]} |f''(x)|.$$

Imponendo che $|R_T| \leq \varepsilon$, precisione prefissata, segue

$$N_\varepsilon \geq \sqrt{\frac{(b-a)^3 M}{12\varepsilon}}. \tag{4.14}$$

Tuttavia questo numero spesso risulta una stima eccessiva a causa della maggiorazione della derivata seconda tramite M.

Capitolo 5

Metodi numerici per equazioni differenziali ordinarie

5.1 Introduzione

Supponiamo che sia assegnato il seguente problema differenziale:

$$\begin{cases} y'(t) = f(t, y(t)) \\ y(t_0) = y_0 \end{cases} \tag{5.1}$$

dove $f : [t_0, T] \times \mathbb{R} \to \mathbb{R}$ è una funzione continua rispetto a t e Lipschitziana rispetto a y, cioè esiste una costante positiva L tale che, per ogni $x, y \in \mathbb{R}$, risulta

$$|f(t, x) - f(t, y)| \leq L|x - y|, \qquad \forall t \in [t_0, T].$$

Il problema (5.1) prende il nome di problema di Cauchy del primo ordine ai valori iniziali. Risolvere (5.1) significa determinare una funzione $y(t)$ di classe $\mathcal{C}^1([t_0, T])$ la cui derivata prima soddisfi l'equazione assegnata e che passi per il punto (t_0, y_0). In base alle ipotesi fatte sulla funzione $f(t, y(t))$ il teorema di Cauchy assicura l'esistenza e l'unicità di tale funzione.

Teorema 5.1.1 (di Cauchy) *Sia $f(t, y) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, una funzione definita e continua per ogni (t, y) appartenente alla regione $[t_0, T] \times \mathbb{R}$, e sia inoltre Lipschitziana rispetto a y allora per ogni condizione iniziale esiste un'unica soluzione continua e differenziabile $y(t)$ del problema (5.1).*

L'equazione (5.1) dipende solo dalla derivata prima della soluzione, mentre si possono avere anche problemi di ordine superiore del tipo:

$$y^{(m)}(t) = f(t, y, y', y'', \ldots, y^{(m-1)}(t)).$$

che tuttavia possono essere ricondotti ad un sistema differenziale del primo ordine, pertanto in questo capitolo saranno considerati metodi numerici per problemi del primo ordine, che possono essere sia di tipo scalare (una singola equazione) che vettoriale (un sistema di equazioni). Purtroppo solo un numero piuttosto basso di equazioni differenziali ordinarie ammette soluzione in forma esplicita. In alcuni casi può essere espressa in forma implicita, ovvero attraverso un'equazione del tipo $G(t, y(t)) = 0$, mentre in altri casi solo in forma di serie numerica. I metodi numerici che saranno descritti in seguito sono in grado di fornire una soluzione approssimata per ogni tipo di equazione differenziale che ammette soluzione. Ovviamente non forniscono la soluzione in forma chiusa. Punto di partenza per tali metodi è quello di dividere l'intervallo di integrazione $[t_0, T]$ in N sottointervalli ciascuno di ampiezza $h = (T - t_0)/N$, dove h è detto passo di discretizzazione. Quindi si definiscono i punti

$$t_n = t_{n-1} + h = t_0 + nh, \qquad n = 0, \ldots, N$$

in ognuno dei quali si cercherà di trovare l'approssimazione y_n tale che

$$y_n \simeq y(t_n), \qquad n = 0, \ldots, N.$$

Prima di studiare le principali classi di metodi numerici per equazioni differenziali vediamo prima di descriverne alcuni piuttosto semplici. descriverne alcuni piuttosto semplici. Partendo dal problema di Cauchy

$$y'(t) = f(t, y(t)) \tag{5.2}$$

e supponendo di voler calcolare la funzione in t_{n+1} noto il suo valore in t_n, andiamo ad integrare membro a membro (5.2):

$$\int_{t_n}^{t_{n+1}} y'(t)dt = \int_{t_n}^{t_{n+1}} f(t, y(t))dt \tag{5.3}$$

cioè

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))dt \tag{5.4}$$

quindi il problema equivale ad approssimare l'integrale a secondo membro. Un primo modo è quello di approssimare tale integrale con l'area del rettangolo avente come base l'intervallo $[t_n, t_{n+1}]$ e come altezza il valore assunto dalla funzione integranda in t_n, quindi

