Analisi Matematica I
Esercitazioni con MATLAB

Dario Bernardi Stefania Malaguti Yuri Taddia Chiara Visentin
Andrea Corli

Indice

Introduzione **vii**

1 Matrici, per cominciare **1**
1.1 Matrici . . . 1
1.2 Chiedere aiuto . . . 1
1.3 Si inizia . . . 2
1.4 Esercizi . . . 4
1.5 Approfondimenti . . . 5

2 Gli script e la grafica in Matlab **7**
2.1 Script . . . 7
2.2 Il comando `plot` . . . 8
2.3 Esercizi . . . 9
2.4 Approfondimenti . . . 9

3 Successioni **11**
3.1 La successione geometrica . . . 11
3.2 Successioni definite per ricorrenza e il ciclo `for` . . . 11
3.3 Complementi . . . 12
3.3.1 Altri esempi . . . 12
3.3.2 La successione di Fibonacci . . . 13
3.4 Esercizi . . . 14
3.5 Approfondimenti . . . 15

4 Serie **17**
4.1 La serie geometrica e l'istruzione condizionale `if` . . . 17
4.2 La serie armonica e il ciclo `while` . . . 18
4.3 Serie a termini di segno variabile . . . 19
4.4 Complementi . . . 19
4.4.1 Stima della divergenza della serie armonica . . . 19
4.4.2 La serie armonica generalizzata . . . 19
4.5 Esercizi . . . 20
4.6 Approfondimenti . . . 21

5 Grafici di funzioni **23**
5.1 La funzione $\sin x/x$ . . . 23
5.2 Operazioni con i grafici . . . 24
5.2.1 Simmetrie . . . 24
5.2.2 Il valore assoluto . . . 25
5.3 Funzioni definite a tratti . . . 25
5.4 Più grafici . . . 26
5.4.1 Con il ciclo `for` . . . 26
5.4.2 Il comando `meshgrid` . . . 26
5.4.3 Minimi e massimi . . . 27
5.5 Complementi . . . 28
5.5.1 Traslazioni e riscalamenti . . . 28
5.5.2 Un'onda (quasi) quadra . . . 29

5.6 Esercizi . . . 29
5.7 Approfondimenti . . . 30

6 Grafici di funzioni: applicazioni **33**
6.1 Grafici in scala logaritmica . . . 33
6.2 Funzioni potenze ed esponenziali . . . 34
6.3 Complementi . . . 34
6.3.1 Applicazione: raffreddamento di un corpo . . . 34
6.3.2 Applicazione: la legge del potere fonoisolante . . . 35
6.3.3 La formula di Stirling . . . 36
6.4 Esercizi . . . 36

7 Polinomi **37**
7.1 I comandi `polyval`, `roots` e `poly` . . . 37
7.2 Il comando `polyfit` . . . 37
7.3 I comandi `conv` e `deconv` . . . 38
7.4 Complementi . . . 38
7.4.1 Applicazione: la legge di Hooke . . . 38
7.5 Esercizi . . . 39
7.6 Approfondimenti . . . 39

8 `Function(s)` **41**
8.1 La mia prima `function` . . . 41
8.2 Zeri di una funzione: il comando `fzero` . . . 42
8.3 Minimi e massimi di una funzione: il comando `fminbnd` . . . 43
8.4 Complementi . . . 44
8.4.1 Una `function` con due output e il ciclo `while` . . . 44
8.4.2 Applicazione: la legge dei gas perfetti . . . 45
8.5 Esercizi . . . 45
8.6 Approfondimenti . . . 46

9 Il calcolo simbolico **49**
9.1 Le variabili simboliche . . . 49
9.2 Manipolare espressioni . . . 49
9.3 Grafici e limiti . . . 50
9.4 Complementi . . . 51
9.4.1 Convergenza delle serie numeriche . . . 51
9.5 Esercizi . . . 52
9.6 Approfondimenti . . . 52

10 Derivate, numeriche e simboliche **53**
10.1 La derivata numerica . . . 53
10.1.1 Difficoltà numeriche (e un'idea dell'Analisi Numerica) . . . 53
10.1.2 Implementazione . . . 54
10.2 La derivata simbolica . . . 54
10.3 Complementi . . . 55
10.3.1 La derivata seconda . . . 55
10.4 Esercizi . . . 56
10.5 Approfondimenti . . . 57

11 Integrazione numerica e simbolica **59**
11.1 Integrazione numerica con il metodo dei trapezi (un metodo di Analisi Numerica...) 59
11.2 Integrazione simbolica . . . 60
11.3 Complementi . . . 61
11.3.1 Ripasso: integrazione numerica tramite punti casuali . . . 61
11.3.2 La funzione integrale . . . 62
11.3.3 Integrali ellittici e altri loro amici . . . 62
11.4 Esercizi . . . 63
11.5 Approfondimenti . . . 63

Bibliografia **65**

Introduzione

Sento, dimentico;
vedo, ricordo;
faccio, capisco.

(Proverbio cinese)

Negli ultimi anni accademici i corsi di Analisi Matematica I, Geometria e Analisi Matematica II, relativi al Corso di Laurea in Ingegneria Civile dell'Università di Ferrara, hanno avviato una sperimentazione che prevede l'insegnamento dei principi fondamentali del software MATLAB[1] all'interno di tali corsi.

Lo scopo di questa sperimentazione è molteplice. Da un punto di vista didattico, la programmazione di un software numerico obbliga ad un ripensamento di quanto appreso tramite l'insegnamento tradizionale, dando poi la possibilità di visualizzare e analizzare concretamente gli aspetti teorici della materia. Da un punto di vista applicativo, lo studente impara così fin dai primi anni di corso un software che è di larghissimo uso nelle Scienze Ingegneristiche.

Queste brevi note sono relative al corso di Analisi Matematica I. Esse non hanno alcuna pretesa di completezza, né intendono essere una presentazione sistematica ed organica a MATLAB; per queste lo studente può riferirsi al testo di Palm [3]. L'approccio che si segue è invece molto più pragmatico e ispirato al libro di Jensen [2]. Questi appunti devono essere letti di fronte ad un computer, programmando direttamente il software (si legga la citazione qui sopra...). Solo così si può avere soddisfazione dello studio e essere stimolati ad andare oltre; a questo scopo, alcuni semplici esercizi sono proposti alla fine di ogni capitolo. In definitiva, è l'ordine usuali degli argomenti del corso di Analisi Matematica I a dettare l'ordine di introduzione dei comandi di MATLAB.

A causa della diversità delle varie versioni di MATLAB, non si sono date indicazioni precise sulla gestione del programma; del resto queste note sono un semplice ausilio alle lezioni che vengono tenute in aule attrezzate, dove tale gestione viene specificata. Le descrizioni di molti comandi non sono esaustive; in generale vengono date solo quelle informazioni utili per gli scopi di queste note. Lo studente energico potrà approfondire caso per caso la sua conoscenza in merito nella documentazione di MATLAB.

MATLAB è un software a pagamento ma liberamante scaricabile dal sito di UNIFE per gli studenti regolarmente iscritti. Esistono tuttavia software analoghi, liberamente scaricabili e (più o meno) facilmente installabili sul proprio computer. I due più popolari sono:

- Octave, `http://www.gnu.org/software/octave/`
- SciLab, `http://www.scilab.org/`

Tra i due, quello maggiormente compatibile con MATLAB è Octave.

Per ragioni di spazio, nel seguito non abbiamo riportato le figure ottenute dalle simulazioni, ma soltanto le istruzioni (complete) per ottenerle. Una (modesta) soddisfazione dello studente sarà quella di scoprire cosa viene fuori da una lunga serie di comandi. Per gli stessi motivi, abbiamo insistito sulla parte di abbellimento grafico (colori, caratteri e così via) solo nella prima parte; nelle parti seguenti, soprattutto per focalizzare l'attenzione sui comandi principali, abbiamo utilizzato al minimo questi comandi di grafica.

Questi appunti sono strutturati come segue. Nelle prime sezioni di ogni capitolo viene presentato dapprima il materiale assolutamente indispensabile. Segue quindi (ma non sempre) una sezione di Complementi; essa può contenere ulteriori semplici esempi, oppure può sviluppare del materiale

[1] MATLAB è un marchio registrato della MathWorks Inc, si veda `http://www.mathworks.com/`

applicativo o, infine, può introdurre alcuni comandi un po' più sofisticati. Segue una sezione di Esercizi (farli) e infine una sezione conclusiva di Approfondimenti, in cui si introduce molto brevemente altro materiale per una migliore comprensione. I Complementi e gli Approfondimenti possono essere tralasciati in prima lettura; le sezioni veramente importanti sono quelle che precedono i Complementi.

Una prima avventurosa stesura di queste note (in Word) è avvenuta durante l'a.a. 2009-10 da parte di Dario Bernardi e Stefania Malaguti. Negli a.a. 2010-11 e 2011-12 Chiara Visentin le ha interamente riviste, aggiunto nuovo materiale e esportate in LaTeX. In seguito, negli a.a. 2011-12 e 2012-13, Dario Bernardi ha di nuovo contribuito a migliorarle, aggiungendo in particolare i dettagli per l'installazione di Octave e le modifiche necessarie per rendere il testo compatibile con Octave. Varie osservazioni sono state poi fatte da Giulia Farina. Una sostanziale revisione è stata poi fatta da Yuri Taddia, che ha anche provveduto ad aggiungere altro materiale. La presente versione è lungi dall'essere definiva; una prova sono l'esiguo numero di esercizi e approfondimenti proposti. Un ringraziamento sentito va a Gaetano Zanghirati, sempre disponibile a chiarire i nostri dubbi.

Buona lettura!

Ferrara, 20 settembre 2019

Dario Bernardi, Stefania Malaguti, Yuri Taddia, Chiara Visentin, Andrea Corli

Capitolo 1

Matrici, per cominciare

Gli inizi sono sempre difficili.

(In principio, Ch. Potok)

In questo capitolo diamo qualche semplice definizione sulle matrici e introduciamo i primi semplici comandi di MATLAB.

1.1 Matrici

Il nome MATLAB è un acronimo per *Matrix Laboratory*, in quanto le matrici formano la struttura di base del programma. Questo verrà chiarito nel seguito e per il momento non deve preoccupare. Una matrice è una lista rettangolare di numeri, ad esempio

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}. \tag{1.1}$$

Si può indicare una matrice usando le parentesi tonde, ma poiché MATLAB usa le quadre ci atterremo a questa notazione. La matrice A in (1.1) ha 2 righe e 3 colonne: si dice che è una matrice 2×3. Un vettore (riga), ad esempio

$$v = \begin{bmatrix} 7 & 8 & 9 & 10 \end{bmatrix},$$

è una particolare matrice; qui sopra v è una matrice 1×4. Nelle matrici si enumerano sempre prima le righe poi le colonne; con la notazione $A(i, j)$ si indica l'elemento della matrice A relativo alla riga i e alla colonna j. Nell'esempio (1.1) si ha

$$A(2, 1) = 4.$$

La trasposta A' di una matrice A è la matrice che si ottiene scambiando le righe con le colonne di A. Dunque

$$A' = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}.$$

Chiaramente A' è una matrice 3×2.

1.2 Chiedere aiuto

Una volta lanciato MATLAB ci si trova solitamente di fronte tre finestre. Quella che ci interessa ora è la *Command Window*, abbreviata nel seguito con *cw*. La riga contrassegnata da

```
>>
```

in cui è posizionata la sbarretta di inserimento è detta riga di comando. Dopo aver scritto un comando... pigiare il tasto di invio.

MATLAB dispone di una ricca documentazione che può essere utile richiamare al momento opportuno. Il comando

```
helpdesk
```

a *cw* ritorna una ricca documentazione on-line. Un aiuto più specifico su singoli comandi, ad esempio sul comando plot, si ottiene con

```
help plot
```

a *cw*. La documentazione relativa si ottiene con

```
doc plot
```

Per argomenti generici, non corrispondenti necessariamente ad un comando, oltre alla documentazione ottenuta con helpdesk, si può usare, ad esempio

```
lookfor sine
```

che cerca essenzialmente tutti i file.m (vedremo dopo cosa vuol dire questo) contenenti la parola sine in modo significativo.

1.3 Si inizia

Il modo più semplice per utilizzare MATLAB consiste nel dare istruzioni a riga di comando che verranno eseguite immediatamente dal programma col tasto "Invio". Si notino le parti commentate, precedute cioè dal simbolo `%`: si inseriscono (facoltativamente) nel codice per chiarire cosa si sta facendo. Non riportiamo i risultati che si visualizzano invece a *cw* sotto il comando.

Le quattro operazioni. Cominciamo con qualcosa di facile.

```
2+3              %il risultato dell'operazione è assegnato
                 %alla variabile predefinita ans
2*3              %il valore della variabile ans è sovrascritto
2/3
3 + 4            %gli spazi non contano
a = 2            %assegniamo un valore ad una variabile
A = 4            %Matlab è un programma "case sensitive"
b = a*2
c = a+b;         %notazione ";", memorizza senza mostrare il risultato
c                %visualizziamo il valore della variabile
```

Le variabili create vengono memorizzate da MATLAB all'interno del *workspace*, il cui contenuto può essere visualizzato a *cw* col comando

```
whos
```

Per ciascuna variabile memorizzata vengono elencati nome, dimensione (in termini di righe e colonne), spazio occupato in memoria e tipologia (variabile numerica, simbolica, logica e così via).

