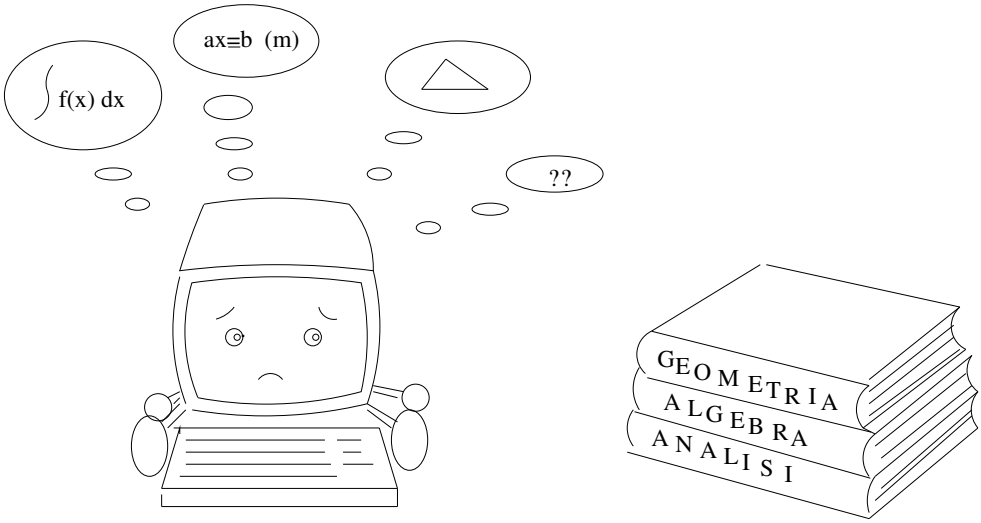
Dispense di
Laboratorio di Matematica

A. Colesanti, S. Dolfi, E. Rubei



Indice

Parte I

Capitolo 1

Introduzione a Matlab

Matlab è un *software* per il calcolo scientifico; fornisce un ambiente in cui sono possibili il calcolo, la programmazione e la visualizzazione grafica dei risultati. Durante il corso impareremo a usare Matlab per verificare e visualizzare alcuni dei risultati che vengono presentati nei corsi di Algebra, Analisi e Geometria, e per *implementare* algoritmi di calcolo la cui validità è stata dimostrata in questi corsi.
E' bene sottolineare che lo scopo del corso *non* è insegnare a usare Matlab, ma mostrare come la matematica possa felicemente interagire con un computer.

Prima di passare a leggere il seguito, avviate Matlab sul vostro computer. Tutte le volte che trovate una parola scritta in `caratteri di questo tipo`, si tratta di un comando di Matlab o più in generale di un comando per il computer.

Nella versione che useremo, ovvero Matlab 6, all'avvio del programma si apre una finestra divisa in tre parti; la parte di destra è la *finestra dei comandi*, in essa vengono scritti i comandi che Matlab può eseguire e vengono visualizzati i risultati che Matlab calcola. Delle altre finestre parleremo in seguito.

1.1 Il computer come calcolatrice

1.1.1 Le operazioni

Una delle cose più elementari che possiamo fare con Matlab è usarlo come una calcolatrice, questo ci consentirà tra l'altro di cominciare a prendere confidenza con i suoi comandi e le sue caratteristiche.
Per fare eseguire una operazione a Matlab è sufficiente scriverla e premere `enter`; ad esempio

```
2+2
enter
```

produce

```
ans = 4
```

Come vedete il risulato compare preceduto da `ans =` (dall'inglese *answer*, risposta).
I simboli corrispondenti all'addizione, la sottrazione, la moltiplicazione e la divisione sono: `+ , - , *` e `/` rispettivamente. Per l'elevamento a potenza si usa il simbolo ^ che deve essere messo tra la base e l'esponente:

```
2^3
ans = 8
```

Per la radice quadrata esiste un comando particolare: `sqrt` (dal corrispondente inglese *square root*); il numero di cui si vuol calcolare la radice è l'argomento della funzione `sqrt` e deve essere scritto tra parentesi tonde dopo la funzione stessa:

```
sqrt(196)
ans = 14
```

1.1.2 Funzioni matematiche predefinite

`sqrt` è il primo esempio che incontriamo di funzione (matematica) predefinita. Matlab possiede molte di queste funzioni, vediamo un elenco di quelle che più comunemente siamo abituati ad usare:

funzione	significato
`sin`	seno
`cos`	coseno
`asin`	arcoseno
`acos`	arcocoseno
`tan`	tangente
`atan`	arcotangente
`exp`	funz. esponenziale in base e
`log`	logaritmo in base e
`sqrt`	radice quadrata
`abs`	valore assoluto
`sign`	funzione segno

Per vedere un elenco completo delle funzioni matematiche di Matlab (solo di quelle considerate elementari) scrivete:

```
more on
help elfun
```

Il comando `more on` fa si che possiate scorrere l'elenco senza che la prima parte di esso resti irrimediabilmente fuori dallo schermo; nella riga successiva abbiamo il primo esempio dell'uso del comando `help`, con cui, in questo caso, si richiedono informazioni sulle funzioni elementari (abbreviato in `elfun`); più avanti torneremo sulle proprietà di `help`.
Come potete vedere nella lista figurano tutte le funzioni trigonometriche e le loro inverse, le funzioni iperboliche con le rispettive funzioni inverse, l'esponenziale, il logaritmo etc. Tutte queste funzioni verranno d'ora in poi correntemente utilizzate.
Come abbiamo visto nel caso di `sqrt`, ogni volta che si usa una funzione matematica per calcolare il suo valore su un certo argomento (numero reale, complesso, vettore etc.), si deve scrivere la funzione seguita dall'argomento incluso tra parentesi tonde.

Altri esempi di funzioni matematiche predefinite da Matlab sono:

- `abs`; questa funzione calcola il valore assoluto di un numero reale (più in generale, il modulo di un numero complesso);

- `fix, floor, ceil` e `round`; sono tutte funzioni che arrotondano il loro argomento, per eccesso o per difetto, ad un numero intero relativo. Ciascuna di queste funzioni arrotonda secondo un criterio diverso dalle altre. Per vedere quali sono questi criteri usate ancora il comando `help`, seguito dal nome della funzione su cui volete avere informazioni. Ad esempio se scrivete `help floor`, verrete a sapere che `floor` arrotonda il proprio argomento verso meno infinito, ovvero all'intero relativo più vicino al numero dato, in direzione di meno infinito:

  ```
  floor(101.5)
  ans = 101
  floor(-18.001)
  ans = -19
  ```

- `rem`; questa funzione, dati due numeri naturali x e y, calcola il resto della divisione intera $x : y$, ovvero la classe di resto di y modulo x; in particolare la funzione `rem` dipende da due argomenti anzichè da uno solo come in tutti gli esempi visti finora. I due argomenti devono essere scritti all'interno della stessa parentesi tonda e separati da una virgola:

  ```
  rem(50,9)
  ans = 5
  ```

  La funzione `mod` è l'analogo della funzione `rem` per i numeri interi relativi.

- Il comando `rand` è un *generatore casuale di numeri*: se scrivete `rand` e date `enter` Matlab genera casualmente, con probabilità uniforme, un numero compreso tra zero e uno. A differenza delle funzioni che abbiamo visto finora, `rand` non prevede argomento.

Le varie funzioni possono essere combinate (composte) tra loro; dunque ha senso scrivere:

```
sin(exp(46))
```

```
atan(rand)
```

```
sqrt(rem(24,50))
```

```
abs(floor(mod(-100,101))).
```

1.1.3 Numeri

Matlab può utilizzare formati diversi per la rappresentazione dei numeri, a seconda delle necessità del computer o delle vostre richieste.

Cominciamo dai numeri interi. La somma, la sottrazione e il prodotto di interi sono ancora numeri interi. Tuttavia se fate eseguire queste operazioni a Matlab, quando il risultato è eccessivamente elevato, in valore assoluto, non viene visualizzato come numero intero; ad esempio scrivete

```
12000 ^ 2
```

e

```
12000*130000.
```

Nel primo caso il risultato viene scritto come numero intero, nel secondo viene rappresentato con una notazione diversa; questo dipende dallo 'spazio' che il computer può dedicare a memorizzare un numero. Il massimo ed il minimo numero intero che Matlab può rappresentare come tale sono dell'ordine di 2^{30} e -2^{30} rispettivamente (cioè dell'ordine di 10 alla decima, in valore assoluto). Quando si esce da questo intervallo, Matlab ricorre alla *notazione esponenziale* di cui parleremo tra poco.

Un generico numero reale può essere rappresentato dal computer:

- in forma di numero decimale con segno, con quattro cifre dopo la virgola (o meglio, dopo il punto):

  ```
  sqrt(2)
  ans = 1.4142
  ```

- in forma esponenziale (o notazione scientifica); in questo caso il numero viene scritto nella forma:

  xey

dove x è un numero decimale con segno che varia tra -10 e 10, e y è un numero intero relativo. Il significato dell'espressione x`e`y è:

$$x \cdot 10^y \, .$$

Quindi

```
10^(3.8)
ans = 6.3096e+03 .
```

ci dice che $10^{3.8}$ è uguale a $6.3096 \cdot 10^3 = 6309.6$.

Come ordine di grandezza, il numero massimo, in valore assoluto, che Matlab può *gestire* è circa 10^{300}. Se chiedete di eseguire un'operazione che abbia come risultato un numero di ordine maggiore di 10^{300} avrete come risposta `ans = Inf` (provate per credere), ovvero per il computer il risultato dell'operazione è infinito.
Dalla parte opposta, la più piccola quantità positiva che Matlab concepisce è dell'ordine di 10^{-300}; questo vuol dire in particolare che ogni numero inferiore a questo viene considerato uguale a zero.

Se scrivete `format long` e premete `enter`, fate in modo che nei risultati di tutte le operazioni che eseguite i numeri decimali abbiano 14 cifre dopo la virgola, anzichè 4. Per ripristinare il formato usuale il comando è `format short` o più semplicemente `format`.

```
format long
sqrt(2)
ans = 1.41421356237310
format
sqrt(2)
ans = 1.4142
```

Esistono altri comandi che fanno in modo che i risultati numerici delle operazioni vengano espressi in formati diversi; per vedere quali sono e come funzionano potete scrivere `help format`. E' bene sottolineare che il comando `format` non altera la precisione con cui il computer esegue le operazioni.

Matlab può rappresentare anche numeri complessi e fare le usuali operazioni con essi. L'unità immaginaria si denota indifferentemente con i simboli `i` o `j`; dunque possiamo far eseguire il prodotto:

```
(1+i)*(2-j) .
```

1.1.4 Variabili

Lavorando con Matlab potete introdurre delle *variabili*: se ad esempio scrivete

```
A = sqrt(2)
```

in ogni successiva operazione in cui compare `A`, questa viene interpretata come radice di due. Le varibili vengono conservate in memoria fino al termine della sessione di lavoro, ovvero fino a che non chiudete il programma Matlab. Il nome di una variabile può essere una qualunque sequenza di lettere e numeri, in cui il primo elemento è una lettera (comunque sono significativi solo i primi 31 caratteri del nome). L'uso di lettere maiuscole e minuscole fa differenza, dunque `a` e `A` sono variabili diverse.
Una delle finestre di sinistra, quella chiamata `workspace`, memorizza e visualizza tutte le variabili che sono state introdotte nella sessione di lavoro corrente.
Per cambiare il valore di una variabile basta ridefinirla; per cancellare tutte le variabili definite si deve dare il comando `clear`.
Matlab possiede numerose variabili predefinite; il primo esempio che abbiamo incontrato è `ans`, una variabile che viene aggiornata ad ogni operazione, e che ha il valore del risultato dell'ultima operazione eseguita. Altre variabili predefinite sono:

variabile	significato
`i , j`	unità immaginaria
`pi`	π, pi greco, 3,1415...
`realmax`	massimo numero macchina positivo
`realmin`	minimo numero macchina positivo
`Inf`	∞, ossia un numero maggiore di `realmax` (o minore di `-realmax`)
`NaN`	"Not a Number", tipicamente il risultato di una espressione 0/0

1.1.5 Help!

Matlab mette a disposizione dei suoi utenti una vasta documentazione in linea, ovvero materiale che può essere consultato tramite computer. Per adesso ci limitiano a vedere il funzionamento del comando `help`, che permette di accedere ad una parte di questa documentazione.
Il comando `help`, che abbiamo già utilizzato in varie occasioni, serve a visualizzare le pagine di un manuale in linea, contenente informazioni sull'uso dei comandi e delle funzioni di Matlab. Se scrivete semplicemente `help`, viene visualizzato l'indice di questo manuale. Ogni riga dell'indice corrisponde ad un capitolo, di cui potete vedere il contenuto usando ancora il comando `help` seguito (da uno spazio e) dal nome del capitolo. Ad esempio potete provare con `help general`. Ancora una volta vi troverete d'avanti una lista di argomenti su ciascuno dei quali potete chiedere informazioni usando ancora `help` seguito dal nome dell'argomento.
In questo modo potete vedere dettagliatamente il funzionamento di due comandi che abbiamo già usato: `more` e `format`.

1.1.6 Esercizi

Esercizio 1.1 *Calcolare:*

$$8!\,, \quad e^{\sin(89)}\,, \quad \sqrt{1+\sqrt{1+\sqrt{2}}}\,.$$

[40320, 2.3633, 1.5981]
(per calcolare il fattoriale di 8, potete anche scrivere `factorial(8)`).

Esercizio 1.2 *Calcolare:*

$$(1+i)^5\,, \quad \sqrt{i}\,, \quad \frac{\pi+2i}{7-i}\,.$$

$[-4-4i,\ 0.7071(1+i),\ 0.3998+0.3428i]$

Esercizio 1.3 *Calcolare:*

$$a^4 - a^3 + a^2 - a + 1\,,$$

dove

$$a = \log(|\sin(e^{2.002})|)\,.$$

[1.1172]

Esercizio 1.4 *È più grande* 100^{120} *o* 120^{100}*?* [Sareste in grado di rispondere a questa domanda senza utilizzare un calcolatore?]

Esercizio 1.5 *Calcolare il resto della divisione di* a *per* b*, dove* $a = 156 \cdot 348 - 1$ *e* $b = a - 11$*.*

[11]

Esercizio 1.6 *Quale tra le funzioni di Matlab* `floor`*,* `fix`*,* `ceil` *e* `round` *coincide con la funzione parte intera?*

Esercizio 1.7 * *Utilizzando la funzione* `rand` *e una delle funzioni di arrotondamento, creare un generatore casuale di numeri naturali compresi tra* 0 *e* 9*.*

Esercizio 1.8 **Creare un generatore casuale di* 0 *e* 1*.*

Riguardo ai due esercizi precedenti: un modo per verificare che si sia trovato un buon generatore casuale di numeri è provarlo molte volte e vedere se i risultati sono effettivamente numeri casuali distribuiti uniformemente nell'intervallo scelto. Osservate che per rieseguire un comando appena eseguito non è necessario riscriverlo ma basta premere il tasto ↑ e il comando appare nuovamente; se premete due volte ↑ tornate al penultimo comando dato, e così via.

1.2 Matrici

1.2.1 Matrici: somma e prodotto per uno scalare

Matlab è particolarmente adatto per maneggiare le matrici (infatti il suo nome è l'abbreviazione di matrix laboratory) e in Matlab anche i numeri sono pensati come matrici (matrici 1×1).
In Matlab per inserire i coefficienti di una matrice si fa nel modo seguente: supponete per esempio di voler chiamare A la matrice 2×3 a coefficienti in $\mathbf{R}$ la cui prima riga è $(-2, 4, 6)$ e la seconda riga è $(45, 0, 19)$; per fare ciò dovete scrivere:

```
A= [-2 4 6; 45 0 19]
```

cioè dovete scrivere prima i coefficienti della prima riga separati da uno spazio, poi un punto e virgola e poi i coefficienti della seconda riga (separati fra loro da uno spazio); se adesso premete `enter`, Matlab vi fa vedere la matrice che avete inserito.
Provate adesso ad inserire un'altra matrice 2×3, chiamatela B e scrivete poi

```
A+B
```

Quando premerete `enter`, Matlab vi mostrerà ovviamente la somma delle due matrici. Cosa succede se provate a sommare due matrici aventi o un numero diverso di righe o un numero diverso di colonne?

Per moltiplicare uno scalare t per una matrice, per esempio la matrice A, basta scrivere

```
t*A
```

per esempio: scrivete `5*A` e poi premete `enter` , oppure scrivete `t=5`, poi `t*A` e infine premete `enter`.

1.2.2 Come si estrae un coefficiente di una matrice, il comando `size` e il prodotto di matrici

L'elemento che sta sulla riga i e colonna j della matrice A si denota con `A(i,j)`. Per esempio scrivete

```
A(2,1)
```

e poi premete `enter`; Matlab vi mostrerà il coefficiente di A che sta sulla seconda riga e la prima colonna, cioè 45.
Il comando `size` dice quante righe e quante colonne ha una matrice; per esempio scrivete

```
size(A)
```

e premete `enter`.
Per avere solo il numero di righe di una matrice A scrivete

```
size(A,1)
```

e premete `enter`.
Per avere solo il numero di colonne, scrivete

```
size(A,2)
```

e premete `enter`.
Il prodotto fra matrici si indica con `*` Provate ad inserire una matrice 3×3, C, ed una matrice 3×4, D, e scrivete

```
C*D
```

e poi premete `enter`.

1.2.3 La trasposta, la traccia e altri comandi

La trasposta di una matrice in Matlab si ottiene scrivendo un punto e un apice dopo la matrice: .' Scrivete ad esempio:

```
A.'
```

e premete enter
Inserite adesso una matrice quadrata; chiamatela C. Provate a calcolare $C + {}^tC$ e $C - {}^tC$, cioè scrivete

```
C+C.'
```

e

```
C-C.'
```

Ovviamente otterrete rispettivamente una matrice simmetrica ed una antisimmetrica (perché?).
Se scrivete `diag(C)` dove C è una matrice quadrata e premete `enter`, matlab vi farà vedere una colonna i cui coefficienti sono i coefficienti della diagonale di C.
La funzione `sum` applicata ad una matrice vi produce una riga i cui coefficienti sono le somme delle colonne della matrice.
Qundi per calcolare la traccia della matrice C possiamo scrivere

```
D= diag(C)
sum(D)
```

oppure direttamente

```
sum(diag(C))
```

(e poi ovviamente premere enter).

In realtà potete calcolare la traccia di una matrice quadrata, oltre che utilizzando le funzioni `diag` e `sum`, direttamente con la funzione `trace`.

Esercizio 1.9 *Dire se la seguente matrice è ortogonale:*

$$A = \begin{pmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

1.2.4 Concatenare matrici, estrarre sottomatrici e alcune matrici particolari

Matlab permette di "concatenare" le matrici, cioè di mettere delle matrici l'una accanto all'altra o l'una sopra l'altra (se il numero delle righe e delle colonne lo consentono) per formare una matrice più grande. Per esempio chiamate A una matrice 2×3, B una matrice 2×5, C una matrice 3×3, D una matrice 3×5 e scrivete

```
S= [A B; C D]
```

Matlab permette anche di estrarre da un matrice una sottomatrice: la matrice i cui elementi sono gli elementi che stanno sulle righe $i_1, ..., i_k$ e sulle colonne $j_1, ..., j_s$ si indica con `A([`$i_1 ... i_k$`],[`$j_1 ... j_s$`])`.
Scrivete ad esempio una matrice 4×5 e chiamatela A;

```
X= A([2 4],[1 2 5])
```

è la matrice i cui elementi sono gli elementi di A che stanno sulle righe $2, 4$ e sulle colonne $1, 2, 5$.
Per estrarre ad esempio la terza colonna possiamo scrivere

```
A([1 2 3 4],[3])
```

ma anche

```
A([1 2 3 4],3)
```

(cioè posso eliminare le parentesi quadre nel secondo argomento in quanto è costituito da un solo numero). In realtà si può anche scrivere più semplicemente

```
A(:,3)
```

il simbolo : sta ad indicare che si considerano tutte le righe possibili. Se avessimo scritto

```
A(2,:)
```

avremmo ottenuto la seconda riga. Se invece avessimo scritto

```
A(:,[3,5])
```

avremmo ottenuto la sottomatrice di A formata dalla terza e quinta colonna, cioè la matrice formata dai coefficienti di A che stanno su una qualsiasi riga e sulla terza e quinta colonna di A.

Apriamo una parentesi.
L'espressione `i:j` dove i e j sono due numeri reali indica il vettore riga i cui coefficienti sono i numeri compresi fra i e j del tipo i, $i+1$ $i+2$..., cioè del tipo $i+n$ con n numero naturale. Provate a vedere che succede se $i > j$.
L'espressione

```
i:k:j
```

dove i, j, k sono tre numeri reali indica il vettore riga i cui coefficienti sono tutti i numeri reali da i a j con passo k, cioè i numeri compresi fra i e j della forma $i+kn$, dove n è un numero naturale.
Provate a scrivere ad esempio

```
1:10
```

```
1:2:10
```

```
pi:2:10
```

```
10:-3:-49
```

```
10:-3:49
```

```
10:3:-49
```

```
1:pi:10
```

etc..

Torniamo a come si estraggono sottomatrici da matrici; utlizzando la notazione appena vista,

```
A([2 3],3:5)
```

è la matrice formata dai coefficienti di A che stanno sulla seconda e terza riga di A e sulle colonne dalla terza alla quinta di A.

Vediamo adesso come si scambiano le righe o le colonne di una matrice. Per scambiare ad esempio la seconda e la quinta colonna di A, basta scrivere

```
A(:,[1 5 3 4 2])
```

Analogamente si possono scambiare due righe.
Scambiate le righe 2 e 4 della matrice A, come sopra.

Vediamo adesso come si possono scrivere in Matlab delle particolari matrici.
La matrice identità $n \times n$ si indica con `eye(n)`
La matrice $m \times n$ con tutti coefficienti nulli si indica con `zeros(m,n)`.
La matrice $m \times n$ con tutti coefficienti uguali a 1 si indica con `ones(m,n)`.

Un ulteriore commento. Se applicate una funzione matematica predefinita di Matlab ad una matrice A il cui generico elemento è a_{ij} ottenete una nuova matrice delle stesse dimensioni di A, il cui generico elemento è il valore della funzione in a_{ij} (questa è un'operazione che in matematica non si fa, ma in Matlab è spesso utile).

Esercizio 1.10 *Sfruttando il prodotto di uno scalare per una matrice e la funzione* `ones`*, scrivete la matrice* 6×7 *con tutti i coefficienti uguali a* 45*.*

Esercizio 1.11 **Create un vettore i cui elementi siano i primi 100 multipli di tre.*

Esercizio 1.12 **Create un vettore* V *che contenga i punti dell'intervallo* $[0, 2\pi]$ *della forma*

$$n\,h\,, \quad n = 0, 1, \ldots, 10000\,,$$

dove $h = \frac{2\pi}{10000}$*. Questo vuol dire che* V *contiene i punti di una* partizione equispaziata *di* $[0, 2\pi]$ *di* passo h*. Calcolate i valori della funzione seno sui punti della partizione.*

1.3 Un po' di calcolo simbolico

Diverse funzioni di Matlab possono essere utilizzate anche con argomenti letterali. Per fare ciò occorre avere il "pacchetto" del calcolo simbolico, che non è una dotazione standard di Matlab.
Per utilizzare il calcolo simbolico, bisogna prima dichiarare quale sono le lettere tramite la funzione `syms`. La somma la differenza il prodotto fra matrici, l'elevamento a potenza di una matrice, il prodotto di uno scalare per una matrice, la trasposta sono tutti comandi che Matlab esegue anche se le matrici in questione hanno coefficienti letterali, purchè prima si dichiarino quali sono le lettere mediante la funzione `syms`; ovviamente quindi Matlab esegue tali operazioni aritmetiche anche fra espressioni letterali (matrici 1×1). Naturalmente potete anche fare operazioni fra matrici i cui coefficienti sono lettere e matrici numeriche.
Per esempio scrivete

```
syms t
C =[t 1 t-1 ; 0 0 3 ; -2 t^2 9]
C
```

Per calcolare la matrice C^2 scrivete

```
C^2
```

Adesso scrivete

```
syms a b c d e f g h
A =[a b ; c d]
B= [e f ; g h]
A
```

```
B
A+B
B+A
A+ zeros(2,2)
A + eye(2)
A + ones(2, 2)
A*B
B*A
```

Inserite adesso la matrice

$$A = \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix}$$

e calcolate I_3A, AI_3, $A + {}^tA$ e $A - {}^tA$. Ricordiamo che per calcolare la trasposta di A dovete scrivere `A.'`

Altre funzioni utilizzabili per il calcolo simbolico in Matlab sono `diag`, `trace`, `size`. Provatele!

In Matlab ci sono diverse funzioni riguardanti i polinomi: `factor`, `expand`, `sym2poly`, `poly2sym`, `simplify`.
La funzione `factor` permette di fattorizzare un polinomio; scrivete ad esempio

```
syms x y
factor(x^2-y^2)
factor(x^3+3*x^2*y+3*x*y^2+y^3 )
```

La funzione `expand` applicata a un prodotto di polinomi calcola il polinomio prodotto sviluppato. Scrivete ad esempio

```
syms x y
expand((x-y)^ 3)
```

La funzione `sym2poly` trasforma un polinomio nel vettore riga dei suoi coefficienti. Ad esempio

```
syms u
sym2poly(u^3-2*u^3+4)
```

La funzione `poly2sym` fa il processo contrario.
La funzione `simplify` semplifica un polinomio, provate ad esempio a scrivere

```
syms a b
simplify(a^3+b+a-a^3)
```

Un'importante funzione è la funzione `subs`. Il comando `subs(S, old, new)` rimpiazza nell'espressione simbolica S `old` con `new`. Per esempio

`subs(a+b, a, 4)`

dà `4+b`.

Osservate che le funzioni `simplify` ed `expand` possono essere anche applicate alle matrici. Utilizzando ciò potete svolgere i seguenti due esercizi:

Esercizio 1.13 *Usando matlab verificate la proprietà associativa del prodotto fra le matrici per le matrici* 3×3.

Esercizio 1.14 *Usando matlab verificate che*

$$^t(AB) = {}^tB{}^tA$$

per le matrici 3×3.

Calcolate infine la diagonale e la traccia di ${}^tAA$ dove A è la seguente matrice:

$$\begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \\ a41 & a42 & a43 \end{pmatrix}$$

Esercizio 1.15 *(teorico) Sia A una matrice $m \times n$. Dimostrate che*
1) $({}^tAA)_{ii} \geq 0$, $\forall i = 1, ..., n$
2) $tr({}^tAA) = 0$ *se e solo se* $A = 0$.

1.4 Operatori relazionali

Se scrivete

```
12>13
```

avrete in risposta

```
ans=0
```

Che cosa significa? Scrivendo la prima riga avete formulato una domanda riguardante il confronto tra due numeri, 12 e 13, e Matlab vi ha risposto che il confronto proposto da voi è falso, ovvero corrisponde alla variabile di verità zero.
Matlab dispone di sei operatori relazionali:

operatore	significato
`>`	maggiore
`>=`	maggiore o uguale
`<`	minore
`<=`	minore o uguale
`==`	uguale
`~=`	diverso

Questi permettono di confrontare tra loro numeri, vettori o matrici. Matlab risponde ad un comando contenente un operatore relazionale con 1 (vero) se il confronto scritto è esatto oppure con 0 (falso) se non è esatto.

Due osservazioni. L'operatore di uguaglianza è dato da due segni di uguale; ricordiamo che il singolo segno di uguale corrisponde all'*assegnazione* di un valore. La riga

```
x=4
```

definisce la variabile `x` assegnandole il valore numerico 4.
Se confrontate due matrici `A` e `B` Matlab confronta i singoli elementi di `A` e `B`; conseguentemente `A` e `B` devono avere le stesse dimensioni; la risposta è una matrice, che ha ancora le dimensioni di `A` e `B`, i cui elementi sono zero o uno. L'unica eccezione si ha quando si confronta una matrice con uno scalare, in questo caso tutti gli elementi della matrice vengono confrontati con lo scalare.

Esercizio 1.16 *Creare una matrice A, 3×3. Senza calcolare A^2, stabilire quanti degli elementi di A^2 sono diversi da 0.*

Esercizio 1.17 **Sia $A = \{\sin(i) : i = 1, 2, \ldots, 10000\}$. Calcolare quanti degli elementi di A sono maggiori o uguali a $1/2$.*

Indichiamo lo schema per risolvere l'ultimo esercizio.

- Creare un vettore `V` i cui elementi sono i primi 10000 numeri naturali;
- utilizzando la funzione `sin` applicata a `V` creare un vettore `W` i cui elementi siano gli elementi di A; a questo proposito ricordiamo che tutte le funzioni matematiche predefinite da Matlab, applicate ad una matrice A, danno come risultato la matrice (della stessa dimensione di A) ottenuta applicando la funzione ad ogni elemento di A;
- confrontate `W` con $1/2$, usando l'operatore maggiore o uguale;
- con il comando `sum`, contate il numero di risposte vere (uno) che avete ottenuto nel precedente confronto.

Capitolo 2

Programmare con Matlab

2.1 Iniziamo a programmare

Matlab è anche, e soprattutto, un linguaggio di programmazione; ovvero lavorando con Matlab si possono *scrivere* dei programmi e *farli eseguire*. Un programma può essere descritto come una sequenza di comandi e istruzioni da fare eseguire al computer, finalizzata a svolgere un calcolo o una grossa mole di operazioni in tempi molto rapidi. Nel corso di laboratorio scriveremo molti programmi, con lo scopo di risolvere problemi (più o meno) semplici, verificare risultati teorici tramite degli esempi, visualizzare il comportamento di esempi troppo complicati per essere studiati 'a mano'.

In questa parte descriveremo i due tipi di programmi che si possono scrivere, ovvero i programmi di tipo *script* e di tipo *function*.

2.2 M-files, scripts e functions

I programmi che Matlab può eseguire devono essere scritti in files chiamati *m-files*, che terminano con il suffisso `.m`
Per creare un m-file si può usare un qualsiasi editor di testi; in particolare la versione Matlab 6 mette a disposizione un proprio editor per la scrittura di m-files. Per aprirlo è sufficiente cliccare sull'icona in alto a sinistra nella finestra di Matlab (sotto la scritta *file*), corrisponedente a *nuovo documento*. Si tratta di un editor standard che utilizzeremo a livello elementare.
Quello che dovete fare per creare e utilizzare un m-file può essere sintetizzato come segue:

- aprite l'editor e scrivete il testo del programma;

- dopo aver completato la scrittura del programma, salvate il file. Per farlo, dal menu *file* in alto a sinistra scegliete l'opzione *Save As*; successivamente vi verrà chiesto il nome da dare al file, ricordate che il nome deve terminare con `.m` Dovrete anche scegliere la collocazione del file in una directory. È consigliabile creare una directory apposita in cui salvare tutti gli m–files che scriverete. **Importante:** potrete fare eseguire da Matlab il programma contenuto nel file, solo se Matlab è stato aperto dalla directory in cui si trova il file stesso;

- dopo aver salvato il file, il programma che esso contiene è operativo, per farlo eseguire basterà scrivere il nome del programma *senza il suffisso* `.m` (seguito dall'argomento, se si tratta di una funzione) nella finestra dei comandi di Matlab.

Vediamo un esempio. Abbiamo visto come creare un generatore casuale di 0 e 1; una delle possibilità è data dalla combinazione di comandi

```
round(rand)
```

Scrivete il programma, formato dalla sola riga `round(rand)`, all'interno di un file che potete chiamare ad esempio `testaocroce.m`. Dopo averlo salvato scrivete `testaocroce` nella finestra dei comandi.

Vedremo due tipi di programmi: *comandi* (*script*) e *funzioni* (*function*). La differenza tra i due sta nel fatto che i secondi possono dipendere da un argomento, mentre i primi no; `testaocroce` è un esempio di comando.

2.2.1 Comandi

I comandi sono programmi la cui esecuzione non dipende da un argomento. Per eseguire un comando all'interno di Matlab se ne scrive semplicemente il nome e Matlab esegue tutte le istruzioni contenute nel file.

2.2.2 Funzioni

Le funzioni sono programmi la cui esecuzione dipende da un argomento. Tutte le funzioni matematiche predefinite sono basate su programmi di questo tipo, ad esempio `sin` è una funzione che permette di sapere il valore del seno di un angolo; per essere eseguita si deve scrivere il suo nome, seguito dal valore dell'argomento tra parentesi tonde. Questo procedimento funziona per l'esecuzione di ogni funzione, anche di quelle create da voi.
Notate che l'argomento di un programma di tipo funzione può essere un numero, una coppia o una terna di numeri, e più in generale una matrice di dimensioni qualsiasi.
La prima riga di un programma di tipo funzione deve essere di questo tipo:

`function` *variabile, es.* `t =` *nome della funzione* `(`*variabile, es.* `x )`

Il nome della variabile che precede l'uguale può essere scelto arbitrariamente. È consigliabile dare al file che contiene un programma di tipo funzione, lo stesso nome che compare a destra dell'uguale nella prima riga (a parte il suffisso `.m`).