$$y(t_{n+1}) - y(t_n) = hf(t_n, y(t_n)) + Errore.$$

Trascurando il termine di errore si ottiene l'uguaglianza approssimata

$$y(t_{n+1}) - y(t_n) \simeq hf(t_n, y(t_n))$$

da cui, definendo le approssimazioni $y_n \simeq y(t_n)$ e $y_{n+1} \simeq y(t_{n+1})$ si ottiene la seguente uguaglianza tra quantità approssimate:

$$y_{n+1} - y_n = hf(t_n, y_n) \quad \Leftrightarrow \quad y_{n+1} = y_n + hf(t_n, y_n).$$

Tale metodo va sotto il nome di Metodo di Eulero Esplicito in quanto consente, noto y_n, di calcolare esplicitamente l'approssimazione nel punto successivo. In modo analogo si può approssimare l'integrale in (5.3) prendendo come punto di riferimento t_{n+1} ottenendo quindi:

$$y(t_{n+1}) - y(t_n) \simeq hf(t_{n+1}, y(t_{n+1}))$$

che fornisce il cosiddetto Metodo di Eulero Implicito:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}).$$

Un ulteriore formula per approssimare l'integrale in (5.3) è quello di usare l'area del trapezio avente come basi il valore della funzione $f(t, y)$ calcolato negli estremi dell'intervallo e come altezza lo stesso intervallo:

$$y(t_{n+1}) - y(t_n) \simeq \frac{h}{2} \left[f(t_{n+1}, y(t_{n+1})) + f(t_n, y(t_n)) \right]$$

che dà luogo al cosiddetto Metodo dei Trapezi:

$$y_{n+1} = y_n + \frac{h}{2} \left[f(t_{n+1}, y_{n+1}) + f(t_n, y_n) \right].$$

I due metodi appena descritti sono di tipo implicito, cioè l'approssimazione y_{n+1} dipende dal valore assunto dalla funzione $f(t, y)$ nell'incognita y_{n+1}. In questo caso è spesso necessario risolvere un'equazione non lineare (o un sistema di equazioni), cosa che, abbiamo visto, può essere fatta numericamente. Un ultimo metodo, applicabile all'intervallo $[t_n, t_{n+2}]$, consiste nell'approssimare l'integrale a secondo membro nell'equazione

$$y(t_{n+2}) - y(t_n) = \int_{t_n}^{t_{n+2}} f(t, y(t))dt$$

con l'area del rettangolo avente come base l'intervallo $[t_n, t_{n+2}]$ e come altezza il valore assunto dalla funzione nel punto medio dello stesso intervallo:

$$y(t_{n+2}) - y(t_n) \simeq 2hf(t_{n+1}, y(t_{n+1}))$$

che fornisce il Metodo del Midpoint Esplicito:

$$y_{n+2} = y_n + 2hf(t_{n+1}, y_{n+1}).$$

5.2 I metodi multistep lineari

L'espressione generale di un metodo multistep lineare è la seguente

$$\sum_{j=0}^{k} \alpha_j y_{n+j} - h \sum_{j=0}^{k} \beta_j f_{n+j} = 0, \tag{5.5}$$

dove $f_{n+j} = f(t_{n+j}, y_{n+j})$. Il valore intero k indica il numero di passi del metodo ed indica il numero di approssimazioni necessarie per poter calcolare un valore y_n. Infatti un metodo multistep lineare funziona in questo modo. Inizialmente si suppone di conoscere le quantità $y_0, y_1, \ldots, y_{k-1}$ e, scrivendo l'espressione (5.5), per $n = 0$ si ricava:

$$\sum_{j=0}^{k} \alpha_j y_j - h \sum_{j=0}^{k} \beta_j f_j = 0,$$

da cui è possibile ricavare (o esplicitamente o attraverso un metodo numerico, come vedremo in seguito) l'unica incognita y_k. Una volta calcolata tale approssimazione si scrive l'equazione (5.5), per $n = 1$:

$$\sum_{j=0}^{k} \alpha_j y_{j+1} - h \sum_{j=0}^{k} \beta_j f_{j+1} = 0,$$

e si ricava l'unica incognita y_{k+1} e, in questo modo, applicando ripetutamente il metodo si calcolano tutte le approssimazioni fino all'ultima y_N. Da quanto detto tuttavia emergono due questioni aperte:

1. Inizialmente si conosce solo il valore y_0 mentre sono necessari anche i valori $y_1, \ldots, y_{k-1}$;

2. Se $\beta_k \neq 0$ e la funzione $f(t, y)$ è non lineare rispetto a y allora ad ogni passo non è possibile calcolare esplicitamente il valore y_{n+k}.

Vedremo in seguito come affrontare queste due problematiche. Adesso facciamo alcune considerazioni preliminari sulle condizioni da imporre ai coefficienti del metodo α_j e β_j.
Osservando che l'espressione di un metodo multistep lineari rimane inalterata se viene moltiplicata per una costante non nulla allora per normalizzare i coefficienti si impone che sia $\alpha_k = 1$. Un altro vincolo sui coefficienti è che sia

$$|\alpha_0| + |\beta_0| \neq 0$$

in modo tale che siano certamente coinvolti nella formula y_n e y_{n+k}. Il motivo di tale scelta è quello di non considerare metodi del tipo

$$y_{n+2} - y_{n+1} - hf_{n+1} = 0$$

che è essenzialmente un metodo ad un passo e non a 2 passi ed è sostanzialmente indistingubile dal metodo ad un passo:

$$y_{n+1} - y_n - hf_n = 0.$$

Prima di vedere quali sono le altre condizioni che i coefficienti del metodo devono soddisfare definiamo i due seguenti polinomi:

$$\rho(z) = \sum_{j=0}^{k} \alpha_j z^j$$

detto primo polinomio caratteristico del metodo, e

$$\sigma(z) = \sum_{j=0}^{k} \beta_j z^j$$

detto secondo polinomio caratteristico del metodo.
Una prima classificazione tra i metodi multistep lineari è quella in base al valore assunto dal coefficiente β_k. Infatti se in (5.5) $\beta_k = 0$ il metodo si dice esplicito, altrimenti si dice implicito. Nel primo caso il metodo si può riscrivere come:

$$y_{n+k} = h \sum_{j=0}^{k-1} \beta_j f_{n+j} - \sum_{j=0}^{k-1} \alpha_j y_{n+j},$$

mettendo in evidenza che il valore y_{n+k} può essere calcolato esplicitamente utilizzando solo valori già noti. Invece se $\beta_k \neq 0$ e la funzione $f(t, y)$ è non lineare rispetto alla variabile y allora ad ogni passo il metodo può essere scritto come:

$$y_{n+k} = h\beta_k f_{n+k} + h\sum_{j=0}^{k-1} \beta_j f_{n+j} - \sum_{j=0}^{k-1} \alpha_j y_{n+j},$$

cosicchè per poter calcolare y_{n+k} è necessario risolvere l'equazione non lineare

$$y_{n+k} = h\beta_k f(t_{n+k}, y_{n+k}) + G_{n,k} \tag{5.6}$$

avendo posto

$$G_{n,k} = h\sum_{j=0}^{k-1} \beta_j f_{n+j} - \sum_{j=0}^{k-1} \alpha_j y_{n+j},$$

che è una quantità nota. L'equazione (5.6) è anche un'equazione di punto fisso

$$x = \Phi(x)$$

dove

$$\Phi(x) = h\beta_k f(t_{n+k}, x) + G_{n,k}.$$

In questo caso è naturale definire il metodo di iterazione funzionale:

$$y_{n+k}^{[m+1]} = \Phi\left(y_{n+k}^{[m]}\right)$$

prendendo come approssimazione iniziale il valore dell'approssimazione più vicina a quella incognita, cioè:

$$y_{n+k}^{[0]} = y_{n+k-1}.$$

Il metodo iterativo è quindi

$$\begin{aligned} y_{n+k}^{[0]} &= y_{n+k-1} \\ y_{n+k}^{[m+1]} &= h\beta_k f(t_{n+k}, y_{n+k}^{[m]}) + G_{n,k}. \end{aligned} \tag{5.7}$$