Ricordiamo che la divisione per zero non ha senso algebrico; MATLAB ci avvisa di questo facendo comparire l'espressione `Inf` (Infinity) se il dividendo è diverso da zero (è un po' come se facesse il limite) e `NaN` (Not a Number) se il dividendo è zero.

```
7/0               %Inf
0/0               %NaN
0/7               %0, ovviamente
```

Pulizia. Le variabili che abbiamo definito sono mostrate nella finestra workspace. Per cancellare le variabili dal workspace, chiudere le figure (quest'ultimo sarà utile in seguito) e ripulire la *cw*, si usano i comandi

```
clear           %per pulire il workspace...
close all       %... chiudere tutte le figure...
clf              %... chiudere la figura corrente... (CLear current Figure)
clc              %... e ripulire la cw (CLear Command window)
```

Matrici. Introduciamo ora alcune semplici operazioni con le matrici, sempre da effettuarsi a *cw*.

```
a = [0 1 2 3]          %un vettore riga...
size(a)                %size: la sua dimensione
max(a)                 %l'elemento massimo di a
min(a)                 %l'elemento minimo di a
abs(a)                 %il valore assoluto (vettoriale) di a
b = [7;5;3;1]          %un vettore colonna
length(b)              %length: il numero di elementi di un vettore
max(b)                 %max e min funzionano anche per i vettori colonna
A = [1 2 3; 4 5 6]     %la matrice A
size(A)                %la sua dimensione
A'                     %la matrice trasposta di A
z = A(2,1)             %estraiamo un elemento dalla matrice A...
b(3,1) = z             %... e lo sostituiamo nel vettore b
```

Si noti la sintassi degli ultimi due comandi: ciò che si definisce va a sinistra del segno di uguaglianza mentre la quantità nota va a destra.

Creazione di vettori. In alcuni casi, per creare vettori riga ci sono modi più rapidi, in modo da evitare di dover inserire manualmente il valore di ciascun elemento.

- Si può far uso del comando colon ":", la cui sintassi generale è

  ```
  v = [valore iniziale:passo:valore finale].
  ```
  (1.2)

  Il passo può anche avere un valore negativo; esso viene assunto uguale ad 1 se non specificato. I simboli di parentesi quadra, qui scritti per chiarezza, possono essere omessi.

- Si può far uso del comando `linspace` (linearly spaced vector), la cui sintassi è

  ```
  v = linspace(valore iniziale, valore finale, n)
  ```
  (1.3)

  e che genera un vettore di `n` elementi equispaziati, compresi tra `valore iniziale` e `valore finale`. Se non altrimenti specificato `n` assume il valore 100.

Il comando colon è molto utile anche per l'estrazione di sottovettori (o sottomatrici), insieme alla parola chiave `end` che permette di accedere all'ultimo elemento del vettore. Riprendiamo la *cw* dalla sessione precedente.

```
v = [1:10]
w = [10:10:100]
z = [7:-2:1]
x = linspace(0,1,5)
c = [a x]                  %concatenazione di vettori
d = [a';x']
e = [b' z]

A = [a;b']                 %creiamo una matrice concatenando vettori
D = A(:,1)                 %estraiamo la prima colonna di A
D = A(1:1,:)               %estraiamo la prima riga di A (altra notazione)
D = A(:,1:2)               %estraiamo le prime due colonne di A
D = A(:,1:3)               %estraiamo le prime tre colonne di A
B = A(:,1:end-1)           %sintassi equivalente: uso la parola chiave "end"
E = A(:,2:4)               %estraiamo le ultime tre colonne di A
F = A(1:2,2:end)           %estraiamo una sottomatrice di A
```

Operatori relazionali. Può essere talvolta utile confrontare due matrici A e B (aventi le stesse dimensioni!) per vedere se esse hanno gli stessi elementi. Per questo MATLAB usa l'operatore relazionale `==` che crea una matrice della stessa dimensione delle matrici A e B i cui elementi sono 1 o 0, a seconda che gli elementi delle matrici A e B coincidano o meno.

```
D == B                     %infatti le due matrici coincidono!
```

Altri operatori relazionali sono i seguenti: `==` (uguale), `~=` (diverso), `>` (maggiore), `<` (minore), `>=` (maggiore o uguale), `<=` (minore o uguale). Facciamo un semplice esempio del loro uso.

```
X = [5 6 7 8]
Y = [5 4 8 10]
X <= Y                    %operatore relazionale
Z = [5;4;8;10]
X <= Z                    %le matrici devono avere le stesse dimensioni!
```

Operazioni con matrici. Le seguenti operazioni elementari tra matrici possono essere eseguite solo se le loro dimensioni sono le stesse. Le operazioni vengono eseguite *elemento per elemento* e per moltiplicazione, divisione ed elevamento a potenza occorre anteporre il punto all'operatore. Si noti bene che la moltiplicazione di matrici così definita *non* coincide con quella comunemente impiegata nell'algebra di matrici (si veda il corso di Geometria) in cui, nella sintassi di MATLAB, il punto viene omesso. In queste note ci riferiremo sempre alla moltiplicazione elemento per elemento. La moltiplicazione di uno scalare x per una matrice A è possibile e dà una matrice i cui elementi si ottengono da quelli di A moltiplicandoli per x; in questo caso la sintassi è `x*A`, che dà lo stesso risultato di `x.*A`.

```
B+E               %somma algebrica
B.*E              %prodotto
3*B               %prodotto di uno scalare per una matrice
3.*B
B./E              %divisione
B.^2              %elevamento a potenza
```

Matrici speciali. Ecco infine alcune matrici un po' speciali:

```
C = ones(2,3)             %matrice (2X3) di soli uno
Z = zeros(2,3)            %matrice (2X3) di zeri
R = rand(3,4)             %matrice (3X4) di numeri casuali
```

Il comando `rand` crea una matrice di numeri casuali compresi tra 0 e 1. Provare a ripetere più di una volta il comando `rand` a riga di comando.

1.4 Esercizi

Tutti gli esercizi seguenti sono da eseguire a *cw*. Eventuali risposte sono indicate brevemente tra parentesi quadre.

1.4.1 Posto $a = 2$ e $b = \frac{1}{3}$, calcolare $c = \frac{a^{-2}+3b}{(9b-a)}$.

1.4.2 Creare il vettore riga `A` formato dagli elementi $5, 6, 7, 8$. Come assegnare ad una nuova variabile `b` il valore del terzo elemento di `A`?

1.4.3 Creare a piacere un vettore riga x di cinque elementi compresi tra 1 e 3.

- assegnare al terzo elemento il valore 3;
- estrarre v, sottovettore di x, che ne contenga gli ultimi tre elementi;
- determinare l, numero di elementi di v;
- determinare il vettore y, prodotto di l e x;
- calcolare il trasposto di y.

1.4.4 Creare una matrice A di dimensioni 3×3, che abbia tutti gli elementi uguali a zero tranne quelli sulla diagonale (cioè $A(1,1)$, $A(2,2)$, $A(3,3)$).

- Siano a e b i vettori di elementi [1 4 7] e [3 2 1], rispettivamente (usare la notazione :);
- costruire la matrice B che abbia come b come prima riga e a come seconda;
- sia C la trasposta di B;

- creare un vettore colonna w di numeri casuali compresi tra 0 e 3 (moltiplicare...);
- aggiungere a C una terza colonna corrispondente al vettore w;
- elevare al cubo gli elementi della matrice C;
- moltiplicare le matrici A e C, ottenendo la matrice D;
- estrarre la sottomatrice E composta dagli elementi (1,1), (1,2), (2,1), (2,2) di D.

1.4.5 Spiegare la differenza tra B e C:

```
A = 1:1:10;
n = 1:1:10;
B = A/n
C = A./n
```

[La prima è una divisione tra matrici, la seconda componente per componente]

1.4.6 Generare un vettore riga le cui componenti sono cinque numeri causali compresi tra 0 e 10.

1.4.7 Ci si può chiedere perché non vada bene il comando `linspace[-2,2]`. E allora si prova con `[-2:0:2]`, ma neanche questo va bene. Perché?

1.4.8 Se `x` è un vettore, il comando `mean(x)` calcola la media aritmetica delle componenti di `x`. Se `A` è una matrice, il comando `mean(A)` calcola le medie aritmetiche dei vettori di `A`. Fare qualche prova.

Se `x` e `y` sono due vettori con la stessa lunghezza, è vero che `mean(x+y) = mean(x) + mean(y)`?

1.5 Approfondimenti

1.5.1 (Creazione di vettori) Notare la differenza tra le due scritture (1.2) e (1.3), apparentemente simili. Nel primo caso si specifica il *passo*, che è compreso tra il valore iniziale e il valore finale; nel secondo, si specifica il *numero di elementi del vettore*, mettendolo dopo il valore finale. Ad esempio, il comando

`v = [0:1:5]` dà il vettore 0 1 2 3 4 5

costituito da 6 elementi, mentre il comando

`w = linspace(0,5,5)` dà il vettore 0 1.2500 2.5000 3.7500 5.0000

costituito da 5 elementi. Si noti che se il passo non è un divisore della differenza tra il valore iniziale e quello finale, allora il comando (1.2) non divide l'intervallo [Valore iniziale, Valore finale] in parti uguali: si provi, ad esempio, `v = [0:4:5]`. Usare il comando `linspace` per generare il vettore 0 1 2 3 4 5.

1.5.2 Da riga di comando calcolare 8/7. Ci si può chiedere il perché di quattro cifre decimali; si provi `help format`. Ripetere lo stesso calcolo con `format long` e `format rat`, dove `format rat` sta per "rational". Ad esempio, calcolare la somma $\frac{3}{4} + \frac{5}{6}$ a *cw* dichiarando prima `format long` poi `format short` e infine `format rat`

1.5.3 MATLAB usa per numeri (molto) grandi o (molto) piccoli la notazione esponenziale. Ad esempio

```
123^123
```

dà il risultato

```
1.1437e+257
```

che sta per $1,1437 \cdot 10^{257}$. La lettera `e` non indica dunque, in questa scrittura, la base dei logaritmi naturali, ma il processo di esponenziazione in base 10.

Analogamente

```
1/1234
```

dà il risultato

```
8.1037e-004
```

che sta per $8,1037 \cdot 10^{-4}$.

1.5.4 Può capitare di imbarcarsi in calcoli veramente complessi che richiedono tempi di esecuzione molto lunghi (questo non succederà mai in queste note). Se si vuole interrompere l'esecuzione, Ctrl + C a *cw*.

Capitolo 2

Gli script e la grafica in Matlab

In questa sezione introduciamo gli script: si tratta di semplici files che contengono varie istruzioni di programmazione e ce ne serviremo sistematicamente nel seguito. Contemporaneamente illustriamo i principali comandi grafici di MATLAB, per non appesantire gli script dei capitoli seguenti con queste istruzioni.

ATTENZIONE! *Da questa sezione inclusa in poi, dunque, non diamo più comandi a* cw *ma apriamo un file di testo sul quale lavoriamo e che facciamo poi eseguire da* MATLAB.

2.1 Script

Tutti i comandi della sezione precedente sono stati dati a *cw*. In molti casi è comodo poter salvare una lista di comandi di MATLAB che verranno eseguiti dal programma uno di seguito all'altro. Un simile file si chiama script. Tutti i file script di MATLAB hanno l'estensione .m (senza tanta fantasia) e sono chiamati M-file; normalmente vengono salvati nella cartella work di MATLAB, ma possiamo salvarli dove preferiamo.

Nel seguito useremo sempre gli script. Ciò ha il vantaggio di poter presentare in maniera compatta le istruzioni di programmazione e, una volta salvati, permettono di creare una biblioteca di file da impiegare quando serve. In particolare, ognuno degli esempi seguenti è un M-file e deve essere eseguito dalla finestra Editor di MATLAB; si ricordi di settare in maniera opportuna il percorso della Current Directory.

Può essere utile iniziare un file.m con il suo nome commentato: nel seguito faremo sempre così. Inoltre, per chiarezza, le prime tre righe possono essere

```
clear
close all
clc
```

il cui significato è stato chiarito nella sezione precedente.

Attenzione: il nome `nome.m` di uno script ("nome", in questo caso) obbedisce ad alcune semplici regole.

- Deve iniziare con una lettera; può contenere numeri, il segno di sottolineatura _ ma non il trattino -; deve contenere al più 31 caratteri. Pertanto `prova_1.m` va bene, `prova-1.m` no. MATLAB è educato e se utilizzate caratteri non ammessi vi avverte quando compilate il file.

- Non deve avere lo stesso nome di una variabile elaborata nel file, di un comando o di una funzione di MATLAB. In particolare non vanno bene i nomi `linspace.m` e `sin.m`.

L'uso degli script obbliga ad avere almeno due finestre aperte nell'interfaccia di MATLAB: la *cw* e quella dello script. In effetti le finestre aperte diventeranno rapidamente tre, perché useremo anche una finestra grafica. L'uso del mouse per cambiare finestra può essere evitato: ogni sistema operativo ha una sequenza di tasti che permettono il cambiamento di finestra da tastiera. Esso è spesso generato dalla sequenza Alt + Tab. Analogamente, spesso i tasti funzionali gestiscono la compilazione degli script; tipicamente il tasto da usare è F5.

2.2 Il comando `plot`

Negli script seguenti vengono introdotti i primi comandi di grafica: visualizzare è uno tra gli scopi principali di questo corso. Il comando principale è `plot`.

Se abbiamo due vettori $x = (x_1, x_2, \ldots, x_n)$ e $y = (y_1, y_2, \ldots, y_n)$ di ugual lunghezza, il comando `plot(x,y)` riporta in un piano cartesiano le coppie $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, collegandole (o meno) con dei segmenti. Pertanto `plot` è un comando lineare. Nel caso del grafico di una funzione, avremo evidentemente $y = (y_1, y_2, \ldots, y_n) = \big(f(x_1), f(x_2, \ldots, f(x_n)\big)$.

Il dominio di una funzione (di una variabile) in MATLAB è dunque un vettore, di solito indicato con `x`. Per generare `x` abbiamo i due modi introdotti nella Sezione 1.3: tramite l'operatore colon o tramite il comando `linspace`. Ed ecco un primo grafico.

```
%gr_x2.m
x = linspace(-2,2);
plot(x,x.^2)
```

Nella prima riga abbiamo scritto il nome con il quale abbiamo salvato lo script. Esso è preceduto dal simbolo `%` in modo che MATLAB non esegua questa istruzione. Non è indispensabile iniziare un file scrivendone il suo nome (commentato), ma può essere utile per capire subito di cosa si tratta. Nel seguito, per brevità, lo ometteremo quasi sempre.