Vediamo un esempio molto semplice: scriviamo un programma che calcola il quadrato del seno di un angolo.

```
function s=senoquadro(x);
s=(sin(x))^2
```

Nella seconda riga vengono date le istruzioni per calcolare la funzione che si sta definendo. Per far girare questo programma, dopo averlo salvato con il nome `senoquadro.m` nella directory in cui è stato lanciato Matlab, tornate nella finestra dei comandi di Matlab e scrivete ad esempio

```
senoquadro(2)
```

Il risultato dell'esecuzione viene visualizzato due volte, una come valore di `s` e la seconda come valore di `ans`. Questo è dovuto al fatto che nell'esecuzione di una funzione, Matlab visualizza *comunque* il valore della funzione stessa. La ripetizione può essere evitata sopprimendo la visualizzazione di `s`, per farlo, aggiungete semplicemente il punto e virgola alla fine della seconda riga del programma.

2.3 Commenti

Una volta che un programma è stato salvato all'interno di un m–file, questo diventa uno dei comandi o delle funzioni che potete usare correntemente con Matlab. In particolare dalla finestra dei comandi si può leggere (ma non modificare) il contenuto di un m-file dall'interno di Matlab con il comando `type` *nome del file*.
È molto utile inserire all'interno di un programma delle *righe di commento*, ovvero righe che non contengono istruzioni per Matlab ma commenti al programma in generale (ad esempio una descrizione dello scopo del programma posta al suo inizio) o a sue parti specifiche. Per inserire un commento, scrivete il simbolo `%` all'inizio di una riga: tutto quello che scrivete in quella riga viene ignorato da Matlab nell'esecuzione del programma. Dalla finestra dei comandi di Matlab si possono visualizzare le righe di commento di un m-file con `help` *nome del file*.

2.4 Esercizi

Ciascuno degli esercizi che seguono prevede la scrittura di un programma. È ovviamente consigliabile verificare la correttezza dei programmi scritti facendoli eseguire da Matlab.

Esercizio 2.1 *Scrivere un programma di tipo comando che visualizzi i primi 546 numeri naturali.*

Esercizio 2.2 **Scrivere u programma di tipo funzione che, dato un numero naturale n (l'argomento) visualizzi i primi n multipli di 7.*

Esercizio 2.3 *Scrivere u programma di tipo funzione che calcoli il valore della funzione* seno iperbolico *al quadrato:*

$$f(x) = (\sinh(x))^2 = \left(\frac{e^x - e^{-x}}{2}\right)^2 , \quad \forall x \in \mathbf{R} .$$

2.5 I comandi `if` e `for`

Affrontiamo ora la parte più importante della programmazione con Matlab. Per scrivere programmi che vadano oltre il livello elementare degli esempi e degli esercizi visti nel paragrafo precedente, occorre utilizzare dei particolari comandi di Matlab con i quali vengono costruiti dei *cicli* all'interno di un programma. In questo paragrafo vedremo in particolare `if` e `for` e più avanti `switch`. Con l'occasione incontreremo anche alcuni operatori logici di Matlab.

2.5.1 Il comando `if`

Il funzionamento di `if` (*se* in inglese) può essere descritto come segue. Quando Matlab incontra una riga contenente il comando `if` seguito da una espressione, controlla questa espressione e, nel caso che questa sia *vera*, esegue i comandi successivi fino a che non trova il comando `end`, o uno dei comandi `elseif` e `else`; ad ogni `if` deve comunque corrispondere un `end`. I due comandi `if` e `end` delimitano un *ciclo if*.

I due comandi `else` e `elseif` possono essere usati con `if`, per valutare più possibilità. Con `elseif` si controlla la verità di un'altra espressione logica, mentre `else` non deve essere seguito da nessuna espressione, con esso si comprendono tutte le possibilità non incluse dai precedenti `if` e `elseif`.
Come abbiamo detto, `if` controlla la verità di una espressione; in molti casi l'espressione consiste di una uguaglianza, di una disuguaglianza o di una non uguaglianza tra due quantità. Per esprimere queste relazioni devono essere usati gli operatori relazionali che abbiamo incontrato nel Paragrafo 1.4.

Esempio 2.1 *L'esempio che segue è un file di tipo function,dipendente da due variabili, che serve a vedere se dati due numeri naturali m e n, il primo è divisibile per il secondo*

```
function d=divisibilita(m,n)
if rem(m,n)==0
      'il secondo numero divide il primo'
  else
      'il secondo numero non divide il primo'
end
```

Nella terza riga e nella quinta riga del programma si dà istruzione a Matlab di scrivere una parte di testo (anziché dei risultati numerici) e questo viene fatto semplicemente mettendo il testo tra due virgolette (si può utilizzare anche il comando `disp`*, vedere la guida in linea).*

La parte di un programma compresa tra l'istruzione `if` e il corrispondente `end` è un esempio di ciclo if. Nel programma dell'esempio precedente alcune righe comincino più avanti delle altre: non è né un errore né un caso, serve a facilitare l'individuazione dei cicli all'interno del programma.

Esercizio 2.4 *Modificando il programma dell'esempio precedente e usando anche il comando* `elseif`*, scrivete un programma ancora di tipo function, che svolga il seguente compito: dati due numeri naturali m e n, dica se m divide n, oppure se n divide m, oppure se nessuno dei due divide l'altro.*

Esempio 2.2 *Vediamo un programma che verifica se un numero naturale è un quadrato perfetto oppure no.*

```
function p=perfetto(n)
R=sqrt(n);
r=R-floor(R);
if r==0
      'il numero e un quadrato perfetto'
   else
      'il numero non e un quadrato perfetto'
end
```

Nella seconda e terza riga vengono introdotte due variabili: la radice di n, `R`*, e* `r` *che è la parte decimale di* `R`*. Ciascuna di queste due righe termina con un punto e virgola, questo fa sì che Matlab quando esegue il programma non visualizzi il valore delle due variabili* `R` *e* `r`*.*

Esercizio 2.5 **Scrivete un programma che, data una matrice dica se è quadrata oppure no (Utilizzate il comando* `size`*).*

Esercizio 2.6 **Scrivere un programma di tipo funzione che, dati tre numeri interi a, b e c verifichi se almeno uno è medio proporzionale tra gli altri due oppure no.*

Esercizio 2.7 **Scrivete un programma di tipo funzione che esprima, per ogni valore reale di x, il valore di $f(x)$ dove f è data da*

$$f(x) = \begin{cases} x^2\,, & \text{se } x \leq 3\,, \\ 9\,, & \text{se } x > 3\,. \end{cases}$$

Esercizio 2.8 **Scrivete un programma che, data una matrice quadrata, dica se questa è simmetrica, antisimmetrica o non ha né la prima né la seconda proprietà; in quest'ultimo caso il programma deve scrivere la matrice come somma di una matrice simmetrica e di una antisimmetrica.*

2.5.2 L'uso di `if` con gli operatori logici *e* e *o*

Gli operatori logici *e* e *o* corrispondono nel linguaggio di Matlab ai simboli `&` e `|` rispettivamente. Utilizzando questi operatori si può fare in modo che `if` controlli più condizioni. Nel caso in cui si vuole che i comandi successivi a `if` siano eseguiti se *almeno una* di queste condizioni è verificata, si deve usare `|` per separare le varie condizioni; si deve invece usare `&` nel caso in cui si vuole che *tutte* le condizioni siano verificate per eseguire i comandi che seguono `if`.
Vediamo attraverso un esempio come questi operatori possono essere utilizzati in concomitanza con `if` per semplificare un programma.

Esempio 2.3 *Vediamo un programma di tipo funzione che verifica se, dati tre numeri reali positivi a, b e c, esiste un triangolo che ha lati lunghi a, b e c rispettivamente. Il programma è basato sul fatto seguente: tre numeri positivi a, b e c sono le lunghezze dei lati di un triangolo se e solo se presi comunque due di essi, la loro somma è maggiore o uguale del terzo, ovvero se e solo se le tre disuguaglianze*

$$a+b \geq c\,, \quad b+c \geq a\,, \quad c+a \geq b\,,$$

sono verificate contemporaneamente. Nella seconda riga del programma il comando `if` *controlla tre espressioni, separate dal segno* `|`*, che sono le negazioni delle disuguaglianze scritte sopra. Se almeno una di queste non è vera, il triangolo non esiste.*

```
function T=esistetriang(a,b,c)
if a+b<c | a+c<b | c+b<a
      'non esiste un triangolo con tali lati'
   else
      'esiste un triangolo con tali lati'
end
```

Esercizio 2.9 *Partendo dall'esempio precedente, scrivete un programma che dica se dati tre numeri reali positivi a, b e c, esiste un triangolo rettangolo i cui lati misurano a, b e c.*

Esercizio 2.10 *Scrivete un programma che dica se, dati tre numeri naturali a, b e c, questi formano una terna pitagorica oppure no.*

2.5.3 Il comando `for`

Il comando `for` fa in modo che un gruppo di operazioni venga eseguito per un numero finito, e prefissato, di volte n. Il gruppo di operazioni, in generale, dipende da un parametro i, che assume n valori, uno corrispondente a ciascuna delle volte che il gruppo viene eseguito. Nel caso più semplice i assume come valori tutti i numeri naturali compresi tra 1 e n. Il numero n ed i valori assunti da i vengono specificati nella stessa riga che contiene il comando `for`. Il gruppo di operazioni deve essere compreso tra `for` e `end`; `for` e `end` delimitano un ciclo for.

Un esempio semplice che illustra il funzionamento di `for` è il *campionamento* di funzioni. Campionare una funzione f di una variabile definita in un intervallo $[a, b]$ a valori reali, significa calcolare esplicitamente i valori che f assume su una *griglia* sufficientemente fitta di punti di $[a, b]$. Ad esempio si può fissare un naturale n abbastanza grande e considerare $n + 1$ punti $x_0 = a, x_1, \ldots x_n = b$, *equispaziati* in $[a, b]$:

$$x_i = a + \frac{i(b-a)}{n} = x_{i-1} + \frac{b-a}{n}\,, \quad i = 1, \ldots, n\,.$$

Esempio 2.4 *Il programma che segue è di tipo comando, e campiona il polinomio* $f(x) = x^5 - 4x^2 + 3x - 2$ *nell'intervallo* $[1, 2]$.

```
for i=1:201
   x(i)=1+(i-1)/200;
   a=x(i);
   y(i)=a^5-4*a^2+3*a-2;
end
A=[x;y];
A.'
```

Nella prima riga si prescrive che l'indice `i` *assuma tutti i numeri naturali compresi tra* 0 *e* $200 = n$. *Nella seconda riga vengono definite le componenti del vettore riga* `x`; `x(i)` *è l'*i*-esimo punto della griglia su cui campioniamo* f; *il punto e virgola finale, al solito, fa sì che questi valori non vengano visualizzati. Nella terza riga viene introdotta una variabile di comodo* `a`, *che ha lo scopo di semplificare la scrittura nella riga successiva, in cui vengono definite le componenti del vettore* `y` *che sono i valori assunti da* f. *Nella penultima riga si affiancano i valori di* `x` *e* `y` *nella matrice* 2×200 `A`, *e infine nell'ultima riga si fa visualizzare la trasposta di* `A`.
Dai risultati che ci mostra Matlab sembra abbastanza chiaro che f *cresce nell'intervallo considerato (il che può essere verificato usando la derivata prima di* f*).*

Esempio 2.5 *Vediamo ora un programma per calcolare* i numeri di Fibonacci. *I numeri di Fibonacci sono gli elementi di una successione infinita di numeri naturali* F_i, $i \in \mathbf{N}$, *definiti per ricorrenza nel modo seguente:*

$$F_0 = 1\,, \quad F_1 = 1\,, \quad F_i = F_{i-1} + F_{i-2}\,, \quad \forall i \geq 2\,.$$

In pratica: i primi due termini sono uguali a 1, e ogni altro termine è ottenuto come somma dei due precedenti.

```
% Programma di tipo funzione
% calcola i primi k numeri di Fibonacci con k maggiore o uguale a 2
function F=numfib(k)
F(1)=1;
F(2)=1;
for i=1:k-2
  F(i+2)= F(i) + F(i+1);
```

```
end
```

Provate a far girare questo programma per vari valori di k.
Modificando leggermente il programma si fa in modo che venga visualizzato solo il k-esimo numero di Fibonacci:

```
% programma di tipo funzione
%calcola il k-esimo numero di Fibonacci
function F=numfibo(k)
F(1)=1;
F(2)=1;
for i=1:k-2
   F(i+2)= F(i) + F(i+1);
end
F=F(k);
```

Esercizio 2.11 **Campionare la funzione $f(x) = \sin(1/x)$ nell'intervallo $[1/1000, 1]$ su una griglia di 5000 punti equispaziati.*

Esercizio 2.12 **Scrivete un programma che calcoli, per un numero naturale k arbitrario, il rapporto*

$$r_k = \frac{F_{k+1}}{F_k}\,.$$

Verificate che per k grande, r_k tende al valore del rapporto aureo $\frac{1+\sqrt{5}}{2}$.

Esercizio 2.13 **Scrivere un programma di tipo funzione che, dato un numero naturale n scriva: tutti i numeri pari minori o uguali di n se questo è pari; tutti i numeri dispari minori o uguali di n se questo è dispari*

2.6 Problemi

2.6.1 Permutare le cifre di un numero

Esempio 2.6 *Supponiamo di voler scrivere tutti i numeri naturali di due cifre, le cui cifre appartengono all'insieme $\{1, 3, 4, 7\}$. Il programma che segue svolge questo compito, utilizzando due cicli uno incluso nell'altro. La prima operazione è quella di definire un vettore* `A` *che abbia come componenti 1, 3, 4 e 7. Successivamente si scrive il generico numero di due cifre che siano componenti di* `A`*, e con due cicli si prendono tutte le possibili combinazioni.*

```
% Comando per scrivere tutti i numeri di due cifre le cui cifre
% appartengono all'insieme 1,3,4,7
A= [1 3 4 7];
for i=1:4
for j=1:4
   disp(A(i)*10 + A(j))
```

```
end
end
```

Notate che ci sono due `end`*, uno in corrispondenza di ciascun* `for`*.*

Esempio 2.7 *Il programma che segue è una leggera modifica di quello precedente. Si tratta di un programma di tipo funzione che ha per argomento quattro numeri naturali a, b, c e d compresi tra 1 e 9 e come risultato scrive tutti i numeri di due cifre le cui cifre appartengono a* $\{a, b, c, d\}$*. Analogamente a prima la prima operazione è quella di definire un vettore* `A` *che abbia come componenti a, b, c, d.*

```
function qcifre(a,b,c,d)
A=[a,b,c,d];
for i=1:4
for j=1:4
   disp(A(i)*10 + A(j))
end
end
```

Esempio 2.8 *Il programma che segue, analogamente al precedente, scrive tutti i numeri di due cifre le cui cifre appartengono ad un insieme dato di quattro numeri naturali compresi fra 1 e 9. La differenza col precedente è consiste nel fatto che adesso l'argomento è direttamente una matrice* 1×4*. Quindi per utilizzare questa funzione, dovete scrivere i quattro numeri naturali compresi fra 1 e 9 fra cui possono variare le cifre come coefficienti di una matrice.*

```
function q=qqcifre(A)
for i=1:4
for j=1:4
   disp(A(i)*10 + A(j))
end
end
```

Esercizio 2.14 **Scrivete un un programma di tipo funzione che, assegnati quattro numeri naturali a, b, c, d compresi tra 1 e 9, scriva tutti i numeri naturali di tre cifre le cui cifre appartengono all'insieme* $\{a, b, c, d\}$*.*

Esercizio 2.15 **Scrivete un programma di tipo funzione che, assegnati n numeri naturali* $\{a_1, \ldots, a_n\}$ *compresi tra 1 e 9, scriva tutti i numeri di tre cifre, le cui cifre appartengono a* $\{a_1, \ldots, a_n\}$*.*

[Suggerimento: la parte nuova rispetto ai casi visti prima, è che n non è noto a priori, e dunque occorre fare in modo che il programma lo *scopra* dopo che è stato immesso l'argomento. Per questo, se `A` è l'argomento del programma, utilizzate il comando `size(A,2)`].

2.6.2 Contare

Esempio 2.9 *Vediamo ora un programma che serve a contare quanti sono gli elementi di un insieme (finito) che soddisfano una certa proprietà. Il problema è lo stesso dell'Esercizio 1.17: calcolare quanti sono i numeri naturali n, con $0 \leq n \leq 1000$ tali che $\sin(n) \geq \frac{1}{2}$; questa volta verrà risolto senza l'aiuto del comando* `sum`*. Nel programma che segue si usa* `if` *all'interno di un ciclo* `for` *per verificare per ciascun n se la disuguaglianza in questione è vera. Inoltre si introduce la variabile* `C` *che fa da contatore, ovvero aumenta di una unità tutte le volte che si trova un elemento che ha la proprietà richiesta.*

```
% script; calcola quanti sono i naturali i tra 0 e 1000
% per cui sin(i) e' maggiore di 1/2
C=0;
for i=1:1000
    if sin(i)>=1/2
      C=C+1;
    end
end
C
```

Osservate che il valore del contatore `C` *viene posto inizialmente uguale a zero, ovvero viene* inizializzato. *Questo serve ad evitare spiacevoli inconvenienti che potrebbero sorgere se, ad esempio, la variabile* `C` *fosse stata già introdotta nella stessa sessione di lavoro in cui viene eseguito il programma.*

Esercizio 2.16 *Dato un numero naturale N, sia n il numero dei naturali i compresi tra 1 e N per cui vale la disuguaglianza $\cos(i^2) \geq 0$. Scrivete un programma di tipo funzione che, dato N, calcoli n e il rapporto n/N.*

[Per $N = 10000$, $n = 5001$, $r = 0.5001$.]

2.6.3 Sommare

Esempio 2.10 *Somma di naturali.*

```
% Programma di tipo funzione
% calcola la somma dei primi n numeri naturali
function s=somnat(n)
s=0;
for i=1:n
  s=s+i;
end
```

Esercizio 2.17 **Calcolate*

$$\sum_{i=5}^{100} \frac{1}{\sqrt{i+i^3}}\,.$$

[0.7374]

Esercizio 2.18 **Scrivete un programma che, calcoli la somma dei numeri naturali i con $0 \leq i < 1000$ tali che 7 non divide i.*

[428429]

2.6.4 Ordinare

Esempio 2.11 *Vediamo adesso un programma di tipo funzione che ha per argomento tre numeri reali e produce come risultato gli stessi tre numeri ordinati in senso crescente.*

```
function ordinare3=ordinare3(a,b,c)
if a>b
   a1=a;
   a=b;
   b=a1;
end
if b>c
   b1=b;
   b=c;
   c=b1;
end
if a>b
   a1=a;
   a=b;
   b=a1;
end
ordinare3=[a b c];
```

Cerchiamo di capire il funzionamento di questo programma. Vi sono tre cicli `if`. Nel primo vengono confrontati `a` e `b`; se `a` è maggiore (strettamente) di `b`, il valore di `a` viene memorizzato in una nuova variabile `a1`, di uso temporaneo, si ridefinisce `a` ponendolo uguale a `b` e infine si ridefinisce `b` ponendolo uguale ad `a1`, ovvero l'originario valore di `a`... Perchè tutti questi scambi? Il risultato finale è che `a` e `b` sono adesso rispettivamente il minimo e il massimo tra i valori originari di `a` e `b`, ovvero la coppia (`a,b`) è stata riordinata in senso crescente. Se il valore originario di `a` è minore o uguale di quello di `b`, la coppia (`a,b`) è già ordinata in senso crescente e questo primo ciclo non ha nessun effetto.
Analogamente al primo, il secondo ciclo serve a ridefinire (eventualmente) `b` e `c` in modo da ordinarli in senso crescente. Al termine di questo ciclo siamo sicuri che l'attuale `c` è il massimo tra i numeri dati inizialmente. Per completare l'opera, si devono ordinare in senso crescente gli attuali `a` e `b`, e questo viene fatto con il ciclo finale.

Esercizio 2.19 *Fate un programma di tipo funzione che ordini quattro numeri in modo crescente.*

Il seguente programma ordina un numero qualsisi di numeri, che vengono dati come coefficienti di una matrice. Usa lo stesso metodo dell'esempio precedente.

```
% function per ordinare un numero qualsiasi di numeri
function ordinare=ordinare(A)
[k,n]=size(A);
for m=n:-1:2
for i=2:m
if A(i)<A(i-1)
x=A(i);
A(i)=A(i-1);
A(i-1)=x;
end
end
end
ordinare=A;
```

Matlab ha un comando predefinito che ordina automaticamente gli elementi di un vettore in ordine crescente: `sort`.

2.7 I comandi `break` e `return`

Il comando `break` può essere utilizzato all'interno di un ciclo per interromperlo, nel caso si verifichi una determinata condizione. Per capire come usare questo comando vediamo un esempio.

Esempio 2.12 *Supponiamo di voler calcolare il più grande numero naturale k tale che la somma*

$$\sum_{i=1}^{k} \sqrt{i}$$

sia minore di 1000. Tale numero è certamente minore di 10^6, la cui radice quadrata è 1000. Possiamo allora determinare il numero con il seguente programma:

```
s=0;
for i=1:1000000
  s=s+sqrt(i);
  if s > = 1000
    break
  end
end
disp(i-1)
```

Quando Matlab incontra `break`*, il che accade se la condizione contenuta nella riga di* `if` *è verificata, salta direttamente al comando* `end` *che conclude il ciclo* `for`*.*
L'istruzione finale del programma fa visualizzare quanti passi del ciclo sono stati eseguiti prima *che la somma divenisse più grande di 1000.*

Esercizio 2.20 *Scrivete un programma di tipo funzione, che dipenda da un vettore* $V = (v_1, \ldots, v_n)$ *e da un numero positivo* a*, che svolga la seguente funzione. Calcola il più grande numero naturale* k*,* $k \leq n$*, tale che*

$$\sum_{i=1}^{k} v_i \leq a\,.$$

Un'altro comando per interrompere l'esecuzione di un programma è `return`. A differenza di quanto accade con `break`, quando Matlab incontra `return` all'interno di un programma non legge nessuna istruzione successiva contenuta nel programma stesso e rende di nuovo attiva la finestra dei comandi.

2.8 Il comando `while`

Il comando `while` fa in modo che una operazione, o un gruppo di operazioni, vengano ripetute fintanto che una certa condizione logica è verificata. La condizione viene specificata nella stessa riga contenente `while`; le operazioni che devono essere ripetute sono contenute nelle linee comprese tra quella contenente `while` e una riga che contiene `end`. Dunque `while` è simile a `for`; la differenza sostanziale è che il numero di iterazioni non viene stabilito all'inizio, ma dipende da una condizione.

Esempio 2.13 *Vediamo un esempio di programma di tipo script in cui viene usato il comando* `while` *per calcolare la somma dei cubi dei primi 100 numeri naturali.*

```
% questo programma calcola la somma dei cubi dei primi 100 numeri naturali
i = 0;
s = 0;
while i<=100
   s=s+i^3;
   i=i+1;
end
disp(s)
```

Notate anche in questo caso che la prima operazione è quella di inizializzare le variabili `i` e `s`.
È chiaro che il programma precedente poteva essere scritto in maniera altrettanto efficiente usando `for` al posto di `while`. Vediamo un secondo esempio che mette in rilievo i vantaggi che `while` può dare. Nella situazione che andiamo a considerare non possiamo infatti determinare a priori quante iterazioni saranno necessarie.

Esempio 2.14 *Scriviamo un programma che determini il quoziente della divisione (con resto) di un intero* a *per un intero* b*.*

L'idea è la seguente: se a e b sono positivi, il quoziente è il massimo intero q tale che $b \cdot q \leq a$.

```
function q = quoz(a,b)
k = 0;
while b * k <= a
   k = k+1;
end
q = k-1;
```

Per interi a e b interi non positivi `quoz` non funziona. Dobbiamo aggiungere alcune correzioni che dipendono dal segno del dividendo `a` e del divisore `b`.

Esempio 2.15
```
function q = quoziente(a,b)
% calcola il quoziente della divisione intera di a per b (b non nullo)
q1 = quoz(abs(a), abs(b));
if a >= 0
   q = sign(b) * q1;
elseif abs(b) * q1 == abs(a)
   q = -1 * sign(b) * q1;
else
   q = -1 * sign(b) * (q1 + 1);
end
```

Esercizio 2.21 **Scrivete la funzione* `resto(a,b)` *che calcola il resto della divisione intera di a per b.*

Esercizio 2.22 ** Scrivete una funzione che, dato un vettore V di numeri reali, calcoli la somma di tutte le componenti di V.*

Esercizio 2.23 **Scrivere un programma di tipo comando che simuli il comportamento di un dado a sei facce, ovvero ogni volta che il programma viene eseguito viene generato un numero intero compreso tra 1 e 6.*

Capitolo 3

Un po' di Algebra

3.1 Operazioni tra insiemi

In Matlab gli insiemi (di numeri) si rappresentano come vettori. Ad esempio scrivendo

```
a = [ 2 2 5 4 4 3 ]
```

definiamo l'insieme

$$a = \{2, 2, 5, 4, 4, 3\}\,.$$

La funzione `unique` elimina le ripetizioni e riscrive gli elementi in ordine crescente. Provate

```
a1 = unique(a)
```

Naturalmente per Matlab `a` e `a1` sono diversi. Per verificarlo scrivete `isequal(a,a1)`, avrete come risultato `0` che in questo caso è un *indicatore logico* ovvero significa che l'affermazione "`a` è uguale a `a1`" è falsa. Se invece provate a scrivere `a == a1` otterrete un messaggio di errore perché i due vettori non hanno la stessa dimensione.
Possiamo controllare l'appartenenza di un elemento ad un insieme mediante la funzione `ismember`; i comandi

```
ismember(2,a), ismember(9,a)
```

danno rispettivamente risultato `1` (*vero*) e `0` (*falso*). Osservate che è possibile affiancare più istruzioni su una riga separandole con virgole; i risultati vengono visualizzati nell'ordine.

Matlab fornisce le funzioni corrispondenti alle operazioni di base tra insiemi:
`union` , `intersect` , `setdiff` e `setxor` calcolano rispettivamente l'unione, intersezione, differenza e differenza simmetrica di due insiemi:

```
b = [ 3 5 9 7 8 ];
union(a,b)
```

calcola l' insieme unione di `a` e `b` . Osservate che "`;` " nella prima riga sopprime la visualizzazione dell'assegnazione della variabile `b` .
Lo stesso risultato si sarebbe ottenuto con

```
b1 = unique(b)
union(a1,b1)
```

Esercitatevi con le funzioni `union` , `intersect` , `setdiff` e `setxor` . (L'insieme vuoto si rappresenta, naturalmente, come il vettore vuoto `[ ]`).

Esempio 3.1 *Vediamo un programma di tipo function che, dati due insiemi A e B calcola l'insieme prodotto cartesiano di A e B.*

```
% programma di tipo function
% dati due insiemi a e b, visualizza l'insieme a×b
function p = prodottocartesiano(a,b)
a1=unique(a);
b1=unique(b);
p=[];
for x=a1
   for y=b1
     p = [p , [x,y]'];
   end
end
```

Nella quinta riga vediamo un uso di `for` *leggermente diverso da quello visto in precedenza. Dopo il segno di uguaglianza non compaiono i due estremi tra cui deve variare* `x` *ma il nome di un insieme, ovvero* `a1`*. Questo vuol dire che la variabile* `x` *deve assumere come valori tutti quelli degli elementi dell'insieme* `a1`*. Una cosa del tutto analoga viene fatta nella riga successiva.*
Nella settima riga è la parte centrale del programma. In essa si ridefinisce l'insieme `p` *aggiungendogli ad ogni passo dei due cicli* `for` *un nuovo elemento, dato in forma di vettore colonna, dell'insieme prodotto cartesiano di* `a` *e* `b`*.*

3.2 Aritmetica

Scriviamo un programma che usa l'algoritmo euclideo delle divisioni successive per determinare il massimo comun divisore tra due interi.
Dati due numeri interi a e b, effettuiamo la divisione di a per b (se $b \neq 0$; se $b = 0$ allora $a = MCD(a, b)$) $a = b \cdot q + r$. Si verifica facilmente che $MCD(a, b) = MCD(b, r)$. Se il resto r è diverso da 0, scambiamo il dividendo a con il divisore b e il divisore b con il resto r ed effettuiamo una nuova divisione. Procediamo in questo modo fino a che non

otteniamo resto nullo. Il massimo comun divisore di a e b è l'ultimo resto non nullo. In altri termini: è il dividendo della divisione che non possiamo effettuare poiché il divisore è 0.
All'inizio del programma viene inserito un test che escluda il caso $a = b = 0$ (notate l'uso del comando `return`).

Esempio 3.2
```
function d = mcd(a,b)
if a==0 & b==0
    'mcd non definito'
    return
end
dividendo = a;
divisore = b;
while divisore ~= 0
    R = resto(dividendo, divisore);
    dividendo = divisore;
    divisore = R;
end
d = abs(dividendo);
```

Possiamo sfruttare la funzione `mcd` appena scritta per trovare il massimo comun divisore di tre numeri interi (che vengono dati come le tre componenti del vettore `lista3`):

Esempio 3.3
```
function d = mcd3(lista3)
%MCD3 massimo comun divisore di tre numeri interi
d = mcd(lista3(1),lista3(2));
d = mcd(d,lista3(3));
```

Esercizio 3.1 **Scrivere una funzione che calcoli il massimo comun divisore di quattro numeri interi. Sapreste farlo per un numero qualunque di interi?*

Esercizio 3.2 **Scrivere una funzione che calcoli il minimo comune multiplo (positivo) di due numeri interi.*

L'algoritmo euclideo generalizzato permette di calcolare, dati due interi positivi a e b, una coppia di interi x e y tali che

$$x \cdot a + y \cdot b = MCD(a, b)\,.$$

Ricordiamo il procedimento di calcolo dei coefficienti x e y. Poniamo $a_1 = a$, $a_2 = b$ ed effettuiamo le divisioni successive come nell'algoritmo euclideo visto sopra:

$$\begin{aligned}
a_1 &= q_1 a_2 + a_3 \\
a_2 &= q_2 a_3 + a_4 \\
&\vdots \\
a_i &= q_i a_{i+1} + a_{i+2} \\
&\vdots \\
a_{n-2} &= q_{n-2} a_{n-1} + a_n \\
a_{n-1} &= q_{n-1} a_n + 0 (= a_{n+1})
\end{aligned}$$

con a_i, q_i interi, $|a_2| > a_3 > \ldots > a_n > a_{n+1} = 0$.
Osserviamo che valgono le relazioni:

$$(A) \quad \begin{cases} a_1 &= 1 \cdot a + 0 \cdot b \\ a_2 &= 0 \cdot a + 1 \cdot b \end{cases}$$

e, per $i \geq 1$,

$$a_{i+2} = x_{i+2} \cdot a + y_{i+2} \cdot b$$

dove

$$(B) \quad \begin{cases} x_{i+2} &= x_i - q_i x_{i+1} \\ y_{i+2} &= y_i - q_i y_{i+1} \end{cases}$$

Supponendo che a_n sia l'ultimo resto non nullo, abbiamo $MCD(a,b) = a_n = x_n \cdot a + y_n \cdot b$, quindi dobbiamo determinare x_n e y_n.
Da (A) abbiamo $x_1 = 1$, $y_1 = 0$ e $x_2 = 0$, $y_2 = 1$. Per $i \geq 3$ da (B) segue che x_i e y_i sono calcolati in termini dei valori x_j, y_j (e q_j) ottenuti nei passi precedenti, precisamente per $j = i - 1$ e $j = i - 2$.