Affinchè tale metodo converga deve essere

$$|\Phi'(x)| < 1$$

in un opportuno intorno della soluzione. Calcolando la derivata prima della funzione $\Phi(x)$ si ha:

$$\Phi'(x) = h\beta_k \frac{\partial f}{\partial y}(t_{n+k}, x)$$

quindi la condizione per la convergenza diviene:

$$h|\beta_k| \left|\frac{\partial f}{\partial y}\right| < 1 \quad \Rightarrow \quad \left|\frac{\partial f}{\partial y}\right| < \frac{1}{h|\beta_k|}.$$

Tale condizione è sicuramente verificata se risulta $h|\beta_k|L < 1$, con L uguale alla costante di Lipschitz della funzione $f(t, y)$, infatti per definizione

$$\left|\frac{\partial f}{\partial y}\right| \leq L$$

quindi

$$h|\beta_k| \left|\frac{\partial f}{\partial y}\right| \leq h|\beta_k|L$$

ed il metodo è convergente se

$$h|\beta_k|L < 1. \tag{5.8}$$

Tale condizione non presenta eccessivi problemi per la gran parte delle equazioni differenziali, se la (5.8) non è rispettata è sufficiente scegliere un valore h più piccolo, in modo tale che essa sia soddisfatta, e procedere. Un caso particolare sono i cosiddetti problemi stiff in cui la costante $L \gg 1$ ed in questo caso la (5.8) rappresenta una restrizione molto forte sul passo h. Un ultimo aspetto da analizzare è il fatto che, per applicare il metodo al primo passo (cioè quando $n = 0$) si devono conoscere le approssimazioni $y_1, y_2, \ldots, y_{k-1}$ per poter calcolare y_k. Un modo per ottenerle è quello di applicare un metodo che richieda un numero di passi inferiore (per esempio metodi ad un passo come Eulero esplicito o implicito) e di applicare il metodo multistep solo quando si abbiano tutte le approssimazioni necessarie. Le proprietà che devono soddisfare i coefficienti del metodo sono la Consistenza, la Zero Stabilità e la Convergenza.

Convergenza

Quando il passo di integrazione h tende a zero l'insieme di punti discreti $\{t_n\}$ diventa l'intero intervallo $[t_0, T]$. Una proprietà ovvia da richiedere ad

un qualsiasi metodo numerico è che, quando $h \to 0$ la soluzione numerica y_n diventa la soluzione teorica $y(t)$, $t \in [t_0, T]$. Questa proprietà è detta Convergenza.

Definizione 5.2.1 *Un metodo numerico si dice convergente se, per ogni problemi ai valori iniziali soddisfacente le ipotesi si ha:*

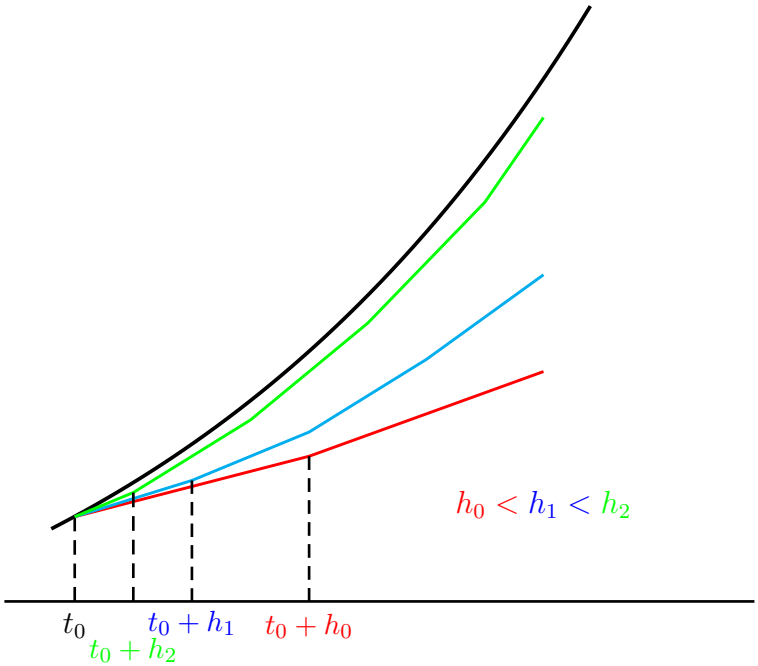
$$\lim_{\substack{h\to 0 \\ t=t_0+nh}} y_n = y(t)$$