Benché il grafico appaia come una linea continua, in effetti così non è. Questo diventa evidente se prendiamo un vettore x con pochi punti (apriamo un nuovo file, gli diamo un nuovo diverso dal precedente):

```
%gr_x2_1.m
x = -2:0.5:2;
plot(x,x.^2)
```

Si noti come le coppie di punti siano congiunte per default da segmenti. Per prendere un po' di dimestichezza con la generazione di vettori, rimpiazzare la prima riga dello script (parte commentata esclusa...) con `x = linspace(-2,2,1000)`, con `x = [-2:0.1:2]`.

Le coppie ottenute con il comando plot possono essere rappresentate in diversi modi tramite il comando `plot(x,y,'s')`, dove `s` può specificare il colore

`b` blu, `g` verde, `r` rosso, `k` nero,

la forma

`.` punto, `o` cerchio, `x` una x, `*` un asterisco,

il modo in cui le coppie sono congiunte

`-` linea continua, `:` linea punteggiata, `--` linea tratteggiata, `-.` linea-punto

o una qualsiasi combinazione delle precedenti tre opportunità. Per esplorare tutti i comandi legati alla grafica nel piano utilizzare `help graph2d`. Per un elenco dettagliato delle specifiche di linea utilizzare invece `doc linespec`.

Ad esempio, lo script

```
x = -2:0.5:2;
plot(x,x.^2,'or')
```

rappresenta le coppie con dei cerchietti rossi (non più collegati tra loro con segmenti).

Ecco in opera un'altra serie di istruzioni grafiche:

```
x = linspace(-2,2);
plot(x,x.^2,'r--')                    %grafico in rosso, tratteggiato
xlabel('x')                           %il nome dell'asse x
ylabel('y')                           %il nome dell'asse y
grid on                               %la griglia
title('Grafico della funzione x^2') %il titolo della figura
```

Notare che, per la gestione delle stringhe di testo, in MATLAB è possibile utilizzare la sintassi TEX (qui le cose si fanno un po' più complicate, ma si può lasciar perdere). Questo tipo di sintassi, che funziona solo per le stringhe di caratteri e non per l'assegnazione delle variabili, permette di scrivere in notazione matematica. Ad esempio il comando `x^2` restituirà nella finestra grafica direttamente x^2; il comando `a_n` produrrà a_n nella finestra grafica.

La grandezza dei marcatori o dei caratteri può essere modificata come segue

```
x = linspace(-2,2);
plot(x,x.^2,'linewidth',4)          %linewidth: spessore di linea
hold on                             %"congela" il grafico
plot(0,0,'or','markersize',14)      %markersize: dimensione di *
xlabel('x','fontsize',12)           %fontsize: dimensione del carattere
ylabel('y','fontsize',12)
```

Ad esempio, `'linewidth', 4` specifica che lo spessore è quadruplicato rispetto a quello di default. Si noti il comando `hold on` che permetta di sovrapporre un secondo grafico (nel caso precedente, costituito da un solo punto) al primo.

Questi sono i principali comandi grafici; qualche altro sarà introdotto nel seguito.

2.3 Esercizi

2.3.1 Cosa non va nel seguente script?

```
x = linspace(0,1)
plot[x,x.^2]
```

2.3.2 Pare che il povero Artemio abbia dei problemi con i grafici. Ora vorrebbe disegnare in un unico grafico le funzioni $\sqrt{x}$ e x^2 e scrive

```
x=linspace(0,1);
plot(x,sqrt(x))
plot(x,x.^2)
```

ma ne vede solo una... Cosa non va?

2.3.3 In un grafico troviamo questa lista di comandi. Cosa specificano?

```
plot(x,y,'or:','markersize',10,'linewidth',2)
xlabel('x','fontsize',10)
title('Figura','fontsize',12)
```

2.4 Approfondimenti

2.4.1 Per brevità abbiamo specificato nel testo solo una lista parziale dei colori e delle forme disponibili. Ecco una lista completa:

- colori: `b g r c m y k w`
- forme dei marcatori dei punti: `. o x + * s d v < > p h`

Capitolo 3

Successioni

In questa sezione iniziamo la scoperta di MATLAB analizzando il comportamento di qualche semplice successione numerica. Per rendere più semplice il codice, gli script seguenti sono quasi totalmente privi delle istruzioni grafiche introdotte nel capitolo precedente. Lasciamo al lettore il compito di migliorare l'aspetto grafico delle elaborazioni seguenti come preferisce.

3.1 La successione geometrica

Iniziamo ora la studio numerico delle successioni con la successione geometrica $\{q^n\}$. Essa converge (a 0) se e soltanto se $|q| < 1$; se $q = 1$ essa vale identicamente 1, ovviamente, se $q > 1$ essa diverge a $+\infty$, se $q \leq -1$ essa è indeterminata.

```
n = 0:20;                          %definisco il vettore n:notare ";"
q = 3/4;                             %assegno la variabile q
an = q.^n;                           %calcolo degli an: notare il punto
plot(n,an,'or')                      %punti: cerchi rossi
```

3.2 Successioni definite per ricorrenza e il ciclo `for`

Nello script seguente si introduce il fondamentale ciclo `for` , che permette di eseguire ripetutamente (per un numero definito di volte) un gruppo di comandi. Esso ha la sintassi generale:

```
for indice = inizio : passo : fine
        istruzioni
end
```

Nell'esecuzione del ciclo viene assegnato il valore di *inizio* al contatore, incrementato poi del valore *passo*; le *istruzioni* vengono eseguite una volta per ogni passaggio, utilizzando il valore corrente dell'*indice*. L'elaborazione continua finché l'*indice* supera il valore di *fine*. Se il passo non è specificato esso è 1 (come abbiamo già visto quando abbiamo introdotto l'operatore colon).

Una successione $\{a_n\}$ definita per ricorrenza è una successione della quale si conosce il primo termine a_1; gli altri termini $a_2, a_3, \ldots$ vengono calcolati tramite una espressione ricorsiva del tipo

$$a_{n+1} = f(a_n), \quad \text{per } n \geq 1,$$

dove f è una funzione data. Pertanto $a_2 = f(a_1)$, $a_3 = f(a_2) = f\left(f(a_1)\right)$ e così via.

Come esempio (semplicissimo) scegliamo $a_1 = 1$ e $f(x) = \frac{1}{1+x}$. Questo vuol dire che

$$a_2 = \frac{1}{1+a_1} = \frac{1}{1+1} = \frac{1}{2},$$
$$a_3 = \frac{1}{1+a_2} = \frac{1}{1+\frac{1}{2}} = \frac{2}{3},$$
$$a_4 = \frac{1}{1+a_3} = \frac{1}{1+\frac{2}{3}} = \frac{3}{5},$$

e così via. In questo caso non è difficile dare un'espressione esplicita del termine a_n, ma si capisce bene che in generale non è così. Ecco lo script nella forma più semplice.

```
n=1:10;
a(1)=1;                        %assegno il primo elemento
for i=2:length(n)              %ciclo for il calcolo degli a_i
    a(i) = 1/(1+a(i-1));
end
plot(n,a,'o')                  %ho costruito il vettore a
```

Si noti che in questo modo abbiamo costruito un vettore $a = \big(a(1), \dots, a(n)\big)$. Il comando `plot(n,a)` dà quindi il grafico della successione.

Perché abbiamo scritto `length(n)`? Bastava scrivere 10, in effetti. Il vantaggio di questo modo di scrivere è che, se vogliamo disegnare i primi 100 termini della successione, basta che cambiamo solo la prima riga con `n=1:100`, senza dover intervenire altrove nello script.

Se diamo un'occhiata ai "warning" (quei trattini orizzontali che compaiono nella schermata dell'editor di MATLAB) vediamo che MATLAB è infastidito dal fatto che la variabile a cambia dimensione ad ogni iterazione (vero: ad ogni iterazione la sua dimensione aumenta di 1!). MATLAB preferirebbe (eseguirebbe più velocemente il file) che dichiarassimo fin dal principio la dimensione finale di a. In questi semplici calcoli la velocità di esecuzione non ha importanza, ma diventa essenziale in situazioni (molto) più complicate. Ecco pertanto lo script di sopra con una riga di codice in più per la *preallocazione* della dimensione di a.

```
n=1:10;
a=zeros(1,length(n));          %creo un vettore di n zeri
a(1)=1;                        %assegno il primo elemento
for i=2:length(n)              %ciclo for il calcolo degli a_i
    a(i) = 1/(1+a(i-1));
end
plot(n,a,'o')                  %ho costruito il vettore a
```

A riga 2 abbiamo creato un vettore con `length(n)` componenti, dunque 10 in questo caso. Questo vettore è composto da zeri (ma anche `ones(size(n))`, in cui tutte le componenti sono 1, sarebbe andato bene) e lo andremo a modificare nel corso dell'iterazione; tuttavia MATLAB sa già che la dimensione di a è `length(n)`.

Nel seguito useremo spesso i numeri e e π. La sintassi di MATLAB per il secondo è semplicemente `pi`; invece il comando `e` non è riconosciuto da MATLAB. Pertanto quando dovremo usare il numero e useremo l'espressione `exp(1)`; se poi ne facciamo un uso ripetuto, definiamo e una volta per tutte all'inizio del file con

```
e=exp(1)
```

e dopo di che possiamo usare il simbolo `e` come al solito. MATLAB riconosce invece correttamente `i` come unità immaginaria, ma questo è argomento di Analisi II.

3.3 Complementi

3.3.1 Altri esempi

Facciamo ora vedere qualche altro esempio. Si ricordi che $\lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n = e$.

```
n = 0:50;
e = exp(1);                  %e=exp(1)
bn = (1+1./n).^n;
plot(n,bn,'ob',n,e,'.r')
legend('successione (1+1/n)^n','e','Location','SouthEast')
```

Si notino in questo script:

- nel comando plot la rappresentazione dei punti (n, b_n) con cerchi blu e dei punti (n, e) con punti rossi, che viene ottenuta scrivendo i comandi uno dopo l'altro;
- la legenda e la specifica della posizione. Altre posizioni (`'Location'`) sono `'NorthEast'`, `'NorthWest'`, `'SouthWest'`.

Confrontiamo ora numericamente le successioni $\{\frac{1}{n}\}$, $\{\frac{1}{\sqrt{n}}\}$ e $\{\frac{1}{n^2}\}$, tutte infinitesime.

```
n = 1:20;
an = 1./n;
bn = 1./sqrt(n);
cn = 1./n.^2;
plot(n,an,'.r',n,bn,'.b',n,cn,'.k')
grid on
legend('1/n','1/n^{1/2}','1/n^2')
```

Ricordiamo infine la formula di Stirling

$$n! \sim n^n e^{-n}\sqrt{2\pi n}, \qquad \text{per } n \to \infty,$$

che dà l'asintotico $\sqrt[n]{n!} \sim \frac{n}{e}$. Verifichiamolo numericamente. Il comando di MATLAB per il fattoriale di n è `factorial(n)`.

```
n=1:100
plot(n,(factorial(n)).^(1./n),'o')
hold on
plot(n,exp(-1)*n)
```

3.3.2 La successione di Fibonacci

Ecco un esempio decisamente più complesso che riguarda una successione più difficile ... che richiede un minimo di lavoro di programmazione.

Questa successione deriva da un semplice problema di dinamica delle popolazioni; in una versione semplificata (mettiamo da parte per un momento l'etica...) il problema è il seguente.

> Una coppia di conigli genera ogni mese una coppia di piccoli, che a loro volta possono riprodursi ad un mese dalla nascita. Supponiamo per semplicità che ogni coppia di nuovi nati sia costituita da un maschio e da una femmina, e che si accoppino tra loro. I conigli non muoiono mai.
>
> All'inizio ho solo una coppia di conigli ancora immaturi. Come evolve questa popolazione?

Indichiamo con f_n il numero di coppie di conigli al mese n. Allora:

- al primo mese, $f_1 = 1$ (i piccoli, ancora immaturi);
- al secondo mese $f_2 = 1$ (sempre loro, ma ora sono maturi);
- al terzo $f_3 = 2$ (hanno generato una coppia di piccoli);
- al quarto $f_4 = 3$ (i genitori hanno generato un'altra coppia di piccoli);
- al quinto $f_5 = 5$ (hanno generato anche i conigli nati il terzo mese)...

La successione di Fibonacci[1] $\{f_n\}$ è definita per ricorrenza così:

$$f_1 = 1, \qquad f_2 = 1, \qquad f_n = f_{n-2} + f_{n-1} \quad \text{per } n \geq 3. \tag{3.1}$$

La formula $f_n = f_{n-2} + f_{n-1}$ (un elemento della successione è la somma dei due precedenti) dice semplicemente che le coppie al mese n sono quelle del mese precedente (f_{n-1}) più i nuovi nati, che sono pari al numero di coppie fertili (f_{n-2}).

Indichiamo con

$$q_n = \frac{f_{n+1}}{f_n}$$

il tasso di crescita della popolazione. Si può dimostrare facilmente che:

$$\lim_{n\to+\infty} q_n = \Phi = \frac{1+\sqrt{5}}{2} \simeq 1.6180\,.$$

Il numero (irrazionale) Φ è anche detto *sezione aurea* o *numero di Fidia* (da cui il simbolo).

Nello script seguente utilizziamo il ciclo `for` per calcolare iterativamente i termini della successione di Fibonacci. Si veda [2, §5.1.3] per un altro modo di produrre la successione di Fibonacci tramite le concatenazioni.