Utilizziamo lo schema appena visto per scrivere una funzione che implementa l'algoritmo euclideo generalizzato. Utilizzeremo i programmi `quoziente` e `resto` dell'Esempio 2.15 e dell'Esercizio 3.1.
Introduciamo due vettori, `U` e `V`. Cominciamo con l'inizializzazione `V` $= [1,0] (= [x_1, y_1])$, `U` $= [0,1] = ([x_2, y_2])$ e poniamo: dividendo $= a (= a_1)$, divisore $= b (= a_2)$.
Se il divisore $(= b)$ è $\neq 0$, procediamo ad effettuare la prima divisione con resto $a_1 = q_1 a_2 + a_3$. Utilizziamo un vettore "di servizio" `T` per conservare il valore di `U` e assegnamo quindi ad `U` i valori in `V` $= [x_2, y_2]$, mentre in `V` vengono memorizzati i nuovi valori $[x_3, y_3]$. Scambiamo infine, per il ciclo successivo, il dividendo con il divisore e il divisore con il resto.
Ripetiamo tale ciclo di istruzioni fintanto che il divisore è diverso da 0.
In questo modo alla $j-$esima esecuzione del ciclo `while` in `V` sono memorizzati $[x_{j+2}, y_{j+2}]$ ed in `U` i valori $[x_{j+1}, y_{j+1}]$ calcolati al passo precedente.
L'ultima divisione $a_{n-1} = q_{n-1} a_n + 0$ (supponendo $a_n \neq 0$, $a_{n+1} = 0$) viene eseguita nell' $n-1$-esimo ciclo `while` e quindi gli interi cercati sono memorizzati in `U` $= [x_n, y_n]$.

```
function C = coeff_mcd(a,b)
%COEFF_MCD(a,b), a,b interi, calcola una coppia di interi [x,y]
%tali che x*a + y*b = MCD(a,b)(massimo comun divisore positivo) usando
%l'algoritmo euclideo generalizzato
U = [1,0];
V = [0,1];
dividendo = a;
divisore = b;
while divisore ~=0
    q = quoziente(dividendo, divisore);
    r = resto(dividendo, divisore);
    T = U;
    U = V;
    V = T - (q * V);
```

```
    dividendo = divisore;
    divisore = r;
end
C = U;
```

Esercizio 3.3 *Scrivere un programma che, dati tre interi a, b e c, determini se l'equazione diofantea $ax + by = c$ ha soluzioni e, in tal caso, ne calcoli una.*

3.3 Cifre e cambi di base per la scrittura dei numeri interi

In questo paragrafo scriveremo un programma che ci permette di conoscere l'espressione di un numero intero in una base qualsiasi (ad esempio nella base binaria).
Il primo passo consiste, nel far *riconoscere* al computer le *cifre* di un numero dato n; più precisamente scriveremo una funzione che abbia come argomento un numero naturale e che crei un vettore le cui componenti sono le cifre del numero. Prima di vedere il programma cerchiamo di spiegare il principio su cui si basa.
Se n ha $k+1$ cifre: $n = c_k c_{k-1} \dots c_1 c_0$, questo vuol dire esattamente che

$$n = 10^k c_k + 10^{k-1} c_{k-1} + \cdots + 10^1 c_1 + 10^0 c_0 = \sum_{i=0}^{k} 10^i c_i \,.$$

Ad esempio

$$3437 = 1000 \cdot 3 + 100 \cdot 4 + 10 \cdot 3 + 1 \cdot 7 \,.$$

La cifra delle unità c_0 è il resto della divisione di n per 10 (e dunque è facilmente calcolabile tramite la funzione `resto`). Se adesso definiamo

$$n_1 = (n - c_0)/10 \,,$$

abbiamo un nuovo numero naturale la cui ultima cifra coincide con la penultima cifra di n, dunque c_1 è il resto della divisione di n_1 per 10. Analogamente posto

$$n_2 = (n_1 - c_1)/10 \,,$$

si ha che c_2 è il resto della divisione di n_2 per 10. E' chiaro che si può iterare il procedimento definendo ricorsivamente i numeri n_i

$$n_i = (n_{i-1} - c_{i-1})/10 \,,$$

e trovando la i-esima cifra di n come il resto della divisione per 10 di n_i. La successione degli n_i decresce, anzi si può affermare che ciascun termine ha una cifra in meno del precedente. In particolare n_k coincide con c_k e n_{k+1} è nullo come tutti i termini successivi. Il procedimento che abbiamo descritto è alla base del programma che segue.

Esempio 3.4 `function L = cifre(n)`

```
% CIFRE(N) produce la lista delle cifre nella espressione decimale
% del numero naturale n
L = [ ];
if n==0
    L = [0];
end
while n > 0
    cifra = resto(n,10);
    L = [cifra, L];
    n = (n -cifra)/10;
end
```

Il vettore `L`, inizializzato come vettore *vuoto*, ha come componenti dopo l'esecuzione del programma le cifre di `n`. Il valore di `n` viene aggiornato ad ogni iterazione del ciclo `while` e assume via via i valori n_i; le iterazioni si arrestano quando `n` diventa uguale a 0. Prima dell'esecuzione del ciclo `while` viene controllato se il numero `n` è zero: in questo caso le istruzioni nel ciclo non vengono eseguite e viene posto `L = [0]`.

A questo punto è più semplice affrontare il problema della scrittura di un numero in una base arbitraria; prendiamo ad esempio la base 2. Se n si scrive in base due come $n = c_k \dots c_0$, questo vuol dire

$$n = 2^k c_k + 2^{k-1} c_{k-1} + \cdots + 2^1 c_1 + 2^0 c_0 = \sum_{i=0}^{k} 2^i c_i \,.$$

e dunque, ripetendo le considerazioni fatte nel caso della base 10, c_0 è la classe di resto di n modulo due, c_1 è la classe di resto di $(n - c_0)/2$ modulo due, e così via.

Esercizio 3.4 **Scrivere una funzione che, dato un numero naturale (in base 10), ne scriva le cifre in base 2.*

Esercizio 3.5 **Scrivere una funzione che, dato un numero n naturale e un secondo numero naturale b che ha il ruolo di base, scriva le cifre di n in base b.*

3.4 Un problema sulle cifre dei numeri interi

In una delle passate edizioni delle selezioni per le Olimpiadi della matematica è stato dato un problema che può essere riassunto così.

Problema. *Ad ogni numero naturale n se ne può associare un altro che la somma delle cifre di n, poi si possono sommare le cifre del nuovo numero e poi quelle del numero ottenuto, fino ad avere un numero di una sola cifra; ad esempio*

$$9965 \to 9 + 9 + 6 + 5 = 29 \to 2 + 9 = 11 \to 1 + 1 = 2 \,.$$

Se applichiamo questo procedimento a tutti i numeri compresi tra 1 e 1000, qual'è la cifra che compare più frequentemente come risultato?

Vi proponiamo di risolvere questo problema scrivendo alcuni programmi.

Esercizio 3.6 *Scrivete una funzione* `scifre` *che, dato un numero n, ne calcoli la somma delle cifre.*

(Suggerimento: usate il programma, visto nel paragrafo precedente, che associa ad n il vettore delle sue cifre in base 10 e la funzione dell'esercizio 2.22)

Esercizio 3.7 *Utilizzando la funzione* `scifre` *scrivete una funzione* `sommacifre` *che, dato un numero naturale n, itera la somma delle sue cifre fino ad ottenere un numero compreso tra 0 e 9, come indicato nel problema.*

Esercizio 3.8 *Scrivete una funzione che, dato un numero naturale n, calcoli quante volte si ottiene ogni numero compreso tra* 0 *e* 9 *applicando la funzione* `sommacifre` *dell'esercizio 3.7 a tutti gli interi compresi tra* 1 *e n.*

(Suggerimento: esprimete il risultato come un vettore di lunghezza 9, la cui componente i-esima sia il numero di volte che si ottiene la cifra i applicando `sommacifre` ai numeri compresi tra 1 e n).

La soluzione del problema posto all'inizio del paragrafo si ottiene, ora, applicando la funzione dell'esercizio 3.8 a $n = 1000$. Sapreste risolvere il problema usando invece la teoria delle congruenze ?

3.5 Aritmetica modulo n

Dato un intero positivo n, identifichiamo gli elementi dell'anello delle classi di resto modulo n

$$\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z} = \{[i] = i + n\mathbb{Z} \mid i = 0, 1, \ldots, n-1\}$$

con i rispettivi rappresentanti $0, 1, \ldots, n-1$.
Per $a, b \in \mathbb{Z}_n = \{0, 1, \ldots, n-1\}$ abbiamo quindi che il prodotto $ab = (a + n\mathbb{Z})(b + n\mathbb{Z}) = ab + n\mathbb{Z} = r + n\mathbb{Z}$ (dove $0 \leq r \leq n-1$ è il resto della divisione di ab per n) è dato da

```
resto(ab,n)
```

e la somma $a + b$ da

```
resto(a+b,n)
```

.

Esercizio 3.9 *Calcolate somme e prodotti di* $15, 25, 32$ *modulo, ad esempio,* 51 *e* 123*.*

Determiniamo ora la "tabella additiva" di $\mathbb{Z}_6$, ovvero la matrice 6×6 $A = (a_{ij})$ dove, per $1 \leq i, j \leq 6$, l'elemento a_{ij} è il rappresentante modulo 6 della somma delle classi (rappresentate da) $i - 1$ e $j - 1$.

Esempio 3.5 `%tabella additiva di Z6`

```
A = zeros(6,6);
    for i = 1:6
        for j= 1:6
            A(i,j) = resto((i-1)+(j-1),6);
        end
    end
A
```

Esercizio 3.10 *Scrivere la "tabella moltiplicativa" di* $\mathbb{Z}_6$.

Esercizio 3.11 *Scrivere due funzioni che, dato un intero positivo* n*, calcolino la tabella additiva e moltiplicativa di* $\mathbb{Z}_n$.

Se $a \in \mathbb{Z}_n$, sappiamo che a è invertibile se e solo se $(a, n) = 1$, ovvero se esistono $x, y \in \mathbb{Z}$ tali che $xa + yn = 1$: in tal caso $x = (x + n\mathbb{Z})$ è l'inverso di $a(= a + n\mathbb{Z})$ in $\mathbb{Z}_n$.

Esercizio 3.12 **Scrivete un programma che, dati* $a, b \in \mathbb{Z}$*,* $n > 0$*, dica se* $a(= a + n\mathbb{Z})$ *è invertibile in* $\mathbb{Z}_n$ *e, in tal caso, ne calcoli l'inverso.*

Esercizio 3.13 **Scrivere una funzione che, dato un intero positivo* n*, determini gli elementi invertibili di* $\mathbb{Z}_n$.

Capitolo 4

Spazi vettoriali e sistemi lineari

4.1 Matrici a scalini, la funzione `rank`

Esercizio 4.1 **(preliminare). Sia v un vettore riga di $\mathbf{R}^n$; definite*

$$pn(v) = max\{i|\ v_j = 0\ per\ j \leq i\}$$

Fate un programma di tipo function che vi calcoli pn.

Suggerimento:
• Chiamate k la lunghezza di v (utilizzate il comando `size(· , 2)`).
• Ponete una certa variabile, ad esempio u, uguale a zero.
• Fate un ciclo, facendo variare i da 1 a k, che incrementi u di 1 se il coefficiente i-esimo di v è zero e che interrompa il programma se è diverso da zero (con il comando `break`).

Esercizio 4.2 **Utilizzando il programma precedente, fate un programma per vedere se una matrice è a scalini.*

Suggerimento:
• Chiamate m il numero di righe di A (utilizzate il comando `size`).
• Ponete una certa variabile, ad esempio r, uguale a zero.
• Sia $A^{(i)}$ la riga i-esima di A. Fate un ciclo facendo variare i da 1 a $m-1$ che incrementi r di 1 se i primi $pn(A^{(i)})+1$ elementi di $A^{(i+1)}$ non sono tutti nulli e lasci r uguale a r se invece i primi $pn(A^{(i)})+1$ elementi di $A^{(i+1)}$ sono tutti nulli.
• Dopo che il ciclo è stato completamente eseguito, se $r = 0$ la matrice è a scalini, se r è maggiore di zero la matrice non è a scalini. Utilizzando questo fatto e il comando `if` potete terminare il programma.

La funzione `rref` riduce una matrice arbitraria in una matrice in forma a scalini i cui primi elementi diversi da zero di ogni riga sono uguali a 1..., facendo operazioni elementari di riga.
La funzione `rrefmovie` fa la stessa cosa mostrandovi i vari passaggi.

Esercizio 4.3 *Una matrice A si dice ridotta a scalini per colonne se soddisfa la seguente condizione:*
se i primi k coefficienti di una colonna sono nulli allora almeno i primi $k+1$ coefficienti della colonna successiva sono nulli (cioè se è la trasposta di una matrice a scalini "classica" cioè a scalini per righe).
Facendo operazioni elementari di colonne si può ridurre una matrice in forma a scalini per colonne. Fate ciò con matlab utilizzando il comando `rref` *e la trasposta.*

Un'altra funzione che Matlab fornisce è `rank`; essa calcola il rango di una matrice. Calcolate:

```
rank(ones(4,5))
```

```
rank(eye(6))
```

Inserite adesso altre matrici, calcolatene il rango; poi controllate che il rango di ciascuna è uguale al numero di scalini della matrice ottenuta da essa riducendo a scalini.

4.2 Sistemi lineari

La function `null(·)` applicata a un matrice A trova una base dello spazio vettoriale delle soluzioni del sistema lineare omogeneo la cui matrice dei coefficienti è A
N.B. Se inserite matrici a coefficienti interi e volete soluzioni "più belle", scrivete invece di `null(·)`

```
null(· , 'r')
```

Ad esempio per trovare una base dello spazio vettoriale delle soluzioni del sistema lineare $Ax = 0$ con

$$A = \begin{pmatrix} 2 & -1 & 4 \\ 1 & 0 & 1 \\ 3 & 0 & 3 \end{pmatrix}$$

scrivete (dopo aver inserito la matrice A)

```
null(A,'r')
```

Vogliamo adesso vedere come si risolve con Matlab un sistema lineare non omogeneo $Ax = b$ (dove b è un vettore colonna $m \times 1$ e A una matrice $m \times n$).

Per vedere se il sistema ha soluzione potete controllare se il rango della matrice che si ottiene affiancando A e b è uguale a quello di A. Se i due ranghi sono uguali allora, per il Teorema di Rouché-Capelli, esiste una soluzione; altrimenti non esiste.
Se una soluzione esiste, l'insieme di tutte le soluzioni di $Ax = b$ si ottiene sommando una soluzione particolare a tutte le soluzioni del sistema linaere omogeneo $Ax = 0$ (una cui base si trova scrivendo `null(A)` o `null(A,'r')`).
Se un sistema ha soluzione, per trovare una particolare soluzione con Matlab si può fare nel seguente modo:
se il rango di A è uguale al minimo fra il numero di righe e il numero di colonne di A allora una soluzione particolare si trova scrivendo:

$x = A\backslash b$

in genere per trovare una soluzione particolare potete fare così:
riducete a scalini la matrice completa associata al sistema; togliete le righe nulle e, (con il comando \ ), trovate una soluzione particolare del sistema associato alla matrice che avete ottenuto.
(**N.B.** Se inserite matrici a coefficienti interi e volete soluzioni "più belle", prima di inziare a fare i conti scrivete

```
format rat
```

(`rat` sta per razionale).)

Provate a risolvere con Matlab i seguenti sistemi lineari:

$$\begin{cases} x - 2y + z = 3 \\ -x + 5y + z = 1 \\ x + 4y + z = 2 \end{cases}$$

$$\begin{cases} x - 2y - 2z = 3 \\ -x + 2y - 2z = -2 \\ -2x + 4y + z = 2 \end{cases}$$

$$\begin{cases} x + y - z + w = 4 \\ -x + y + z + w = 0 \\ -2x + y + 2z + w = -2 \end{cases}$$

Esercizio 4.4 *Dite se il vettore* $v = \begin{pmatrix} 1 \\ 0 \\ 1 \\ -7 \end{pmatrix}$ *di* $\mathbf{R}^4$ *appartiene al sottospazio generato dai seguenti vettori:*

$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 5 \end{pmatrix} \begin{pmatrix} -9 \\ 3 \\ 2 \\ 4 \end{pmatrix} \begin{pmatrix} 0 \\ 3 \\ 0 \\ 5 \end{pmatrix}$$

cioè se esistono x_1, x_2, x_3 numeri reali tali che

$$x_1 \begin{pmatrix} 1 \\ 0 \\ 1 \\ 5 \end{pmatrix} + x_2 \begin{pmatrix} -9 \\ 3 \\ 2 \\ 4 \end{pmatrix} + x_3 \begin{pmatrix} 0 \\ 3 \\ 0 \\ 5 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ -7 \end{pmatrix}$$

Ovviamente questo è vero se e solo se il seguente sistema lineare ha soluzione:

$$\begin{pmatrix} 1 & -9 & 0 \\ 0 & 3 & 3 \\ 1 & 2 & 0 \\ 5 & 4 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 7 \end{pmatrix}$$

(quindi per rispondere potete utilizzare il Teorema di Rouché-Capelli e la funzione `rank`*).*

Esercizio 4.5 *Dite se i seguenti vettori di $\mathbf{R}^4$ sono linearmente indipendenti e se sono linearmente dipendenti trovate una loro combinazione lineare nulla con coefficienti non tutti nulli:*

$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 2 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 3 \\ 1 \\ 2 \\ 6 \end{pmatrix}$$

(Sono linearmente indipendenti se e solo se l'unica soluzione di

$$x_1 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 5 \end{pmatrix} + x_2 \begin{pmatrix} 1 \\ 1 \\ 2 \\ 1 \end{pmatrix} + x_3 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + x_4 \begin{pmatrix} 3 \\ 1 \\ 2 \\ 6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

è $x_1 = x_2 = x_3 = x_4 = 0$ cioè se e solo se il sistema lineare

$$\begin{pmatrix} 0 & 1 & 1 & 3 \\ 0 & 1 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 5 & 1 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

ha solo la soluzione $x_1 = x_2 = x_3 = x_4 = 0$.)

Esercizio 4.6 *Dite se il vettore $v = \begin{pmatrix} 1 \\ 0 \\ 2 \\ 0 \\ 0 \end{pmatrix}$ di $\mathbf{R}^5$ appartiene al sottospazio generato dai seguenti vettori:*

$$\begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \\ 2 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 3 \\ 3 \\ 0 \\ 0 \end{pmatrix}$$

Esercizio 4.7 *Dite se i seguenti vettori di* $\mathbf{R}^3$ *sono linearmente indipendenti e se sono linearmente dipendenti trovate una loro combinazione lineare nulla con coefficienti non tutti nulli:*

$$\begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix} \begin{pmatrix} 3 \\ 2 \\ 4 \end{pmatrix} \begin{pmatrix} 4 \\ 3 \\ 4 \end{pmatrix}$$

4.3 Inversa di una matrice

Esercizio 4.8 *Inserite la seguente matrice invertibile A*

$$\begin{pmatrix} 1 & 2 & 0 & 0 \\ 1 & 3 & 13 & 0 \\ 0 & 0 & 1 & 1 \\ 2 & 0 & 0 & -1 \end{pmatrix}$$

Utilzzando la concatenazione e la funzione `rref`*, calcolate l'inversa di A.*

In realtà Matlab fornisce già una funzione per calcolare l'inversa di una matrice: `inv`.

Esercizio 4.9 *Risolvete il sistema lineare* $Ax = b$ *dove A è la matrice dell'esercizio precedente e b è il vettore:*

$$\begin{pmatrix} 2 \\ 45 \\ 0 \\ 2 \end{pmatrix}$$

4.4 Basi di sottospazi vettoriali, estrazione di basi da un insieme di generatori, estensione di insiemi di vettori indipendenti a basi di $\mathbf{R}^n$

Esercizio 4.10 *Sia W il sottospazio vettoriale di* $\mathbf{R}^4$ *generato dalle colonne della seguente matrice*

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 3 & 3 \\ 1 & 2 & 0 \\ 5 & 0 & -10 \end{pmatrix}$$

i) Calcolare la dimensione di W.
ii) Trovare una base di W.

(Per risolvere ii potete fare operazioni elementari di colonna fino ad ottenere una matrice a scalini per colonna; le colonne non nulle di tale matrice costituiscono una base del sottospazio generato dalle colonne della matrice iniziale, infatti sono linearmente indipendenti e facendo operazione elementari di colonna non si altera il sottospazio generato dalle colonne di una matrice).

Esercizio 4.11 **Fate un programma per estrarre una base da un insieme di generatori di un sottospazio di $\mathbf{R}^m$; precisamente sia A una matrice $m \times n$; considerate il sottospazio vettoriale di $\mathbf{R}^m$ generato dalle colonne di A; scrivete un programma che estragga una base di $< A_{(1)}, ..., A_{(n)} >$ da $\{A_{(1)}, ..., A_{(n)}\}$*

Esercizio 4.12 *Fate un programma che estenda un insieme di vettori di $\mathbf{R}^m$ linearmente indipendenti ad una base di $\mathbf{R}^m$.*

Suggerimento:
sia B la matrice che si ottiene affiancando l'insieme di vettori linearmente indipendenti dato; affiancate a tale matrice la matrice identità, chiamate A la matrice così ottenuta e utilizzate il programma precedente.

Esercizio 4.13 *Sia S il sottospazio di $\mathbf{R}^4$ generato da*

$$v_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 2 \\ 1 \\ 0 \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ 4 \\ 4 \\ 5 \end{pmatrix}, v_4 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Estrarre una base di S da $\{v_1, v_2, v_3, v_4\}$ (cioè trovare una base di S contenuta in $\{v_1, v_2, v_3, v_4\}$).

Esercizio 4.14 *Estendere ad un base di $\mathbf{R}^4$ il seguente insieme di vettori:*

$$v_1 = \begin{pmatrix} 1 \\ 2 \\ -2 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 3 \\ -1 \\ 1 \\ 0 \end{pmatrix}$$

Esercizio 4.15 *Sia S il sottospazio di $\mathbf{R}^5$ generato da*

$$v_1 = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 1 \\ 3 \\ 1 \\ 2 \\ 12 \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 5 \\ 5 \end{pmatrix}, v_4 = \begin{pmatrix} 5 \\ 5 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Estrarre una base di S da $\{v_1, v_2, v_3, v_4\}$ (cioè trovare una base di S contenuta in $\{v_1, v_2, v_3, v_4\}$).

Esercizio 4.16 *Estendere ad un base di $\mathbf{R}^5$ il seguente insieme di vettori:*

$$v_1 = \begin{pmatrix} 3 \\ 2 \\ -2 \\ 3 \\ 0 \end{pmatrix}, v_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

4.5 Estrazione di basi da un insieme di generatori, estensione di insiemi di vettori indipendenti a basi di $\mathbf{R}^n$ II

Come abbiamo già visto il comando `rref` riduce a scalini una matrice con operazioni elementari di righe. Quindi se si scrive

```
R = rref(A)
```

R è una matrice a scalini ottenuta da A con operazioni elementari di righe.
Osservate che se invece si scrive

```
[R, jb] = rref(A)
```

si ottengono due matrici: la matrice R che è una matrice a scalini ottenuta da A con operazioni elementari di righe e una riga i cui coefficienti sono gli indici delle colonne dove compaiono i pivots in R.
Quindi se abbiamo un sottospazio W di $\mathbf{R}^n$ generato da certi vettori colonna e vogliamo estrarre una base da tale insieme di generatori, possiamo fare nel seguente modo: affiancare questi vettori colonna formando una matrice che chiamiamo A, poi possiamo scrivere

```
[R, jb] = rref(A)
A(:, jb)
```

l'ultima riga produce una matrice le cui colonne sono le colonne di A aventi come indici gli elementi di jb. Quindi le colonne di questa ultima matrice sono un base di W estratta dall'insieme di generatori iniziale.

Esercizio 4.17 *Trovate una base del sottospazio vettoriale di* $\mathbf{R}^3$ *generato dalle colonne della seguente matrice*

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 3 & 3 \\ 1 & 2 & 0 \end{pmatrix}$$

Esercizio 4.18 *Sia* S *il sottospazio di* $\mathbf{R}^4$ *generato da*

$$v_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 2 \\ 1 \\ 0 \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ 4 \\ 4 \\ 5 \end{pmatrix}, v_4 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

Estrarre una base di S *da* $\{v_1, v_2, v_3, v_4\}$ *(cioè trovare una base di* S *contenuta in* $\{v_1, v_2, v_3, v_4\}$*).*

Esercizio 4.19 *Estendere ad un base di* $\mathbf{R}^4$ *il seguente insieme di vettori:*

$$v_1 = \begin{pmatrix} 1 \\ 2 \\ -2 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 3 \\ -1 \\ 1 \\ 0 \end{pmatrix}$$

Esercizio 4.20 *Sia* S *il sottospazio di* $\mathbf{R}^5$ *generato da*

$$v_1 = \begin{pmatrix} 0 \\ 2 \\ 0 \\ 2 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 1 \\ 3 \\ 1 \\ 2 \\ 12 \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 5 \\ 5 \end{pmatrix}, v_4 = \begin{pmatrix} 5 \\ 5 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

Estrarre una base di S *da* $\{v_1, v_2, v_3, v_4\}$ *(cioè trovare una base di* S *contenuta in* $\{v_1, v_2, v_3, v_4\}$*).*

Esercizio 4.21 *Estendere ad un base di* $\mathbf{R}^5$ *il seguente insieme di vettori:*

$$v_1 = \begin{pmatrix} 3 \\ 2 \\ -2 \\ 3 \\ 0 \end{pmatrix}, v_2 = \begin{pmatrix} 4 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

Esercizio 4.22 *Fare un programma per estendere ad un base di* $\mathbf{R}^n$ *un insieme di vettori indipendenti dato dalle colonne di una matrice* C*.*

4.6 Forme parametriche e cartesiane di sottospazi vettoriali

Esercizio 4.23 *Calcolate una base del seguente sottospazio vettoriale di* $\mathbf{R}^4$*:*

$$B = \{(x_1, x_2, x_3, x_4) \in \mathbf{R}^4 |\ x_1 + 3x_4 = 0\}$$

(ricordate che la function `null` applicata a un matrice A trova una base dello spazio vettoriale delle soluzioni del sistema lineare omogeneo la cui matrice dei coefficienti è A; il comando `null(·,'r')` vi dà soluzioni "più" belle se inserite matrici a coefficienti interi).

Esercizio 4.24 *Trovate l'espressione cartesiana del sottospazio di* $\mathbf{R}^4$ *generato dai seguenti vettori:*

$$v_1 = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

Esercizio 4.25 *Sia W il seguente sottospazio vettoriale di $\mathbf{R}^5$:*

$$W = \{(x_1, x_2, x_3, x_4, x_5) \in \mathbf{R}^5 |\ x_1 + 2x_4 = 0,\ \ 2x_2 + x_3 = 0\ \ -2x_1 + 2x_2 + x_3 - 4x_4 = 0\}$$

i) Calcolare la dimensione di W senza utilizzare il comando `null`.
ii) Trovare una base di W.

Esercizio 4.26 *Trovate l'espressione cartesiana del sottospazio di $\mathbf{R}^5$ generato dai seguenti vettori:*

$$v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 5 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} v_3 = \begin{pmatrix} 0 \\ 1 \\ 3 \\ 3 \\ 3 \end{pmatrix}$$

4.7 Somme, intersezioni, unioni di sottospazi vettoriali

Esercizio 4.27 *Trovare l'intersezione dei due seguenti sottospazi di $\mathbf{R}^4$:*

$$W = < \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 5 \end{pmatrix} >$$

$$Z = < \begin{pmatrix} -5 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 4 \\ 0 \\ -1 \\ -1 \end{pmatrix} >$$

Esercizio 4.28 *Sia A il sottospazio vettoriale di $\mathbf{R}^4$ generato da* $\begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix}$ *e* $\begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \end{pmatrix}$. *Sia B il sottospazio vettoriale di $\mathbf{R}^4$ generato da* $\begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$, $\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$ *e* $\begin{pmatrix} -3 \\ 0 \\ 0 \\ 1 \end{pmatrix}$. *Trovate una base del sottospazio vettoriale di $A + B$.*

Esercizio 4.29 *Fate un programma che calcoli una base della somma di due sottospazi vettoriali di $\mathbf{R}^n$ generati dalle colonne di due matrici, A e B.*

Esercizio 4.30 *Fate un programma che calcoli una base della intersezione di due sottospazi vettoriali di $\mathbf{R}^n$ dati come luoghi di zeri di due sistemi lineari omogenei $Ax = 0$ e $Bx = 0$.*

Esercizio 4.31 *Fate un programma che calcoli una base della intersezione di due sottospazi vettoriali di* $\mathbf{R}^n$ *generati dalle colonne di due matrici,* C *e* D *(una via possibile è la seguente: trovate una forma cartesiana dei due sottospazi e utilizzate l'esercizio precedente).*

Esercizio 4.32 *i) Dire se* $v \in W$ *(suggerimento:* $v \in W$ *se e solo se un certo sistema lineare ha soluzione; utilizzare il teorema di Rouché-Capelli):*

$$v = \begin{pmatrix} 1 \\ 1 \\ 2 \\ 0 \end{pmatrix},\ W = < \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 2 \\ 0 \end{pmatrix} >$$

ii) Dire se $v \in Z = \{x |\ Ax = 0\}$ *dove*

$$A = \begin{pmatrix} 0 & 0 & 0 & 7 \\ 2 & 2 & -4 & 1 \\ 1 & -1 & 0 & 0 \end{pmatrix}$$

Esercizio 4.33 *Dire se l'unione dei due seguenti sottospazi di* $\mathbf{R}^4$ *è un sottospazio (ricordatevi che l'unione di due sottospazi* W_1, W_2 *è un sottospazio se e solo se o* $W_1 \subset W_2$ *o* $W_2 \subset W_1$*; per verificare se un sottospazio è contenuto in un altro basta verificare se tutti i generatori del primo sottospazio appartengono al secondo sottospazio):*

$$W_1 = < \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} >, \quad W_2 = < \begin{pmatrix} 0 \\ 1 \\ 3 \\ 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 1 \\ 0 \end{pmatrix} >$$

Esercizio 4.34 *Fate un programma che vi dica se, dati due sottospazi* W_1 *e* W_2 *di* $\mathbf{R}^n$*, generati dalle colonne di due matrici* A *e* B*, si ha* $W_1 \subset W_2$*.*

Esercizio 4.35 *Fate un programma che vi dica se, dati due sottospazi* W_1 *e* W_2 *di* $\mathbf{R}^n$*, generati dalle colonne di due matrici* A *e* B*, si ha* $W_1 = W_2$*.*

Esercizio 4.36 *Fate un programma che vi dica se l'unione di due sottospazi di* $\mathbf{R}^n$ *(generati dalle colonne di due matrici* A *e* B*) è un sottospazio.*

Capitolo 5

Determinanti e applicazioni lineari

5.1 Determinanti

Il comando `det` calcola il determinante di una matrice quadrata ed è utilizzabile anche nel calcolo simbolico, cioè si possono calcolare i determinanti di matrici i cui coefficienti sono espressioni letterali.

Esercizio 5.1 *Calcolate il determinante delle seguenti matrici*

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & m \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0 & 0 \\ d & e & f \\ g & h & m \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ a & b & c \end{pmatrix}$$

$$\begin{pmatrix} a & b & c \\ d & e & f \\ a+d & b+e & c+f \end{pmatrix}$$

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ x & y & w & z \\ x^2 & y^2 & w^2 & z^2 \\ x^3 & y^3 & w^3 & z^3 \end{pmatrix}$$

e fattorizzate il risultato.

Esercizio 5.2 *(teorico) siano $x_1, ..., x_n$ dei numeri reali; calcolare il determinante della matrice $n \times n$ la cui colonna j-esima ha coefficienti $1, x_j, (x_j)^2, ...(x_j)^{n-1}$; tale determinante si chiama determinante di Vandermonde; l'ultimo determinante del precedente esercizio è il caso $n = 4$ del determinante di Vandermonde.*

5.2 Applicazioni lineari I

Come abbiamo più volte detto, i comandi `null(A)` e `null(A, 'r')` calcolano una base delle soluzioni del sistema lineare omogeneo $Ax = 0$ dove A è una matrice $m \times n$. Questo è equivalente a dire che trovano una base del nucleo della applicazione lineare $f_A : \mathbf{R}^n \to \mathbf{R}^m$ associata alla matrice A.

Esercizio 5.3 *Sia $f_A : \mathbf{R}^4 \to \mathbf{R}^3$ l'applicazione lineare associata dalla seguente matrice:*

$$A = \begin{pmatrix} 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 10 & 9 \end{pmatrix}$$

i) Calcolate la dimensione del nucleo e la dimensione della immagine di f_A, utilizzando solo il comando `rank`*.*
ii) Calcolate una base del nucleo e una base della immagine di f_A.
iii) Il vettore

$$\begin{pmatrix} -1 \\ 1 \\ 10 \end{pmatrix}$$

appartiene alla immagine di f_A?
iv) Il vettore

$$\begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

appartiene al nucleo di f_A?

Esercizio 5.4 *i) Dimostatrate che*

$$\begin{pmatrix} h \\ h \\ h \\ h \\ h \end{pmatrix}$$

appartiene al nucleo dell'applicazione lineare associata a

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

$\forall h \in \mathbf{R}$ *(utilizzate il calcolo simbolico).*
ii) Dimostatrate che

$$\begin{pmatrix} h \\ -h \\ h \\ -h \\ h \end{pmatrix}$$

appartiene al nucleo dell'applicazione lineare associata a

$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$\forall h \in \mathbf{R}$.

Esercizio 5.5 *Siano $f : \mathbf{R}^4 \to \mathbf{R}^3$ l'applicazione lineare data dalla seguente matrice:*

$$\begin{pmatrix} 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 10 & 9 \end{pmatrix}$$

e $g : \mathbf{R}^3 \to \mathbf{R}^3$ l'applicazione lineare data dalla seguente matrice:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

Calcolate una base di: $Ker(f)$, $Ker(g \circ f)$, $Im(g)$, $Im(g \circ f)$. Dimostrate che $Kerf \subset Ker(g \circ f)$ e $Im(g \circ f) \subset Im(g)$.