per ogni $t \in [t_0, T]$. *Un metodo che non è convergente si dice divergente.*

Tale definizione necessita di alcuni chiarimenti. Consideriamo infatti un punto t della discretizzazione (cioè tale che $t = t_n = t_0 + nh$), un metodo convergente deve essere tale che la soluzione numerica y_n nel punto della discretizzazione $t = t_n$ tende a quella teorica $y(t)$ quando $h \to 0$. La definizione puntualizza l'esigenza che, anche se h tende a zero (e quindi $n \to \infty$), la quantità nh si mantiene costante all'ampiezza dell'intervallo $[t_0, t]$. Una definizione alternativa di convergenza richiede che

$$\lim_{h\to 0} \max_{0\le n\le N} |y(t_n) - y_n| = 0$$

quando il metodo numerico viene applicato ad un qualsiasi probema ai valori iniziali che soddisfa le ipotesi del Teorema 5.1.1.



Adesso l'obiettivo diventa quello di stabilire quali sono le condizioni che consentano di stabilire la convergenza del metodo. Un primo concetto è quello di consistenza.

Consistenza

Sia $z(t) \in \mathcal{C}^1([t_0, T])$ e definiamo il seguente operatore differenza lineare:

$$\mathcal{L}[z(t); h] = \sum_{j=0}^{k} \left[\alpha_j z(t + jh) - h\beta_j z'(t + jh)\right].$$

Se calcoliamo tale operatore nella soluzione del problema di Cauchy assegnato in un punto della discretizzazione t_n otteniamo un'espressione del residuo, detto anche errore locale di troncamento:

$$\begin{aligned} \tau_{n+k}(h) &= \sum_{j=0}^{k} \left[\alpha_j y(t_{n+j}) - h\beta_j y'(t_{n+j})\right] = \\ &= \sum_{j=0}^{k} \left[\alpha_j y(t_{n+j}) - h\beta_j f(t_{n+j}, y(t_{n+j}))\right]. \end{aligned}$$

Se le approssimazioni y_n coincidessero con la soluzione teorica allora il residuo sarebbe nullo pertanto possiamo prendere tale quantità come possibile misura della qualità dell'approssimazione fornita dal metodo. Una condizione ovvia che il residuo deve soddisfare è che

$$\lim_{h \to 0} \tau_{n+k}(h) = 0$$

ma questo implicherebbe una condizione solo sui coefficienti α_j, pertanto si preferisce la seguente definizione.

Definizione 5.2.2 *Un metodo numerico si dice consistente (con il problema differenziale) se*

$$\lim_{h \to 0} \frac{\tau_{n+k}(h)}{h} = 0 \tag{5.9}$$

Per esplicitare le condizioni per la consistenza di un metodo cerchiamo un'espressione alternativa del residuo. Sostituiamo infatti al posto di $y(t_{n+j})$ e

di $y'(t_{n+j})$ gli sviluppi in serie di Taylor (ipotizzando ovviamente che $y(t)$ sia una funzione di classe $\mathcal{C}^\infty([t_0, T])$):

$$\begin{aligned} y(t_{n+j}) &= y(t_n + jh) = \sum_{i=0}^{\infty} \frac{(jh)^i}{i!} y^{(i)}(t_n), \\ y'(t_{n+j}) &= y'(t_n + jh) = \sum_{i=0}^{\infty} \frac{(jh)^i}{i!} y^{(i+1)}(t_n). \end{aligned}$$