[1] Leonardo Fibonacci (Pisa, 1170 – Pisa, 1240 ca.)

```
n=1:10;
f=zeros(1,length(n));                  %fisso la dimensione
f(1)=1;                                %assegno il primo elemento...
f(2)=1;                                %...e il secondo
for i=3:length(n)                      %ciclo for per il calcolo degli f_i
        f(i)=f(i-2)+f(i-1);
end
display(f)                             %per vedere i numeri a cw
figure(1)                              %apro figura 1, numeri di Fibonacci
plot(n,f,'.')
grid on
xlabel('n')
ylabel('f_n')
title('Successione di Fibonacci')
%
q=zeros(1,length(f)-1);                %fisso la dimensione
for i=1:length(n)-1                    %ciclo for per il tasso di crescita
        q(i)=f(i+1)/f(i);              %length(q) = length(f) - 1
end
phi=(1+sqrt(5))/2;                     %sezione aurea
figure(2)                              %apro figura 2, tasso di crescita
plot(n(1:end-1),q,'or')                %i quozienti sono uno di meno
hold on                                %hold on: "congelo" il grafico
plot(n(1:end-1),phi,'.k')              %rappresento la sezione aurea
grid on
xlabel('n')
ylabel('q_n')
title('Tasso di crescita')
legend('f(n+1)/f(n)','\Phi',1)         %1: posizione legenda
```

Nello script precedente abbiamo creato due figure, una per la successione $\{f_n\}$ e una per il tasso di crescita $\{q_n\}$. La specifica del loro numero poteva essere omessa. Per maggior chiarezza abbiamo invece specificato i nomi degli assi e i titoli delle figure. Di solito, l'apertura di più figure in uno stesso script è comoda per evitare di creare altri file; è però raccomandabile, soprattutto agli inizi, di scrivere script brevi, ognuno con una sola figura. Si noti di nuovo qui sopra la sintassi TEX: il comando `\Phi` restituisce nella finestra grafica la lettera greca maiuscola Φ.

3.4 Esercizi

3.4.1 Confrontare tra loro i primi 10 termini delle successioni $a_n = \ln(n)$, $b_n = \log_{10} x$ e $c_n = \ln_2 x$, $n \geq 1$, rappresentandole nello stesso grafico con colori e simboli diversi, utilizzando il comando `plot`. Il comando `log(x)` calcola il logaritmo naturale di `x`, mentre `log10(x)` e `log2(x)`... Aggiungere il titolo del grafico, i titoli degli assi ed una legenda opportunamente collocata. Sorpresa: cosa dà `log3(x)`?

3.4.2 Si riprenda l'esempio della Sezione 3.3.1. Per definizione di limite si ha che per ogni $\epsilon > 0$ esiste $N \in \mathbb{N}$ tale che

$$0 < e - \left(1 + \frac{1}{n}\right)^n < \epsilon \quad \text{se } n > N. \tag{3.2}$$

Si rappresenti graficamente la successione $e - \left(1 + \frac{1}{n}\right)^n$ e si determini numericamente l'N di soglia nei casi $\epsilon = 0.1$ e $\epsilon = 0.01$. Si usi il comando `axis`, che ha la sintassi

```
axis([xmin xmax ymin ymax])
```

per centrare il grafico nella regione di interesse (le x nell'intervallo $[xmin, xmax]$, le y nell'intervallo $[ymin, ymax]$). Nel secondo caso ($\epsilon = 0.01$) può essere utile il calcolo diretto della differenza in (3.2); usare allora `format long`. Per chiarezza rappresentare i punti della successione con dei semplici puntini `'.'` nel plot.

[13, 135]

3.4.3 Rappresentare la successione a_n definita in maniera ricorsiva come: $a_1 = \sqrt{2}$, $a_{n+1} = \sqrt{2 + a_n}$ per $n > 1$, utilizzando un ciclo `for` e settando opportunamente gli assi. Identificare graficamente a che valore converge la successione.

3.4.4 Si migliori graficamente lo script relativo alla successione di Fibonacci introducendo uno spessore di linea 2 e una grandezza dei marcatori 14 nella seconda figura.

3.4.5 (Il bravo allevatore) Dopo quanti mesi il nostro allevatore avrà almeno 1000 conigli?

L'allevatore vuole poi perfezionare il suo modello, includendovi il fatto che dopo sei mesi una coppia di conigli muore di vecchiaia. Come deve cambiare la (3.1)? Rielaborare lo script sulla base di questo nuovo modello.

3.4.6 Cosa fa il seguente script di MATLAB?

```
somma=0;
for(n=1:10)
   somma=somma+n
end
```

3.5 Approfondimenti

3.5.1 Come sapere se il nome `pippo.m` che stiamo dando ad uno script è già presente come nome di un comando o di una funzione? A *cw* scrivere

```
exist('pippo')
```

Se il risultato è 0, tutto a posto, non c'è. Se invece troviamo come risposta i numeri 1, 2, 5, allora questo vuol dire che esiste una variabile, un file, una funzione, rispettivamente, con quel nome. Provare ad esempio i nomi `linspace`, `sin`. Molto faticoso, ma ne capiremo il senso quando introdurremo le `function`.

3.5.2 Si riprenda l'esempio della sezione 3.1 e si elimini `'or'` dal comando `plot`. Risultato? Per capire meglio considerare il caso `n = [1:3]`. Dunque...

3.5.3 Si consideri lo script della Sezione 3.3.2; si può pensare di semplificarlo togliendo il comando `display(f)` e togliendo il punto e virgola due righe sopra. Cosa si trova?

3.5.4 Chissà perché Φ è detto sezione aurea... e poi chi era Fidia? Una piccola ricerca in rete può aiutare lo studente curioso.

3.5.5 Abbiamo visto che la radice quadrata di un numero x è calcolata col comando `sqrt(x)`. Il comando

```
nthroot(x,n)
```

calcola la radice n-esima di x. Ad esempio, $\sqrt[3]{2}$ si calcola col comando `nthroot(2,3)`. Il numero n può essere anche non naturale. Ad esempio, il comando `nthroot(2,1.5)` dà $2^{\frac{3}{2}}$.

3.5.6 In questa sezione abbiamo visto il comando `legend`. Cambiare i font in una legenda è leggermente più complicato che cambiarli, ad esempio, nelle etichette degli assi. La cosa più semplice è procedere nel modo seguente, cioè trattando la legenda come una specie di variabile, usando poi il comando `set` per impostarne alcuni parametri. In riferimento alla successione di Fibonacci, rimpiazzare

```
legend('f(n+1)/f(n)','\Phi',1)
```

con

```
nuova_legenda = legend('f(n+1)/f(n)','\Phi',1);
set(nuova_legenda,'fontsize',14)
```

3.5.7 Nel plottare successioni un comando utile è `scatter`, che crea dei grafici in cui i punti relativi sono rappresentati da cerchietti. Provare ad esempio:

```
n=1:10;
scatter(n,(-1).^n./n)
```

Si possono ottenere più effetti: provare ad esempio

```
scatter(n,(-1).^n./n,30,'r','filled')
```

e magari indagare altre proprietà con `help scatter`.

3.5.8 In questi appunti cerchiamo di scrivere codici non troppo lunghi, ma può capitare di dover scrivere un codice per produrre più figure, come ad esempio nella Sezione 3.3.2. Per default MATLAB mette queste figure una sull'altra; in questo modo si vede a schermo solo l'ultima generata. E' possibile cambiare questa impostazione di MATLAB in modo da disporre le figure sullo schermo come vogliamo noi. Per farlo, vi sono vari modi, ma forse il più semplice è di usare il comando

```
figure('units','normalized','position',[a b L A])
```

quando si crea una figura. Il vettore `[a b L A]` contiene le coordinate `(a,b)` dell'angolo in basso a sinistra della figura e le dimensioni `(L, A)` = (larghezza, altezza) della finestra. In questo caso abbiamo usato le unità `normalized` che si riferiscono alla frazione dell'intero schermo. Ad esempio

```
figure('units','normalized','position',[0 0.5 0.3 0.4])
```

creerà una figura il cui angolo sinistro in basso è a metà del lato sinistro del monitor ($x = 0$, $y = 0.5$), larga 0.3 volte la larghezza del monitor e alta 0.4 volte l'altezza del monitor.

Perciò, per avere quattro figure visibili contemporaneamente sullo schermo si usano di seguito i comandi

```
figure('units','normalized','position',[0 0 0.4 0.4])
figure('units','normalized','position',[0 0.5 0.4 0.4])
figure('units','normalized','position',[0.5 0 0.4 0.4])
figure('units','normalized','position',[0.5 0.5 0.4 0.4])
```

Abbiamo tenuto le dimensioni un po' inferiori (0.4) per avere un po' di distanza tra le figure. Esse sono collocate, in ordine: a sinistra in basso, a sinistra in alto, a destra in basso, a destra in alto.

Capitolo 4

Serie

In questo capitolo analizziamo numericamente il comportamento di alcune serie. Per semplificare il più possibile gli script, non utilizzeremo da questo capitolo in poi le istruzioni grafiche introdotte nel capitolo precedente (ad esempio: il nome degli assi, il titolo della figura, l'uso dei colori e così via). Lasciamo allo studente l'aggiunta di queste istruzioni che, pur non indispensabili, rendono nondimeno più chiara la lettura di una finestra grafica.

Importanti sono i comandi

`sum`, `cumsum`.

Nel caso di un vettore `x`:

(i) il comando `sum(x)` somma semplicemente gli elementi di `x` e dà quindi uno scalare;

(ii) Il comando `cumsum(x)` fa invece la somma cumulativa, un vettore che ha per i-esima componente la somma degli elementi di `x` da `x(1)` a `x(i)`.

Petanto, se assimiliamo `x` ad una successione (composta da un numero finito di termini), `sum(x)` dà la somma e `cumsum(x)` la successione delle somme parziali della successione.

Infine, il comando

`factorial(n)`

calcola il fattoriale del numero intero n.

4.1 La serie geometrica e l'istruzione condizionale `if`

Ricordiamo la serie geometrica

$$\sum_{n=0}^{\infty} q^n,$$

che converge a $\frac{1}{1-q}$ se $|q| < 1$, diverge a $+\infty$ se $q \geq 1$ ed è indeterminata se $q \leq -1$.

Nello script seguente introduciamo l'istruzione condizionale `if`, che permette di eseguire in maniera selettiva gruppi di istruzioni. Essa ha la sintassi generale:

```
if espressione logica 1 = true
   istruzioni 1
elseif espressione logica 2 = true
   istruzioni 2
else
   istruzioni 3
end
```

Ed ecco lo script.

```
n = 0:20;
q = 3/5;
an = q.^n;
sn = cumsum(an)                          %a cw i valori
plot(n,sn,'or')
```

```
grid on
xlabel('n')
ylabel('s_n')
hold on
if abs(q)<1                                    %istruzione if
        somma=1/(1-q);
        plot(n,somma,'ob')
end
```

Nell'esempio precedente, l'istruzione condizionale `if` è utilizzata per calcolare e visualizzare la somma della serie quando (e solo quando!) la serie converge; altrimenti non dà informazioni.

4.2 La serie armonica e il ciclo `while`

Sappiamo che la serie armonica

$$\sum_{n=1}^{\infty} \frac{1}{n}$$

è divergente. Questo vuol dire che, indicata con

$$s_n = \sum_{k=1}^{n} \frac{1}{k}$$

la successione delle sue somme parziali, si ha che $s_n \to +\infty$ per $n \to \infty$. Questo, a sua volta, vuol dire che, fissato un numero positivo arbitrario M, esiste un numero naturale N tale che $s_n > M$ se $n > N$. Il calcolo esplicito della successione $\{s_n\}$ non dà luogo ad una espressione semplice, come invece succedeva nel caso della serie geometrica e dunque, da un punto di vista teorico, non è facile stabilire per quali n si ha $s_n > M$ (si pensi alla dimostrazione svolta a lezione). MATLAB può darci una mano.

Per far questo, introduciamo il fondamentale ciclo `while`. Esso viene utilizzato, diversamente dal ciclo `for`, quando *non si conosce in anticipo il numero di iterazioni da effettuare*: l'elaborazione del ciclo termina quando è soddisfatta una certa condizione. La sintassi è:

```
while espressione logica = true
   istruzioni
end
```

Le istruzioni contenute nel ciclo vengono eseguite utilizzando il valore corrente della variabile di ciclo (che deve essere contenuta nell'espressione logica) finchè l'espressione logica rimane vera, altrimenti il ciclo termina e vengono eseguite le istruzioni che si trovano dopo la parola chiave `end`.

Ecco lo script che ci serve.

```
spz = 0;                %spz: somma parziale
k = 0;
while spz < 8           %quando spz supera 8 il ciclo si ferma
    k = k+1;            %se no, incrementiamo k di 1...
    spz = 1/k + spz;    %...e addizioniamo un termine alla spz
end
disp(['Il numero dei termini è:' num2str(k)])
disp(['La somma parziale dei termini è:' num2str(spz)])
```

Le ultime due righe richiedono una spiegazione: il comando `disp(x)` serve per visualizzare `x` a *cw*; in molti casi, ciò è equivalente a togliere il punto e virgola nell'espressione che dà `x`. Il comando `num2str` (da numero a stringa di testo) converte appunto un numero (come qui sopra) in un testo. Questo comando è anche utile, ad esempio, per inserire numeri nel titolo di una figura. La sintassi con le parentesi quadre (l'output è un vettore di due componenti, ecco perché lo spazio tra il secondo apice e `num2str`; altrimenti, si cancella lo spazio ma si mette una virgola) e gli apici è dovuta all'uso particolare che si fa di `disp` qui sopra.

4.3 Serie a termini di segno variabile

Vogliamo ora studiare le serie a termini di segno variabile. La più semplice, a termini di segno alterno, è senz'altro la serie

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

Si tratta di una serie convergente (non assolutamente), in quanto, essenzialmente, nelle somme parziali hanno luogo delle cancellazioni importanti.

Nello script seguente rappresentiamo la successione delle somme parziali; per evidenziare i diversi termini li congiungiamo con un segmento.

Per esercizio sull'istruzione condizionale `if` e per avere qualche informazione numerica sul limite, valutiamo inoltre la semisomma $\frac{s_n+s_{n-1}}{2}$, che graficamente sembra approssimare bene il limite della successione, ponendo come condizione che la differenza $s_n - s_{n-1}$ sia abbastanza piccola (più piccola di 0.1) in modo da ritenere la valutazione attendibile.

```
n = 1:50;
an = ((-1).^n)./n;
sn = cumsum(an);
plot(n,sn,'x-b')
grid on
xlabel('n')
ylabel('s_n')
%                                    %a quale valore converge la serie?
d = sn(end)-sn(end-1);
m = (sn(end)+sn(end-1))/2;
if d<0.1
    disp(['lim. appr.=',num2str(m)]) %disp: per rappresentare sulla cw
end                                  %num2str: da numero a stringa di testo
```

Informazioni più precise sulle somme delle serie (in particolare di questa) verranno date nel Capitolo 9.