Esercizio 5.6 *(teorico, molto semplice) Dimostrate che se $f : V \to W$ e $g : W \to U$ sono due applicazioni lineari, si ha*

$$Ker(f) \subset Ker(g \circ f)$$

$$Im(g \circ f) \subset Im(g)$$

Esercizio 5.7 *i) Sia g l'applicazione lineare $\mathbf{R}^5 \to \mathbf{R}^5$ data dalla seguente matrice:*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Calcolate $Ker(g)$, $Ker(g^2)$, $Ker(g^3)$, $Ker(g^4)$.... Per quali numeri naturali s si ha $g^s = 0$? Calcolatevi (utilizzando il calcolo simbolico) $g(\begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{pmatrix})$.

ii) (Teorico) Sia g l'applicazione lineare $\mathbf{R}^n \to \mathbf{R}^n$ data dalla matrice G $n \times n$ i cui coefficienti sono tutti nulli tranne quelli della sopradiagonale che sono uguali a 1, precisamente:

$$G_{ij} = \begin{cases} 0 & j \neq i+1 \\ 1 & j = i+1 \end{cases}$$

Dimostrate che g è nilpotente, cioè una sua potenza è nulla (suggerimento osservate chi sono $g(e_1), g(e_2), ...$).

Esercizio 5.8 *i) Sia g l'applicazione lineare $\mathbf{R}^5 \to \mathbf{R}^5$ data dalla seguente matrice:*

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

i) Qual'è il più piccolo numero naturale s_1 tale che $Ker(g^{s_1}) = Ker(g^{s_1+1}) = Ker(g^{s_1+2}) = ...$?
ii) Qual'è il più piccolo numero naturale s_2 tale che $Im(g^{s_2}) = Im(g^{s_2+1}) = Im(g^{s_2+2}) = ...$?
iii) (Teorico) Sia $f : V \to W$ un'applicazione lineare fra due spazi vettoriali di dimensione finita.
Sia s_1 il più piccolo numero naturale tale che $Ker(g^{s_1}) = Ker(g^{s_1+1}) = Ker(g^{s_1+2}) = ...$
Sia s_2 il più piccolo numero naturale tale che $Im(g^{s_2}) = Im(g^{s_2+1}) = Im(g^{s_2+2}) = ...$
Dimostrare che $s_1 = s_2$.

Esercizio 5.9 **Sia A la seguente matrice:*

$$\begin{pmatrix} 1 & 2 & 3 & 7 \\ 1 & 0 & 1 & 5 \\ 1 & 1 & 2 & 6 \end{pmatrix}$$

i) Trovare una base del nucleo dell'applicazione lineare associata ad A.
ii) Trovare una matrice 4×1, B, tale che $AB = 0$.
iii) Trovare una matrice 4×3, B, tale che $AB = 0$.
iv) (Teorico) Come sono fatte le matrici B 4×3 tali che $AB = 0$? Quant'è la dimensione del seguente sottospazio vettoriale di $M(4 \times 3, \mathbf{R})$?

$$\mathcal{S} = \{B \in M(4 \times 3) | \, AB = 0\}$$

v) Fare un programma che calcoli una base di $\mathcal{S}$.
vi) (Teorico) Sia $A \in M(m \times n, \mathbf{R})$ di rango r. Quant'è la dimensione del seguente sottospazio vettoriale di $M(n \times k, \mathbf{R})$?

$$\mathcal{S}_A = \{B \in M(n \times k) | \, AB = 0\}$$

5.3 Applicazioni lineari II: cambi di base e endomorfismi diagonalizzabili

Esercizio 5.10 *Sia $f : \mathbf{R}^4 \to \mathbf{R}^3$ l'applicazione lineare associata alla seguente matrice (nelle basi canoniche):*

$$\begin{pmatrix} 1 & 2 & 3 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 10 & 9 \end{pmatrix}$$

Scrivete la matrice della applicazione f nella base di $\mathbf{R}^4$ data da

$$v_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 4 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, v_4 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}$$

e nella base di $\mathbf{R}^3$ data da

$$w_1 = \begin{pmatrix} 0 \\ 2 \\ 2 \end{pmatrix}, v_2 = \begin{pmatrix} 0 \\ 5 \\ 1 \end{pmatrix}, v_3 = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

Esercizio 5.11 *Trovare la matrice che esprime nella base canonica di $\mathbf{R}^3$ l'applicazione lineare f tale che*

$$f(\begin{pmatrix} 1 \\ 0 \\ -0.5 \end{pmatrix}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \quad f(\begin{pmatrix} 0 \\ 3 \\ 0 \end{pmatrix}) = \begin{pmatrix} 1 \\ 0 \\ -4 \end{pmatrix}, f(\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}) = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

Esercizio 5.12 *Dite se la seguente matrice a coefficienti reali è diagonalizzabile sui reali:*

$$\begin{pmatrix} 6 & 0 & 12 & -5 \\ 2 & 1 & 4 & -2 \\ 0 & 0 & 3 & 0 \\ 4 & 0 & 12 & -3 \end{pmatrix}$$

[Suggerimento:
1) Con il calcolo simbolico calcolatevi $det(A - tI)$.
2) Trovate le radici complesse di tale polinomio col comando `solve`; chiamate l la colonna delle soluzioni.
3) Controllate se tutte le soluzioni sono reali (utilizzate il comando `isreal` applicato a l; il comando `isreal` applicato ad un vettore vi dice se tutti i coefficienti sono reali oppure se ce ne è qualcuno con parte immaginaria diversa da zero).
Se non lo sono la matrice non è diagonalizzabile, se lo sono passate al punto 4..
4) Controllate per ciascun autovalore trovato che la molteplicità algebrica sia uguale a quella geometrica.]
In realtà esiste un metodo molto migliore per vedere col computer se una matrice è diagonalizzabile, basato sul cosidetto polinomio minimo.

Capitolo 6

Esercizi di riepilogo I

Esercizio 6.1 **Scrivere tutti i numeri della forma $7k + 2$, dove k è un intero realtivo compreso tra -100 e 100, estremi inclusi.*

Esercizio 6.2 **Quanti elementi ha l'insieme*

$$\{x \in \mathbf{N} : \exists\, y \in \mathbf{N} : x = y^2 - 1\,, \exists\, k \in \mathbf{N} : x = 4k\,, x \leq 10000\}\,?$$

Esercizio 6.3 **Scrivete un programma di tipo funzione, dipendente da tre numeri a, b ($a \leq b$) e $h > 0$. Il risultato del programma deve essere calcolare i valori della funzione*

$$f(x) = 2\sin(8x) - \ln(x^2 + 1)$$

su una griglia di punti equispaziati dell'intervallo $[a, b]$, di passo h (il passo è la distanza tra due punti successivi della griglia).

Esercizio 6.4 **Scrivete un programma che, dato n, calcoli*

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

Esercizio 6.5 **Scrivere un programma di tipo funzione che, dato $n \in \mathbf{N}$, calcoli i primi n termini della successione definita per ricorrenza:*

$$a_1 = 1 \quad a_n = \sin(a_{n-1}) \quad n = 2, 3, \ldots$$

Esercizio 6.6 **Scrivete un programma che, dato n, calcoli*

$$\sum_{i=1}^{n} a_i$$

dove a_i è la successione definita per ricorrenza, introdotta nell'esercizio precedente.

Esercizio 6.7 **Scrivete un programma che, dati tre numeri reali, ne trovi il massimo.*
[Suggerimento. Supponiamo che a, b e c siano i numeri dati. Fate in modo che il programma trovi prima il massimo m tra a e b, e succesivamente il massimo M tra m e c. Chiaramente M è il massimo tra a, b e c.]

Esercizio 6.8 **Basandovi sull'esercizio precedente, scrivete un programma che, dati n numeri reali, ne trovi il massimo.*
[Suggerimento. In questo caso si tratta di fare un ciclo: siano $a_1, a_2, \ldots a_n$, i numeri dati; introducete una variabile, ad esempio m, che potete porre inzialmente uguale ad a_1; fate un ciclo su un indice i, da 1 a $n-1$, al passo i-esimo aggiornate il valore di m ponendolo uguale al massimo tra il vecchio valore (di m) e a_{i+1}. In questo modo il valore finale di m è il massimo cercato. In margine a questo esercizio osserviamo che esiste una funzione predefinita di Matlab: `max`, che dato un vettore V visualizza la componente massima di V.]

Esercizio 6.9 **Scrivete un programma che determini gli accoppiamenti di un torneo all'italiana con n giocatori (n pari).*

Suggerimenti:

a) Potete rappresentare il tabellone degli incontri come una matrice $A = (a_{ij})$, $n \times n-1$, in cui le righe rappresentino i giocatori e le colonne i turni. L'elemento a_{ij} sarà il numero corrispondente al giocatore che incontrerà l'i-esimo giocatore nel j-esimo turno.

b) Ricordiamo, per comodità, la regola per calcolare gli accoppiamenti: nel turno r il giocatore x, $x \in \{1, 2, \ldots, n-1\}$, incontra y, $y \in \{1, 2, \ldots, n-1\}$, se $y \equiv r - x \pmod{n-1}$ e $y \neq x$; altrimenti x incontra n.

Esercizio 6.10 **Scrivete, usando la tabella degli accoppiamenti dell'esercizio precedente, un programma* `turno(k, n)` *che visualizzi tutti gli incontri previsti per il k-esimo turno di un torneo di n giocatori.*

Parte II

Capitolo 7

Ricerca di zeri di funzioni

In questa parte ci occupiamo del seguente problema: data una funzione $f(x)$, continua in un intervallo $[a, b]$, trovare eventuali punti x dell'intervallo in cui f si annulla, ovvero trovare le soluzioni dell'equazione:

$$f(x) = 0\,. \tag{7.1}$$

Affronteremo la questione sia dal punto di vista teorico che da quello operativo. Ovvero enunceremo risultati astratti (teoremi) che ci dicano in quali ipotesi su f il problema ammette una soluzione, vedremo dei procedimenti teorici (algoritmi) per trovare le soluzioni e infine tradurremo questi procedimenti in programmi per Matlab.

7.1 Il metodo di bisezione

Il metodo di bisezione è il più semplice procedimento che permette di attaccare l'equazione (7.1); esso viene utilizzato solitamente nella dimostrazione del teorema dell'esistenza degli zeri.
Supponiamo che f sia continua in $I = [a, b]$ e che assuma valori di segno opposto agli estremi di I, ovvero

$$f(a)f(b) \leq 0\,.$$

Il teorema del'esistenza degli zeri afferma che, in queste ipotesi, esiste almeno un punto $x_0 \in I$ tale che $f(x_0) = 0$. Il punto x_0 può essere trovato seguendo appunto il metodo di bisezione. Si costruiscono due successioni definite per ricorrenza:

$$a_0 = a\,, \quad b_0 = b\,;$$

e per $n > 1$, posto

$$c = \frac{a_{n-1} + b_{n-1}}{2}\,,$$

si pone:

$$\begin{cases} a_n = a_{n-1}\,, \quad b_n = c\,, & \text{se} \quad f(a_{n-1})f(c) \le 0\,, \\ \\ a_n = c\,, \quad b_n = b_{n-1}\,, & \text{altrimenti}\,. \end{cases}$$

Si verifica facilmente che le successioni a_n e b_n, $n \in \mathbf{N}$, hanno le seguenti proprietà: sono monotone (crescente la prima e decrescente la seconda) e sono contenute in $[a, b]$ (e quindi sono in particolare limtate). Ne segue che ammettono entrambe limite finito, appartenente ad $[a, b]$. Inoltre:

$$b_n - a_n = \frac{b_{n-1} - a_{n-1}}{2}\,, \quad \forall n \in \mathbf{N}\,,$$

da cui

$$b_n - a_n = \frac{b-a}{2^n}\,, \quad \forall n \in \mathbf{N}\,. \tag{7.2}$$

Dunque i limiti delle due successioni coincidono; indichiamo con x_0 il valore comune di questi due limiti: $x_0 \in [a, b]$. Osserviamo ora che le successioni $f(a_n)$ e $f(b_n)$ hanno ciascuna segno costante, ma opposto l'una rispetto all'altra; questa informazione, unita alla continuità di f, assicura che

$$f(x_0) = 0\,.$$

Il metodo di bisezione si presta bene ad essere tradotto in un programma in quanto è basato su un procedimento iterativo di facile descrizione. È bene comunque sottolineare fin da subito che il metodo teorico prevede infinite iterazioni, se si eccettuano i casi fortunati in cui dopo un numero finito di passi un termine della successione a_n o b_n coincide con una soluzione della (7.1); un programma invece, prevede lo svolgimento di un numero *finito* di operazioni. Usando altri termini, il metodo di bisezione prevede un passaggio al limite che nessun programma può eseguire. Quindi il programma metterà in atto il metodo solo parzialmente, ovvero calcolerà solo un numero finito N, scelto da noi, di termini delle successioni a_n e b_n, e conseguentemente non produrrà la soluzione esatta x_0 ma solo una sua approssimazione espressa da a_N oppure b_N. Il grado di approssimazione è legato a N, infatti sappiamo dalla definizione di limite che le successioni a_n e b_n diventano *arbitrariamente vicine* x_0, per n sufficientemente grande. Indichiamo con $\epsilon(N)$ l'errore che commettiamo rimpiazzando x_0 con, ad esempio, a_N (ma la scelta di b_N sarebbe del tutto equivalemnte):

$$\epsilon(N) = |x_0 - a_N|\,, \quad N \in \mathbf{N}\,.$$

Osserviamo che x_0 è compresa tra a_N e b_N per ogni N, dunque

$$\epsilon(N) \le b_N - a_N\,, \quad N \in \mathbf{N}\,. \tag{7.3}$$

Dunque per calcolare x_0 con precisione ϵ, possiamo scrivere un programma che calcoli i primi N valori delle successioni a_n e b_n, dove N è tale che

$$b_N - a_N \le \epsilon\,, \tag{7.4}$$

e prendere come valore approssimato di x_0, a_N, b_N oppure

$$\frac{a_N + b_N}{2}\,.$$

Poichè a_n e b_n hanno lo stesso limite (ovvero la differenza tende a zero) la condizione (7.4) sarà verificata dopo un numero finito di passi N. Osserviamo che, utilizzando la (7.3), N potrebbe essere facilmente calcolato in dipendenza di ϵ, a e b. Come vedremo nel programma che segue, questo non è necessario. Infatti grazie al comando `while` possiamo fare in modo che Matlab cominci a calcolare i primi termini delle successioni a_n e b_n e prosegua finchè non risulta verificata la condizione (7.4).

Programma che implementa il metodo di bisezione per la funzione $\sin(x)$*, in un intervallo arbitrario, con grado di approssimazione arbitrario*

```
function c=bisezione(a,b,e)
if b-a<e
  c=(b+a)/2;
  return
end
while b-a>=e
  c=(b+a)/2;
  if sin(c)*sin(a)<=0
    b=c;
  else
    a=c;
  end
end
```

Vediamo ora una forma *più evoluta* del precedente programma. Le novità sono due: la funzione a cui si applica il metodo è uno degli argomenti del programma, inoltre all'inizio del programma viene eseguito un controllo sui dati per verificare se sono compatibili con il teorema dell'esistenza degli zeri.

Programma che implementa il metodo di bisezione per una funzione arbitraria

```
function c=bisezione2(f,a,b,e)
fa=feval(f,a);
fb=feval(f,b);
if fa*fb>0
  disp('dati non accettabili')
  return
end
c=(b+a)/2;
while b-a>=e
  c=(b+a)/2;
  fa=feval(f,a);
  fc=feval(f,c);
  if fa*fc<=0
```

```
    b=c;
  else
    a=c;
  end
end
```

Nel programma è presente un nuovo comando, `feval`, che permette di calcolare il valore di una funzione in un punto. La sua *sintassi* è

`feval(`*nome della funzione*`,` *valore*`)`

e la funzione deve essere identificata con il suo nome e non con la sua espressione.
Se si vuole far girare il precedente programma, ad esempio, alla funzione arcotangente nell'intervallo $[-2, 1]$, con $\epsilon = 0.001$, si deve scrivere:

```
>> bisezione2('atan',-1,2,.001)
```

Ovvero il nome della funzione deve essere messo tra virgolette. Se si vuol applicare il programma ad una funzione che non è tra quelle predefinite da Matlab, ad esempio un polinomio:

$$f(x) = x^3 - 3x^2 - 1\,, \quad x \in [0, 10]\,,$$

si può usare il comando `inline`:

```
>> f=inline('x*x*x-3*x*x-1');
>> bisezione2(f,0,10,.001)
```

7.2 L'algoritmo di Erone

L'algoritmo di Erone è un procedimento ricorsivo che serve a calcolare la radice quadrata di un numero reale positivo a, ovvero l'unica radice positiva dell'equazione

$$f(x) := x^2 - a = 0\,, \quad a > 0\,.$$

Osserviamo subito che a questo problema potremmo applicare il metodo di bisezione appena visto; per fare questo è sufficiente trovare un intervallo I tale che la funzione f abbia valori di segno discorde agli estremi: se $a \in (0, 1)$, $I = [0, 1]$ serve allo scopo, mentre se $a > 1$, si può prendere ad esempio $I = [1, a]$.
Il metodo che vedremo ha il vantaggio di *convergere più rapidamente* alla soluzione. L'algoritmo è basato su una successione definita per ricorrenza nel modo seguente:

$$\begin{cases} x_0 = \text{ numero positivo arbitrario} \\ \\ x_{n+1} = \frac{1}{2}\left(\frac{a}{x_n} + x_n\right)\,, \quad n = 0, 1\ldots \end{cases} \tag{7.5}$$

Nel paragrafo successivo vedremo da dove proviene questa particolare successione. Adesso proviamo un certo numero di proprietà di x_n, che porteranno come conseguenza il fatto che questa successione converge alla radice quadrata di a.

(*i*) $x_n \geq 0$, per ogni $n \in \mathbf{N}$; questo si può dimostrare facilmente con il principio di induzione;
(*ii*) $x_n^2 \leq a$, per ogni $n \in \mathbf{N}$; infatti, elevando al quadrato la seconda delle equazioni (7.5) otteniamo:

$$x_{n+1}^2 = a + \frac{1}{4}\left(x_n - \frac{x_n}{a}\right)^2 ,$$

l'affermazione fatta segue immediatamente;
(*iii*) la successione è monotona decrescente, infatti, ancora dalla (7.5) segue che:

$$x_{n+1} = x_n + \frac{1}{2x_n}(a - x_n^2) \leq x_n ,$$

dove nell'ultimo passaggio si è usata la proprietà (*iii*).
Tirando le somme, x_n è una successione decrescente e limitata inferiormente da $a > 0$, dunque ammette limite $l > 0$. Passando al limite in ambo i termini della seconda delle (7.5) otteniamo

$$l = \frac{1}{2}\left(\frac{a}{l} + l\right) \quad \Longrightarrow \quad l = \sqrt{a}$$

che è quanto volevamo dimostrare.
La (7.5) ci dà un'informazione riguardante la velocità di convergenza della successione:

$$(x_{n+1} - \sqrt{a}) = \frac{1}{2x_n}(a + x_n^2 - 2\sqrt{a}) = \frac{1}{2x_n}(x_n - \sqrt{a})^2$$

e dunque:

$$(x_{n+1} - \sqrt{a}) \leq \frac{1}{2a}(x_n - \sqrt{a})^2 , \tag{7.6}$$

ovvero *la distanza tra un termine della successione e* $\sqrt{a}$ *è minore di una costante per il quadrato della distanza del termine precedente da* $\sqrt{a}$. Perchè questo fatto ha importanza? Osserviamo che la distanza tra x_n ed $\sqrt{a}$ diviene arbitrariamente piccola per n sufficientemente grande. In particolare sia N tale che

$$x_N - a \leq \frac{\sqrt{a}}{2} .$$

Sia ora $n \geq N$, applicando iterativamente la (7.6) si ottiene:

$$(x_{n+1} - \sqrt{a}) \leq \frac{1}{2^{n-N}}\left(\frac{x_N - \sqrt{a}}{\sqrt{a}}\right)^{2^{n-N}} \leq \frac{1}{2^{n-N}}\frac{1}{2^{2^{n-N}}} ,$$

per ogni $n > N$; il secondo fattore del termine di destra tende a zero molto rapidamente con n.
Quando vale una relazione come la (7.6) tra i termini di una successione definita per ricorrenza si dice che la velocità di convergenza è *quadratica*.
Per tornare al confronto tra algoritmo di Erone e metodo di bisezione, si dimostra che quest'ultimo ha velocità di convergenza meno che quadratica.

Veniamo ora al programma che implementa l'algoritmo di Erone. Prima di scriverlo dobbiamo stabilire, come per la bisezione, un criterio di arresto. Quante iterazioni dobbiamo

fare se vogliamo calcolare la radice di a con precisione maggiore di ϵ? Utilizzando ancora la (7.5) otteniamo:

$$\frac{1}{2}(x_n + \sqrt{a}) - x_{n+1} = \frac{\sqrt{a}}{2}\left(1 - \frac{\sqrt{a}}{x_n}\right) ;$$

il termine di destra di questa uguaglianza è positivo, quindi

$$x_{n+1} \leq \frac{1}{2}(x_n + \sqrt{a})$$

e questa equivale a:

$$(x_{n+1} - \sqrt{a}) \leq (x_n - x_{n+1}) ,$$

che ci dice che un termine della successione è più vicino a $\sqrt{a}$ di quanto non sia vicino al precedente. Dunque se vogliamo ottenere una precisione $\epsilon > 0$, possiamo porre come condizione di arresto dell'algoritmo il raggiungimento di N iterazioni dove N è un numero naturale tale che:

$$x_N - x_{N+1} \leq \epsilon .$$

Poichè la successione x_n è convergente, possimo essere sicuri che, se $\epsilon > 0$, questa condizione sarà verificata dopo un numero finito di iterazioni.

Programma che implementa il metodo di Erone per calcolare la radice di un numero positivo a con precisione e

```
function r=erone(a,e)
if a<0 | e<0
  disp('dati non accettabili')
  return
end
c=1;
s=(a+1)/2;
r=.5*(a/s+s);
while abs(r-s)>=e
  c=c+1;
  s=r;
  r=.5*(a/s+s);
end
disp('numero di iterazioni') , disp(c)
```

Il programma tiene anche conto del numero `c` di iterazioni fatte per arrivare alla precisione voluta. Questo numero viene visualizzato quando viene eseguito il programma.

7.3 Il metodo di Newton

Il metodo di Newton, così come quello di bisezione, è un procedimento generale per la risoluzione dell'equazione:

$$f(x) = 0 \tag{7.7}$$

dove f è una qualunque funzione continua in un intervallo, che assume valori di segno opposto agli estremi. Il metodo di Newton è basato sulla costruzione geometrica che descriviamo di seguito.
Supponiamo che f sia una funzione *derivabile* in $I = [a, b]$, strettamente crescente, con $f(a) < 0 < f(b)$. Possiamo così essere sicuri che la (7.7) ha esattamente una soluzione x^* in I. L'algoritmo che stiamo per illustrare è, come i precedenti, iterativo e prevede la costruzione di una successione definita per ricorrenza x_n, convergente a x^*. Come punto iniziale della successione scegliamo un punto arbitrario x_0 di I.
Tracciamo la retta r tangente al grafico di f nel punto $(x_0, f(x_0))$ e indichiamo con $(c, 0)$ il punto di intersezione di questa retta con l'asse delle x; c è il secondo elemento della successione che stiamo costruendo:

$$x_1 = c\,.$$

Per procedere oltre, ripetiamo, a partire da x_1, il procedimento geometrico ora descritto, ovvero tracciamo la retta tangente al grafico di f nel punto $(x_1, f(x_1))$ e consideriamo il punto di intersezione $(c_1, 0)$ di questa retta con l'asse delle x; c_1 fornisce il valore di x_2, e così via per i successivi valori di x_n.
Affinchè il procedimento iterativo che abbiamo descritto sia attuabile, è necessario che, per ogni punto $\overline{x}$ di I, il punto di intersezione $(c, 0)$ della retta r tangente al grafico di f in $(\overline{x}, f(\overline{x}))$:
(i) sia ben definito, ovvero r deve intersecare l'asse delle x; questo equivale al fatto che r non sia orizzontale. Poichè il coefficiente angolare di r è $f'(\overline{x})$, introduciamo l'ipotesi:

$$f'(x) \neq 0\,, \quad \forall x \in I\,;$$

(ii) c appartenga ad I. Questa condizione è meno semplice da verificare rispetto alla precedente. Se f è *convessa*, o *concava*, in I, oltre ad essere strettamente monotona, affidandosi all'intuizione geometrica che può dare un disegno, sembra garantito che $c \in I$; vedremo più avanti che questa intuizione è *sostanzialmente* vera.

Il punto c ottenuto partendo da $\overline{x}$, può essere espresso analiticamente mediante $\overline{x}$ stesso, f e f'. Infatti la retta tangente al grafico di f in $(\overline{x}, f(\overline{x}))$ ha equazione

$$y = f'(\overline{x})(x - \overline{x}) + f(\overline{x})\,,$$

ponendo $y = 0$ si trova

$$c = \overline{x} - \frac{f(\overline{x})}{f'(\overline{x})}\,. \tag{7.8}$$

Grazie a questa equazione possiamo scrivere esplicitamente la successione definita per ricorrenza su cui è basato l'algoritmo di Newton:

$$\begin{cases} x_0 = \text{ punto arbitrario di } [a, b] \\ \\ x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}\,, \quad n = 1, 2, \ldots \end{cases} \tag{7.9}$$

La successione x_n così costruita converge alla radice x^* cercata di f. Vale infatti il seguente risultato, che riportiamo senza dimostrazione (per questa e per tutte le altre dimostrazioni omesse in questo paragrafo rimandiamo a [1], §3.7).

Teorema 7.1 *Sia f una funzione con derivate prima e seconda continue in $I = [a, b]$, tale che: $f(a)f(b) < 0$, f è monotona in I,*

$$f'(x) \neq 0\,, \quad \forall x \in I$$

e f è concava, o convessa, in I. Sia $x_0 \in I$ tale che

$$x_1 := x_0 - \frac{f(x_0)}{f'(x_0)}$$

sia ancora un punto di I, allora la successione definita dalle (7.9) è interamente contenuta in I e si ha

$$\lim_{n\to\infty} x_n = x^*\,,$$

dove x^* è l'unica radice di f in I.

Il Teorema 7.1 garantisce che se f è monotona e convessa (o concava), e ha segno opposto agli estremi dell'intervallo, il metodo di Newton funziona, *purchè il secondo punto dell'iterazione, cioè x_1, sia ancora all'interno dell'intervallo.* È facile convincersi con esempi che questo non è sempre vero, e dipende dalla scelta del punto iniziale x_0.

Prima di vedere il programma che implementa il metodo di Newton, facciamo alcune osservazioni. Innanzitutto la convergenza è di tipo quadratico; più precisamente si dimostra che

$$|x_n - x^*| \leq C(x_{n-1} - x^*)^2\,, \quad \forall n = 1, 2, \ldots$$

per una opportuna costante C, che dipende solo dalla funzione f:

$$C = \max_I \frac{|f''|}{|f'|}\,.$$

L'algoritmo di Erone visto nel paragrafo precedente non è altro che il metodo di Newton applicato alla funzione

$$f(x) = x^2 - a\,, \quad a > 0\,.$$

Infatti, consideriamo per semplicià il caso $a > 1$, prendiamo $I = [1, a]$ e costruiamo la successione x_n, definita dalle (7.9), tenendo presente che $f'(x) = 2x$. Otteniamo:

$$\begin{cases} x_0 = \text{ punto arbitrario di } [1, a] \\ \\ x_n = x_{n-1} - \frac{x_{n-1}^2}{2x_{n-1}} = \frac{1}{2}\left(\frac{a}{x_{n-1}} + x_{n-1}\right)\,, \quad n = 1, 2, \ldots \end{cases}$$

Per quanto riguarda il confronto tra il metodo di bisezione e quello di Newton, volendo semplificare possiamo dire che il primo è piu affidabile (in quanto converge in ipotesi molto meno restrittive) ma meno rapido del secondo.

Se qualcuna delle ipotesi del teorema enunciato prima viene a mancare, in generale il metodo di Newton può non convergere. Un esempio è dato dalla funzione:

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

se si scelgono $I = [-2, 2]$ e, come punto di partenza, $x_0 = 1$. In questo caso non è verificata l'ipotesi di convessità in I (f cambia concavità in $x = 0$). La successione x_n non converge, infatti si verifica immediatamente che:

$$x_n = (-1)^n\,, \quad \forall n\,.$$

Tuttavia questo esempio non è molto indicativo; anzi si può affermare che il metodo di Newton, anche senza che tutte le ipotesi siano soddisfatte, in generale converge ad una soluzione di $f(x) = 0$.

Vediamo ora un programma che contiene l'algoritmo di Newton. In questo caso il criterio d'arresto contiene due condizioni; il programma deve fermarsi se:

(i) *la distanza tra due termini successivi di x_n è minore di una quantità positiva data come argomento del programma (*`e`*);*

oppure se:

(ii) *il numero delle iterazioni compiute ha superato una soglia,* `N`*, anch'essa data come argomento.*

Poichè la successione x_n è convergente, la condizione (i) risulterà sicuramente verificata dopo un numero finito di passi.
Il risultato finale del programma è il valore x_N, dove N è il più piccolo naturale per cui almeno una delle condizioni (i), (ii) risulta verificata. A differenza degli algoritmi di bisezione e di Erone, non abbiamo un'informazione precisa sulla distanza di x_N dal valore cercato x^*.

Programma che implementa il metodo di Newton per calcolare una radice di una funzione arbitraria

```
function r=newton(f,d,a,b,X,N,e)
if X<a | X>b | N<0 | e<=0
  disp('dati non accettabili')
  return
end
s=X;
fs=feval(f,s);
ds=feval(d,s);
r=s-fs/ds;
n=1;
if r<a | r>b
  disp('punto iniziale non accettabile')
  return
end
while abs(s-r)>e & n<=N
  s=r;
  fs=feval(f,s);
  ds=feval(d,s);
  r=s-fs/ds;
  n=n+1;
end
disp('numero di iterazioni') , disp(n)
```

Osserviamo che tra gli argomenti del programma, deve essere data anche la derivata prima `d` della funzione `f`.
Supponiamo ad esempio di voler utilizzare il programma per calcolare la radice cubica di 5. La funzione in questione sarà allora $f(x) = x^3 - 5$, questa è monotona e convessa per $x > 0$; inoltre, la radice cercata si trova sicuramente in $(1, 5)$. Quindi dobbiamo scrivere:

```
>> f=inline(x*x*x-5);
>> d=inline(3*x*x);
>> newton(f,d,1,5,2,10,.001)
```

Abbiamo scelto come punto di partenza 2, come numero massimo di iterazioni 10 e come grado di approssimazione un millesimo.

7.4 Esercizi

Esercizio 7.1 *Trascrivete e salvate i programmi relativi al metodo di bisezione, all'algoritmo di Erone e al metodo di Newton. Provate a farli girare su semplici esempi per verificare che funzionino correttamente.*
[Che cosa è un semplice esempio? È un esempio di cui sapete già che risposta vi deve dare il programma se applicato ad esso. Quindi una funzione di cui sono perfettamente note

le radici ($f(x) = x - 1$, $f(x) = \log(x) - 1$) per la bisezione o Newton, oppure un numero che sia un quadrato perfetto nel caso di Erone.]

Esercizio 7.2 **Modificate il programma di tipo funzione* `bisezione2.m` *in modo che alla fine venga visualizzato il numero di iterazioni fatte.*
[Salvate il programma modificato, ad esempio, con il nome `bisezione3.m`]

Esercizio 7.3 **Calcolate la radice quarta di 9 in due modi distinti: utilizzate* `bisezione3` *(vedere l'esercizio precedente) e* `newton` *(applicati alla funzione opportuna), con una precisione di un millesimo. Confrontate il numero di iterazioni fatte nei due casi. Inoltre confrontate i risultati con il valore calcolato da Matlab:*

```
>> 9 ^(.25)
```

Esercizio 7.4 **Scrivete un programma di tipo funzione che, dato un numero positivo a (l'argomento del programma) calcoli, utilizzando il metodo di Newton, la sua radice cubica.*
[Dovete scrivere un programma simile a `erone.m`, in cui la successione definita per ricorrenza che entra in gioco si ottiene scrivendo la successione data dall'algoritmo di Newton:

$$x_0 = \text{numero positivo}\,, \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\,, \quad n = 1, 2, \ldots$$

per la funzione $f(x) = x^3 - a$. Come punto di partenza si può scegliere, come nel caso di Erone, $(1 + a)/2$. Ricordatevi di inserire un criterio di arresto.]

Esercizio 7.5 **Modificando opportunamente il programma* `bisezione`*, scrivere un programma di tipo funzione che, dato un numero $y_0 \in [-1, 1]$ trovi l'unico numero $x_0 \in [-\pi/2, \pi/2]$ tale che*

$$\sin(x_0) = y_0\,.$$

[Si tratta di risolvere l'equazione:

$$f(x) := \sin(x) - y_0 = 0\,,$$

in $[-\pi/2, \pi/2]$, con il metodo di bisezione. Quindi dovete scrivere un programma simile a `bisezione` che abbia per argomento y_0. Gli estremi dell'intervallo saranno fissati: $a = -\pi/2$, $b = \pi/2$, e la precisione `e` dovrà essere impostata all'interno del programma con un valore fisso (piccolo) scelto da voi. Per verificare se avete scritto il programma giusto, tenete presente che il valore che vi dà il programma per un dato y_0 non è altro che $\arcsin(y_0)$; quindi dovete confrontare, per varie scelte di y_0, il risultato dato dal vostro programma con il valore in y_0 della funzione arcoseno predefinita in Matlab.]