Riscriviamo le due espansioni in serie in modo tale da uniformare gli indici delle sommatorie e l'ordine delle derivate:

$$\begin{aligned} y(t_{n+j}) &= y(t_n) + \sum_{i=1}^{\infty} \frac{(jh)^i}{i!} y^{(i)}(t_n), \\ y'(t_{n+j}) &= \sum_{i=1}^{\infty} \frac{(jh)^{i-1}}{(i-1)!} y^{(i)}(t_n) \end{aligned}$$

e andiamo a sostituire tali serie nell'espressione del residuo:

$$\begin{aligned} \tau_{n+k}(h) &= \sum_{j=0}^{k} [\alpha_j y(t_{n+j}) - h\beta_j y'(t_{n+j})] = \\ &= \sum_{j=0}^{k} \left[\alpha_j \left(y(t_n) + \sum_{i=1}^{\infty} \frac{(jh)^i}{i!} y^{(i)}(t_n) \right) - h\beta_j \sum_{i=1}^{\infty} \frac{(jh)^{i-1}}{(i-1)!} y^{(i+1)}(t_n) \right] = \\ &= y(t_n) \sum_{j=0}^{k} \alpha_j + \sum_{i=1}^{\infty} \left[\sum_{j=0}^{k} \left(\alpha_j \frac{(jh)^i}{i!} - h\beta_j \frac{(jh)^{i-1}}{(i-1)!} \right) \right] y^{(i)}(t_n) = \\ &= y(t_n) \sum_{j=0}^{k} \alpha_j + \sum_{i=1}^{\infty} \left[\sum_{j=0}^{k} \left(\alpha_j \frac{j^i}{i!} - \beta_j \frac{j^{i-1}}{(i-1)!} \right) \right] h^i y^{(i)}(t_n). \end{aligned}$$

Ponendo

$$C_0 = \sum_{j=0}^{k} \alpha_j, \qquad C_i = \sum_{j=0}^{k} \left(\alpha_j \frac{j^i}{i!} - \beta_j \frac{j^{i-1}}{(i-1)!} \right)$$

si ha che il residuo $\tau_{n+k}(h)$ ammette uno sviluppo asintotico del tipo:

$$\tau_{n+k}(h) = C_0 y(t_n) + C_1 h y'(t_n) + C_2 h^2 y''(t_n) + C_3 h^3 y'''(t_n) + \dots.$$

Da tale proprietà segue che il metodo multistep può essere consistente solo se i primi due coefficienti dello sviluppo, C_0 e C_1, sono nulli, cioè:

$$C_0 = \sum_{j=0}^{k} \alpha_j = 0$$

$$C_1 = \sum_{j=0}^{k} (\alpha_j j - \beta_j) = 0.$$

In termini di coefficienti dei polinomi caratteristici

$$C_0 = \sum_{j=0}^{k} \alpha_j = \rho(1) = 0$$

$$C_1 = \sum_{j=0}^{k} (\alpha_j j - \beta_j) = \rho'(1) - \sigma(1) = 0$$

Riassumendo abbiamo il seguente risultato:

Teorema 5.2.1 *Un metodo multistep lineare avente $\rho(z)$ e $\sigma(z)$ come primo e secondo polinomi caratteristici, rispettivemente, è consistente se e solo se:*

$$\rho(1) = 0, \qquad \rho'(1) - \sigma(1) = 0.$$

Lo sviluppo in serie del residuo (o dell'errore locale di troncamente) induce un'altra considerazione. Poichè esso può essere considerato come una misura dell'errore commesso e di solito il passo di integrazione h è una quantità relativamente piccola, possiamo ipotizzare che i metodi che forniscono risultati più accurati (a parità di passo) sono quelli il cui residuo dipende da potenze di h di ordine più grande. Tale considerazione può essere formalizzata nella seguente definizione.

Definizione 5.2.3 *Un metodo multistep lineare si dice di ordine p se risulta $C_0 = C_1 = \dots = C_p = 0$ e $C_{p+1} \neq 0$. La costante C_{p+1} viene detta costante dell'errore.*