4.4 Complementi

4.4.1 Stima della divergenza della serie armonica

La teoria ci può aiutare per capire "quanto rapidamente" diverga la serie armonica. Si può provare che la successione delle somme parziali $s_n = \sum_{k=1}^{n}\frac{1}{k}$ si comporta come $\ln(n)$, a meno di una costante. Più precisamente,

$$\lim_{n\to\infty}\left(\sum_{k=1}^{n}\frac{1}{k}-\ln(n)\right)=\gamma\simeq 0.5772.$$

Il numero γ è detto *costante di Eulero-Mascheroni* . Ecco lo script.

```
n = 1:50;                            %oppure: n=[1:50];
sn = cumsum(1./n);
plot(n,sn,'.r',n,log(n),'.b')
hold on
gamma = 0.5772;                      %cost Eulero-Mascheroni
plot(n,sn-log(n),'.g',n,gamma,'.k')
grid on
xlabel('n')
legend('s_n','ln(n)','s_n-ln(n)','\gamma',2)
```

4.4.2 La serie armonica generalizzata

Si tratta della serie

$$\sum_{n=1}^{\infty}\frac{1}{n^{\alpha}}$$

con $\alpha > 0$. Essa converge se $\alpha > 1$ e diverge a $+\infty$ se $\alpha \leq 1$.

Nello script seguente vogliamo analizzare il comportamento della serie nei casi $\alpha > 1$ e $\alpha < 1$. Poiché questo è piuttosto banale, vogliamo inserire una stringa di testo che ci ricordi che nel primo caso la serie converge, nel secondo diverge. Questo è ottenuto col comando `gtext`, che permette di rappresentare una stringa di testo all'interno di un grafico, scegliendone la posizione con un click del mouse.

Cambiare il valore di α per esplorare i vari casi.

```
n = 1:50;
alfa = 2;
sn = cumsum(1./(n.^alfa));
plot(n,sn,'.r')
grid on
xlabel('n')
if alfa>1                                          %istruzione if...
      gtext('La serie converge')
   elseif alfa <1                                  %...elseif...
      gtext('La serie diverge')
   else                                            %...else
      gtext('\alpha=1: La serie diverge')
end
```

Qui sopra siamo stati ridondanti nell'uso dell'istruzione condizionale `if` per mettere in rilievo l'uso di `elseif` e `else`. Possiamo rimpiazzare le ultime righe con il più breve

```
if alfa>1                                          %if...
      gtext('La serie converge')
elseif alfa <=1                                    %...elseif
      gtext('La serie diverge')
end
```

o meglio ancora con

```
if alfa>1                                          %if...
      gtext('La serie converge')
else                                               %...else
      gtext('La serie diverge')
end
```

4.5 Esercizi

4.5.1 Quanto vale la variabile `a` dopo avere eseguito il seguente codice MATLAB?

```
x = 10;
if(x > 0)
    a = 1
else
    a = 0
end
```

4.5.2 Rappresentare i primi 50 termini della serie $s_n = \sum_{n=1}^{\infty} n^{\alpha} a^n$, per $\alpha > 0$ e a <1. Modificare i valori di α e a e verificare che la convergenza della serie dipende solo da a.

4.5.3 Verificare graficamente che la serie $\sum_{n=0}^{\infty} \frac{1}{n!}$ converge al numero e. Analogamente, verificare che la serie $\sum_{n=1}^{\infty} \frac{1}{n^2}$ converge a $\frac{\pi^2}{6}$. [Si cominci con

```
n = [1:1:7];
an = zeros(1,length(n));
an(1) = 1;
an(2:end) = 1./factorial(n(2:end));
bn = 1 + sum(an(n(1:end)))
```

e poi rappresentare in un grafico...]

4.6 Approfondimenti

I comandi

`prod`, `cumprod`

sono completamente analoghi a `sum` e `cumsum` ma eseguono prodotti invece di somme. In particolare il comando `cumprod` permette di calcolare i fattoriali; convincersene inserendo a *cw*

```
n = [1 2 3 4 5 6 7];
cumprod(n)              %se li vogliamo tutti
prod(1:7)               %se ne vogliamo uno (passo 1 se non specificato)
```

Capitolo 5

Grafici di funzioni

MATLAB contiene al suo interno (*built-in*) una serie di funzioni matematiche elementari di uso comune, ad esempio la funzione esponenziale `exp`, le funzioni trigonometriche `sin`, `cos`, `tan`, e così via. Per accedere all'elenco di queste funzioni ed avere informazioni sulla loro sintassi, digitare a *cw* il comando `help elfun` (elementary functions).

5.1 La funzione $\sin x/x$

Vogliamo disegnare il grafico della funzione

$$f(x) = \frac{\sin x}{x}.$$

Abbiamo un problema: in 0 la funzione non è definita (benché ne esista il limite, ma questa è un'altra cosa). Nell'esempio seguente introduciamo il comando

`find`

che cerca gli indici di un vettore per il quale una certa espressione logica è vera.

```
x = -30:0.01:30;              %attenzione! lo zero fa parte del dominio
k = find(x == 0);             %find: cerca l'indice per cui x è nullo...
x(k) = NaN;                   %...e sostituisci 0 con NaN
y=sin(x)./x;
plot(x,y)
```

Utilizzando lo strumento *Zoom In* della finestra grafica, indagare che cosa succede in corrispondenza di $x = 0$, a cui era stato associato il valore `NaN` (ingrandire molto...). Cosa sarebbe successo se non fosse stata data questa istruzione? In proposito, ricordarsi di controllare sempre i messaggi di warning sulla *cw*.

Per evidenziare il problema precedente, consideriamo la funzione $f(x) = 1/x$ nell'intervallo $[-1, 1]$ e analizziamo i due seguenti script:

```
x = -1:0.01:1        x = linspace(-1,1)
plot(x,1./x)         plot(x,1./x)
```

Mentre nel primo caso siamo soddisfatti del risultato, nel secondo no: ci compare una riga verticale attraverso l'origine degli assi. Questo si spiega come segue.

- Nel primo caso il punto 0 fa parte del vettore delle x (come si vede ad esempio a *cw* scommentando la prima riga); MATLAB calcola `1/0 = Inf` e non collega il punto "$(0, \infty)$" con il punto precedente (né con quello seguente) del grafico.
- Nel secondo caso il punto 0 non fa parte del vettore delle x (verificarlo con `k = find(x==0)`). Pertanto MATLAB calcola i punti del grafico le cui ascisse sono immediatamente a sinistra e a destra di 0 e li unisce con un segmento. Per rendersi conto del fatto che tale linea non è verticale si può sostituire la prima riga del secondo script con `x = linspace(-1,1,10)`. Se vogliamo mantenere la campionatura delle x considerata in questo caso ed avere nello stesso tempo un grafico che ci soddisfi, basta aggiungere le istruzioni considerate nel caso della funzione $\frac{\sin x}{x}$.

5.2 Operazioni con i grafici

I seguenti script non offrono difficoltà ma vogliono aiutare lo studente a familiarizzarsi con le manipolazioni dei grafici. Ne approfittiamo per introdurre qualche comando nuovo.

5.2.1 Simmetrie

Data una funzione f definita in $\mathbb{R}$, vogliamo rappresentare il grafico delle funzioni

$$f(-x), \quad -f(x), \quad -f(-x).$$

Questi grafici sono ottenuti da quello di f per simmetria: simmetria rispetto all'asse y e rispetto all'asse x nei primi due casi, simmetria centrale rispetto all'origine degli assi nel terzo. Per evidenziare l'effetto della simmetria scegliamo come cavia una funzione che non sia né pari né dispari, cioè

$$f(x) = e^x.$$

Simmetria rispetto all'asse y Il grafico di $f(-x)$ si può banalmente ottenere nel seguente modo:

```
x = linspace(-3,3);
y = exp(x);
plot(x,y,'k',x,exp(-x),'r')
legend('f(x)','f(-x)')
```

Tuttavia, avendo già definito la funzione `y=exp(x)`, non serve calcolare esplicitamente anche `exp(-x)`: possiamo pensare di *plottare* `y` *in funzione di* `-x`:

```
x = linspace(-3,3);
y = exp(x);
plot(x,y,'k',-x,y,'r')
legend('f(x)','f(-x)')
```

Questo "trucco" corrisponde a plottare la funzione f *da destra verso sinistra* anziché *da sinistra verso destra* come si fa solitamente. Per convincersene provare il listato seguente:

```
x = linspace(-3,3);
y = exp(x);
plot(x,y,'k',-x,y,'r')
hold on
t = linspace(-3,1);
z = exp(t);
plot(t,z,'k','linewidth',2)
plot(-t,z,'r','linewidth',2)
```

Simmetria rispetto all'asse x Questo caso è più semplice: basta plottare `-y` in funzione di `x`:

```
x = linspace(-3,3);
y = exp(x);
plot(x,y,'k',x,-y,'r')
legend ('f(x)','-f(x)')
```

Simmetria rispetto all'origine degli assi Questa simmetria può essere pensata come la composizione delle simmetrie rispetto ai due singoli assi x e y. Si tratta dunque di *plottare* `-y` *in funzione di* `-x`:

```
x = linspace(-3,3);
y = exp(x);
plot(x,y,'k',-x,-y,'r')
legend ('f(x)','-f(-x)')
```

Sostituire la funzione e^x con altre funzioni per vedere l'effetto della simmetria.

5.2.2 Il valore assoluto

Ora abbiamo f e vogliamo rappresentare il grafico di $|f(x)|$. Troppo facile. Vogliamo allora rappresentare i due grafici come sottografici di una stessa finestra. Per questo usiamo il comando `subplot`. La sua sintassi è `subplot(m,n,p)`: crea una matrice $m \times n$ di grafici, e il corrente, quello che si sta rappresentando è il numero p (si conta da sinistra a destra, dall'alto in basso).

```
x = -2:0.01:2;
p = (x+3/2).*(x+1).*x.*(x-1).*(x-2); %un polinomio, p
%
subplot(1,2,1)                %2 grafici uno a lato dell'altro, faccio l'1
plot(x,p,'k')
grid on
xlabel('x')
ylabel('p')
axis([min(x) max(x) min(p) max(abs(p))]) %il comando axis
title('Il polinomio p')
%
subplot(1,2,2)                %2 grafici uno a lato dell'altro, faccio il 2
plot(x,abs(p),'r')
xlabel('x')
ylabel('|p|')
grid on
axis([min(x) max(x) min(p) max(abs(p))])
title('La funzione |p|')      %non e' piu' un polinomio...
```

Si noti il comando

```
axis([a b c d])
```

che mostra graficamente solo le ascisse comprese tra a e b e le ordinate comprese tra c e d. L'argomento di questo comando è dunque un vettore a quattro componenti. Poiché il comando `axis` ha lo stesso vettore in entrambi i grafici, possiamo meglio visualizzare l'effetto del valore assoluto. Provare a togliere entrambi i comandi `axis` dai grafici: ogni singolo grafico è più centrato rispetto al suo sistema di riferimento, ma si perde l'effetto di insieme. Nel caso si voglia limitare solo l'asse x o l'asse y si usino i comandi più semplici `xlim([a b])` e `ylim([c d])`.

Lasciamo allo studente la rappresentazione di $f(|x|)$.

5.3 Funzioni definite a tratti

Molto spesso risulta comodo definire una funzione a tratti. Consideriamo ad esempio la funzione f definita da

$$f(x) = \begin{cases} 2x, & \text{se } -2 \le x \le 0, \\ x^2, & \text{se } 0 < x \le 2. \end{cases} \tag{5.1}$$

Per definire a tratti una funzione con Matlab possiamo utilizzare diversi metodi (a seconda della complessità della funzione in questione).

- Separando il dominio. Il più breve.

```
x = -2:0.01:0;
plot(x,2*x)
hold on
x = 0:0.01:2;
plot(x,x.^2)
```

- Le concatenazioni. Per avere un solo dominio.

```
x1 = -2:0.01:0;          %dominio della "prima" funzione
f1 = 2*x1;               %prima funzione
x2 = 0:0.01:2;           %dominio della "seconda" funzione
f2 = x2.^2;              %seconda funzione
x = [x1 x2];             %concateniamo i domini
f = [f1 f2];             %concateniamo le funzioni
plot(x,f)
```

- Una combinazione dei comandi `for` e `if`.

```
x = -2:0.01:2;
for i=1:length(x)
  if x(i)<=0
   w(i) = 2*x(i);
  else
   w(i) = (x(i)).^2;
  end
end
plot(x,w)
```

- Gli operatori relazionali, che al solito restituiscono il valore 1 se la relazione è vera oppure 0 se è falsa. Il più logico.

```
x = -2:0.01:2;
z = 2*x.*(x>=-2 & x<=0)+x.^2.*(x>0 & x<=2);
plot(x,z)
```

Ci si può chiedere se questi tre metodi siano equivalenti. A riga di comando scrivere allora

```
>> y == w;
```

e si ha una prima sorpresa. Questo dipende dal fatto che, nel caso delle concatenazioni, abbiamo una coppia in più (l'ultimo `x1` coincide col primo `x2`). Da un punto di vista teorico, abbiamo esteso la definizione della funzione x^2 a tutto l'intervallo $[0, 2]$, mentre invece in f essa risulta definita solo in $(0, 2]$. Una sintassi più corretta si ottiene pertanto modificando lo script così:

```
x = [x1 x2(2:end)]; y = [y1 y2(2:end)];
```

Riprovare allora

```
>> y == w;
>> find(ans == 0)
```

5.4 Più grafici

Spesso capita di dover riportare in uno stesso piano cartesiano (o in una stessa finestra grafica) i grafici di più funzioni. Si può naturalmente iterare il comando `plot`, ma questo risulta noioso quando i grafici sono tanti. Vi sono varie possibilità e in questa sezione ne presentiamo due.

5.4.1 Con il ciclo `for`

Il più semplice. Consideriamo per esempio le funzioni x^n per $x \in [0, 1]$ e $n = 1, \ldots, 10$.

```
x=linspace(0,1);
for k=1:10
    plot(x,x.^k)
    hold on
end
```

Si noti che, per rendere più chiaro il ciclo `for`, si poteva mettere il comando `hold on` prima (e fuori) del ciclo. Il comando `hold on` è però indispensabile: se viene omesso si otterrà soltanto il grafico dell'ultima funzione (x^{10}).