Capitolo 8

Grafica e applicazioni alla Analisi e alla Geometria nel piano e nello spazio

8.1 Disegnare nel piano

Il primo comando grafico che vediamo è `plot`. Se scrivete

```
plot(1,1)
```

Matlab apre automaticamente una nuova finestra in cui compare un sistema di riferimento cartesiano nel piano all'interno del quale è disegnato il punto $(1, 1)$.
Con `plot (1,1,'*')` si fa in modo che il punto $(1, 1)$ sia contrassegnato con un asterisco; questo è un esempio di uso di `plot` con una *opzione*.
Se `X` e `Y` sono due vettori *con lo stesso numero di componenti* n, con il comando

```
plot(X,Y,'.')
```

si disegnano gli n punti (x_1, y_1), $(x_2, y_2), \ldots,$ (x_n, y_n), dove x_i e y_i, $i = 1, 2, \ldots, n$, sono le componenti di `X` e `Y` rispettivamente; verificate facendo qualche esempio. In questo caso si è scelta l'opzione `'.'` per fare in modo che i punti siano disegnati effettivamente come tali nella figura; notate che con il comando `plot (X,Y)` si crea invece una poligonale che ha per estremi i punti scelti.
Con il comando possono essere date tre opzioni; la prima riguarda il colore della linea, la seconda lo stile della poligonale (tratteggiata, a punti...), la terza indica come si "marcano" i punti vertici della poligonale (ad esempio con un asterisco...). Per informazioni più precise sull'uso di queste ozioni consultate il manuale in linea alla voce `plot`.

Con il comando `line` si possono disegnare dei segmenti di retta. Ad esempio con

`line`([x_1 x_2],[y_1 y_2])

si disegna il segmento che unisce il punto di coordinate x_1 e y_1 con il punto di coordinate x_2 e y_2. Più in generale, se `X=[`$x_1\, x_2\, \ldots\, x_n$`]` e `Y=[`$y_1\, y_2\, \ldots\, y_n$`]` sono due vettori di n componenti, il comando

```
line(X,Y)
```

fa in modo che venga disegnata una poligonale, che unisce nell'ordine i punti (x_1, y_1), (x_2, y_2), ..., fino a (x_n, y_n); fate qualche prova. Anche per conoscere le varie opzioni di `line` potete consultare il manuale in linea.

Più oggetti in una stessa figura.
Se all'interno di un programma compaiono più comandi grafici, e volete fare in modo che tutti gli oggetti vengano disegnati nella stessa figura, inserite all'inizio del programma il comando `hold on`. Per ripristinare la situazione precedente il comando è `hold off`.

Dimensioni del sistema di riferimento.
Se non viene data nessuna istruzione particolare Matlab sceglie le dimensioni del riferimento cartesiano in base alle dimensioni e alle coordinate degli oggetti che deve disegnare. Se invece si vuole imporre una particolare scelta, si può usare il comando

`axis`([*x-min x-max y-min y-max*]).

Per capire come funziona questo comando, fate qualche prova, con varie scelte dei quattro numeri da inserire al posto di *x-min, x-max, y-min*, e *y-max*.
Altri comandi utili che riguardano gli assi sono:

- `axis square` seleziona i limiti degli assi in modo che il diagramma sia quadrato;
- `axis equal` che fa in modo che i due assi abbiano la stessa unità di misura;
- `axis auto` ripristina il controllo automatico degli assi da parte di Matlab;
- `axis off` e `axis on` rispettivamente eliminano e ripristinano gli assi cartesiani;
- `grid on` fa comparire una griglia equispaziata di rette parallele agli assi coordinati; per eliminarla il comando è `grid off`.
- `xlabel` e `ylabel` permette di mettere delle "etichette" agli assi; per scrivere "asse delle ascisse" accanto all'asse delle x, dovete scrivere

  ```
  xlabel('asse delle ascisse')
  ```

Analogamente per dare un titolo al vostro grafico scrivete

```
title('il titolo che vi pare')
```

Potete anche inserire un testo nel disegno (per esempio per dare un nome ai punti); il comando
`text(1,2,'testo')`
colloca la parola 'testo' a partire dal punto (1,2).

Esercizio 8.1 *Disegnate un triangolo, un quadrato, un trapezio ... Osservate che nel caso del quadrato, se non avete usato il comando* `axis equal`*, il disegno che ottenete è quello di un rettangolo.*

8.2 Un po' di geometria del piano

Esercizio 8.2 *Scrivete un programma di tipo funzione, che, dati due numeri reali positivi* `a` *e* `b`*, disegni il rombo (con il centro nell'origine) che ha le diagonali parallele agli assi cartesiani, di lunghezza* `a` *e* `b` *rispettivamente.*

Esercizio 8.3 *Scrivete un programma che dati due punti disegni un tratto della retta passante per essi tratteggiando la parte finale e iniziale.*

Esercizio 8.4 *Scrivete un programma che dati un punto e una retta in forma parametrica disegni il punto, la sua proiezione ortogonale sulla retta, contrassegnandola con un asterisco, e un tratto della retta contente la proiezione del punto (tutto nello stesso disegno).*

N.B. Per calcolare l'inversa di una matrice A utilizzate il comando `inv(A)`.

Esercizio 8.5 *Scrivete un programma che dati un punto P e una retta r in forma parametrica disegni il punto P, un tratto della retta r contente la proiezione di P su r e un tratto della retta perpendicolare a r e passante per P.*

Esercizio 8.6 *Scrivete un programma che dati un punto P e una retta r in forma parametrica disegni il punto P, un tratto della retta r e un tratto della retta parallela a r e passante per P.*

Esercizio 8.7 *Risolvete i precedenti esercizi supponendo che la retta r sia data in forma cartesiana.*

Esercizio 8.8 *Disegnate una casetta.*

Esercizio 8.9 *Scrivete un programma che sia funzione di un vettore di $\mathbf{R}^2$ e di un punto del piano, che disegni il punto e il suo traslato rispetto al vettore (contrassegnandolo con un asterisco).*

Esercizio 8.10 *Scrivete un programma che sia funzione di un angolo θ, di un punto del piano C e di un punto P, che disegni il punto C, il punto P e il ruotato di P per la rotazione di centro C e angolo θ.*

Esercizio 8.11 *Scrivete un programma che dati un punto e una retta nel piano li disegni e disegni il simmetrico del punto rispetto alla retta.*

8.3 Grafici di funzioni di una variabile

Consideriamo una funzione f definita e continua in un intervallo $[a, b]$, dove a e b sono due numeri reali con $a < b$. Il grafico di f è definito come l'insieme

$$\text{graf}(f) = \{(x, y) \,:\, x \in [a, b]\,,\, y = f(x)\}\,,$$

ed è una curva che rappresenta in un sistema di riferimento cartesiano xy, l'andamento di f; vogliamo utilizzare Matlab per disegnare questa curva. Chiaramente non è possibile far disegnare al computer *tutti* i punti che appartengono al grafico di f per il semplice fatto che questi sono infiniti. Quello che possiamo fare è prendere una *griglia* abbastanza fitta di punti nell'intervallo $[a, b]$

$$x_0 = a < x_1 < \cdots < x_n = b\,,$$

dove n è un numero naturale, e considerare i punti corrispondenti sul grafico

$$P_i = (x_i, f(x_i))\,, \quad i = 0, 1, \ldots, n\,.$$

Possiamo adesso disegnare tutti i punti P_i, che sono in numero finito, o ancora meglio, possiamo disegnare una *poligonale* che li unisca, ovvero la poligonale che ha come primo estremo P_0, come secondo P_1 e così via, fino a P_n che è l'ultimo estremo, La curva che otteniamo ci darà una buona approssimazione del grafico di f.
Una scelta naturale della griglia di punti x_i è quella di n punti equispaziati in $[a, b]$, ovvero tale che tra due punti consecutivi ci sia una distanza pari a $(b - a)/n$:

$$x_0 = a\,, \quad x_i = x_{i-1} + \frac{(b-a)}{n}\,, \quad i = 1, 2, \ldots n\,.$$

Il numero $h = (b - a)/n$ viene chiamato *passo* della griglia. Nel programma che segue, il procedimento che abbiamo descritto viene applicato alla funzione

$$f(x) = 1 - (x^2 - 1)^2\,,$$

nell'intervallo $[-1.5, 1.5]$.

```
% programma grafico1.m.  Disegna il grafico di 1-(1-x^2)^2
% nell'intervallo [-1.5,1.5]
n=300;
h=1/100;
x=zeros(1,n);
y=zeros(1,n);
x(1)=-1.5;
y(1)=1-(1-(-1.5)^2)^2;
for i=2:300
    x(i)=x(i-1)+h;
    y(i)=1-(1-x(i)^2)^2;
end
```

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

Osservate l'uso di `plot` con argomento i due vettori `x` e `y`, che, come abbiamo detto, disegna la poligonale che unisce i punti `(x(i),y(i))` (avremmo potuto usare anche il comando `line`). Se scrivete `plot(x,y,'.')`, otterrete i soli punti al posto della poligonale. Vediamo ancora un esempio. Supponiamo di voler disegnare il grafico della funzione $f(x) = \sin(x)$, relativamente all'intervallo $I = [-\pi, \pi]$.

```
N=1e6 % numero di punti della partizione
h=2*pi/N % passo della partizione
X=[];
Y=[];
for i=0:N
   x=-pi+i*h;
   y=sin(x);
   X=[X x];
   Y=[Y y];
end
line(X,Y)
```

Notate il diverso modo di inizializzare e di formare i due vettori delle ascisse e delle ordinate dei punti del grafico, adottati nei due esempi.

Esercizio 8.12 *Scrivete un programma che disegni il grafico della funzione*

$$f(x) = 8\sin x + 4\sin(2x) + 2\sin(6x) + \sin(24x)\,,$$

nell'intervallo $[0, 2\pi]$.

Esercizio 8.13 **Scrivete un programma di tipo funzione dipendente da quattro argomenti: una funzione* `f`*, due numeri reali* `a` *e* `b`*, il primo minore del secondo, e un numero naturale* `n`*. Lo scopo del programma è quello di disegnare il grafico di* `f` *nell'intervallo di estremi* `a` *e* `b`*, utilizzando una griglia di* `n` *punti equispaziati.*

Esercizio 8.14 *Utilizzando il programma dell'esercizio precedente, disegnate i grafici delle funzioni*

$$\begin{aligned} &f(x) = \log(1+x)\,, \quad f(x) = \sin(1/(1+x^2))\,, \quad f(x) = e^{1/x^2}\,,\\ &f(x) = \cos(x^2)\,, \quad f(x) = 1-(1-x^2)^2\,, \quad f(x) = \log(1+x)\,,\\ &f(x) = \log(1+\sin^2(x))\,, f(x) = \sin(x^{2/5})\,, \quad f(x) = [x]\,. \end{aligned}$$

(ricordando che $[x]$ *indica la parte intera di* x*). Per ciascuna funzione potete scegliere vari intervalli in cui rappresentare il grafico (purché contenuti nel dominio).*

8.4 Curve parametriche nel piano

Una *curva parametrica* $\mathcal{C}$ nel piano è il luogo dei punti descritti da un parametro t, può essere utile immaginarla come la traiettria di un punto materiale la cui posizione varia al variare del tempo t. Dal punto di vista matematico una curva è data dalle sue *equazioni parametriche*:

$$x = x(t)\,, \quad y = y(t)\,,$$

al variare di t in un certo intervallo. Al variare di t il punto di coordinate $(x(t), y(t))$ descrive la curva, che quindi può essere espressa come:

$$\mathcal{C} = \{(x(t), y(t)) \,:\, t \in [a, b]\}\,.$$

Vediamo alcuni esempi. Le equazioni parametriche:

$$x(t) = t\,, \quad y(t) = t\,, \quad t \in [-1, 1]\,,$$

descrivono un segmento della bisettrice dei primo e quarto quadrante. Le equazioni

$$x(t) = t\,, \quad y(t) = t^2\,, \quad t \in [-1, 1]\,,$$

descrivono l'arco della parabola di equazione cartesiana $y = x^2$, corrispondente a $x \in [-1, 1]$. Le equazioni

$$x(t) = \sin(t)\,, \quad y(t) = \cos(t)\,, \quad t \in [-\pi, \pi]\,,$$

descrivono la circonferenza centrata nell'origine e di raggio 1.
Possiamo far disegnare una curva parametrica a Matlab con un procedimento simile a quello appena usato per il grafico di una funzione. Supponiamo per esempio di voler disegnare la curva di equazioni parametriche

$$x(t) = t\,, \quad y(t) = t^2\,, \quad t \in [-4, 4]\,.$$

Anche stavolta approssimeremo la curva con una poligonale. In questo caso scegliamo una partizione equispaziata dello spazio in cui varia il parametro: $[-4, 4]$, disegnamo i punti della curva corrispondenti ai valori della partizione, e infine uniamo questi punti con una poligonale.

```
t= -4:0.001:4;
x=t;
y=t.^2;
plot(x,y)
```

Notate la riga che definisce il vettore `y`, in essa viene applicato a `t` il simbolo di elevamento al quadrato preceduto da un punto, questo fa in modo che ogni componente di `t` venga elevata al quadrato.

Esercizio 8.15 *Disegnate la curva parametrica di equazioni*

$$x(t) = t^2\,, \quad y(t) = t^3\,, \quad t \in [0, 3]\,.$$

Esercizio 8.16 **Scrivete un programma avente per argomento: una coppia di funzioni* `x` *e* `y`*, due nueri reali* `a` *e* `b` *(il primo minore del secondo) e un numero naturale* `n`*. Lo scopo del programma è quello di disegnare la curva parametrica di equazioni* `x` *e* `y`*, al variare del parametro nell'intervallo di estremi* `a` *e* `b`*, utilizzando una partizione equispaziata di* `n` *punti dell'intervallo in cui varia il parametro.*

Esercizio 8.17 *Utilizzando il programma precedente, disegnate le curve di equazioni parametriche:*

$$x(t) = 2\cos(t)\,, \quad y(t) = \sin(t)\,, \quad t \in [0, 2\pi]\,;$$
$$x(t) = t\cos(t)\,, \quad y(t) = t\sin(t)\,, \quad t \in [0, 10\pi]\,;$$
$$x(t) = (1+\cos(t))\cos(t)\,, \quad y(t) = (1+\cos(t))\sin(t)\,, \quad t \in [0, 2\pi]\,.$$

Esercizio 8.18 *L'ellisse $\mathcal{E}(a,b)$ centrata nell'origine, con assi paralleli agli assi coordinati e semiassi di lunghezza $a > 0$ e $b > 0$ ha equazione cartesiana:*

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1\,.$$

La stessa ellisse ammette la seguente rappresentazione parametrica:

$$x(t) = a\cos(t)\,, \quad y(t) = b\sin(t)\,, \quad t \in [0, 2\pi]\,.$$

Scrivete un programma di tipo funzione che, dati $a, b > 0$ disegni l'ellisse $\mathcal{E}(a,b)$.

8.5 Istogrammi di successioni e serie numeriche

Per *istogramma* intendiamo una figura, in un riferimento cartesiano, formata da un certo numero di colonne (rettangoli) con basi della stessa lunghezza giacenti sull'asse delle x, e altezza variabile (si possono avere anche altezze *negative*, ovvero colonne che giacciono nel semipiano $y \leq 0$).
Per disegnare istogrammi con Matlab, si può usare il comando `bar`. Se X è un vettore di n componenti, $x_1, x_2, \ldots x_n$, scrivendo

```
bar(X)
```

si ottiene un istogramma di n colonne, di altezze x_i, $i = 1, \ldots, n$. Provate a fare qualche esempio. Il comando `bar` ha molte opzioni. In particolare si può fare in modo che le colonne dell'istogramma abbiano basi di larghezza assegnata, oppure siano di colori diversi etc. Ancora una volta rimandiamo al manuale in linea per vedere in dettaglio queste opzioni.

8.5.1 Successioni

Gli istogrammi possono essere utili per visualizzare l'andamento di una successione o di una serie numerica. Il programma che segue è una funzione dipendente da un parametro `n`, che serve a visualizare mediante un istogramma i primi `n` termini della successione numerica

$$a_n = \frac{\sin(10\,n)}{\sqrt{n}+1}\,, \quad n \in \mathbf{N}\,.$$

```
% programma per disegnare l'istogramma di una succesione
function I=istogramma1(n)
X=zeros(1,n);
for j=1:n
  X(j)=sin(10*j)/(j^(1/2)+1);
end
bar(X)
```

Esercizio 8.19 **Consideriamo la successione definita per ricorrenza:*

$$a_1 = 1\,, \quad a_n = \sin(a_{n-1})\,, \quad n = 2, 3, \ldots$$

Scrivete una funzione che disegni l'istogramma dei primi n termini di questa successione.

[Se il programma è corretto, dall'istogramma dovrebbe essere chiaro che: i termini della successione sono positivi, la successione è monotona decrescente e tende a zero. Sareste in grado di dimostrare tutto ciò con i risultati di teoria che avete a disposizione?]

Esercizio 8.20 *Applicate lo stesso programma dell'esercizio precedente alla successione dei numeri Fibonacci*

$$F_1 = 1\,, \quad F_2 = 1\,, \quad F_n = F_{n-1} + F_{n-2}\,, \quad n = 3, 4, \ldots$$

e alla successione dei rapporti dei numeri di Fibonacci $r_n = \frac{F_{n+1}}{F_n}$.

Esercizio 8.21 **Supponiamo che una successione* a_n, $n \in \mathbf{N}$, *sia data per ricorrenza nel modo seguente:*

$$a_0 = a\,, \quad a_{n+1} = f(a_n)\,, \quad \forall n \geq 1\,, \tag{8.1}$$

dove a è un numero reale dato e f una certa funzione di una variabile che per comodità supporremo definita in $\mathbf{R}$ *(la successione dell'Esercizio 8.19 ha questa proprietà, con* $f(t) = \sin(t)$*). Scrivete un programma di tipo funzione che, data una funzione f, un numero reale a e un numero naturale n, disegni l'istogramma relativo ai primi n termini della successione (8.1).*

8.5.2 Serie

Il comportamento di una serie numerica

$$\sum a_i \,,$$

generata da una successione reale a_i, $i \in \mathbf{N}$, è riconducibile a quello della successione delle sue somme ridotte o parziali:

$$\sigma_n = \sum_{i=1}^{n} a_i \,, \quad n \in \mathbf{N} \,.$$

Ovvero la serie data è convergente, divergente o indeterminata, a seconda che, rispettivamente, σ_n abbia limite finito, abbia limite infinito oppure non ammetta limite. Dunque per visualizzare il comportamento della serie è utile disegnare un istogramma della successione delle sue somme ridotte. Il programma che segue disegna gli istogrammi relativi alla serie armonica

$$\sum_{i=1}^{+\infty} \frac{1}{i} \,.$$

```
function T=istogramma2(n)
X=zeros(1,n);
s=0;
for j=1:n
  s=s+1/j;
  X(j)=s;
end
bar(X)
```

Esercizio 8.22 **Molti degli esempi di serie che avete visto sono del tipo:*

$$\sum f(i) \,, \tag{8.2}$$

dove f è una certa funzione di una variabile, definita in $[0, +\infty)$. Ad esempio

$$\sum \frac{1}{i} = \sum f(n) \,, \quad \textit{con} \quad f(t) = \frac{1}{t} \,.$$

Scrivete un programma di tipo funzione che, data una funzione f e un numero naturale n disegni l'istogramma delle prime n ridotte della serie (8.2).

Esercizio 8.23 *Utilizzando l'esercizio precedente, disegnare gli istogrammi relativi alle serie:*

$$\sum \frac{1}{i^{3/2}} \,, \quad \sum \frac{\sin(i)}{i^2} \,, \quad ; \sum \frac{\sin(i)}{i} \,, \quad \sum \frac{\sin(i)}{\sqrt{i}} \,.$$

Dagli istogrammi che raffigurano l'andamento delle ridotte, per valori abbastanza grandi di n, si intuisce il comportamento della serie in questione. Alcuni dei risultati che trovate possono essere giustificati dal punto di vista teorico con i criteri di convergenza che avete a disposizione. Quali?

Esercizio 8.24 *Sia a_i, $i \in \mathbf{N}$, la successione (vista in uno dei precedenti esercizi)*

$$a_1 = 1\,, \quad a_i = \sin(a_{i-1})\,, \quad i = 2, 3, \ldots$$

Scrivete un programma che, dato n, disegni l'istogramma dei primi n termini della successione delle ridotte della serie

$$\sum a_i\,.$$

Esercizio 8.25 *Scrivete un programma che, dati un numero reale a, una funzione di una variabile f definita in $\mathbf{R}$ e un numero naturale n disegni l'istogramma della successione delle prime n ridotte della serie:*

$$\sum a_i\,,$$

dove a_i è la successione definita per ricorrenza come segue:

$$a_0 = a\,, \quad a_{i+1} = f(a_i)\,, \quad \forall\, i \geq 2\,.$$

La serie armonica $\sum \frac{1}{n}$ è divergente; la serie a segno alterno

$$\sum_{n=1}^{+\infty} \frac{(-1)^n}{n}\,,$$

è invece convergente per il criterio di Leibnitz. Consideriamo più in generale la serie

$$\sum_{n=1}^{+\infty} \frac{\varepsilon_n}{n}\,,$$

dove ε_n, $n \in \mathbf{N}$, è una successione i cui termini possono assumere solo i valori -1 e 1. Evidentemente per alcune successioni ε_n, $n \in \mathbf{N}$, la serie in questione è convergente e per altre no.
Proviamo a immaginare il caso in cui la successione ε_n sia *costruita in modo casuale*, ovvero il suo n-esimo termine è un numero scelto con uguale probabilità tra -1 e 1. Questa situazione può essere *simulata* con il seguente programma che disegna l'istogramma delle prime n ridotte della serie di cui stiamo parlando.

```
function I=istogramma3(n)
s=0;
X=zeros(1,n);
for i=1:n
    c=2*round(rand)-1;
    s=s+c/i;
    X(i)=s;
end
bar(X)
```

Osservate che la variabile `c` definita alla quinta riga è un generatore casuale di -1 e 1.
Provate a far girare il programma più volte; ovviamente il risultato non è sempre lo stesso, questo perchè la successione ε_n, $n \in \mathbf{N}$, è diversa in ogni esecuzione. Ciò nonostante vi è nell'andamento della serie un comportamento decisamente *più frequente*.

Esercizio 8.26 **Data una funzione f di una variabile, consideriamo la successione a_i, $i \in \mathbf{N}$ definita da*

$$a_i = \varepsilon_i f(i)\,, \quad i \in \mathbf{N}\,,$$

dove $\varepsilon_i \in \{-1, 1\}$ con probabilità uniforme. Scrivete un programma di tipo funzione che dati f e n disegni l'istogramma della successione delle prime n ridotte della serie

$$\sum a_i\,.$$

Esercizio 8.27 *Applicate il programma dell'esercizio precedente alle serie:*

$$\sum_{n=1}^{+\infty} \frac{\varepsilon_n}{n^2}\,, \quad \sum_{n=1}^{+\infty} \frac{\varepsilon_n}{n^{1/3}}\,, \quad \sum_{n=1}^{+\infty} \frac{\varepsilon_n \cdot \sin(n)}{n}\,.$$

8.6 Punti e linee nello spazio

Il comando `plot3` permette di disegnare punti nello spazio tridimensionale; provate a scrivere

```
plot3(2,1,3)
```

Per migliorare il risultato conviene scrivere `grid` in modo che compaia la "griglia" e anche dare le "etichette" agli assi (con i comandi `xlabel`....).
Analogamente a prima, se `X`, `Y` e `Z` sono tre vettori *con lo stesso numero di componenti* n, con il comando

```
plot3(X,Y,Z,'.')
```

si disegnano gli n punti (x_1, y_1, z_1), $(x_2, y_2, z_2), \ldots, (x_n, y_n, z_n)$, dove x_i,y_i, $i = 1, 2, \ldots, n$, sono le componenti di `X`,`Y` e `Z` rispettivamente; verificate facendo qualche esempio. In questo caso si è scelta l'opzione `'.'` per fare in modo che i punti siano disegnati effettivamente come tali nella figura; notate che con il comando `plot3(X,Y,Z)` si crea invece una poligonale che ha per estremi i punti scelti.
Con il comando `plot3` si possono quindi rappresentare le linee nello spazio tridimensionale.

Nello spazio, come nel piano, si possono definire delle curve parametriche. Una curva parametrica $\mathcal{C}$ nello spazio è data da tre equazioni:

$$x = x(t)\,, \quad y = y(t)\,, \quad z = z(t)\,, \quad t \in [a, b]\,.$$

Al variare di $t \in [a, b]$, il punto di coordinate $(x(t), y(t), z(t))$ descrive $\mathcal{C}$, che quindi può essere espressa nella forma:

$$\mathcal{C} = \{(x(t), y(t).z(t)) \,:\, t \in [a, b]\}\,.$$

Per disegnare una curva nello spazio con Matlab, usiamo un procedimento del tutto analogo a quello visto nel piano. Supponiamo di voler disegnare la curva

$$\mathcal{C} = \{(\sin(t), \cos(t), t^2) |\; t \in [-10, 10]\}\,.$$

Questo può essere fatto con le istruzioni:

```
t=-10:0.1:10;
plot3(sin(t),cos(t),t.^2)
```

il punto e virgola alla fine della prima riga serve solo affinché non compaiano sullo schermo tutti i valori che assume t; il punto nell'espressione `t.^2` sta ad indicare che ogni coefficiente del vettore t è elevato al quadrato.

Esercizio 8.28 *Disegnate le curve*

$$\mathcal{C}_1 = \{(t, t, t^2) \,:\, t \in I\}\,, \quad \mathcal{C}_2 = \{(t, -2t), t + 3) \,:\, t \in J\}\,,$$

dove I e J sono intervalli a vostra scelta.

Osservate che la curva $\mathcal{C}_2$ dell'esercizio precedente, essendo un segmento di retta poteva essere disegnata semplicemente individuando due suoi punti (x_1, y_1) e (x_2, y_2) e scrivendo `plot(X,Y)` dove `X` e `Y` sono...

Esercizio 8.29 *Scrivete un programma che dati un punto e una retta in forma parametrica nello spazio tridimensionale disegni il punto, la sua proiezione ortogonale sulla retta, contrassegnandola con un asterisco, e un tratto della retta contente la proiezione del punto (tutto nello stesso disegno).*

8.7 Superfici nello spazio

In questo paragrafo vedremo come disegnare superfici nello spazio ottenute come grafici di funzioni di due variabili oppure come superfici parametriche.

8.7.1 Grafici di funzioni di due variabili

Supponiamo che $f = f(x, y)$ sia una funzione di due variabili a valori in $\mathbf{R}$, definita in un rettangolo $R = [a, b] \times [c, d]$ del piano xy. Il grafico di f relativo a R è l'insieme:

$$G\{(x, y, f(x, y)) \,:\, (x, y) \in R\}\,.$$

G è una superficie nello spazio tridimensionale. Vediamo come possiamo far disegnare questa superficie a Matlab.
Occorre anzitutto generare una griglia di punti di R su cui calcolare la funzione $f(x, y)$; per griglia intediamo l'insieme dei punti di R ottenuti intersecando due famiglie di rette parallele ciascuna ad uno dei due assi coordinati. Questo vuol dire che il rettangolo R viene diviso in tanti sottorettangoli i cui vertici sono i punti della griglia. Per creare una griglia si può utilizzare la funzione `meshgrid`; più precisamente la successione di istruzioni:

```
x= [a:s:b]
y= [c:l:d]
[X,Y] = meshgrid(x,y)
```

fa in modo che vengano generati i punti di una griglia di R in cui ogni sottorettangolo è largo `s` e alto `l`. Le ascisse di tali punti vengono memorizzate nella matrice `X` e le ordinate nella matrice `Y`. Osserviamo che le due istruzioni

```
[X,Y] = meshgrid(x)
[X,Y] = meshgrid(x,x)
```

sono equivalenti, quindi possiamo utilizzare la prima delle due scritture tutte le volte che vogliamo creare una griglia con sottorettangoli quadrati.
Dopo avere creato la griglia possiamo disegnare il grafico con il comando `mesh` la cui sintassi è `mesh(X,Y,Z)`.
Proviamo per esempio a disegnare il grafico della funzione $f(x,y) = x^2 + y^2$ in $R = [-2,2] \times [-2,2]$. Scegliamo una griglia quadrata con spaziatura 0.2.

```
[X,Y]= meshgrid(-2:0.2:2);
Z=X.^2 + Y.^2;
mesh(X,Y,Z)
```

I punti dopo la matrice `X` e dopo la matrice `Y` stanno ad indicare che l'operazione di elevamento al quadrato è fatta su ogni coefficiente della matrice.
Se invece volete disegnare il grafico della funzione $f(x,y) = x + yx$ in $R = [0,4] \times [-2,2]$ scegliendo una griglia con spaziatura 0.2 per la x e 0.3 per la y, scrivete:

```
x=[0:0.2:4]
y=[-2:0.3:2]
[X,Y]= meshgrid(x,y);
Z=X +Y.*X;
mesh(X,Y,Z)
```

Con Matlab si possono disegnare anche le *curve di livello* di una funzione di due variabili $f(x,y)$. Supponiamo che f assuma valori compresi tra due numeri reali m e M, con $m < M$. Se t appartiene a $[m,M]$, la linea di livello t di f è l'insieme dei punti in cui f assume il valore t:

$$\Gamma_t = \{(x,y) \,:\, f(x,y) = t\}\,.$$

In particolare osserviamo che Γ_t è un sottoinsieme del piano xy. Se ad esempio $f(x,y) = x^2 + y^2$, si verifica facilmente che Γ_t, per $t \geq 0$, è la circonferenza centrata nell'origine di raggio $\sqrt{t}$. Un altro fatto di semplice verifica è che le linee di livello di funzioni lienari o affini:

$$f(x,y) = ax + by + c\,, \quad a,b,c \in \mathbf{R}\,,$$

sono rette. Ad esempio le linee di livello x $f(x,y) = x - y$ sono tutte e sole le rette paralle alla bisettrice del primo e terzo quadrante del piano xy.

Torniamo adesso al procedimento, che abbiamo visto prima, per disegnare il grafico di una funzione con Matlab. Se invece del comando `mesh` utilizzate `meshc`, vengono disegnate oltre alla superficie data dal grafico di f, anche alcune delle curve di livello di f. Il comando `contour` disegna invece solo le curve di livello su un piano.

8.7.2 Superfici in forma parametrica

Una superficie Σ nello spazio tridimensionale può essere data anche attraverso le sue *equazioni parametriche*, ovvero

$$x = x(s,t)\,, \quad y = y(s,t)\,, \quad z = z(s,t)\,, \quad (s,t) \in R = [a,b] \times [c,d]\,.$$

Al variare della coppia (s,t) in R, il punto $(x(s,t), y(s,t), z(s,t))$ descrive Σ che quindi può essere definita come

$$\Sigma = \{(x(s,t), y(s,t), z(s,t)) \,:\, (s,t) \in R\}\,.$$

Ad esempio le equazioni parametriche

$$x(s,t) = s\,, \quad y(s,t) = t\,, \quad z = 1 - s - t\,, \quad (s,t) \in R = [a,b] \times [c,d]\,,$$

descrivono una porzione del piano di equazione cartesiana $x + y + z = 1$.
Le equazioni

$$x(\phi,t) = \cos\phi\,, \quad y(\phi,t) = \sin\phi\,, \quad z(\phi,t) = t\,, \quad (\phi,t) \in R = [0,2\pi] \times [0,1]\,,$$

descrivono la porzione, compresa tra i piani $z = 0$ e $z = 1$, di un cilindro circolare retto avente per asse l'asse delle z e raggio di base 1.
Le equazioni

$$x(\phi,t) = t\cos\phi\,, \quad y(\phi,t) = t\sin\phi\,, \quad z(\phi,t) = t\,, \quad (\phi,t) \in R = [0,2\pi] \times [0,1]\,,$$

descrivono invece una porzione di un cono circolare retto avente l'asse delle zeta come asse.
La superficie di una sfera centrata nell'origine di raggio $r > 0$ ammette la seguente rappresentazione parametrica:

$$x(\phi,\theta) = r\cos\phi\cos\theta\,, \quad y(\phi,\theta) = r\cos\phi\sin\theta\,, \quad z(s,t) = r\sin\phi\,,$$
$$(\phi,\theta) \in R = [-\pi/2, \pi/2] \times [0, 2\pi]\,.$$

I due parametri ϕ e θ rappresentano rispettivamente la longitudine e la latitudine del punto $(x(\phi,\theta), y(\phi,\theta), z(\phi,\theta))$.