Se un metodo multistep ha ordine p allora

$$\tau_{n+k}(h) = C_{p+1}h^{p+1}y^{(p+1)}(t_n) + C_{p+2}h^{p+2}y^{(p+2)}(t_n) + \dots$$

ovvero si scrive

$$\tau_{n+k}(h) = O(h^{p+1})$$

quindi dal punto di vista teorico i metodi che dovrebbero fornire un errore più piccolo sono quelli che hanno con ordine più elevato.
Osserviamo che le condizioni per la consistenza di un metodo multistep lineare sono equivalenti alla richiesta che il metodo numerico abbia almeno ordine $p = 1$. Abbiamo visto che la convergenza è un processo che avviene al limite, quando $h \simeq 0$, quindi può essere interessante studiare la soluzione numerica y_n quando $h = 0$. Così facendo il metodo diventa

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = 0. \tag{5.10}$$

Tale equazione prende il nome di equazione alle differenze lineare a coefficienti costanti e può essere considerata come l'analogo discreto delle equazioni differenziali a coefficienti costanti. Per risolvere (5.10) si procede esattamente come nel caso delle analoghe equazioni differenziali a coefficienti costanti, cioè si cerca una soluzione di tipo esponenziale. In questo caso specifico si cerca una soluzione del tipo

$$y_n = \xi^n$$

con $\xi \in \mathbb{C}$. Imponendo che tale y_n sia soluzione di (5.10) segue

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = \sum_{j=0}^{k} \alpha_j \xi^{n+j} = \xi^n \sum_{j=0}^{k} \alpha_j \xi^j = \xi^n \rho(\xi).$$

Quindi $y_n = \xi^n$ è soluzione di (5.10) solo se ξ è uno zero del polinomio $\rho(z)$. Se questo ammette k radici distinte ognuna dà luogo ad una soluzione dell'equazione (5.10) pertanto, proprio in analogia con quanto accade nel caso delle equazioni differenziali, dette $\xi_1, \dots \xi_k$ tali radici, la soluzione generale della (5.10) è:

$$y_n = \sum_{j=1}^{k} c_j \xi_j^n$$

dove le c_j sono delle costanti che dipendono dalle condizioni iniziali imposte su y_n. Nel caso in cui una delle radici ξ abbia molteplicità doppia, cioè

$$\rho(\xi) = \rho'(\xi) = 0$$

allora si può provare che $y_n = n\xi^n$ è soluzione di (5.10). Infatti:

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = \sum_{j=0}^{k} \alpha_j (n+j)\xi^{n+j} = \xi^n \left[n \sum_{j=0}^{k} \alpha_j \xi^j + \sum_{j=0}^{k} \alpha_j j \xi^j \right] =$$

$$= \xi^n \left[n\rho(\xi) + \sum_{j=1}^{k} \alpha_j j \xi^j \right] = \xi^n \left[n\rho(\xi) + \xi \sum_{j=1}^{k} \alpha_j j \xi^{j-1} \right] =$$

$$= \xi^n \left[n\rho(\xi) + \xi\rho'(\xi) \right] = 0.$$

Consideriamo ora che, posto $h = 0$, è come se y_n fosse la soluzione numerica del problema di Cauchy

$$y'(t) = 0, \qquad y(t_0) = y_0$$

che ammette come soluzione $y(t) = y_0$ per $t \geq t_0$. Poichè la soluzione teorica è limitata è naturale richiedere che anche la soluzione numerica lo sia. In base al discorso fatto sulle radici del polinomio $\rho(z)$ valgono le seguenti considerazioni:

1. Il valore $z = 1$ è radice del polinomio $\rho(z)$;

2. Se tutte le radici ξ_j di $\rho(z)$ sono distinte e tali che

   $$|\xi_j| \leq 1, \qquad j = 1, \ldots, k$$

   allora la soluzione numerica y_n si mantiene limitata (come quella teorica). Tale situazione resta invariata se $\rho(z)$ ha una radice multipla ξ_j di modulo minore di 1, infatti in questo caso $n\xi_j$ è soluzione dell'equazione alle differenza 5.10 ma

   $$\lim_{n\to\infty} n\xi_j^n = 0.$$

3. Se invece $\rho(z)$ ha una radice ξ_j di modulo maggiore di 1 oppure di modulo 1 ma di molteplicità maggiore di 1 allora la soluzione y_n non può mantenersi limitata.

Riassumiamo quanto appena detto nella seguente definizione.