5.4.2 Il comando `meshgrid`

Il comando `meshgrid` può essere utile in questi casi; tuttavia, poiché esso sarà fondamentale per i grafici di funzioni di due variabili, ci accontentiamo qui di mostrare come funziona per quello che ci interessa.

La sintassi del comando è `meshgrid(x,y)`, dove `x` e `y` sono vettori. Ad esempio

```
meshgrid(1:3,1:2)
```

genera la matrice

$$\begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix},$$

formata da due righe del vettore `x = [1 2 3]`. Il comando

```
[X,Y] = meshgrid(1:3,1:2)
```

produce le seguenti due matrici:

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} \quad \text{e} \quad Y = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{bmatrix}.$$

Le righe della matrice X (tante quante la lunghezza del vettore `y`) sono copie del vettore `x`; le colonne della matrice Y (tante quante la lunghezza del vettore `x`) sono copie del vettore `y`. La matrice potenza X^Y è allora quella matrice in cui ogni componente x della matrice X è elevata alla stessa componente y della matrice Y; in simboli $(X^Y)_{ij} = x_{ij}^{y_{ij}}$. Pauroso. In particolare la matrice potenza diventa

$$X^Y = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 9 \end{bmatrix}.$$

Questo viene implementato nello script seguente, in cui vogliamo rappresentare i grafici delle funzioni x^n, nell'intervallo $[0, 1]$, per diversi valori di n.

```
x = 0:0.01:1;                %vettore delle x
n = [1 2 3 4 5];             %vettore delle n
[X,N] = meshgrid(x,n);
plot(x,X.^N)                 %(x,X^1) (x,X^2) ...
grid on
legend('n=1','n=2','n=3','n=4','n=5',2)
```

5.4.3 Minimi e massimi

Per trovare i valori minimi e massimi di una funzione si usano i comandi `min` e `max`... con cautela. Consideriamo ad esempio una funzione f. Il comando `min(f)`, già visto nel Capitolo 1, cerca il più piccolo valore del vettore `f`, che dipende ovviamente dai punti `x` che sono stati usati per produrre `f`.

```
x = linspace(0,3);
f = x.*exp(-2*x);
plot(x,f)
grid on
%
M = max(f)                   %massimo, valore a cw; oppure, ; e display(M)
i = find(f == M);            %indice del massimo
xM = x(i)                    %punto di massimo, valore a cw
```

Ma, attenzione!

```
x = linspace(-2*pi,2*pi,100);
f = cos(x);
plot(x,f)
grid on
axis([min(x) max(x) -1 1])
m = min(f)                   %minimo, il valore a cw
i = find(f == m);            %indice del minimo
xm = x(i)                    %punto di minimo, valore a cw
```

In questo caso, a causa del basso valore `n=100`, troviamo solo un punto di minimo. Aumentando n (ad esempio `n=1000`) si trova anche l'altro.

Vedremo nella Sezione 8.3 un altro modo per calcolare minimi e massimi.

5.5 Complementi

5.5.1 Traslazioni e riscalamenti

Ovvero, come cambia il grafico di $f(x)$ considerando $f(x-a)$ (traslazione a destra se $a>0$, a sinistra se $a<0$) o $f(kx)$ (compressione se $k>1$, dilatazione se $0<k<1$). A questo proposito consideriamo la gaussiana

$$f(x) = e^{-x^2}.$$

Diamo inoltre la possibilità di inserire da *cw* i valori desiderati di a e di k; questo è fatto col comando `input`. L'istruzione condizionale `if` infine ricorda agli smemorati che tipo di operazioni stiamo facendo.

```
x = linspace(-3,3,1000);
y = exp(-x.^2);                          %una gaussiana
                                         %una traslazione orizzontale...
a = input('Digitare il valore di a:')    %input: per assegnare il valore
                                         %alla variabile da cw
z = exp(-(x-a).^2);
if a>0
      s='Traslazione a destra';          %definiamo una stringa di testo
elseif a<0
      s='Traslazione a sinistra';
else
      s='Nessuna traslazione';
end
                                         %...e un riscalamento
k = input('Digitare il valore di k:')    %input: per assegnare il valore
                                         %alla variabile da cw
w = exp(-(x.*k).^2);
if k>1
  t='Compressione orizzontale';
elseif k==1
  t='Nessun riscalamento';
elseif k>0 & k<1                         %prendiamo k>0...
  t='Dilatazione orizzontale';
else
  error('k deve essere maggiore di 0!')%error: errori a cw
end
                                         %arrivano i grafici
figure('name','Operazioni con le funzioni')  %name: nome finestra
subplot(3,1,1)                           %subplot: grafico 1...
plot(x,y,'k')
title('e^{-x^2}')
grid on
%
subplot(3,1,2)                           %... grafico 2...
plot(x,z,'r')
grid on
title([s,' a= ',num2str(a)])
%
subplot(3,1,3)                           %... e grafico 3.
plot(x,w,'b')
grid on
title([t,' k= ',num2str(k)])
```

Il comando `error` produce un errore a *cw* che riporta il messaggio specificato nella stringa di testo: da notare che il comando produce l'interruzione del codice in esecuzione. Si notino inoltre le istruzioni date al comando `figure` per far apparire il nome. Abbiamo dunque i comandi `title'ABC'` e `figure('name','ABC')` che attribuiscono rispettivamente dei nomi al plot o alla figura. Quali usare? Il primo compare nella stampa mentre il secondo no. Essi sono utili congiuntamente quando abbiamo diversi subplot (ognuno col suo titolo) in più figure, a ciascuna delle quali viene utile dare un nome.

5.5.2 Un'onda (quasi) quadra

In questo esempio generiamo automaticamente molti grafici in vari subplot. Si consideri la funzione

$$f(x) = e^{-x^n}.$$

Per n pari e grande la funzione restituisce una buona approssimazione di un'onda quadra.

```
x = linspace(-3,3);
n = [2 4 8 16 30 60];                  %valori del parametro n
L = length(n);                         %6
q = 1*(-1<=x & x<=1)+0.*(x<-1 & x>1);
for i = 1:L
    subplot(L/2-1,L/2,i)
    f = exp(-x.^n(i));
    plot(x,f,'b',x,q,'-.k')
    title(['n=', num2str(n(i))])
    %grid on
    xlabel('x')
    ylabel('y')
    axis([min(x) max(x) 0 1.1])
end
```

5.6 Esercizi

5.6.1 Si consideri la funzione $f(x) = 3(x-2)^3 + 6x^2 - 1$, definita tra 0 e 3. Creare il grafico di f; creare il grafico della funzione g, ottenuta da f traslando verticalmente il grafico di f di 4 e riscalandolo di $\alpha = 0.5$.

5.6.2 Si consideri la funzione $f(x) = \frac{\sin x}{x}$, definita nell'intervallo $[-2\pi, 2\pi]$. Rappresentare in tre grafici separati, ma all'interno della stessa finestra grafica (`subplot`) la funzione, la funzione traslata verticalmente di a e la funzione traslata orizzontalmente di b, dove a e b sono scelti a *cw* utilizzando il comando `input`.

5.6.3 Rappresentare graficamente la funzione f definita da:

$$f(x) = \begin{cases} e^x & \text{se } x \le 0, \\ 1 - x & \text{se } 0 < x \le 1, \\ \ln x & \text{se } x > 1. \end{cases}$$

5.6.4 Disegnare il grafico della funzione

$$f(x) = \begin{cases} -1 & \text{se } -2 \le x \le 0 \\ 1 & \text{se } 0 < x \le 2. \end{cases}$$

Se usiamo uno qualsiasi dei metodi introdotti nella Sezione 5.3 abbiamo una sorpresa: compare un tratto verticale passante per 0. Ovvio, MATLAB fa il suo dovere, dunque congiunge le coppie di valori. Ma in Analisi i tratti verticali non sono ammessi nei grafici di funzioni... Ovviare al problema usando il trucco introdotto nella Sezione 5.1 (`NaN`).

Stesso esercizio per la funzione $f(x) = \tan x$ in $(-\pi, \pi)$ (non è definita in $\pm\frac{\pi}{2}$).

Stesso esercizio per la funzione $f(x) = \frac{1}{\sqrt{x} - \sqrt[3]{x}}$ in $(0, 2)$ (non è definita in 1). Quest'ultimo esempio è subdolo: se si usa `x = linspace(0,2)` pare che $\lim_{x\to 0+} f(x) = 0$ ma se si raffina a `x = linspace(0,0.5)` o a `x = linspace(0,0.1)` si scopre che ...

5.6.5 Procedendo come nella Sezione 5.4 rappresentare in uno stesso piano i grafici di alcune funzioni esponenziali, al variare della base. Analogamente per i logaritmi.

5.6.6 Disegnare in quattro sottofinestre i grafici delle funzioni:

$$f(x) = e^{-x^2} \sin(k\pi x) \qquad k = 1, 2, 3, 4, \ x \in [-1, 1].$$

Utilizzare un ciclo `for`.

5.6.7 Familiarizzarsi col comando `input` scrivendo uno script che converte gradi in radianti. Ad esempio,

```
phi = input('Valore dell''angolo in gradi: ');
theta = (pi/180).*phi;
disp(['Valore dell''angolo in radianti: ' num2str(theta)])
```

Fare ora il contrario, cioè la conversione di radianti in gradi: facile. Un po' più difficile: nella conversione di radianti in gradi scrivere la misura dell'angolo modulo 360 (ad esempio, invece di 400 gradi si vorrebbe vedere 40, cioè $400 - 360$).

5.6.8 MATLAB permette di visualizzare molto bene come i polinomi di Taylor (MacLaurin) approssimano una funzione. A titolo di esempio, plottare in una stessa finestra grafica il grafico di $\sin x$ e dei suoi primi polinomi di MacLaurin x, $x - \frac{x^3}{3!}$, $x - \frac{x^3}{3!} + \frac{x^5}{5!}$.

5.7 Approfondimenti

5.7.1 Si vogliono disegnare i grafici delle funzioni $\sqrt[n]{x}$ nell'intervallo $[0, 1]$ per $n = 1, 2, 3, 4, 5$. E ci si riesce bene usando il comando `nthroot(x,n)` e `meshgrid`. Analogamente si possono disegnare i grafici di $\sqrt[n]{x}$ nell'intervallo $[-1, 1]$ per $n = 1, 3, 5$. Se però proviamo a cambiare comando e, nell'intervallo $[-1, 1]$, usiamo il comando `x.^(1/3)` invece di `nthroot(x,3)`, si ha una sorpresa notevole...

A *cw* si capisce che qualcosa non va...

Si ricorda che anche con lo script

```
x=linspace(-5,5);
plot(x,log(x))
```

si aveva ottenuto la stessa risposta...

Vedere le voci `power` e `log` nella documentazione di MATLAB (`doc power` o `doc log` a *cw*).

5.7.2 Come cambiare le unità di misura degli assi? Se abbiamo a che fare con funzioni trigonometriche ci piacerebbe di vedere qualche π sull'asse x.... Il semplice comando

```
x=linspace(0,2*pi);
plot(x,sin(x))
axis([0 2*pi -1 1])
```

non funziona: l'asse x è diviso con i numeri da 0 a 7. Per ovviare si usa il comando `XTick` (o `YTick` per l'asse y), aggiungendo alle tre righe di sopra

```
set(gca,'XTick',0:pi/2:2*pi)   %"tacche" da 0 a 2pi con passo pi/2
set(gca,'XTickLabel',{'0','pi/2','pi','3pi/2','2pi'})  %...
                               %... il nome delle "tacche"
```

Plottare i grafici di arcsin e di arccos.

5.7.3 Vi sono anche altri metodi per disegnare il grafico di una funzione definita a tratti oltre a quelli mostrati nella Sezione 5.3. Ad esempio, se si vuole evitare di concatenare gli intervalli, si può scrivere

```
x1 = -2:0.01:0;
x2 = 0:0.01:2;
plot(x1,2*x1,x2,x2.^2)
```

Si noti la differenza nei colori dei grafici. Usando le concatenazioni MATLAB interpreta f come un'unica funzione, mentre nel caso di sopra le funzioni sono due.

5.7.4 (Tic-toc) Quanto tempo ci mette MATLAB a svolgere una serie di comandi? Facile saperlo: si scrive `tic` quando si vuole far partire il cronometro e `toc` quando lo si vuole fermare. Naturalmente, dipende dal computer che si usa... e, di solito, i tempi di esecuzione variano anche per la stessa elaborazione. Vogliamo vedere in particolare quale dei procedimenti illustrati per calcolare una funzione definita a tratti è più rapido. Non consideriamo il metodo delle

concatenazioni, che è chiaramente una versione più elaborata del metodo di separazione. Inoltre, per evidenziare i tempi di esecuzione, aumentiamo i punti di campionamento. Scriviamo allora lo script seguente:

```
                          %separazione
tic                       %via!
x = -2:0.00001:0;
f1 = 2*x;
x = 0:0.00001:2;
f2 = x.^2;
Tsep = toc;               %stop!
                          %for-if
tic                       %via!
x = -2:0.00001:2;
for i=1:length(x)
  if x(i)<=0
   w(i) = 2*x(i);
  else
   w(i) = (x(i)).^2;
  end
end
Tfif = toc;               %stop!
                          %operatori relazionali
tic                       %via!
x = -2:0.00001:2;
z = 2*x.*(x>=-2 & x<=0)+x.^2.*(x>0 & x<=2);
Topr = toc;               %stop!
%
[Tsep Tfif Topr]          %i tre tempi
```

Come si vede anche eseguendo varie volte lo script, il metodo di separazione e quello che usa gli operatori relazionali sono equivalenti, mentre il metodo che usa una combinazione di `for` e `if` è decisamente più lento.

Capitolo 6

Grafici di funzioni: applicazioni

In questo capitolo continuiamo lo studio dei grafici di funzioni dando qualche applicazione.