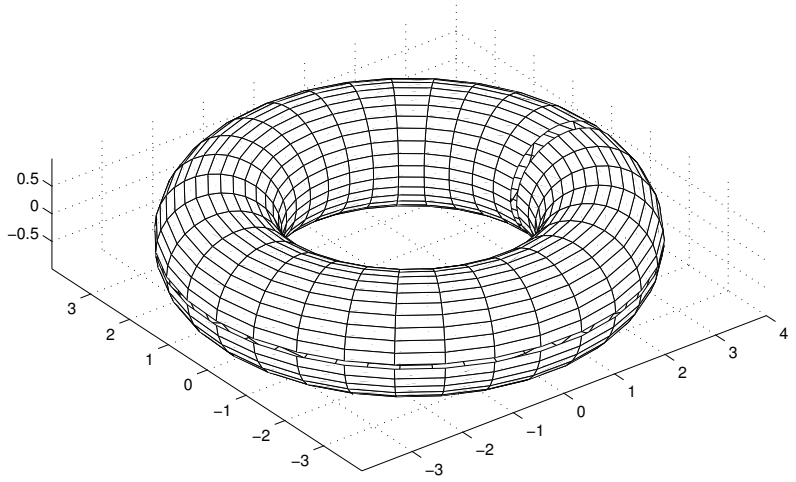
Con i comandi `mesh` e `meshgrid` si possono disegnare anche le superfici parametriche. Il procedimento è del tutto analogo a quello visto per i grafici di funzioni di due variabili, con la differenza che per le superfici parametrica la griglia di punti viene fatta sul rettangolo in cui variano i parametri, ovvero sul rettangolo R.
Ad esempio per disegnare la superficie sferica scrivete:

```
axis equal
s=[-0.5*pi:0.2:0.5*pi ];
t=[0:0.2:2*pi];
[T,S]=meshgrid(t,s);
Z=sin(S);
```

```
X=cos(S).*cos(T);
Y=cos(S).*sin(T);
mesh(X,Y,Z)
```

Provate adesso a eseguire i comandi:

```
axis equal
t=[0:0.2:2*pi];
s=[0:0.2:2*pi];
[T,S]=meshgrid(t,s);
X=(3+cos(S)).*cos(T);
Y=(3+cos(S)).*sin(T);
Z=sin(S);
mesh(X,Y,Z)
```



Quello che ottenete è un *toro*, ovvero la superficie ottenuta facendo ruotare una circonferenza contenuta nel piano zy e non intersecante l'asse z, attorno a tale asse.

Esercizio 8.30 *Disegnate le seguenti superfici (prima di disegnare le superfici 4, 5, 6, 7, 8, provate ad a capire se sono dei sottospazi vettoriali):*
1) il piano $\{(1+t+s, 3t, 2s+1) |\ t, s \in \mathbf{R}\}$*;*
2) un cilindro con asse l'asse delle z*;*
3) un cono con asse l'asse delle z *e vertice l'origine;*
4) la superficie $\{(t^2, t^2, s^2) |\ t, s \in \mathbf{R}\}$*;*
5) la superficie $\{(t^2, t^3, s^2) |\ t, s \in \mathbf{R}\}$*;*
6) la superficie $\{(t^3, t^3, s^2) |\ t, s \in \mathbf{R}\}$*;*
7) la superficie $\{(t^3, t^3, s^3) |\ t, s \in \mathbf{R}\}$*;*
8) la superficie $\{(t^3, t^3, s^5) |\ t, s \in \mathbf{R}\}$*;*
9) un cono con asse l'asse delle x *e vertice l'origine;*
10) un cono con asse la retta $\{(x, y, z) |\ z = x, \quad y = 0\}$ *(suggerimento: scrivete prima l'espressione di una opportuna rotazione e poi sfruttate i punti 3 o 9).*

Esercizio 8.31 *Che curve sono le curve di livello dei coni in 3) 9) 10)?*

Esercizio 8.32 *Disegnate una casetta tridimensionale (per esempio potete disegnare il tetto con il comando* `mesh` *e disegnare solo gli spigoli delle pareti laterali col comando* `plot3`*).*

Esercizio 8.33 *Scrivete un programma che dato un piano in* $\mathbf{R}^3$ *in forma parametrica e un punto* P*, disegni il piano e la retta perpendicolare al piano passante per il punto* P*.*

Esercizio 8.34 *Scrivete un programma che, dato un piano* π *in* $\mathbf{R}^3$ *in forma parametrica e un punto* P*, disegni il piano* π *e il piano passante per il punto* P *e parallelo a* π*.*

Esercizio 8.35 *Scrivete un programma che, dato un piano* π *in* $\mathbf{R}^3$ *in forma parametrica e una retta* r *in forma parametrica parallela al piano, disegni il piano* π *e il piano* σ *contenente* r *e perpendicolare al piano* π*.*

Esercizio 8.36 *Scrivete un programma che, dati due punti distinti* P *e* Q *in* $\mathbf{R}^3$*, li disegni e disegni il piano* H *tale che* P *e* Q *sono l'uno il simmetrico dell'altro rispetto al piano* H*.*

Capitolo 9

Geometria nel piano e nello spazio

9.1 Posizione reciproca di rette e piani nello spazio affine

Vogliamo adesso scrivere dei programmi che ci dicano la posizione reciproca di rette e piani nello spazio affine (utilizzando il Teorema di Rouché-Capelli).

Esempio 9.1 *Il seguente programma ci dice la posizione reciproca di due piani nello spazio affine; i due piani sono dati in forma cartesiana, precisamente sono le soluzioni di due sistemi lineari* $Ax = b$ *e* $Cx = d$*, dove* A *e* C *sono due matrici* 1×3 *e* b *e* c *sono due matrici* 1×1*.*
Osservate che nel quarto, quinto e sesto rigo viene "scartato" il caso in cui una delle due equazioni non sia l'equazione di un piano (ricordatevi che il simbolo | è un operatore logico e significa "o"). Nei righi successivi si considera invece il caso in cui entrambe le equazioni siano equazioni di piani e si esaminano le posizioni reciproche di questi due piani.

```
% Function per calcolare la posizione reciproca di due piani
% dati in forma cartesiana Ax=b , Cx=d nello spazio affine
function duepiani=duepiani(A,b,C,d)
if rank(A)==0 | rank(C)==0
'almeno uno dei due non e un piano'
else
   if rank([A; C]) == 2
   'i due piani sono incidenti distinti'
   elseif rank([A b; C d]) == 1
   'i due piani sono coincidenti'
```

```
   else
   'i due piani sono paralleli distinti'
   end
end
```

Esercizio 9.1 **Fare un programma che vi dica la posizione reciproca di una retta e un piano, dati in forma cartesiana, nello spazio affine.*

Guida per lo svolgimento dell'esercizio:
sia la retta l'insieme delle soluzioni del sistema lineare $Ax = b$ e e sia il piano l'insieme delle soluzioni del sistema lineare $Cx = d$, dove A è una matrice 2×3, b una matrice 2×1, C una matrice 1×3 e d una matrice 1×1;
costruite un programma che sia funzione delle quattro matrici A, b, C, d nel seguente modo:
• anzitutto "scartate" il caso in cui o l'insieme delle soluzioni del sistema lineare $Ax = b$ non sia una retta o l'insieme delle soluzioni del sistema lineare $Cx = d$ non sia un piano, cioè fate in modo che in tal caso vi compaia un avvertimento;
• nel caso che l'insieme delle soluzioni del sistema lineare $Ax = b$ sia una retta e l'insieme delle soluzioni del sistema lineare $Cx = d$ sia un piano, distinguete i seguenti tre casi:
i) la retta e il piano sono incidenti in un punto
ii) la retta è contenuta nel piano
iii) la retta e il piano sono paralleli e la retta non è contenuta nel piano.

Esercizio 9.2 **Scrivete un programma che vi dica la posizione reciproca di due rette, date in forma cartesiana, nello spazio affine (può essere utile utilizzare l'operatore logico "e" che si scrive &).*

Esercizio 9.3 *Scrivete un programma che, dato un piano in forma cartesiana nello spazio affine, dia una forma parametrica di tale piano.*

Esercizio 9.4 *Scrivete un programma che, dato un piano in forma parametrica nello spazio affine, dia una forma cartesiana di tale piano.*

9.2 Inviluppi convessi

Vogliamo scrivere un programma per verificare se un punto appartiene all'inviluppo convesso di alcuni altri punti del piano affine.
Cominciamo con alcuni esercizi:

Esercizio 9.5 *Scrivete un programma che dati due punti distinti del piano affine calcoli un'equazione della retta passante per i due punti. Ricordate che la retta passante per due punti* $A = \begin{pmatrix} x^A \\ y^A \end{pmatrix}$, $B = \begin{pmatrix} x^B \\ y^B \end{pmatrix}$, *è definita dalla seguente equazione (nelle incognite* x, y*):*

$$\det \begin{pmatrix} x - x^A & x^B - x^A \\ y - y^A & y^B - y^A \end{pmatrix} = 0$$

Prima di continuare con i programmi riguardanti gli inviluppi convessi, introduciamo il comando `all` e facciamo qualche osservazione.
Il comando `all` applicato ad un vettore b, dà 0 se b ha almeno un elemento uguale a zero ed è 1 se b ha tutti gli elementi diversi da zero.
Osservate inoltre che se A è una matrice e scrivete ad esempio

```
A < 55
```

ottenete una matrice con le stesse dimensioni di A i cui coefficienti sono tutti o 0 o 1 e precisamente il coefficiente i, j è 0 se $A_{i,j} \geq 55$, è 1 se $A_{i,j} < 55$.
Facciamo un esempio per illustrare una possibile utilizzazione di quanto detto sopra: il seguente programma, dato un vettore v in $\mathbf{R}^5$, definsce $w = -v$ se v ha tutti gli elementi minori di zero, $w = v$ altrimenti:

```
if all(v < 0)
w = -v
else w= v
end
```

Notate che in realtà si può fare a meno del comando `all` nel seguente modo:

```
c=0;
for i=1:5
   if v(i) < 0
     c=c+1;
   end
end
if c=5
   w=-v
else
   w=v
end
```

Esercizio 9.6 *Scrivete un programma che, dati k punti del piano affine, di cui i primi tre non allineati, $P_1, ..., P_k$ ($k \geq 4$), vi dice se i punti $P_4, ..P_k$ appartengono al semipiano contenente P_1, P_2, P_3 e avente come "retta di frontiera" la retta passante per P_1, P_2.*

Il seguente programma verifica se un punto P appartiene all'inviluppo convesso di altri cinque punti A, B, C, D, E nel piano affine.
Esso si basa sul seguente fatto: l'inviluppo convesso di k punti $P_1, ..., P_k$ nel piano affine è l'intersezione di tutti i semipiani contenenti i k punti e che hanno come "retta di frontiera" una delle rette P_iP_j.

```
% Function per calcolare se un punto P appartiene all'inviluppo convesso di
5
```

```
% punti, A, B, C, D, E
function apparinv= apparinv(A,B,C,D,E,P)
X=[A B C D E P];
for i=1:5
   for j=1:5
    for k=1:6
       L(i,j,k)=det([X(1,k)-X(1,i) X(1,j)-X(1,i) ; X(2,k)-X(2,i) X(2,j)-X(2,i)]);
     end
   end
end
for i=1:5
   for j=1:5
     if all( L(1:5,i,j) >= 0)
       M(i,j)= L(i,j,6);
       elseif all( L(1:5,i,j) <= 0)
       M(i,j)= -L(i,j,6);
       else
       M(i,j)=0;
     end
   end
end
if all( M(:,:)  >= 0)
'il punto P appartiene allo inviluppo convesso'
else
'il punto P non appartiene allo inviluppo convesso'
end
```

Commenti:
• anzitutto viene definita una matrice X ottenuta affiancando le colonne le cui coordinate sono le coordinate dei sei punti; quindi $A = \begin{pmatrix} X(1,1) \\ X(2,1) \end{pmatrix}$, $B = \begin{pmatrix} X(1,2) \\ X(2,2) \end{pmatrix}$, $C = \begin{pmatrix} X(1,3) \\ X(2,3) \end{pmatrix}$, $D = \begin{pmatrix} X(1,4) \\ X(2,4) \end{pmatrix}$, $E = \begin{pmatrix} X(1,5) \\ X(2,5) \end{pmatrix}$, $P = \begin{pmatrix} X(1,6) \\ X(2,6) \end{pmatrix}$,
• poi vengono calcolati i determinanti delle matrici 2×2

$$\begin{pmatrix} X(1,k) - X(1,i) & X(1,j) - X(1,i) \\ X(2,k) - X(2,i) & X(2,j) - X(2,i) \end{pmatrix}$$

per $i = 1, ..., 5$, $j = 1, ..., 5$, $k = 1, ..., 6$; (precisamente si forma un matrice tridimensionale (cioè i cui coefficienti dipendono da tre indici), L, il cui coefficiente di indice i, j, k è il determinante scritto sopra).
• Osservate che la retta passante per il punto i-esimo e j-esimo individua un semipiano contenente anche gli altri tre punti di A, B, C, D, E se e solo se

$$\det \begin{pmatrix} X(1,k) - X(1,i) & X(1,j) - X(1,i) \\ X(2,k) - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} \geq 0$$

$\forall k = 1, ..., 5$ oppure

$$\det \begin{pmatrix} X(1,k) - X(1,i) & X(1,j) - X(1,i) \\ X(2,k) - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} \leq 0$$

$\forall k = 1, ..., 5$. Infatti: la retta passante per il i-esimo e il j-esimo punto ha equazione (nelle incognite x_1 e x_2)

$$\det \begin{pmatrix} x_1 - X(1,i) & X(1,j) - X(1,i) \\ x_2 - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} = 0,$$

quindi i due semipiani individuati da tale retta avranno disequazioni rispettivamente

$$\det \begin{pmatrix} x_1 - X(1,i) & X(1,j) - X(1,i) \\ x_2 - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} \geq 0$$

$$\det \begin{pmatrix} x_1 - X(1,i) & X(1,j) - X(1,i) \\ x_2 - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} \leq 0$$

• Nel linguaggio del programma le condizioni affinché la retta passante per il punto i-esimo e j-esimo individui un semipiano contenente anche gli altri tre punti di A, B, C, D, E, (cioè

$$\det \begin{pmatrix} X(1,k) - X(1,i) & X(1,j) - X(1,i) \\ X(2,k) - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} \geq 0$$

$\forall k = 1, ..., 5$ oppure

$$\det \begin{pmatrix} X(1,k) - X(1,i) & X(1,j) - X(1,i) \\ X(2,k) - X(2,i) & X(2,j) - X(2,i) \end{pmatrix} \leq 0$$

$\forall k = 1, ..., 5$) equivalgono a `all( L(1:5,i,j) >= 0)` e a `all( L(1:5,i,j) <= 0)` .
• Quindi alle righe 14-20 del programma, consideriamo una matrice M il cui coefficiente i, j è così definito: per ogni i e j tali che la retta passante per il punto i-esimo e j-esimo individua un semipiano contenente anche gli altri tre punti di A, B, C, D, E, è il valore del polinomio

$$\det \begin{pmatrix} x_1 - X(1,i) & X(1,j) - X(1,i) \\ x_2 - X(2,i) & X(2,j) - X(2,i) \end{pmatrix}$$

valutato nelle coordinate di P se tale polinomio valutato nei restanti punti di A, B, C, D, E è positivo, il valore opposto se tale polinomio valutato nei restanti punti di A, B, C, D, E è negativo.
Quindi la condizione affinché P appartenga all'inviluppo convesso di A, B, C, D, E è che tutti i coefficienti di M siano positivi (ultimi righi del programma).

Esercizio 9.7 *Scrivete un programma che verifichi se un punto P appartiene all'inviluppo convesso di un numero qualsiasi di punti P_i nel piano affine.*

(Suggerimento:
potete scrivere tale programma come funzione di due matrici: la prima le cui colonne sono i punti P_i, la seconda 2×1 data dal punto P;
chiamate X la matrice concatenazione delle due e con il comando `size` "scoprite" quanto è il numero delle colonne di X (nel programma precedente tale numero era 6); il resto del programma è analogo al precedente).

9.3 Esercizi vari di Geometria: coniche, prodotto vettoriale, aree, forme bilineari

Esercizio 9.8 *Scrivete un programma che, data la matrice associata ad una conica, classifichi tale conica.*

Esercizio 9.9 *Scrivete un programma che calcoli l'area di un poligono.*

Esercizio 9.10 *Scrivete un programma che calcoli il prodotto vettoriale di due vettori.*

Esercizio 9.11 *Scrivete un programma che, data una matrice simmetrica, verifichi se la forma bilineare associata è definita positiva.*

Capitolo 10

Numeri primi e fattorizzazioni

10.1 Un semplice test di primalità

Lo scopo di questo paragrafo è costruire un algoritmo, che possa essere implementato su computer, che ci permetta di affermare se un numero naturale n è primo oppure no. Algoritmi di questo genere si chiamano *test di primalità* e si rivelano molto importanti per le loro applicazioni, soprattutto alla moderna crittografia.
I più sofisticati test di primalità oggi in uso nascono da nozioni molto avanzate di teoria dei numeri, decisamente al di fuori della nostra portata. Vedremo invece un test molto semplice, basato su considerazioni elementari, che possiamo chiamare il *test della divisione*.
Questo test si basa sulla definizione stessa di numero primo: n è primo se non è divisibile per nessun numero intero eccettuati 1 e n stesso. L'idea quindi è quella di provare a dividere n per ogni intero $m \in \{2, \ldots, n-1\}$, se il resto della divisione è sempre diverso da zero, n è primo.
Questo procedimento può essere raffinato sulla base di due semplici considerazioni.

(i) Se n ha un divisore intero $m \in \{2, \ldots n-1\}$ (e quindi non è primo) allora sicuramente ha anche un divisore $m' \in \{2, \ldots [\sqrt{n}\,]\}$, dove [ ] indica al solito la parte intera.
Questa affermazione, che si prova facilmente, consente di risparmiare una buona parte delle operazioni previste dalla versione iniziale dell'algoritmo, infatti per verificare se n è primo è sufficiente controllare se n diviso m ha resto diverso da zero per $m \in \{2, \ldots, [\sqrt{n}\,]\}$.

(ii) Una volta verificato che n non è pari, ovvero dopo aver fatto la divisione per due, si può procedere considerando solo gli eventuali divisori dispari. Infatti se n non è pari, non è certamente divisibile per nessun numero pari. Questa osservazione consente un ulteriore dimezzamento del numero delle operazioni.

Il numero di operazioni necessarie (o meglio il suo ordine di grandezza) per eseguire un algoritmo destinato ad essere implementato su un computer, è una caratteristica per niente trascurabile dell'algoritmo stesso. Nel caso dei test di primalità, se questo numero rimane contenuto rispetto a n, si può sperare di applicare il test a valori grandi di n. Vediamo ora un programma che implementa il test della divisione.

Test della divisione

```
function primalita1(n)
if n==2
  disp('numero primo')
  return
end
if mod(n,2)==0
  disp('numero composto')
  return
end
N=floor(sqrt(n));
for i=3:2:N
  if mod(n,i)==0
    disp('numero composto')
    return
  end
end
disp('numero primo')
```

Come vedrete, questo programma permette di identificare in un tempo ragionevole numeri primi con 10 o 11 cifre. Un test del genere si adatta invece poco ai numeri primi più grandi noti al giorno d'oggi, che contano centinaia (e in alcuni casi decine di migliaia) di cifre.

10.2 Il crivello di Eratostene

Il *crivello di Eratostene* è un metodo per trovare *tutti* i primi compresi tra 1 e un numero naturale dato n. Il principio utilizzato è molto semplice: si parte dall'insieme iniziale

$$2\,,\,3\,,\,4\,,\,\ldots n\,,$$

che sicuramente contiene la lista dei primi cercati. Successivamente si prende il primo elemento della lista, cioè 2, che è primo, e si eliminano tutti i suoi multipli eccettuato 2 stesso:

$$2\,,\,3\,,\,\underline{4}\,,\,5\,,\,\underline{6}\,,\,7\,,\,\underline{8}\,,\,\ldots\,,\,n\,,$$

$$2\,,\,3\,,\,5\,,\,7\,,\,9\,,\ldots\,,n$$

(supponendo per semplicità che n sia dispari). Ora si prende il secondo elemento della nuova lista, che è ancora primo, e si sopprimono tutti i suoi multipli eccettuato lui stesso:

$$2\,,\,3\,,\,5\,,\,7\,,\,\underline{9}\,,\,11\,,\,13\,,\,\underline{15}\,,\,17\,,\,19\,,\,\underline{21}\,,\,\ldots\,,\,n\,,$$

$$2\,,\,3\,,\,5\,,\,7\,,\,11\,,\,13\,,\,17\,,\,19\,,\ldots\,,n\,.$$

A questo punto è chiaro quale sarà il passo successivo: prendere il terzo elemento della nuova lista ed eliminare tutti i suoi multipli eccettuato lui stesso; e così via.
In questo modo si passa la lista iniziale attraverso numerosi *setacci* (crivello è sinonimo di setaccio) fino ad eliminare tutti i numeri non primi.
Quanti passi è necessario fare? Anche in questo caso è sufficiente eliminare tutti i multipli dei primi p delle liste via via ridotte, fino a che $p^2 \leq n$. Infatti, supponiamo di aver fatto un certo numero di iterazioni, e che il successivo numero primo p di cui dobbiamo eliminare i multipli sia strettamente maggiore di $\sqrt{n}$. Supponiamo per assurdo che ci sia un multiplo di p ancora presente nella lista, questo avrà la forma

$$kp\,,\quad k \in \mathbf{N}\,.$$

Ma

$$kp \leq n\,,\quad p > \sqrt{n} \quad \Longrightarrow k < \quad n\,,$$

dunque la lista contiene un multiplo di k che è più piccolo di p, e questo è assurdo.
Vogliamo ora implementare il crivello di Eratostene in un programma per Matlab. Procediamo nel modo seguente:

- dato il numero n, definiamo un vettore X fatto da n componenti,

  $$X = (x_1, \ldots, x_n)\,,$$

  inizialmente tutte uguali ad 1

  $$x_i = 1\,,\quad \forall i = 1, \ldots n\,.$$

  Pensiamo gli elementi di X in corrispondenza biunivoca con i numeri $1, 2, \ldots, n$ (1 corrisponde a x_1, 2 a x_2 e così via);

- Poniamo $N = [\sqrt{n}\,]$; per ogni j tra 1 e N, consideriamo tutti i multipli $s = kj$ di j, diversi da j e minori di n; questi in base al procedimento del crivello, devono essere eliminati. Realizziamo questa eliminazione ponendo la s–esima componente di X uguale a zero, per ogni s siffatto. Inoltre poniamo $x_1 = 0$ poichè 1 non è primo (questo passo può essere fatto inizialmente).

- Al termine di tutte le iterazioni del tipo descritto al passo precedente, le componenti del vettore risultante X verificano:

  $$x_i = \begin{cases} 1 & \text{se } i \text{ è primo,} \\ \\ 0 & \text{se } i \text{ non è primo.} \end{cases}$$

- A questo punto è facile passare da X ad un vettore contenente la lista dei primi minori di n: innanzitutto poniamo

$$x_i = i\, x_i \quad i = 1, 2, \ldots, n\,,$$

  e infine eliminiamo da X tutte le componenti uguali a zero.

Il procedimento che abbiamo descritto viene messo in pratica nel programma che segue.

Il crivello di Eratostene

```
function X=eratostene(n)
X=ones(1,n);
N=floor(sqrt(n));
X(1)=0;
for j=2:N
  r=2;
  while r*j<=n
    s=r*j;
    X(s)=0;
    r=r+1;
  end
end
for k=1:n
  X(k)=k*X(k);
end
X=setdiff(X,0);
```

Il programma che segue è un diverso modo di implementare il crivello di Eratostene.

```
function L=eratostene2(n)
N=2:n;
L[];
while ~isempty(N)
   m=N(1);
   L=[L m];
   mult=m:m:n; % multipli di m
   N=setdiff(N,mult);
end
```

In questo programma `N` è il vettore formato inizialmente da tutti i naturali tra 2 e `n`, mentre `L` è la variabile destinata a contenere la lista dei primi cercata, inizializzata come vettore *vuoto*. Il ciclo while viene iterato finchè `N` è diverso dal vettore vuoto. Ad ogni iterazione del ciclo si aggiunge a `L` il primo elemento `m` di `N`, e si eliminano da `N` stesso tutti i multipli di `m`, lui compreso.

Due numeri primi p e q che differiscono di due unità:

$$p = q \pm 2\,,$$

sono detti *primi gemelli*. Esempi di coppie di primi gemelli alla portata di semplici calcoli, sono $(11, 13)$, $(197, 199)$, ma sono state scoperte anche coppie di primi gemelli straordinariamente grandi come:

$$311700055 \cdot 2^{39020} \pm 1\,.$$

Una questione che ancora non è stata risolta è se esistano infinite coppie di numeri primi gemelli oppure no. Nell'Esercizio 10.6 vi verrà chiesto di scrivere un programma, basato sul crivello di Eratostene, che elenchi tutte le coppie di primi gemelli più piccoli di un numero assegnato.

10.3 La funzione di Gauss

I numeri primi sono infiniti; questo semplice risultato, noto da molto tempo, risponde alla domanda: *quanti sono i numeri primi?* Eppure la domanda merita ancora di essere posta, per avere una risposta in qualche modo più precisa. Potremmo ad esempio chiedere: *come sono distribuiti i numeri primi tra i numeri naturali?* Per rendere più concreto questo interrogativo, supponiamo di avere una scatola con n palline ciascuna delle quali è contrassegnata da uno dei numeri interi tra 1 e n. Immaginiamo che n sia abbastanza grande. Quante possibilità abbiamo, pescando una pallina a caso, di prenderne una contrassegnata da un numero primo? Questa probabilità è pari al numero di primi minori di n diviso n stesso.
Definiamo allora la funzione di Gauss:

$$\pi(n) = \#\{p \,\text{primo} \,:\, p \leq n\}\,, \quad n \in \mathbf{N}\,.$$

Dunque $\pi(n)$ è il numero di primi minori di n. Il fatto che i numeri primi siano infiniti ci dice subito che:

$$\lim_{n\to\infty} \pi(n) = +\infty\,.$$

D'altra parte il problema che emerge dall'esempio precedente equivale a determinare:

$$\frac{\pi(n)}{n}\,, \quad n \in \mathbf{N}\,. \tag{10.1}$$

Conoscere questa quantità vuol dire determinare un'espressione esplicita per $\pi(n)$ e questo non è attuabile. Quello che per molto tempo è stato uno dei problemi più studiati in teoria dei numeri, è il *comportamento asintotico* di $\pi(n)$ al tendere di n all'infinito, ovvero l'ordine di grandezza con cui la funzione di Gauss cresce. Determinare il comportamento asintotico, permetterebbe ad esempio di avere un valore approssimato del rapporto (10.1) per n grande.
Proviamo ad affrontare il problema sperimentalmente, con gli strumenti che abbiamo a disposizione. Cominciamo con lo scrivere un programma che ci dia il valore numerico della funzione di Gauss per ogni intero n. Utilizzando il programma che implementa il crivello di Eratostene, questo è semplicissimo:

```
function f=gauss(n)
f=size(eratostene(n),2);
```

Un unico commento: poichè la funzione `eratostene` applicata a un numero n produce come risultato un vettore riga con tutti i numeri primi minori di n, per ottenere $\pi(n)$ è sufficiente contare le colonne di questo vettore.

Tramite questo programma possiamo fare un *campionamento* di $\pi(n)$ per una gamma abbastanza estesa di n. Questo campionamento è riportato nella tabella che segue. Come vedete, per ogni n non viene calcolato solo $\pi(n)$, ma anche altre due quantità:

$$\frac{\pi(n)}{n}\,, \qquad \frac{\pi(n)\log n}{n}\,.$$

La prima l'abbiamo già incontrata, della seconda parleremo tra poco.

n	$\pi(n)$	$\pi(n)/n$	$(\pi(n)\log(n))/n$
1	0	0	0
2	1	0.5000	0.3466
3	2	0.6667	0.7324
4	2	0.5000	0.6931
5	3	0.6000	0.9657
6	3	0.5000	0.8959
7	4	0.5714	1.1119
8	4	0.5000	1.0397
9	4	0.4444	0.9765
10	4	0.4000	0.9210
20	8	0.4000	1.1983
30	10	0.3333	1.1337
40	12	0.3000	1.1067
50	15	0.3000	1.1736
60	17	0.2833	1.1601
70	19	0.2714	1.1532
80	22	0.2750	1.2051
90	24	0.2667	1.1999
100	25	0.2500	1.1513
200	46	0.2300	1.2186
300	62	0.2067	1.1788
400	78	0.1950	1.1683
500	95	0.1900	1.1808
600	109	0.1817	1.1621
700	125	0.1786	1.1698
800	139	0.1737	1.1615
900	154	0.1711	1.1640
1000	168	0.1680	1.1605
10000	1229	0.1229	1.1319
100000	9592	0.0959	1.1043
1000000	78498	0.0784	1.0844

Proviamo a fare alcune considerazioni. Il valore $\pi(n)$ cresce, anche se non troppo velocemente; il rapporto $\pi(n)/n$ invece è sostanzialmente decrescente, dunque i numeri primi si diradano. Se pescate a caso un numero tra 1 e 100, questo è primo con probabilità un quarto. Se fate la stessa cosa con i primi centomila numeri naturali, avete meno del dieci per cento di possibilità di pescare un numero primo. La quarta quantità sembra stabilizzarsi, anche se molto lentamente, per n grande. Questa impressione trova la sua conferma in uno dei più importanti risultati della teoria dei numeri, noto come il Teorema

dei Numeri Primi, che è stato dimostrato alla fine del 1800. Il teorema afferma che:

$$\lim_{n\to\infty} \frac{\pi(n)\log(n)}{n} = 1\,.$$

In base a questo risultato, possiamo fare la seguente affermazione: *per n grande, la probabilità che un numero compreso tra 1 e n sia primo è circa il reciproco di* $\log(n)$.

10.4 Fattorizzazione

In questo paragrafo vediamo un programma di tipo funzione che, dato un numero naturale n dà come risultato la fattorizzazione in primi di n.
Il programma è basato sul procedimento elementare che noi tutti eseguiremmo *a mano* per fattorizzare n. L'unico punto da sottolineare è che possiamo servirci della lista di primi minori di n che ci viene data dal crivello di Eratostene.
Dato n, prendiamo l'insieme V formato da tutti i numeri primi minori di n. Si comincia a dividere n per il primo elemento p di V (cioè 2). Contemporaneamente inizializziamo un contatore c uguale a zero, destinato ad essere l'esponenente della potenza massima di p che divide n. Se il resto della divisione di n per p è zero ridefiniamo $n = n : p$ e aumentiamo di una unità c; ripetiamo questa operazione fino a che n diviso p dà resto zero. Al termine di queste iterazioni, mettiamo da parte la coppia (p, c) (se c è maggiore di zero), questa farà parte della fattorizzazione di n.
Questo procedimento deve essere ripetuto per ogni elemento p di V. Alla fine avremo messo da parte un insieme di coppie (p_i, c_i), con $i = 1, 2, \ldots k$ per un certo intero k, formate ciascuna da un numero primo e dal corrispondente esponente; queste coppie consentono di scrivere la fattorizzazione di n:

$$n = p_1^{c_1} \cdot p_2^{c_2} \cdot \cdots \cdot p_k^{c_k}\,.$$

Programma per la fattorizzazione in primi di un numero naturale

```
function M=fattorizzazione(n)
M=ones(0,2);
V=eratostene(n);
for p=V
  c=0;
  while mod(n,p)==0
    n=n/p;
    c=c+1;
  end
  if c>0
    M=[M ;[p c]];
  end
  if n==1
```

```
    return
  end
end
```

Osservate che l'ultimo ciclo `if` serve ad evitare ulteriori iterazioni nel caso in cui la scomposizione sia completamente avvenuta prima di aver fatto assumere a p tutti i valori di V.

10.5 Crittografia: il metodo RSA

La *crittografia* è la disciplina che si occupa di cifrare messaggi. La situazione tipica in cui la crittografia viene impiegata è quella di una persona, il mittente, che deve mandare un messaggio ad un'altra persona, il destinatario, senza che durante il viaggio il contenuto del messaggio possa essere compreso da altri. Il mittente altera la forma del messaggio, seguendo un opportuno metodo, ovvero *codifica* il messaggio, in modo da renderlo incomprensibile a chiunque non sia a conoscenza del metodo stesso. Il destinatario, applica al messaggio ricevuto un procedimento di *decodifica* che gli permette di riottenere il testo originale.
La crittografia è una scienza molto antica e nel corso della storia sono stati utilizzati molti metodi distinti di cifratura (codifica). In questo paragrafo ci occupiamo del metodo noto come RSA, messo a punto negli anni 1970 che è attualmente uno dei più diffusi per quanto riguarda la trasmissione elettronica di informazioni.
In questo metodo, a differenza di quanto accade nella maggior parte dei metodi di cifratura, la codifica e la decodifica del messaggio non sono due operazioni del tutto simmetriche.

Il primo passo da fare per codificare un messaggio è quello di passare da lettere dell'alfabeto a numeri. Questo viene fatto in modo molto semplice, stabilendo una corrispondenza biunivoca tra le 26 lettere del nostro alfabeto e i numeri

$$01 \quad 02 \quad 03 \quad \dots \quad 14 \quad 15 \quad \dots \quad 25 \quad 26\,,$$

in modo che A corrisponda a 01, B a 02 e così via. Al simbolo 00 corrisponde lo spazio tra due parole. Secondo questo schema

PAROLE E NUMERI

diventa:

16 01 18 15 12 05 00 05 00 14 21 13 05 18 09

Successivamente scegliamo un numero intero r e raggruppiamo le coppie in blocchi di r elementi ciscuno. Se ad esempio scegliamo $r = 3$ otteniamo

160118 151205 000500 142113 051809

(se il numero di coppie non è un multiplo di r, alla fine si possono aggiungere delle coppie di zeri per avere tutti blocchi della stessa lunghezza). In questa nuova forma, ciascun

blocco del messaggio viene detta una *parola*; quindi una parola è un numero di al più $2r$ cifre.