Definizione 5.2.4 *Un polinomio $\rho(z)$, di grado k, soddisfa il Criterio delle radici se le sue radici $\xi_1, \ldots, \xi_k$ sono tali che*

$$|\xi_i| \leq 1, \qquad i = 1, \ldots, k$$

e ogni radice di modulo 1 è semplice.

Se il primo polinomio caratteristico $\rho(z)$ associato ad un metodo multistep lineare soddisfa il criterio delle radici allora il metodo numerico si dice Zero-stabile, intendendo in questo modo che la soluzione numerica di una particolare equazione differenziale che ammette soluzione teorica limitata è a sua volta limitata. Le condizioni di Consistenza e Zero-stabilità di un metodo multistep lineare possono essere utilizzate per dimostrare il seguente risultato che enunciamo senza dimostrare.

Teorema 5.2.2 *Un metodo multistep lineare è convergente se e solo se è consistente e zero-stabile.*

Avendo già fatto alcune considerazioni su consistenza e zero-stabilità possiamo enunciare il precedente risultato in modo equivalente:

Teorema 5.2.3 *Un metodo multistep lineare (5.5) è convergente se e solo se:*

1. $\rho(1) = 0$;
2. $\rho'(1) - \sigma(1) = 0$;
3. *il polinomio $\rho(z)$ soddisfa la condizione delle radici.*

Concludiamo questo paragrafo sui metodi multistep lineari descrivendo in breve le classi di metodi più utilizzate.

5.2.1 Alcune classi di metodi multistep lineari

La classe più famosa di metodi multistep sono i Metodi di Adams, che hanno come primo polinomio caratteristico:

$$\rho(z) = z^k - z^{k-1}.$$

Tale polinomio soddisfa ovviamente il criterio delle radici perchè i suoi zeri sono $z = 1$ (radice semplice) e $z = 0$ (radice di molteplicità $k - 1$). Un metodo di Adams ha la forma:

$$y_{n+k} - y_{n+k-1} - h \sum_{j=0}^{k} \beta_j f_{n+j} = 0. \tag{5.11}$$

I metodi di Adams espliciti ($\beta_k = 0$) sono noti come Metodi di Adams-Bashforth, mentre quelli impliciti sono detti Metodi di Adams-Moulton. Il metodo di Adams-Bashforth ad un passo coincide con il metodo di Eulero Esplicito, mentre il metodo di Adams-Moulton ad un passo è il metodo dei Trapezi. I metodi di Adams sono tra i più antichi (erano noti già nel diciannovesimo secolo) eppure continuano ad essere utilizzati anche negli algoritmi più moderni.
Una seconda classe di metodi è definita dal primo polinomio caratteristico:

$$\rho(z) = z^k - z^{k-2}.$$

Anche questi metodi sono zero-stabili perchè il polinomio $\rho(z)$ ha come radici $z = \pm 1$ (radici semplici) e $z = 0$ (radice di molteplicità $k - 2$). I metodi espliciti di questa classe sono i Metodi di Nyström, quelli impliciti sono detti Metodi di Milne-Simpson Generalizzati. Un esempio di metodo di Nyström è il Metodo del Midpoint Esplicito, mentre tra i metodi di Milne-Simpson Generalizzati, possiamo citare il Metodo di Simpson:

$$y_{n+2} - y_n - \frac{h}{3}\left(f_{n+2} + 4f_{n+1} + f_n\right) = 0.$$

Un'ultima classe di metodi multistep che vale la pena di citare sono le cosiddette Backward Differentiation Formulae (BDF in breve), che sono metodi impliciti definiti dal secondo polinomio caratteristico:

$$\sigma(z) = \beta_k z^k.$$

Per $k = 1$ il metodo appartenente a questa classe è il metodo di Eulero Implicito, per $k = 2$ abbiamo il seguente metodo che non ha un nome particolare:

$$y_{n+2} - \frac{4}{3}y_{n+1} + \frac{1}{3}y_n - \frac{2}{3}hf_{n+2} = 0,$$

mentre per $k = 3$ abbiamo:

$$y_{n+3} - \frac{18}{11}y_{n+2} + \frac{9}{11}y_{n+1} - \frac{2}{11}y_n - \frac{6}{11}hf_{n+3} = 0.$$