6.1 Grafici in scala logaritmica

In Matlab è possibile creare grafici in scala (semi)logaritmica utilizzando i seguenti comandi:

`loglog(x,y)` :	entrambi gli assi in scala logaritmica;
`semilogx(x,y)` :	asse x in scala logaritmica;
`semilogy(x,y)` :	asse y in scala logaritmica.

Ricordiamo che fissare le scale logaritmiche corrisponde ad eseguire le trasformazioni:

$$x \to X = \log_{10} x, \quad y \to Y = \log_{10} y.$$

Ricordiamo come vengono modificati i grafici delle funzioni passando da una scala lineare ad una scala logaritmica.

- In scala logaritmica una funzione potenza, del tipo $y = bx^m$, ha per grafico la retta $Y = B + mX$, dove $B = \log_{10} b$. Infatti

$$Y = \log_{10} y = \log_{10}(bx^m) = \log_{10} b + m \log_{10} x = B + mX.$$

- In scala semilogaritmica (asse y logaritmico) una funzione esponenziale, del tipo $y = ba^{mx}$, ha per grafico la retta $Y = B + mAx$, dove $A = \log_{10} a$. Infatti

$$Y = \log_{10} y = \log_{10}(ba^{mx}) = \log_{10} b + mx \log_{10} a = Y = B + mAx.$$

Una tabella aiuta a ricordare:

scala	funzione	diventa	dove
`loglog(x,y)`	$y = bx^m$	$Y = B + mX$	$B = \log_{10} b$,
`semilogy(x,y)`	$y = ba^{mx}$	$Y = B + mAx$	$B = \log_{10} b$, $A = \log_{10} a$.

I grafici in scala logaritmica vengono utilizzati per rappresentare valori che si estendono in intervalli molto ampi o per mettere in evidenza andamenti particolari nella variazione dei dati. Va da sè che si useranno:

- scale logaritmiche per confrontare funzioni potenze,

- scale y-semi-logaritmiche per funzioni esponenziali.

6.2 Funzioni potenze ed esponenziali

Facciamo un esempio relativo a una scala logaritmica, rappresentando i grafici delle funzioni x^2, x^3, x^4.

```
x = linspace(0,3,100);
y1 = x.^2;
y2 = x.^3;
y3 = x.^4
%
subplot(2,1,1)
plot(x,y1,'k',x,y2,'r',x,y3,'b')
title('Scala lineare')
xlabel('x')
ylabel('y')
grid on
axis([min(x) max(x) 0 max(y3)])
legend('x^2','x^3','x^4',2)
%
subplot(2,1,2)
loglog(x,y1,'k',x,y2,'r',x,y3,'b')
title('Scala logaritmica')
xlabel('X')
ylabel('Y')
grid on
axis([min(x) max(x) 0 max(y3)])
legend('x^2','x^3','x^4',4)
```

Si noti l'uso del comando `axis` per rendere più leggibili i grafici. Provare a togliere entrambi i comandi dallo script.

Ed ora un esempio relativo a una scala semilogaritmica (rispetto ad y). In questo caso rappresentiamo i grafici delle funzioni 2^x, 3^x, 4^x.

```
x = linspace(0,3,100);
y1 = 2.^x;
y2 = 3.^x;
y3 = 4.^x;
%
subplot(2,1,1)
plot(x,y1,'k',x,y2,'r',x,y3,'b')
title('Scala lineare')
xlabel('x')
ylabel('y')
grid on
axis([min(x) max(x) 0 max(y3)])
legend('2^x','3^x','4^x',2)
%
subplot(2,1,2)
semilogy(x,y1,'k',x,y2,'r',x,y3,'b')
title('Scala semilogaritmica')
xlabel('x')
ylabel('Y')
grid on
axis([min(x) max(x) 0 max(y3)])
legend('2^x','3^x','4^x',2)
```

6.3 Complementi

6.3.1 Applicazione: raffreddamento di un corpo

Vogliamo studiare come varia la temperatura $T = T(t)$ di un corpo che si raffredda. Il corpo, posto in un ambiente a temperatura T_a, ha una temperatura $T_0 > T_a$ al tempo $t = 0$ e si raffredda secondo

la legge esponenziale di Newton:

$$T(t) = (T_0 - T_a)e^{-ct} + T_a.$$

Qui c è una costante caratteristica del corpo. Il grafico di T in funzione del tempo è quello di una funzione esponenziale asintotica al valore T_a per $t \to +\infty$. Per evidenziare meglio l'andamento esponenziale, rappresentiamo qui sotto la temperatura relativa T_r del corpo rispetto all'ambiente, cioè

$$T_r(t) = (T_0 - T_a)e^{-ct} = T(t) - T_a.$$

```
t = 0:0.01:200;                          %il tempo
T0 = 50;                                 %temperatura a t=0 [°C]
Ta = 23;                                 %temperatura ambiente [°C]
c = 0.014;
Tr= (T0-Ta)*exp(-c*t);                   %legge di raffreddamento di Newton
%                                        %Tr=T-Ta;
subplot(1,2,1)
plot(t,Tr,'r')
xlabel('tempo [s]')
ylabel('Temperatura relativa [°C]')
title('Scala lineare')
grid on
%
subplot(1,2,2)
semilogy(t,Tr,'r')
xlabel('tempo [s]')
ylabel('Temperatura relativa [°C]')
title('Scala semilogaritmica')
grid on
axis([min(t) max(t) min(Tr) max(Tr)])
```

6.3.2 Applicazione: la legge del potere fonoisolante

Nell'acustica applicata all'edilizia, la legge del potere fonoisolante consente di prevedere le proprietà di isolamento superficiale di una parete, note le caratteristiche della parete stessa. Il parametro che interessa è il *potere fonoisolante* R della parete, espresso in [dB] e definito (empiricamente...) da:

$$R = 10\log_{10}\left(\frac{m\pi f}{\rho c}\right)^2 \approx 20\log_{10}(mf) - 42,$$

dove m è la densità superficiale di massa della parete [kg/m^2], f la frequenza del suono [Hz], ρ la densità dell'aria [kg/m^3] e c la velocità del suono in aria [m/s]. Più alto è R, meglio la parete isola. In altre parole, per suoni di data frequenza, l'isolamento sarà tanto migliore quanto più grande sarà m, e questo miglioramento avviene in maniera logaritmica; inoltre, è più difficile isolare i suoni gravi che non quelli acuti.

Senza entrare nei dettagli di questa formula, vogliamo solo rappresentare graficamente l'andamento di R. Utilizzeremo una scala x-semilogaritmica rispetto alla frequenza f. Ovviamente prendiamo in considerazione solo le frequenze udibili, che variano da 20Hz a 20KHz.

```
f = 20:2000;                 %frequenza [Hz]
m = [0.8 2.3 11.2];          %massa per unità di area
[F,M] = meshgrid(f,m);
R = 20*log10(M.*F) - 42;
semilogx(f,R)
grid on
xlabel('frequenza f [Hz]')
ylabel('R [dB]')
title('Potere fonoisolante di una parete')
legend('cartongesso 0.8', 'calcestr. arm. 2.3', 'piombo 11.2',2)
```

6.3.3 La formula di Stirling

Riprendiamo ora un argomento relativo alle successioni numeriche ma che può essere ben rappresentato utilizzando le scale logaritmiche. La formula di Stirling

$$n! \sim n^n e^{-n}\sqrt{2\pi n}, \qquad \text{per } n \to \infty,$$

fornisce un asintotico di $n!$. Ciò vuol dire che, posto

$$a_n = \frac{\sqrt{2\pi n}\, n^n e^{-n}}{n!},$$

si ha

$$\lim_{n\to\infty} a_n = 1.$$

L'uso della scala semilogaritmica non è tanto di aiuto nella visualizzazione di a_n quanto in quella dell'errore $e_n = 1 - a_n$. I più curiosi potranno verificare che $e_n \sim e^{\frac{1}{12n}} - 1$.

```
n = 1:100;
a = (sqrt(2*pi.*n).*(n./exp(1)).^n)./cumprod(n);
lim = 1;
err = lim - a;
%
figure('name','Grafici')
subplot(2,1,1)
plot(n,a,'.b',n,lim,'.r')
axis([n(1) n(end) min(a) 1.01*lim])
grid on
title('Scala lineare')
legend('a_n','lim',4)
%
subplot(2,1,2)
semilogy(n,a,'.b',n,1,'.r')
axis([n(1) n(end) min(a) 1.01*lim])
grid on
title('Scala semilogaritmica')
legend('a_n','lim',4)
%
figure('name','Errori')
subplot(2,1,1)
plot(n,err,'.b')
grid on
title('Errore, scala lineare')
%
subplot(2,1,2)
semilogy(n,err,'.b')
grid on
title('Errore, scala semilogaritmica')
```

Si noti il riscalamento dell'asse delle ordinate per meglio visualizzare il limite (provare ad omettere il fattore di scala).

6.4 Esercizi

6.4.1 Uno studente abita in una stanza non riscaldata in cui la temperatura è di 15 °C. Pensa allora di scaldarsi almeno le mani chiedendo ad un vicino una pentola di acqua bollente (100 °C). Si rende conto ben presto che il metodo non funziona. Perché? Basandosi sulla Sezione 6.3.1, plottare nello stesso sistema di riferimento alcuni grafici della temperatura T_r al variare di $T_0 = 50, 60, \ldots, 100$ per rendersi conto di come il decadimento esponenziale porti rapidamente la temperatura relativa vicino alla temperatura ambiente. Calcolare numericamente per ognuno di questi casi il tempo al quale T_r raggiunge il valore 5.

Capitolo 7

Polinomi

Una classe particolarmente utile di funzioni è rappresentata dalle funzioni polinomiali; brevemente, i polinomi. MATLAB dispone per questi di alcuni comandi interessanti.

7.1 I comandi `polyval`, `roots` e `poly`

Per creare un polinomio e valutarlo su alcuni punti utilizziamo il comando `polyval`. La sintassi del comando è `polyval(p,x)`, dove `p` è il vettore costituito dai coefficienti (tutti, anche quelli nulli) del polinomio ordinati secondo le potenze decrescenti, e `x` i punti sui quali è valutato. Ad esempio, il polinomio

$$p(x) = x^3 + 2x^2 - x - 2 \tag{7.1}$$

è rappresentato dal vettore `[1 2 -1 -2]`; il polinomio $q(x) = x^2 - \sqrt{2}$ è rappresentato dal vettore `[1 0 -sqrt(2)]`. Come esempio consideriamo il polinomio (7.1). Nello script seguente calcoliamo anche gli zeri del polinomio col comando `roots`. La sua sintassi è `roots(p)`

```
x = -2:0.1:2;                  %punti su cui valutare il polinomio
p = [1 2 -1 -2];               %vettore dei coefficienti del polinomio
y = polyval(p,x);              %polyval
plot(x,y)
hold on
roots(p)                       %radici, a cw
plot(roots(p),0,'or')
grid on
```

Viceversa, qualora siano note le radici (gli zeri) di un polinomio, è possibile ricavarne i coefficienti con il comando `poly`:

```
p = [1 2 -1 -2]
r = roots(p)
coeff = poly(r)
```

Ovviamente in questo caso si ritrovano gli stessi coefficienti.

Il comando `roots` può riservare delle sorprese. Se infatti consideriamo il polinomio $x^2 + 1$, che non ha radici *reali*, MATLAB risponde che ha due radici ... *complesse*. Non approfondiamo questo argomento, materia di Analisi II.

7.2 Il comando `polyfit`

Supponiamo siano dati m punti $(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)$ nel piano. Cerchiamo il polinomio di grado n il cui grafico "passa il più possibile vicino" a questi punti. Si noti che n ed m possono essere diversi: ad esempio, se $n = 1$ il problema consiste nel cercare una retta (il grafico di un polinomio di grado 1) che meglio "approssima" questi punti. MATLAB ha un comando efficace per questo problema; l'approssimazione usata, che qui non spieghiamo, è quella dei *minimi quadrati*, e questo spiega le virgolette di sopra che vogliono dare un'idea intuitiva del problema.

Supporremo che le ascisse dei punti da approssimare siano tutte diverse tra loro. Si può dimostrare allora che per $n+1$ punti passa esattamente un polinomio di grado n (per due punti passa una sola retta), si veda la Sezione 7.6.

Dati due vettori `X` e `Y` aventi la stessa lunghezza, il comando `polyfit(X,Y,n)` calcola i coefficienti del polinomio P di grado `n` che meglio approssima i dati `(X,Y)` nel senso specificato sopra. Il comando restituisce i coefficienti di P (che sono $n+1$), ordinati secondo le potenze decrescenti.

```
X = [0 1 2 3];              %ascisse dei punti da approssimare
Y = [0 -1 0 1];             %ordinate dei punti da approssimare
P = polyfit(X,Y,3)          %coeff. del polinomio a cw
x = linspace(0,3);
p = polyval(P,x);           %creiamo il polinomio con polyval
plot(x,p,X,Y,'or')          %cerchietti rossi nei punti da approssimare
grid on
```

7.3 I comandi `conv` e `deconv`

Per effettuare il prodotto e la divisione tra polinomi, esistono rispettivamente i comandi `conv` e `deconv`. Consideriamo ad esempio i polinomi $A(x) = x^3$ e $B(x) = x^2 + 1$:

```
A=[1 0 0 0];        % A=x^3
B=[1 0 1];          % B=x^2+1
```

La sintassi del primo comando è `conv(p1,p2)` dove `p1` e `p2` sono i vettori che contengono i coefficienti dei polinomi di cui eseguire il prodotto. Il comando restituisce a sua volta un vettore contenente i coefficienti del polinomio risultante dal prodotto:

```
C=conv(A,B)         %C=[1 0 1 0 0 0] ovvero x^5+x^3
```

Il comando `deconv` esegue invece la divisione con la sintassi `[Q,R]=deconv(p1,p2)`, dove stavolta sono previsti *due output*: questo perché in generale la divisione tra polinomi (come ben noto...) fornisce anche un resto, oltre al solito quoziente.

```
[Q,R]=deconv(A,B)  %A = Q*B + R; Q=x e R=-x
```

Ricordiamo che la divisione tra polinomi è fondamentale quando si devono integrare delle funzioni razionali in cui il numeratore ha grado maggiore del denominatore.