Veniamo adesso alla codifica vera e propria. Questa si basa su alcuni risultati di algebra. Il punto di partenza può essere considerato il Teorema di Fermat, che afferma che se p è un numero primo e a è un numero intero non divisibile per p, allora:

$$a^{p-1} \equiv 1 \pmod{p}\,.$$

Un forma più generale di questo risultato è il Teorema di Eulero. Ricordiamo che dato un numero naturale m, la funzione di Eulero $\phi(m)$ è data dal numero di naturali strettamente minori di m e primi con m. Il Teorema di Eulero afferma che, se a e m sono numeri naturali con $a < m$ e a *è primo con* m, allora:

$$a^{\phi(m)} \equiv 1 \pmod{m}\,.$$

Il metodo RSA utilizza una forma ancora più generale di questi risultati. Prima di enunciarla ricordiamo che un numero naturale m si dice *libero da quadrati* se non è divisibile per il quadrato di nessun numero intero, ovvero se nella scomposizione in fattori primi di m ogni fattore compare con esponente uno.

Proposizione. *Sia m un numero naturale libero da quadrati, allora per ogni intero a e per ogni naturale k:*

$$a^{k\phi(m)+1} \equiv a \pmod{m}\,.$$

Gli ingredienti per codificare un messaggio sono:

1. un numero naturale r, che determina la lunghezza delle parole nel modo descritto sopra;
2. un numero naturale m, libero da quadrati, con almeno $2r+1$ cifre;
3. un numero naturale s primo con $\phi(m)$;
4. un numero t tale che
$$st \equiv 1 \pmod{\phi(m)}\,, \tag{10.2}$$
t esiste, questo segue dal fatto che s è primo con $\phi(m)$.

La codifica viene fatta come segue. Si prende una parola w; w è un numero di al più $2r$ cifre e quindi minore di m. Si considera

$$w^s\,,$$

e si trova la classe di resto modulo m di w^s, ovvero un naturale z, con $z < m$, tale che

$$z \equiv w^s \pmod{m}\,.$$

Il numero z è la forma codificata della parola w ed è quello che viene spedito dal mittente.

Il destinatario riceve z; per decodificarlo calcola:

$$z^t$$

e poi trova la classe di resto modulo m di z^t, ovvero un numero w' tale che $w' < m$ e

$$w' \equiv z^t \pmod{m}\,.$$

La codifica è avvenuta, cioè affermiamo che

$$w = w'\,.$$

Infatti

$$w' \equiv z^t \equiv w^{st} \pmod{m}\,;$$

inoltre, poichè $st \equiv 1 \pmod{\phi(m)}$,

$$st = k\phi(m) + 1$$

per un opportuno numero naturale k. Quindi

$$w^{st} = w^{k\phi(m)+1} \equiv w \pmod{m}\,,$$

grazie alla Proposizione enunciata prima. In conclusione w e w' sono congruenti modulo m, ed essendo entrambi strettamente minori di m coincidono.

Nella pratica questo metodo viene fatto funzionare scegliendo m uguale al prodotto di due numeri primi molto grandi p e q. Quindi m è libero da quadrati e

$$\phi(m) = (p-1)(q-1)\,.$$

Successivamente m può essere reso di pubblico dominio, così come s, l'importante è che rimangano segreti p e q. Con la coppia (m, s) chiunque può codificare un messaggio, per decodificarlo però sarà necessario conoscere t e per conoscere t non si può prescindere da $\phi(m)$ in base alla (10.2). Dunque se qualcuno entra in possesso di un messaggio codificato, deve risalire a $\phi(m)$ conoscendo m, ovvero deve fattorizzare m. Questo è il punto di forza del metodo RSA, infatti non esistono algoritmi che, implementati sui calcolatori oggi a disposizione, permettano di fattorizzare numeri dell'ordine di 10^{200} in tempi ragionevoli.

10.6 Codici a correzione di errore

I codici a correzione di errore permettono di trasmettere dati in modo tale che, se qualche dato viene alterato nella trasmissione, si possa ricostruire il dato iniziale.

Sia F un campo finito di cardinalità q; chiameremo F "insieme dei simboli" e chiameremo F^n "insieme delle parole".
Nello spazio vettoriale F^n si introduce una metrica, detta metrica di Hamming, definita nel modo seguente:

$$d(x, y) = \#\{i |\ x_i \neq y_i\}$$

(quindi due parole si dicono distanti d se d coordinate sono differenti e le restanti sono uguali).
Se x è un punto di F^n si definisce la palla di centro x e raggio t, che si indica con $B(x,t)$, nel modo seguente:

$$B(x,t) = \{y \in F^n |\ d(x,y) \leq t\}$$

Un codice a correzione di e errori in F^n è un insieme C di punti di F^n (che saranno le parole del nostro vocabolario) tali che $B(x,e) \cap B(y,e) = \emptyset\ \forall x, y \in C$ (ovviamente ciò è equivalente a dire che $d(x,y) > 2e\ \forall x, y \in C$).
Questo significa che se considero una parola del codice e faccio meno di e "errori di ortografia" cioè sbaglio meno di e coordinate della parola, la parola alterata che ho così ottenuto sta in una sola delle palle di centro una parola del codice e raggio e e quindi sono in grado di ricostruire da quale parola del codice proveniva.

Definizione 10.1 *Un codice C a correzione di e errori in F^n si dice perfetto se F^n è ricoperto dalle palle di centro una parola del codice e raggio e.*

Definizione 10.2 *Un codice si dice lineare di tipo (n,k) se è un sottospazio vettoriale di F^n di dimensione k.*

Definizione 10.3 *Si dice peso di una parola x la sua distanza da 0 (quindi la cardinalità delle coordinate di x diverse da 0).*

Osservazione 10.1 *Il minimo fra le distanze dei punti di un codice lineare è uguale al minimo dei pesi delle parole del codice diverse da 0.*

Studieremo adesso dei particolari codici lineari, detti codici di Hamming, che correggono un errore e sono perfetti.
Sia m un numero naturale. Per ognuna delle rette per l'origine di F^m si considera un punto diverso da 0; si ottengono così $n := (q^m - 1)/(q-1)$ colonne di lunghezza m (infatti su ogni retta per l'origine di F^m ci sono $q-1$ punti diversi da 0 e i punti diversi da 0 di F^m sono ovviamente $q^m - 1$). Affiancando queste n colonne, si ottiene una matrice H $m \times n$ di rango m. Si definsce codice di Hamming di tipo $(n, n-m)$ il seguente sottospazio di F^n di dimensione $n-m$:

$$C = \{x \in F^n \mid Hx = 0\}$$

Teorema 10.1 *Il peso minimo di ogni parola diversa da 0 in un codice di Hamming è ≥ 3 (quindi i codici di Hamming sono codici capaci di correggere un errore).*

Dimostrazione. Se per assurdo ci fosse un codice di Hamming con una parola di peso ≤ 2, si avrebbe che tale parola avrebbe al più due coordinate diverse da 0 e quindi la matrice H che definsce il codice avrebbe due colonne linearmente dipendenti e ciò è assurdo per come è stata costruita la matrice H. *Q.e.d.*

Teorema 10.2 *I codici di Hamming sono perfetti.*

Dimostrazione. Consideriamo un codice di Hamming di tipo $(n, n-m)$. Occorre dimostrare che le palle di centro una parola del codice e di raggio 1 ricoprono tutto F^n.
Il numero delle parole del codice è q^{n-m}.
Il numero dei punti in ogni palla di raggio 1 è $1 + n(q-1)$, infatti ogni palla di raggio 1 contiene il centro e gli $n(q-1)$ punti che si ottengono dal centro modificando una coordinata (ognuna delle n coordinate può essere modificata in $q-1$ modi); dato che $n = (q^m - 1)/(q-1)$, si ha che $1 + n(q-1) = q^m$.
Quindi la cardinalità dell'unione delle palle di raggio 1 e centro una parola del codice è $q^{n-m}q^m = q^n$, che è anche la cardinalità di F^n. *Q.e.d.*

10.7 Esercizi

Esercizio 10.1 *Trascrivete sul computer e salvate il programma* `primalita1`*, verificando che funzioni. In particolare potete provare il programma sui seguenti numeri interi, che sono tutti primi:*

$$n_1 = 641\,, \quad n_2 = 65537\,, \quad n_3 = 274177\,, \quad n_5 = 6700417\,, \quad n_6 = 45592577$$
$$n_7 = 825753601\,, \quad n_8 = 1214251009\,, \quad n_9 = 70125124609\,,$$

e sui primi cinque numeri di Fermat*:*

$$F_i = 2^{2^i} + 1\,, \quad i = 1, 2, 3, 4, 5\,.$$

Esercizio 10.2 *Trascrivete il programma* `eratostene` *e provate a farlo girare per vari valori di n.*

Esercizio 10.3 **Scrivete un programma che, dati due numeri naturali N e M, con $N \leq M$, determini tutti i numeri primi compresi tra N e M, estremi inclusi.*

Esercizio 10.4 **Scrivete un programma che, dato un numero naturale N, calcoli il numero $f(N)$ di numeri primi compresi tra N e $N + 100$ e calcoli il rapporto:*

$$\frac{f(N)}{N}\,.$$

Calcolate questo rapporto per valori grandi di N.

Esercizio 10.5 **Costruite un test di primalità basato sul crivello di Eratostene, nel modo seguente:*
(i) dato un numero naturale n, definite v come l'insieme di tutti i numeri primi minori o uguali a n; in altre parole:

```
v=eratostene(n);
```

(ii) tramite la funzione di Matlab `ismember`*, verificate se n appartiene a v oppure no; se vi appartiene allora è primo.*

Esercizio 10.6 **Scrivete un programma di tipo funzione che, dato un numero naturale n trovi tutte le coppie di primi gemelli minori o uguali a n.*
[Suggerimento. Per prima cosa, dato n, tramite la funzione `eratostene`, create una lista V di tutti i primi minori di n. Successivamente, con un ciclo `for`, identificate tutte le coppie di elementi consecutivi di V la cui differenza è due (infatti, due primi gemelli sono necessariamente consecutivi).]

Esercizio 10.7 **In questo esercizio ci proponiamo di visualizzare graficamente l'andamento della funzione di Gauss $\pi(n)$, $n \in \mathbf{N}$. Prima di tutto scrivete e salvate il programma* `gauss` *per calcolare il valore di π. Adesso scrivete un programma che, dato un numero N* campioni *la funzione di Gauss per tutti gli interi tra 1 e N, ovvero crei un vettore di N componenti, la cui n-esima componente è il valore $\pi(n)$.*
Ora potete creare un istogramma del vettore `camgauss(N)`*, mediante il comando* `bar`*:*

```
>> bar(camgauss(100))
```

Esercizio 10.8 *Seguendo lo schema dell'esercizio precedente, campionate le successioni:*

$$\frac{\pi(n)}{n}\,, \quad \frac{\pi(n)\log(n)}{n}\,,$$

e visualizzatene l'andamento mediante un istogramma.

Esercizio 10.9 *Scrivete il programma* `fattorizzazione`*, e verificate il suo funzionamento su semplici esempi.*

Esercizio 10.10 **Scrivete un programma di tipo funzione che ad un numero naturale n associ il numero di tutti i naturali che dividono n, eccettuati 1 e n stesso. Ad esempio*

$$\begin{aligned} 2 &\longrightarrow 0 \\ 6 &\longrightarrow 2 \\ 12 &\longrightarrow 4 \\ 20 &\longrightarrow 4 \\ 25 &\longrightarrow 1 \end{aligned}$$

Salvate il programma con il nome, ad esempio, `numdidiv`*.*

Esercizio 10.11 **Scrivete un programma per campionare la funzione* `numdidiv` *definita nell'esercizio precedente, in un generico intervallo $[1, N]$. Successivamente disegnate un suo istogramma con la funzione* `bar` *(vedi Esercizio 10.7) per vari valori di N.*

Esercizio 10.12 **Scrivete un programma che dato un numero n calcoli il numero di divisori* primi *di n.*

Esercizio 10.13 *Scrivete un programma che associ ad un numero naturale n un vettore formato da tutti i divisori primi di n*

Esercizio 10.14 **Scrivete un programma di tipo funzione che dato n naturale calcoli la funzione di Eulero $\phi(n)$, ovvero il numero di naturali $k < n$ tali che $\mathrm{mcd}(k, n) = 1$. Per scrivere il programma dovete utilizzare il programma* `mcd.m`*, descritto nella prima parte del corso, che calcola il massimo comun divisore tra due numeri. Verificate di aver scritto un buon programma calcolando ad esempio:*

$$\phi(p\,q)$$

con p e q primi; la risposta deve essere $(p-1)(q-1)$.

Esercizio 10.15 **Scrivete un programma che verifichi se un numero n è libero da quadrati oppure no.*

Esercizio 10.16 **Implementate il metodo RSA mediante due programmi, uno di codifica e uno di decodifica. Prendete $r = 1$ (ogni* parola *è quindi un numero minore o uguale a 26 e corrisponde ad una singola lettera dell'alfabeto), $m = 33 = 3 \times 11$, e scegliete s e t di conseguenza. Per verificare di aver scritto i programmi giusti date l'istruzione*

`decodifica(codifica(`i`))` ,

per ogni i compreso tra 1 e 26 la risposta deve essere i.

Nei due esercizi successivi le notazioni sono quelle del Paragrafo 10.6.

Esercizio 10.17 *Sia $q = 2$, quindi $F = \mathbf{Z}/2$; sia $m = 3$ e $n = (2^3 - 1)/(2 - 1) = 7$; costruite "a mano" la matrice H che definsce il codice di Hamming di tipo $(n, n - m)$ e fate un programma che vi scriva tutte le parole del codice e che, se inserite una parola di F^n, vi dica se tale parola appartiene al codice o no ed eventualmente se non appartiene al codice ve la corregga.*

Esercizio 10.18 *Sia $q = 2$, quindi $F = \mathbf{Z}/2$; fate un programma di tipo funzione tale che, se inserite una parola di F^n, vi dica se tale parola appartiene al codice di Hamming di tipo $(n, n - m)$ ed eventualmente, se non appartiene al codice, ve la corregga (quindi il programma sarà funzione di m e di una parola in F^n con $n = (2^m - 1)/(2 - 1)$).*

Capitolo 11

Formule di quadratura

Il Teorema Fondamentale del Calcolo Integrale ci dice che, se f è una funzione continua in $[a, b]$ e F è una primitiva di f, allora

$$\int_a^b f(x)\,dx = F(b) - F(a)\,.$$

Questa formula permette di calcolare il valore esatto dell'integrale di f, ma richiede di conoscere un'espressione esplicita della primitiva F; questo obiettivo può presentare delle serie difficoltà, basti pensare che funzioni come

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

non ammettono primitive che siano esprimibili come *combinazioni finite* di funzioni elementari.
In questa parte ci occupiamo di formule che servono a determinare, in modo approssimato, l'integrale definito di una funzione di una variabile reale:

$$\int_a^b f(x)\,dx\,. \tag{11.1}$$

Queste formule nascono per la maggior parte dalla definizione stessa di integrale; più precisamente si rifanno al principio per cui l'integrale di f può essere approssimato da una sommatoria del tipo:

$$\sum_{k=1}^{n} f(\xi_k)(x_k - x_{k-1}) \tag{11.2}$$

dove

$$x_0 = a < x_1 < \cdots < x_n = b\,,$$

è una partizione di $[a,b]$ e $\xi_k \in [x_{k-1} - x_k]$ per ogni k. Osserviamo che l'integrabilità di f garantisce che, indicata con δ la massima ampiezza degli intervalli della partizione:

$$\delta := \max\{x_k - x_{k-1} \,:\, k = 1, \ldots, n\}\,,$$

la sommatoria (11.2) è arbitariamente vicina la valore dell'integrale (11.1) purché δ sia sufficientemente piccolo.
In questo capitolo ci limitiamo a considerare *partizioni equispaziate* di $[a,b]$, ovvero tali che:

$$x_k - x_{k-1} = \frac{b-a}{n} =: h\,, \quad \forall\, k = 1, \ldots, n\,. \tag{11.3}$$

In queste ipotesi abbiamo

$$x_k = a + k\,h\,, \quad k = 0, 1, \ldots, n \tag{11.4}$$

e

$$\sum_{k=1}^{n} f(\xi_k)(x_k - x_{k-1}) = h \sum_{k=1}^{n} f(\xi_k)\,. \tag{11.5}$$

Inoltre, trattiamo solo il caso di funzioni integrande con doti di sufficiente regolarità, più precisamente supponiamo che f abbia derivate di ogni ordine, continue in $[a,b]$.

11.1 Prime formule di quadratura

In questo paragrafo vedremo quattro distinte formule di quadratura. Le prime tre si ottengono direttamente dalla (11.5) per opportune scelte dei punti ξ_k. Se per ogni k scegliamo come ξ_k l'estremo sinistro di $[x_{k-1}, x_k]$, ovvero x_{k-1}, otteniamo:

$$ps(n) := h \sum_{k=1}^{n} f(x_{k-1}) = h \sum_{k=1}^{n} f(a + (k-1)h) \tag{11.6}$$

dove si è usata anche la (11.4). Se scegliamo l'estremo di destra abbiamo:

$$pd(n) := h \sum_{k=1}^{n} f(a + kh)\,. \tag{11.7}$$

Infine possiamo scegliere il punto medio:

$$\xi_k = \frac{x_{k-1} + x_k}{2} = a + (k - 1/2)h\,, \quad k = 1, \ldots, n$$

ottenendo:

$$pm(n) := h \sum_{k=1}^{n} f(a + (k - 1/2)h)\,. \tag{11.8}$$

Volendo semplificare (cioè supponendo f positiva), possiamo dire che nelle formule precedenti si calcola, come approssimazione dell'integrale di f, la somma delle aree di rettangoli con base di ampiezza fissata e altezza uguale al valore di f in un punto della base. Nella

formula successiva, detta *formula dei trapezi*, si considera, per il k-esimo intervallo della partizione, il trapezio (rettangolo) che ha per vertici i punti $(x_{k-1}, 0)$, $(x_k, 0)$, $(x_k, f(x_k))$ e $(x_{k-1}, f(x_{k-1}))$; la sua area è data da

$$\frac{f(x_{k-1}) + f(x_k)}{2} (x_k - x_{k-1}) = h \, \frac{f(a + (k-1)h) + f(a + kh)}{2} \,,$$

sommando rispetto a k si ottiene:

$$\begin{aligned} t(n) &:= h \sum_{k=1}^{n-1} \frac{f(a + (k-1)h) + f(a + kh)}{2} \\ &= h \left[\frac{1}{2}(f(a) + f(b)) + \sum_{k=1}^{n-1} f(a + kh) \right] . \end{aligned} \tag{11.9}$$

Osserviamo che un'immediata conseguenza delle precedenti definizioni è l'uguaglianza

$$t(n) = \frac{ps(n) + pd(n)}{2} \,. \tag{11.10}$$

Da quest'ultima formula e da quanto abbiamo detto nel paragrafo precedente, segue che:

$$\lim_{n \to +\infty} ps(n) = \lim_{n \to +\infty} pd(n) = \lim_{n \to +\infty} pm(n) = \lim_{n \to +\infty} t(n) = \int_a^b f(x) \, dx \,.$$

Nel seguito cercheremo di determinare e confrontare tra loro le velocità di convergenza delle varie formule di quadratura. Osserviamo che, per quanto riguarda il confronto per valori grandi di n, si possono utilizzare le seguenti formule:

$$\begin{aligned} ps(2^{p+1}) &= \frac{1}{2} \left[ps(2^p) + pm(2^p) \right] \\ pd(2^{p+1}) &= \frac{1}{2} \left[pd(2^p) + pm(2^p) \right] \\ t(2^{p+1}) &= \frac{1}{2} \left[t(2^p) + pm(2^p) \right] , \quad p = 1, 2, \ldots \end{aligned} \tag{11.11}$$

Le prime due formule sono intuitivamente immediate. Vediamo comunque una dimostrazione della prima:

$$\begin{aligned} ps(2^p) &= \frac{b-a}{2^p} \sum_{k=1}^{2^p} f(a + (k-1)(b-a)/2^p) \\ &= 2 \, \frac{b-a}{2^{p+1}} \sum_{k \, \text{pari} \,, \, k=2}^{2^{p+1}} f(a + (k-2)(b-a)/2^{p+1}) \,. \end{aligned}$$

Analogamente:

$$\begin{aligned} pm(2^p) &= \frac{b-a}{2^p} \sum_{k=1}^{2^p} f(a+(k-1/2)(b-a)/2^p) \\ &= 2\,\frac{b-a}{2^{p+1}} \sum_{k\,\text{pari}\,,\,k=2}^{2^{p+1}} f(a+(k-1)(b-a)/2^{p+1}) \\ &= 2\,\frac{b-a}{2^{p+1}} \sum_{k\,\text{dispari}\,,\,k=1}^{2^{p+1}-1} f(a+k(b-a)/2^{p+1})\,. \end{aligned}$$

Sommando termine a termine le ultime si ottiene la prima delle (11.11). La dimostrazione della seconda è del tutto analoga, mentre la terza si ottiene dalle prime due e dalla (11.10). Le formule (11.11) verranno utilizzate negli esercizi per confrontare le formule di quadratura viste sinora su un esempio che farà da banco di prova:

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi\,. \tag{11.12}$$

Questo esempio può essere utilizzato in due modi. Assumendo come noto il valore dell'integrale, si può analizzare la velocità di convergenza delle successioni $ps(n)$, $pd(n)$, $pm(n)$ e $t(n)$; questa analisi ci permetterà di verificare che le ultime due convergono più rapidamente delle altre al valore esatto.
Rovesciando il punto di vista, si potrebbe pensare di determinare delle buone approssimazioni di π calcolando l'integrale della (11.12) con le formule di quadratura. Per far questo è necessario avere delle informazioni a priori sugli errori che si commettono con ciascuna delle varie formule, in dipendenza da n:

$$\epsilon_1(n) := \left|\int_a^b f(x)\,dx - ps(n)\right|\,, \quad \epsilon_2(n) := \left|\int_a^b f(x)\,dx - pd(n)\right|\,,$$

$$\epsilon_3(n) := \left|\int_a^b f(x)\,dx - pm(n)\right|\,, \quad \epsilon_4(n) := \left|\int_a^b f(x)\,dx - t(n)\right|\,.$$

Prima di enunciare il risultato che ci darà questo tipo di informazione, introduciamo due numeri positivi M_1 e M_2 con la seguente proprietà:

$$|f'(x)| \le M_1\,, \quad |f''(x)| \le M_2\,, \quad \forall\, x \in [a,b]\,. \tag{11.13}$$

Osserviamo che M_1 e M_2 sono ben definiti perchè abbiamo supposto che f abbia derivate di ogni ordine continue in $[a,b]$.

Teorema 11.1 *Per ogni numero naturale n valgono le disuguaglianze:*

$$\begin{aligned} &\epsilon_1(n)\,,\,\epsilon_2(n) \le M_1\,\frac{(b-a)^2}{2n}\,, \\ &\epsilon_3(n) \le M_2\,\frac{(b-a)^3}{6n^2}\,, \quad \epsilon_4(n) \le M_2\,\frac{(b-a)^3}{12n^2} \end{aligned} \tag{11.14}$$

Prima di vedere la dimostrazione, torniamo all'esempio precedente. Per

$$f(x) = \frac{4}{1+x^2}\,, \quad x \in [a,b] = [0,1]\,,$$

si trova

$$|f'(x)| = \frac{8x}{(1+x^2)^2} \le 8\,, \quad \forall x \in [0,1]$$

(questa maggiorazione potrebbe essere migliorata trovando il massimo di $|f'(x)|$ in $[0,1]$) e

$$|f''(x)| = \frac{8(1-3x^2)}{(1+x^2)^3} \le 8\,, \quad \forall x \in [0,1]\,.$$

Dunque

$$|\pi - ps(n)|\,,\ |\pi - pd(n)| \le \frac{4}{n}\,, \quad \forall\, n \tag{11.15}$$

$$|\pi - pm(n)| \le \frac{4}{3n^2}\,, \quad |\pi - t(n)| \le \frac{2}{3n^2}\,, \quad \forall\, n\,.$$

Quindi, *in teoria*, se vogliamo calcolare il valore di π con precisione di un milionesimo (le prime cinque cifre decimali esatte) usando, ad esempio, la formula dei trapezi, dobbiamo scegliere

$$n \ge 1000\,\frac{\sqrt{2}}{\sqrt{3}} \approx 854\ldots$$

Dimostrazione del Teorema 11.1. Abbiamo:

$$ps(n) = \sum_{k=1}^{n} f(x_k)(x_k - x_{k-1}) = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} f(x_k)\,dx\,,$$

da cui

$$\int_a^b f(x)\,dx - ps(n) = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} (f(x) - f(x_k))\,dx\,.$$

D'altra parte

$$f(x) - f(x_k) = \int_{x_{k-1}}^{x} f'(t)\,dt\,;$$

sostituendo e passando ai valori assoluti si ha

$$\begin{aligned}
\left|\int_a^b f(x)\,dx - ps(n)\right| &\le \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} \left(\int_{x_{k-1}}^{x} |f'(t)|\,dt\right) dx \\
&\le M_1 \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} (x - x_{k-1})\,dx \\
&= M_1 \sum_{k=1}^{n} \frac{(x_k - x_{k-1})^2}{2} = M_1 \frac{(b-a)^2}{2n}\,.
\end{aligned}$$

La disuguaglianza per $pd(n)$ si dimostra in modo del tutto analogo.
Proviamo ora la disuguaglianza relativa a $\epsilon_3(n)$. Poniamo

$$y_k = \frac{x_{k-1} + x_k}{2}\,, \quad k = 1, \ldots, n\,.$$

Analogamente a quanto fatto prima possiamo scrivere:

$$\int_a^b f(x)\,dx - pm(n) = \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} (f(x) - f(y_k))\,dx\,.$$

Sviluppiamo la quantità $f(x) - f(y_k)$ con la formula di Taylor al primo ordine, con il resto in forma integrale:

$$f(x) - f(y_k) = f'(y_k)(x - y_k) + \int_{y_k}^{x} (x - t) f''(t)\,dt\,;$$

integrando ambo i membri rispetto a x tra x_{k-1} e x_k si ha:

$$\int_{x_{k-1}}^{x_k} (f(x) - f(y_k))\,dx = \int_{x_{k-1}}^{x_k} \left(\int_{y_k}^{x} (x - t) f''(t)\,dt \right) dx\,,$$

poichè, come si verifica facilmente,

$$\int_{x_{k-1}}^{x_k} (x - y_k)\,dx = 0\,.$$

Dunque:

$$\begin{aligned}
\epsilon_3(n) = \left| \int_a^b f(x)\,dx - pm(n) \right| &\leq \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} |f(x) - f(y_k)|\,dx \\
&\leq \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} \left(\int_{y_k}^{x} (x - t) |f''(t)|\,dt \right) dx \\
&\leq M_2 \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} \left(\int_{y_k}^{x} (x - t)\,dt \right) dx \\
&= \frac{M_2}{6} \sum_{k=1}^{n} (x_k - x_{k-1})^3 = M_2\,\frac{(b-a)^3}{6n^2}
\end{aligned}$$

Veniamo infine alla disuguaglianza riguardante $t(n)$. Abbiamo

$$\begin{aligned}
\int_a^b f(x)\,dx - t(n) &= \int_a^b f(x)\,dx - \sum_{k=1}^{n} \frac{f(x_{k-1}) + f(x_k)}{2} (x_k - x_{k-1}) \qquad (11.16) \\
&= \sum_{k=1}^{n} \int_{x_{k-1}}^{x_k} \left(f(x) - \frac{f(x_{k-1}) + f(x_k)}{2} \right) dx\,.
\end{aligned}$$

Sviluppiamo le differenze $f(x_k) - f(x)$ e $f(x_{k-1}) - f(x)$ con la formula di Taylor al primo ordine, con resto in forma integrale:

$$f(x_k) - f(x) = f'(x)(x_k - x) + \int_x^{x_k} (x_k - t) f''(t)\, dt\,,$$

$$f(x_{k-1}) - f(x) = f'(x)(x_{k-1} - x) + \int_x^{x_{k-1}} (x_{k-1} - t) f''(t)\, dt\,.$$

Moltiplichiamo la prima equazione per $(x_{k-1} - x)$, la seconda per $(x_k - x)$ e sottraiamo termine a termine:

$$f(x) - \left[\frac{x_k - x}{x_k - x_{k-1}} f(x_{k-1}) - \frac{x - x_{k-1}}{x_k - x_{k-1}} f(x_k)\right] =$$
$$= -\frac{1}{x_k - x_{k-1}} \left[(x_k - x) \int_{x_{k-1}}^{x} (t - x_{k-1}) f''(t)\, dt + (x - x_{k-1}) \int_x^{x_k} (x_k - t) f''(t)\, dt\right]$$

Integriamo ambo i membri di questa uguaglianza rispetto a x, tra x_{k-1} e x_k. Otteniamo così:

$$\int_{x_{k-1}}^{x_k} f(x)\, dx - \frac{f(x_{k-1}) + f(x_k)}{2}(x_k - x_{k-1}) =$$
$$= -\frac{1}{x_k - x_{k-1}} \left[\int_{x_{k-1}}^{x_k} (x - x_{k-1}) \left(\int_x^{x_k} (x_k - t) f''(t)\, dt\right) dx + \right.$$
$$\left. + \int_{x_{k-1}}^{x_k} (x_k - x) \left(\int_{x_{k-1}}^{x} (t - x_{k-1}) f''(t)\, dt\right) dx\right].$$

Da questa uguaglianza si ottiene, con facili passaggi:

$$\left|\int_{x_{k-1}}^{x_k} f(x)\, dx - \frac{f(x_{k-1}) + f(x_k)}{2}(x_k - x_{k-1})\right| =$$
$$\leq \frac{M_2}{6(x_k - x_{k-1})} \left[\int_{x_{k-1}}^{x_k} (x_k - x)^3\, dx + \int_{x_{k-1}}^{x_k} (x - x_{k-1})^3\, dx\right] = \frac{M_2\,(x_k - x_{k-1})^3}{12}\,.$$

La conclusione segue dalla (11.16).
□

11.2 La formula di Simpson

Si tratta della formula di quadratura più efficace e attualmente più usata nei programmi di calcolo.
Consideriamo una funzione regolare f in $[a, b]$ e, fissato n, prendiamo la usuale partizione equispaziata di $[a, b]$:

$$x_k = a + k\,\frac{b - a}{n}\,, \quad k = 0, 1, \ldots, n\,.$$

Fissiamo un intervallo della partizione: $[x_{k-1}, x_k]$. Si prova facilmente che esiste uno e un solo polinomio di secondo grado p che coincide con f nei punti x_{k-1}, $\frac{x_{k-1}+x_k}{2}$ e x_k (questo polinomio può essere determinato esplicitamente ma ciò non sarà necessario). Nella formula di Simpson l'integrale di f in $[x_{k-1}, x_k]$ viene approssimato dall'integrale di p. Osserviamo che, in generale, se p è un polinomio di secondo grado in x:

$$p(x) = Ax^2 + Bx + C\,, \quad A\,,\, B\,,\, C\, \in \mathbf{R}\,,$$

l'integrale di p in un intervallo $[\alpha, \beta]$ può essere scritto nel modo seguente:

$$\begin{aligned}\int_a^b p(x)\,dx &= \frac{1}{6}\left[2A(b^3-a^3) + 3B(b^2-a^2) + 6C(b-a)\right] \\ &= \frac{b-a}{6}\left[p(a) + p(b) + 4p\left(\frac{a+b}{2}\right)\right]\,.\end{aligned}$$

Ovvero l'integrale dipende solo dai valori che p assume negli estremi e nel punto medio dell'intervallo. In base alla formula appena trovata abbiamo:

$$\begin{aligned}\int_{x_{k-1}}^{x_k} p(x)\,dx &= \frac{x_k - x_{k-1}}{6}\left[p(x_{k-1}) + p(x_k) + 4p\left(\frac{x_{k-1}+x_k}{2}\right)\right] \\ &= \frac{x_k - x_{k-1}}{6}\left[f(x_{k-1}) + f(x_k) + 4f\left(\frac{x_{k-1}+x_k}{2}\right)\right]\,.\end{aligned}$$

Per avere un'approssimazione dell'integrale di f in $[a, b]$ sommiamo queste quantità rispetto a k; la somma definisce la formula di Simpson:

$$s(2n) := \sum_{k=1}^{n} \frac{x_k - x_{k-1}}{6}\left[f(x_{k-1}) + f(x_k) + 4f\left(\frac{x_{k-1}+x_k}{2}\right)\right]\,.$$

In base alla definizione che abbiamo dato, ad una partizione in n punti corrisponde la formula di Simpson di ordine $2n$; questo perchè in realtà nella formula appena scritta intervengono i valori di f in $2n$ punti, ovvero tutti i punti della partizione ed i punti medi di ogni singolo intervallo della partizione stessa.
Per ricollegarci alle formule di quadratura definite in precedenza, notiamo che una ovvia conseguenza della definizione è l'uguaglianza:

$$s(2n) = \frac{1}{3}\left[t(n) + 2pm(n)\right]\,, \quad \forall\, n \in \mathbf{N}\,.$$

In particolare da questa formula segue che $s(2n)$ converge all'integrale di f al tendere di n a $+\infty$. Inoltre, ricordando che la partizione è equispaziata, possiamo scrivere:

$$\begin{aligned}s(2n) &= \frac{h}{6}\sum_{k=1}^{n}\left[f(a+(k-1)h) + f(a+kh) + 4f(a+(k-1/2)h)\right] \\ &= \frac{h}{3}\left[\frac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(a+kh) + 2\sum_{k=1}^{n}(a+(k-1/2)h)\right]\,.\end{aligned}$$

Infine veniamo alla velocità di convergenza. Il teorema seguente, che riportiamo senza dimostrazione, afferma che in sufficienti ipotesi di regolarità l'errore che si commette calcolando $s(2n)$ al posto dell'effettivo integrale di f, tende a zero come $\frac{1}{n^4}$.

Teorema 11.2 *Sia M_4 una costante positiva tale che:*

$$|f^{(4)}(x)| \leq M_4\,, \quad \forall\, x \in [a,b]\,.$$

Allora

$$\left| \int_a^b f(x)\,dx - s(2n) \right| \leq \frac{M_4\,(b-a)^5}{10^3\,n^4}\,.$$

11.3 Esercizi

Esercizio 11.1 **Scrivere un programma di tipo funzione, che dipenda da una funzione f, da due numeri reali a e b e da un naturale n. Il risultato del programma deve essere calcolare il valore approssimato dell'integrale:*

$$\int_a^b f(x)\,dx$$

mediante la formula dei punti sinistri, su una partizione equispaziata di $[a,b]$ in n intervalli.

Esercizio 11.2 **Ripetere l'esercizio precedente per le formule di quadratura del punto destro, del punto medio e dei trapezi.*

Esercizio 11.3 **Confrontare l'efficienza delle varie formule di quadratura dei due precedenti esercizi, per calcolare l'integrale:*

$$\int_0^1 \frac{4}{1+x^2}\,dx = \pi = 3,14159265358979\ldots$$

Per fare questo, scrivete un programma (di tipo comando) che calcoli $ps(2^p)$, $pd(2^p)$, $pm(2^p)$ e $t(2^p)$ per ogni p da 1 a 10, utilizzando le formule:

$$\begin{aligned} ps(2^{p+1}) &= \frac{1}{2}\,[ps(2^p) + pm(2^p)] \\ pd(2^{p+1}) &= \frac{1}{2}\,[pd(2^p) + pm(2^p)] \\ t(2^{p+1}) &= \frac{1}{2}\,[t(2^p) + pm(2^p)]\,, \quad p = 1,2,\ldots \end{aligned}$$

(per la dimostrazione vedere il primo paragrafo di questo Capitolo).
Osservate che per $p = 5$ il numero di cifre decimali esatte è 1 per $ps(2^p)$ e $pd(2^p)$ e 3 per $pm(2^p)$ e $t(2^p)$, mentre per $p = 10$ questo numero è 2 per $ps(2^p)$ e $pd(2^p)$ e 6 per $pm(2^p)$ e $t(2^p)$.

Esercizio 11.4 **Svolgere l'analogo dell'Esercizio 11.1 per la formula di quadratura di Simpson.*

Esercizio 11.5 **Scrivere un programma di tipo funzione, dipendente da un numero reale positivo x, che calcoli il logaritmo di x nella base naturale e utilizzando la formula:*

$$\log(x) = \int_1^x \frac{1}{t}\, dt\,, \quad \forall x > 0\,,$$

e calcoli l'integrale con la formula di quadratura di Simpson. Per verificare di aver scritto un buon programma, confrontate i valori che ottenete per varie scelte di x, con quelli dati dalla funzione logaritmo di Matlab. In particolare osservate che la precisione peggiora sensibilmente per valori di x molto grandi o molto vicini a zero.

Esercizio 11.6 **Analogo dell'esercizio precedente con la funzione arcotangente di x (definita stavolta per ogni x reale) al posto della funzione logaritmo.*

Esercizio 11.7 **Analogo dell'esercizio precedente per la funzione:*

$$f(x) = \int_0^x e^{-t^2}\, dt\,, \quad x \in \mathbf{R}\,.$$

Esercizio 11.8 **Scrivere un programma di tipo funzione che, data una funzione f definita in $[a, b]$, disegni il grafico della funzione primitiva di f:*

$$F(x) = \int_a^x f(t)\, dt\,, \quad x \in [a, b]\,,$$

in $[a, b]$. Successivamente scrivere una versione più elaborata del programma che dipenda da un ulteriore parametro x_0 (compreso tra a e b) e disegni il grafico della primitiva:

$$F(x) = \int_{x_0}^x f(t)\, dt\,, \quad x \in [a, b]\,.$$

Utilizzate questi programmi su esempi noti per verificare che siano corretti, e successivamente su funzioni di cui non siamo in grado di scrivere esplicitamente le primitive.

Capitolo 12

Esercizi di riepilogo II

Esercizio 12.1 **Considerate il polinomio*

$$p(x) = x^3 + x + \lambda$$

dove λ è un parametro reale. Utilizzando risultati noti di analisi si dimostra facilmente (provate) che per ogni scelta di λ l'equazione

$$p(x) = 0$$

ha una e una sola soluzione reale $r(\lambda)$.
Scrivete un programma che, utilizzando il metodo di bisezione, determini $r(\lambda)$ una volta assegnato λ, con precisione di 10^{-3}.
Suggerimento: osservate che $p(-\lambda)\,p(0) < 0$ per ogni $\lambda \neq 0$.

Esercizio 12.2 **Scrivere un programma di tipo funzione che, dati due numeri reali a e b trovi le soluzioni di*

$$x^2 + ax + b = 0$$

utilizzando il metodo di Newton e con precisione scelta da voi. Non *dovete usare la formula risolutiva (se non come verifica).*
Suggerimento: Il metodo di Newton funziona in intervalli in cui la funzione ha derivata diversa da zero e non ha cambi di concavità; la derivata di $f(x) = x^2 + ax + b$ si annulla solo in un punto, che è in mezzo alle due radici cercate, inoltre f è convessa ovunque. Il metodo deve essere applicato due volte, una per ogni radice, scegliendo il punto di partenza in modo da evitare in entrambi i casi il punto a derivata nulla.

Esercizio 12.3 **La funzione*

$$f(x) = x + x^3 + x^5$$

è strettamente monotona in $\mathbf{R}$ *e dunque in particolare è invertibile in* $[0,1]$*. L'immagine di* $[0,1]$ *tramite* f *è l'intervallo* $[0,3]$*. Scrivete un programma che dato* $y \in [0,3]$ *determini* $x = f^{-1}(y)$*. Potete utilizzare l'algoritmo della bisezione o quello di Newton.*

Esercizio 12.4 **Scrivere un programma di tipo funzione che, dato un numero naturale* n *trovi il più grande numero primo minore o uguale a* n*.*

Esercizio 12.5 **Un teorema afferma che se* n *è un numero naturale, esiste un numero primo compreso tra* n *e* $2n$*. Utilizzando questo risultato, scrivete un programma di tipo funzione che, dato* n*, trovi il più piccolo numero primo maggiore o uguale a* n*.*

Esercizio 12.6 **Scrivete un programma di tipo funzione che, dato* n *naturale, vi dica se esistono due numeri primi (e in caso affermativo li scriva)* p_1 *e* p_2 *tali che*

$$n = p_1 + p_2 \,.$$

Questo esercizio è legato alla famosa congettura di Goldbach che afferma che la scomposizione in somma di due primi è vera per ogni numero n *pari.*

Esercizio 12.7 **Scrivere un programma di tipo funzione che, dato* n *naturale, trovi, se esistono, tre numeri* p_1*,* p_2 *e* p_3 *primi o uguali a uno, tali che*

$$n = p_1 + p_2 + p_3 \,.$$

Esercizio 12.8 **La lunghezza di un'ellisse centrata nell'origine di un sistema di riferimento cartesiano, con assi paralleli agli assi coordinati e lunghezze dei semiassi* a*,* $b > 0$ *è data dalla formula:*

$$\int_0^{2\pi} \sqrt{a^2 \sin^2 x + b^2 \cos^2 x}\, dx \,.$$

Utilizzando questa formula scrivete un programma di tipo funzione che dati due numeri reali positivi a *e* b*, calcoli la lunghezza dell'ellisse che ha semiassi lunghi* a *e* b*, tramite una qualsiasi delle formule di quadratura viste, su una partizione equispaziata dell'intervallo di integrazione in* 10^6 *sottointervalli.*

Esercizio 12.9 **Si dimostra la seguente uguaglianza*

$$\int_{-\infty}^{+\infty} e^{-x^2}\, dx = \sqrt{\pi} \,.$$

Per verificarla numericamente scrivete un programma di tipo funzione che, dati n *e* N *naturali, calcoli*

$$\int_{-N}^{N} e^{-x^2}\, dx$$

con la formula di Simpson, su una partizione equispaziata di $[-N, N]$ *in* n *sottointervalli.*

Capitolo 13

Soluzioni di alcuni esercizi

13.1 Capitolo 1

1.7

```
floor(10*rand)
```

1.8

```
round(rand)
```

1.11

```
v=1:100;
w=3*v
```

1.12

```
h=2*pi*1e-4;
N=0:10000;
V=h*N
sin(V)
```

13.2 Capitolo 2

1.17

```
V=1:10000;
W=sin(V);
C=W>=.5;
sum(C)
```

2.2

```
function V=ese2_4(n)
A=1:n;
V=7*A;
```

2.5

```
function ese2_8(A)
if size(A,1)==size(A,2)
      'matrice quadrata'
   else
      'matrice non quadrata'
end
```

2.6

```
function ese2_9(a,b,c)
if a*c==b*b
      'b medio proporzionale'
   elseif a*b==c*c
      'c medio proporzionale'
   elseif b*c==a*a
      'a medio proporzionale'
   else
      'nessun dei tre numeri e medio proporzionale'
end
```

2.7

```
function f=ese2_10(x)
if x<=3
      f=x*x;
   else
      f=9;
end
```

2.8

```
function ese2_11(A)
if A==A.'
      'matrice simmetrica'
   elseif A==-A.'
      'matrice antisimmetrica'
   else
      'matrice non simmetrica e non antisimmetrica'
      B=0.5*(A+A.');
      C=0.5*(A-A.');
      A
      '='
      B
      '+'
      C
end
```

2.14

```
a=.001;
b=1;
h=(b-a)/4999;
% h = passo della partizione
for i=0:4999
   x(i)=a+h*i;
   y(i)=sin(1/x(i));
end
C=[x;y];
C=C.'
```

2.12

```
function r=ese2_15(k)
F(1)=1;
F(2)=1;
for i=1:k-2
   F(i+2)=F(i+1)+F(i);
end
r=F(k)/F(k-1);
```

2.13

```
function V=ese2_16(n)
if rem(n,2)==0
      V=2:2:n;
   else
      V=1:2:n;
end
```

2.14

```
function ese2_17(a,b,c,d)
A=[a,b,c,d];
for i=1:4
   for j=1:4
      for k=1:4
         disp(A(i)*100+A(j)*10+A(k))
      end
   end
end
```

2.15

```
function ese2.18(A)
n=size(A,2);
for i=1:n
   for j=1:n
      for k=1:n
         100*A(i)+10*A(j)+A(k)
      end
   end
end
```

2.17

```
S=0;
for i=5:100
   S=S+1/sqrt(i+i*i*i);
end
S
```

2.18

```
S=0;
for i=1:999
   if rem(i,7)=~0
      S=S+i;
   end
end
```

2.21

```
function r = resto(a,b)
% calcola il resto della divisione intera di a per b
r = a - quoziente(a,b) * b;
```

2.22

```
function s = sommavettore(V)
% calcola la somma delle componenti del vettore V
s = 0;
while  ~isempty(V)
   s = s + V(1);
   V(1) = [];
end
```

Usando for possiamo alternativamente scrivere:

```
function s = sommavettore(V)
s = 0;
for x = V
   s = s + x;
end
```

2.23

```
a=0;
while a==0
a=rand*6;
end
ceil(a)
```

13.3 Capitolo 3

3.1

```
function d = mcd4(L)
% mcd4 calcola il mcd di quattro numeri interi (dati come vettore L)
d = mcd(L(1), L(2));
d = mcd(d , L(3));
d = mcd(d , L(4));
```

```
function d = mcdn(L)
%calcola il MCD degli interi della lista L
d = L(1);
for i = L
   d = mcd(d,i);
end
```

3.2

```
function  m = mcm(a,b)
m = abs( ( a * b) / mcd(a, b));
```

3.3

```
function L = diofantea(a, b, c)
L = [];
d = mcd(a,b);
if resto(c, d) ~= 0
      'nessuna soluzione'
   else
      K = coeff_mcd(a,b);
      L(1) = K(1) * quoziente(c,d);
      L(2) = K(2) * quoziente(c,d);
end
```

3.4

```
function L = cifre2(n)
% CIFRE2(n) produce  la lista delle cifre della espressione in base 2
%del numero naturale n
L = [];
if n == 0
  L = [0];
  return
end
while  n > 0
   cifra = resto(n,2);
   L = [cifra, L];
   n = (n -cifra)/2;
end
```

3.5

```
function L = cifre_base(n, b)
% CIFRE_BASE(n,b) produce  la lista delle cifre della espressione in
%base b di n
L = [];
if n == 0
   L = [0];
   return
end
while  n > 0
   cifra = resto(n,b);
   L = [cifra, L];
   n = (n -cifra)/b;
end
```

3.6

```
function s = scifre(n)
% somma le  cifre della espressione di n in base 10
s = sommavettore(cifre(n));
```

3.7

```
function c = sommacifre(n)
%SOMMACIFRE(N) somma le cifre della espressione decimale di n
% fino ad ottenere una cifra tra 0 e 9
c = n;
while c > 9
  c = scifre(c);
end
```

3.8

```
 function L = contacifre(n)
L =  zeros(1,9);
for i = 1:n
  posizione = sommacifre(i);
  L(posizione) = L(posizione) + 1;
end
```

3.12

```
function c = inverso(a,n)
%determina se a e' invertibile modulo n e ne calcola l'inverso
if mcd(a,n) ~= 1
   'non  invertibile '
else
    V = coeff_mcd(a,n);
    c1 = V(1);
    c = resto(c1,n);
end
```

(Osservate che alla riga 8 l'intero c1 viene ridotto modulo n, in modo da avere il risultato c compreso tra 0 e n-1, per coerenza con le notazioni adottate.)

3.13

```
function  L = invertibili(n)
%determina la lista degli elementi invertibili in Z/nZ
L = [];
for i = 1:n
  if mcd(i, n) == 1
  L = [L, i];
  end
end
```

13.4 Capitolo 4

4.1

```
function pn=pn(v)
k=size(v,2);
u=0;
for i=1:k
  if v(i)==0
     u=u+1;
  else
     break
  end
end
pn=u;
```

4.2

```
% function per vedere se una matrice e a scalini
function scalini=scalini(A)
[m,n]=size(A);
r=0;
for i=1:m-1
   if  pn(A(i+1,:))>=pn(A(i,:))+1 | pn(A(i+1,:))==n
      r=r;
   else
      r=r+1;
   end
end
if r==0
  'matrice  a scalini'
else
  'matrice non a scalini'
```

4.11

```
% function per estrarre una base da un insieme di generatori di un
% sottospazio di Rn
function estrbase0=estrbase0(A)
[m,n]=size(A);
B=zeros(m,0);
for i=1:n
  if rank([B A(:,i)])==size(B,2)+1
    B=[B A(:,i)];
  else
    B=B;
  end
end
estrbase0=B;
end
```

13.5 Capitolo 5

5.9

```
function sa=sa(A,k)
U=null(A);
[m,n]=size(A);
v=size(U,2);
for j=1:k
  for l=1:v
     Z=zeros(n,k);
     Z(:,j)=U(:,l)
  end
end
```

13.6 Capitolo 6

6.1

```
R=[];
for k=-100:1:100
  R=[R 7*k+2];
end
R
```

6.2

```
s=0;
for x=1:10000
   if (round(sqrt(x+1))==sqrt(x+1) & rem(x,4)==0
        s=s+1;
   end
end
s
```

6.3

```
function P=ese183(a,b,h)
C=zeros(2,0);
i=0;
while a+i*h<=b
x=a+i*h;
y=2*sin(8*x)-log(x^2+1);
  V=[x;y];
  C=[C V];
  i=i+1;
end
P=C.';
```

6.4

```
function P=ese184(n)
S=0;
for i=1:n
    S=S+((-1)^i)/((sin(i))^2+1);
end
P=S;
```

6.5

```
function V=ese185(n)
V=zeros(1,n);
V(1)=1;
for i=1:(n-1)
  V(i+1)=sin(V(i));
end
```

6.6

```
function S=ese186(n)
T=0;
a=sin(1);
for i=1:n
  T=T+a;
  a=sin(a);
end
S=T;
```

6.7

```
function M=ese188(a,b,c)
M=a;
if b>=M
 M=b;
end
if c>=M
 M=c;
end
```

6.8

```
function M=ese189(V)
n=size(V,2);
M=V(1);
for i=2:n
  if V(i)>=M
    M=V(i);
  end
end
```

6.9

```
function A = torneo(n)

%A e' la matrice degli accoppiamenti di un torneo
%all'italiana con n giocatori (n pari). L'elemento
%A(i,j) e' il giocatore che i incontra al turno j.

A = zeros(n, n-1);

for j = 1:n-1
  for i = 1:n-1
    s = mod(j - i, n-1);
    if s == 0
       t = n-1;
    else
       t = s;
    end

    if ~(t == i)
       A(i,j) = t;
    else
       A(i,j) = n;
       A(n,j) = i;
    end
   end
end
```

6.10

```
function turno(k, n)
A = torneo(n);
L = 1:n;
while ~isempty(L)
   disp([L(1)  A(L(1), k)])
   L = setdiff(L, [L(1) A(L(1),k)]);
end
```

13.7 Capitolo 7

7.2

```
function c=bisezione3(f,a,b,e)
fa=feval(f,a);
fb=feval(f,b);
if fa*fb>0
```

```
  disp('dati non accettabili')
  return
end
c=(b+a)/2;
n=1;
while b-a>=e
  c=(b+a)/2;
  fa=feval(f,a);
  fc=feval(f,c);
  if fa*fc<=0
      b=c;
  else
      a=c;
  end
  n=n+1;
end
disp ('numero di iterazioni') , disp(n)
```

7.3

```
>> format long
>> f=inline(x*x*x*x-9);
>> d=inline(4*x*x*x);
>> bisezione3(f,1,9,.0001)
>> newton(f,d,1,9,9,20,.0001)
>> 9 ${}^\wedge$(.25)
```

Osservate che, dando per buono che tutte le cifre decimali del valore calcolato da Matlab siano esatte, il metodo di bisezione con 18 iterazioni ha solo quattro cifre decimali esatte, mentre il risultato trovato con il metodo di Newton, con dieci iterazioni, contiene 13 cifre decimali esatte.

7.4

```
function r=cubica(a)
s=(1+a)/2;
fs=s*s*s-a;
ds=3*s*s;
r=s-(fs)/(ds);
e=.000001;
N=1;
while abs(r-s)>=e & N<=20
  s=r;
  fs=s*s*s-a;
  ds=3*s*s;
  r=s-(fs)/(ds);
  N=N+1;
end
```

7.5

```
function x=arcoseno(y)
if abs(y)>1
  disp('valore non accettabile')
  return
end
e=1e-14;
a=-pi/2;
b=pi/2;
while b-a>=e;
  x=(a+b)/2;
  if (sin(x)-y)*(sin(a)-y)<=0
     b=x;
  else
     a=x;
  end
end
```

13.8 Capitolo 8

8.13

```
function grafico(f,a,b,n)
h=(b-a)/n; % passo della partizione
X=[];
Y=[];
for i=0:n;
    x=a+i*h;
    X=[X x];
    y=feval(f,x);
    Y=[Y y];
end
plot(X,Y)
```

8.16

```
function curva(x,y,a,b,n)
h=(b-a)/n; % passo della partizione
t=a;
X=[];
Y=[];
for i=0:n;
    t=a+i*h;
    xt=feval(x,t);
    X=[X xt];
    yt=feval(y,t);
    Y=[Y yt];
end
plot(X,Y)
```

8.19

```
function ese8_19(n)
a=1;
S=[];
s=0;
for i=1:(n-1)
   a=sin(a)
   s=s+a;
   S=[S s];
end
bar(S)
```

8.21

```
function ese8_21(f,a,n)
A=[a];
for i=1:(n-1)
   a=feval(f,a);
   A=[A a];
end
bar(A)
```

8.22

```
function ese8_22(f,n)
S=[];
a=0;
s=0;
for i=1:n
   a=feval(f,i);
   s=s+a;
   S=[S s];
end
bar(S)
```

8.26

```
function ese8_26(f,n)
S=[];
s=0;
for i=1:n
   c=2*round(rand)-1;
   a=c*feval(f,i);
   s=s+a;
   S=[S s];
end
bar(S)
```

13.9 Capitolo 9

9.1

```
%Function per calcolare la posizione reciproca di una retta e un piano
% dati in forma cartesiana Ax=b , Cx=d nello spazio affine
function rettaepiano=rettaepiano(A,b,C,d)
if rank(A)<=1 | rank(C)==0
 'la prima non e un retta o la seconda non un piano'
else
  if rank([A; C]) == 3
  'la retta e il piano  sono incidenti in un punto'
  elseif rank([A b; C d]) == 2
  'la retta e contenuta nel piano'
  else
  'la retta e il piano sono paralleli e la retta non e contenuta nel piano'
  end
end
```

9.2

```
%Function per calcolare la posizione reciproca due rette
% date in forma cartesiana Ax=b , Cx=d nello spazio affine
function duerette=duerette(A,b,C,d)
if rank(A)<=1 | rank(C)<=1
 'almeno una delle due non e un retta'
else
  if rank([A; C])==3 & rank([A b; C d])==3
  'le due rette   sono incidenti in un punto'
  elseif rank([A; C])==2 & rank([A b; C d])==2
  'le due rette sono uguali'
  elseif rank([A; C])==2 & rank([A b; C d])==3
  'le due rette sono parallele distinte'
  else
  'le due rette sono sghembe'
  end
end
```

13.10 Capitolo 10

10.3
Proponiamo due soluzioni. La prima è più immediata e sfrutta il programma `eratostene` che viene eseguito due volte, una per N e una per M. La seconda invece è una variazione di `eratostene` ed ha il vantaggio, rispetto alla precedente, che il crivello di Eratostene viene impiegato una sola volta, per M.

```
function V=eratosdiff(N,M)
X=eratostene(M);
Y=eratostene(N-1);
V=setdiff(X,Y);
```

```
function X=eratosdiff2(N,M)
X=ones(1,M);
I=floor(sqrt(M));
X(1)=0;
for j=2:I
    r=2;
    while r*j<=M
        s=r*j;
        X(s)=0;
        r=r+1;
    end
end
for k=N:M
    X(k)=k*X(k);
end
X=setdiff(X,0);
X=setdiff(X,1);
```

10.4

```
function gauss100(n)
m=n+100;
v=eratosdiff2(n,m);
p=size(v,2);
r=p/n;
disp(p) , disp(r)
```

10.5

```
function primalita2(n)
v=eratostene(n);
if ismember(n,v)==1
    disp('numero primo')
else
    disp('numero composto')
end
```

10.6

```
function X=gemelli(n)
X=ones(0,2);
v=eratostene(n);
N=size(v,2);
for i=1:N-1
  if v(i)+2==v(i+1)
     X=[X;[v(i) v(i+1)]];
  end
end
```

10.7

```
function V=camgauss(N)
V=zeros(1,N);
for n=1:N
  V(n)=gauss(n);
end
```

10.10

```
% conta il numero di tutti i divisori
% di n, diversi da 1 e n
function c=numdidiv(n)
c=0;
for d=2:(n-1)
    if mod(n,d)==0
        c=c+1;
    end
end
```

10.11

```
% campiona la funzione numdidiv(i)
% per i da 1 a n
function X=camnd(N)
X=[];
for i=1:N
\quad    X=[X numdidiv(i)];
```

10.12

```
% conta il numero di divisori primi di n
function c=numdidivp(n)
c=0;
v=eratostene(n);
for p=v
    if mod(n,p)==0
        c=c+1;
        n=n/c;
    end
end
```

10.14

```
function c=fieulero(n)
c=0;
for k=1:(n-1)
    if mcd(n,k)==1
        c=c+1;
    end
end
```

10.15

```
function libquad(n)
M=fattorizzazione(n);
M=M.';
V=M(2,:); % vettore contenente le potenze dei div. primi di n
k=size(V,2);
for i=1:k
   if V(i)>=2
      'non libero da quadrati'
      return
   end
end
'libero da quadrati'
```

10.16

```
function z=codifica(w)
if w>26;
    'input non corretto'
    return
end
p=w^3;
z=rem(p,33);

function z=decodifica(w)
p=w^7;
z=rem(p,33);
```

13.11 Capitolo 11

11.1

```
%formula di quadratura del punto sinistro
function q=psx(f,a,b,n)
q=0;
```

```
h=(b-a)/n;
for k=1:n
    fk=feval(f,a+(k-1)*h);
    q=q+fk;
end
q=h*q;
```

11.2

```
% formula di quadratura del punto destro
function q=pdx(f,a,b,n)
q=0;
h=(b-a)/n;
for k=1:n
    fk=feval(f,a+k*h);
    q=q+fk;
end
q=h*q;

% formula di quadratura del punto medio
function q=pm(f,a,b,n)
q=0;
h=(b-a)/n;
for k=1:n
    fk=feval(f,a+(k-.5)*h);
    q=q+fk;
end
q=h*q;

% formula di quadratura dei trapezi
function q=trapezi(f,a,b,n)
q=0;
N=n-1;
h=(b-a)/n;
for k=1:N
    fk=feval(f,a+k*h);
    q=q+fk;
end
fa=feval(f,a);
fb=feval(f,b);
q=h*(.5*(fa+fb)+q);
```

11.3

```
format long
f=inline('4/(1+x^2)');
V=zeros(1,4);
```

```
V(1)=psx(f,0,1,1);
V(2)=pdx(f,0,1,1);
V(3)=pm(f,0,1,1);
V(4)=trapezi(f,0,1,1);
disp(V);
for p=1:10
    n=2^p;
    V(1)=.5*(V(1)+V(3));
    V(2)=.5*(V(2)+V(3));
    V(4)=.5*(V(4)+V(3));
    V(3)=pm(f,0,1,n);
    disp(V)
end
```

11.4

```
% formula di quadratura di Simpson
function q=simpson(f,a,b,n)
h=(b-a)/n;
q=0;
for k=1:n
    f1=feval(f,a+h*(k-1));
    f2=feval(f,a+h*k);
    f3=feval(f,a+h*(k-.5));
    q=q+f1+f2+4*f3;
end
q=h*q/6;
```

11.5

```
% Questo programma calcola il logaritmo di un numero positivo x
% come integrale di 1/t tra 1 e x. Per il calcolo dell'integrale
% utilizza la formula di Simpson. Il programma da' buoni
% risultati se x non e' ne' troppo grande ne' troppo vicino a zero.
function l=logaritmo(t)
format long
n=100;
f=inline('1/x');
if t<=0
    disp('argomento non accettabile')
    return
end
if t==1
    l=0;
    return
end
if t>1
```

```
    l=simpson(f,1,t,n);
    return
end
if t<1
    l=-simpson(f,t,1,n);
end
```

11.6

```
% Questo programma calcola la funzione arcotangente
% come primitiva di 1/(1+x^2)
function a=arcotangente(t)
f=inline('1/(1+x*x)');
n=30;
P=4*simpson(f,0,1,n);
T=abs(t);
if T==0
    a=0;
    return
end
if T<=1
    A=simpson(f,0,T,n);
else
    A=.5*P-simpson(f,0,1/T,n);
end
if t>0
    a=A;
else
    a=-A;
end
```

11.7

```
function y=primgaussiana(x)
f=inline('exp(-x*x)');
n=30;
X=abs(x);
Y=simpson(f,0,X,n);
if x>0
    y=Y;
else
    y=-Y;
end
```

11.8

```
function graficoprimitiva(f,a,b,n)
x=zeros(1,n+1);
y=zeros(1,n+1);
h=(b-a)/n;
H=h/6;
x(1)=a;
y(1)=0;
for k=1:n
    x(k+1)=a+h*k;
    f1=feval(f,a+h*(k-1));
    f2=feval(f,a+h*k);
    f3=feval(f,a+h*(k-.5));
    y(k+1)=y(k)+H*(f1+f2+4*f3);
end
plot(x,y)

function graficoprimitiva2(f,a,b,X,n)
h=(b-a)/n;
H=h/6;
C=simpson(f,a,X,n);
x=zeros(1,n+1);
y=zeros(1,n+1);
x(1)=a;
y(1)=0;
for k=1:n
    x(k+1)=a+h*k;
    f1=feval(f,a+h*(k-1));
    f2=feval(f,a+h*k);
    f3=feval(f,a+h*(k-.5));
    y(k+1)=y(k)+H*(f1+f2+4*f3);
end
for k=1:n+1
    y(k)=y(k)-C;
end
plot(x,y)
```

13.12 Capitolo 12

12.1

```
% risolve numericamente, con il metodo di
% bisezione, l'equazione x^3+x+l=0 in funzione di l
% (questa equazione ha una e una sola soluzione reale
% per ogni valore di l). La precisione e' 10^(-5)
function c=ese1(l)
a=min(-l,0);
b=max(-l,0);
```

```
c=(b+a)/2;
while b-a>.00001
    c=(b+a)/2;
    fc=c^3+c+l;
    if fc>=0
        b=c;
    else
        a=c;
    end
end
```

12.2

```
% questo programma determina le (eventuali) soluzioni
% reali dell'equazione x^2+a*x+b=0
% utilizzando il metodo di Newton, con precisione
% 10^(-6)
function ese2(a,b)
%controllo del discriminante
if a^2-4*b<0
    disp('non esistono radici reali')
    return
end
v=-.5*a;
% radice maggiore
r=v+1;
fr=r^2+a*r+b;
dr=2*r+a;
s=r-fr/dr;
n=0;
while abs(r-s)>1e-6;
    r=s;
    fr=r^2+a*r+b;
    dr=2*r+a;
    s=r-fr/dr;
    n=n+1;
end
disp(s)
% radice minore
r=v-1;
fr=r^2+a*r+b;
dr=2*r+a;
s=r-fr/dr;
while abs(r-s)>1e-6
    r=s;
    fr=r^2+a*r+b;
    dr=2*r+a;
    s=r-fr/dr;
```

```
end
disp(s)
```

12.3

```
function x=ese3(y)
if y<0 | y>3
    disp('dato non accettabile')
    return
end
a=0;
b=1;
x=.5;
while b-a>1e-6
    fx=x^5+x^3+x-y;
    if fx>=0
        b=x;
    else
        a=x;
    end
        x=(b+a)/2;
end
```

12.4

```
function p=ese4(n)
v=eratostene(n);
m=size(v,2);
p=v(m);
```

12.5

```
function p=ese5(n)
N=2*n;
v=eratostene(N);
i=1;
while v(i)<n
    i=i+1;
end
p=v(i);
```

12.6

```
function ese6(n)
v=eratostene(n);
m=size(v,2);
for i=1:m
    m=n-v(i);
    if ismember(m,v)==1
        disp(v(i)) , disp(m)
        return
    end
end
disp('il numero dato non si scrivve come somma di due primi')
```

12.7

```
function ese7(n)
v=[1 eratostene(n)];
m=size(v,2);
for i=1:m
    for j=i:m
        if v(i)+v(j)<=n
            N=n-v(i)-v(j);
            if ismember(N,v)==1
                disp(v(i))
                disp(v(j))
                disp(N)
                return
            end
        end
    end
end
disp('il numero dato non si scrive come somma di tre primi')
```

12.8

```
function I=ese8(a,b)
n=1e6;
h=(2*pi)/n;
q=0;
for i=1:n-1
    x=i*h;
    fx=sqrt(a^2*(sin(x))^2+b^2*(cos(x))^2);
    q=q+fx;
end
I=h*(b+q);
```

12.9

```
function ese9(n,N)
f=inline('exp(-x^2)');
simpson(f,-n,n,N)
```

Bibliografia

[1] S. Campi, M. Picardello e G. Talenti, *Analisi matematica e calcolatori*, Bollati Boringhieri, Torino, 1990.

[2] L. Childs, *Algebra un'introduzione concreta*, ETS editrice, Pisa, 1989.

[3] G. Naldi, L. Pareschi e G. Russo, *Introduzione al calcolo scientifico - metodi e applicazioni con Matlab*, McGraw-Hill, Milano, 2001.

[4] W. J. Palm III, *MATLAB6 per l'ingegneria e le scienze*, McGraw-Hill, Milano, 2001.