7.4 Complementi

7.4.1 Applicazione: la legge di Hooke

Questo esempio è stato tratto da [3]. Consideriamo il problema dell'allungamento di una molla sottoposta ad un carico. Da un esperimento viene raccolta la seguente serie di dati, dove F indica la forza applicata e Δl l'allungamento prodotto:

F [N]	Δl [cm]
1.96	1.41
2.94	1.81
3.92	2.21
4.91	3.11
5.89	3.71
6.87	3.81
7.85	4.91
8.83	5.71

Assunta la legge di Hooke

$$F = k\Delta l$$

vogliamo calcolare la costante elastica k della molla. Il polinomio che interpola i dati sarà dunque di grado 1 e il suo grafico sarà una retta.

```
F = [1.96 2.94 3.92 4.91 5.89 6.87 7.85 8.83];%forze
L = [1.41 1.81 2.21 3.11 3.71 3.81 4.91 5.71];%allungamenti
Z = L.*10^(-2);                                %da cm a m
p = polyfit(F,Z,1);                            %interpolazione con una retta
x = linspace(1,10,1000);
y = polyval(p,x);
%
plot(F,Z,'.r',x,y,'b')
xlabel('Forza impressa [N]')
ylabel('Allungamento [m]')
legend('dati sperimentali','polinomio',2)
grid on
k = 1/(p(1))                                  %cost. elast. della molla a cw
```

7.5 Esercizi

7.5.1 Approssimare con un polinomio di grado $n = 3$ la seguente serie di dati:

$$\begin{aligned} x &= \begin{bmatrix} -3 & -1 & 0 & 2 & 6 \end{bmatrix} \\ y &= \begin{bmatrix} -1.5 & 0.36 & 4 & 5.1 & 7.5 \end{bmatrix} \end{aligned}$$

e rappresentare graficamente il risultato (sia la serie di valori che la curva che la approssima). Provare ad approssimare la stessa serie di dati con un polinomio 2 e poi con uno di grado 5; cosa succede?

7.6 Approfondimenti

7.6.1 MATLAB ha parecchi comandi specifici che riguardano i polinomi. Dare ad esempio un'occhiata al comando `polyder`.

7.6.2 Vogliamo dimostrare che per $n + 1$ punti del piano, aventi ascisse diverse tra loro, passa esattamente il grafico di un polinomio di grado n.

Invece di fare una dimostrazione generale, consideriamo prima il caso di 2 punti (x_1, y_1), (x_2, y_2), con $x_1 \neq x_2$. Cerchiamo a e b in modo che la retta $y = ax + b$ passi per questi punti. La condizione di passaggio è allora

$$\begin{cases} ax_1 + b = y_1 \\ ax_2 + b = y_2. \end{cases}$$

Troveremo a e b se il determinante della matrice dei coefficienti di a e b è diverso da 0. Ma esso è

$$\det \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \end{pmatrix} = x_1 - x_2,$$

che è diverso da 0 per ipotesi. Analogamente, dati tre punti (x_1, y_1), (x_2, y_2), (x_3, y_3), cerchiamo $y = ax^2 + bx + c$ che passi per questi punti. La condizione di passaggio è allora

$$\begin{cases} ax_1^2 + bx_1 + c = y_1 \\ ax_2^2 + bx_2 + c = y_2 \\ ax_3^2 + bx_3 + c = y_3, \end{cases}$$

il cui determinante vale $(x_1 - x_2)(x_1 - x_3)(x_2 - x_3)$, che è diverso da 0.

Il caso generale si tratta in maniera analoga; si è così ricondotti al *determinante di Vandermonde*:

$$\det \begin{pmatrix} x_1^n & x_1^{n-1} & \dots & 1 \\ x_2^n & x_2^{n-1} & \dots & 1 \\ \vdots & \vdots & & \vdots \\ x_n^n & x_n^{n-1} & \dots & 1 \end{pmatrix} = \prod_{1 \leq i < j \leq n} (x_i - x_j),$$

che è diverso da 0 per ipotesi.

Capitolo 8

Function(s)

Matlab utilizza due tipi di M-files: `script` e `function`. I primi, ormai li conosciamo; vediamo ora i secondi. Premettiamo però che, in questi appunti, l'uso delle `function` è molto modesto e in (quasi) tutti i casi può essere aggirato programmando diversamente. Si tratta però di un tipo fondamentale di file nel caso di programmazioni un po' più avanzate.

Ecco una breve lista di differenze tra i due.

- Gli `script` contengono semplicemente una lista di istruzioni, le `function` possono accettare dati in ingresso (input) e danno argomenti in uscita (output).
- L'unica avvertenza che si deve avere con gli `script` è di evitare alcuni caratteri nel loro nome. Le `function`, invece, sono più schizzinose:
  - il file dei comandi che costituiscono la function deve avere lo stesso nome della function, con estensione .m. Ad esempio la function *nomefun* deve essere salvata nel file nomefun.m;
  - la prima riga (a parte commenti) di tale file deve contenere la sintassi della function, ad esempio

    ```
    function [out1,out2...outn] = nomefun(in1,in2...inn)
    ```

Tutto questo è perfettamente incomprensibile. Vediamo allora un esempio...

8.1 La mia prima `function`

Supponiamo di dover usare più volte, in una simulazione, una funzione matematica un po' astrusa, ad esempio

$$f(x) = e^{-(x+3)^2} + e^{-x^2} + e^{-(x-3)^2}.$$

Naturalmente sappiamo che si tratta di una gaussiana e di due sue traslate, sommate tra loro; il grafico non presenta problemi. Per evitare di scrivere quell'espressione molte volte (o per evitare numerosi copia e incolla) possiamo aggiungerla alla libreria di funzioni di MATLAB. Ecco la ricetta.

- Apriamo un nuovo file che chiamiamo `fork.m` (ad esempio perché il grafico ci ricorda una forchetta; notare che MATLAB suggerisce il nome corretto del file quando lo salviamo, in quanto ha riconosciuto che si tratta di una function); questo file ha il seguente contenuto (abbiamo lasciato il nome commentato per ricordare che il nome del file deve essere quello della function; anche se questo non è necessario, faremo così anche in seguito):

  ```
  %fork.m
  function y = fork(x)
  y = exp(-x.^2) + exp(-(x+3).^2) + exp(-(x-3).^2);
  ```

  e basta. Notare che questo rientra nel quadro generale tracciato sopra: diamo le `x` (input) e la function ci ritorna le `y` (output); il nome del file è lo stesso di quello della function.
- Per usare la `function` ora definita creiamo ad esempio uno `script` (i nostri due tipi di M-file!) di questo tipo

```
x = linspace(-6,6,1000);
plot(x,fork(x))           %richiamiamo la function definita prima
```

Si noti che è indispensabile specificare la variabile nell'argomento della function: il solo comando `plot(x,fork)` non funziona.

Tutto qua. Notare che, quando viene richiamata la function, non conta il nome della variabile (`x`, qui sopra).

8.2 Zeri di una funzione: il comando `fzero`

Passiamo ora allo studio degli zeri di una funzione, ovvero dei punti x in cui

$$f(x) = 0.$$

Nel caso di polinomi, abbiamo visto che il semplice comando `roots` risolve il nostro problema. Non è così per funzioni più complesse. Chiamiamo `fun` la funzione di cui vogliamo calcolare gli zeri. Il comando `fzero` utilizzato qui sotto consente di calcolare uno zero di una funzione *nelle vicinanze di un punto* `x0`; è basato su un algoritmo numerico che sfrutta il metodo di bisezione. Risulta quindi conveniente individuare prima (ad esempio, sul grafico della funzione) l'intervallo in cui lo zero potrebbe trovarsi e poi utilizzare il comando `fzero`. Il comando non funziona se si omette di specificare il punto `x0`.

Il comando `fzero` ha varie sintassi:

(1) nel caso di una funzione interna di MATLAB, la sintassi è semplicemente

```
fzero(@fun,x0)
```

ad esempio, `fzero(@cos,1)` ci dà $\pi/2$;

(2) nel caso di funzioni non interne si usa la sintassi

```
fzero(@(x) fun(x), x0)
```

ad esempio, `fzero(@(x) exp(x)-2,0)`, oppure

```
fzero('fun(x)',x0)
```

ad esempio, `fzero('exp(x)-2',0)`; quest'ultimo è forse il metodo più semplice;

(3) infine, sempre nel caso di funzioni non interne ma "complicate", possiamo creare una `function` che definisca la nostra funzione e quindi richiamarla tramite la sintassi

```
fzero('fun',x0)
```

Si noti che, in questo caso, non bisogna specificare la variabile dell'argomento della funzione, diversamente dal caso visto nella Sezione 8.1.

Cominciamo facendo vedere un esempio di quest'ultimo metodo. Creiamo una `function` che definisca la funzione ("complicata")

$$r(x) = 1 - x^{1/2} + x^{2/3} - 2x^{3/4}, \tag{8.1}$$

cioè

```
%rad.m
function y = rad(x)
y = 1 - x.^(1/2) + x.^(2/3) - 2*x.^(3/4);
```

Apriamo ora un nuovo M-file e richiamiamo quello appena definito.

```
x = linspace(0,1);
r = rad(x);
plot(x,r)
grid on
hold on
z = fzero('rad',1);           %calcolo gli zeri
plot(z,0,'or')
```

Il comando `fzero` è anche utile per risolvere equazioni non omogenee. Nell'esempio seguente viene risolta l'equazione

$$xe^x = 1.$$

In questo esempio usiamo la sintassi (2) qui sopra.

```
x = linspace(-1,1,1000);
y = x.*exp(x)-1;
plot(x,y)
grid on
hold on
z = fzero('x.*exp(x)-1',-0.5)    %il valore a cw
plot(z,0,'or')
```

Inoltre `fzero` può essere utilizzato per trovare i punti di intersezione tra i grafici di due funzioni, come nell'esempio che segue. Le due funzioni in questione sono $\ln x$ e e^{-x}; chiaramente l'equazione $\ln x = e^{-x}$ si legge come

$$\ln x - e^{-x} = 0.$$

```
x = linspace(0.5,2.5,1000);
y = log(x);
z = exp(-x);
d = y-z;
plot(x,y,'k',x,z,'b',x,d,'r')
grid on
hold on
legend('ln x','e^{-x}','ln x - e^{-x}',4)
hold on
int = fzero('log(x)-exp(-x)',1.5)          %il valore a cw
plot(i,log(int),'ok')
```

Invece di specificare un punto vicino al quale cercare una funzione, possiamo specificare un intervallo tramite il comando

```
fzero('fun',[a,b])
```

dove `[a,b]` è l'intervallo in questione. Tipicamente (Teorema degli zeri per le funzioni continue) si inseriranno valori a e b in cui il segno di f è diverso, cioè $f(a) < 0 < f(b)$ o $f(b) < 0 < f(a)$. Questo è usato nell'esempio seguente, in cui la funzione ha due zeri.

```
x = linspace(0,3);
f = x.^3 - 5.*x + 3;
plot(x,f)
grid on
z1 = fzero('x.^3-5.*x +3',[0,1])
z2 = fzero('x.^3-5.*x +3',[1,3])
```

8.3 Minimi e massimi di una funzione: il comando `fminbnd`

Passiamo ora ai massimi e minimi di una funzione. La sintassi del comando `fminbnd` (*bounded minimization*), che consente il calcolo dei *punti di minimo locali* di una funzione, è analoga, anche se non uguale, a quella del comando `fzero` (si veda la Sezione 8.2). Essa è

```
fminbnd(fun,a,b)
```

dove `a`< `b` sono gli estremi dell'intervallo in cui si vuole minimizzare la funzione. Il comando

```
[xmin min] = fminbnd(fun,a,b)
```

restituisce il punto di minimo e il minimo. Il calcolo dei *massimi* locali può essere effettuato utilizzando lo stesso comando, ricordando che

$$\max(f) = -\min(-f).$$

Come funzione f consideriamo il polinomio

$$P(x) = x^5 - 3x^4 - 11x^3 + 27x^2 + 10x - 24,$$

e, poiché la sua espressione è abbastanza complicata, creiamo una function, cioè

```
%polinomio.m
function y = polinomio(x)
p=[1 -3 -11 27 10 -24];
y=polyval(p,x);
```

Per calcolarne i massimi definiamo una nuova `function` nel file polinomiom.m, corrispondente al polinomio $-P(x)$:

```
%polinomiom.m
function y = polinomiom(x)
y = - polinomio(x);
```

E' ora possibile calcolare minimi e massimi di P:

```
a = linspace(-2,4);
b = polinomio(a);
[xmin,min] = fminbnd('polinomio',-2,4)      %valori a cw
[xmax,max] = fminbnd('polinomiom',-2,4)     %valori a cw
plot(a,b,'k',xmin,min,'*r',xmax,-max,'*b')
grid on
```

Come evitare di creare una function *ad hoc*? Per le funzioni *built-in* di MATLAB, ovvero per tutte le funzioni elementari, basta premettere il simbolo `@`, senza dover creare una function. Ricordando il Capitolo 1, lo script per il calcolo dei punti di minimo e del minimo di $\cos x$ si riduce allora a

```
[xmin min] = fminbnd(@cos,-2*pi,2*pi)
```

In questo caso MATLAB ci dà il valore esatto 1. Si noti la differenza col comando `min` esaminato nella Sezione 5.4.3; in questo caso i comandi

```
x=linspace(0,2*pi);
min(cos(x))
```

danno il valore approssimato -0.9995.

In presenza di più punti di minimo, MATLAB si limita a segnalare il primo. Per funzioni non *built-in* di MATLAB, ad esempio per la funzione $-\cos x$, la sintassi è la seguente:

```
[xmin min] = fminbnd(@(x) -cos(x),-2*pi,2*pi)
```

Notare che non c'è la virgola tra `@(x)` e `-cos(x)`.

8.4 Complementi

8.4.1 Una `function` con due output e il ciclo `while`

Ora ci possiamo scatenare. Vogliamo dividere due numeri interi positivi a e b (con $a \geq b$) per ottenere un quoziente q e un resto $0 \leq r < b$:

$$a : b = q + r.$$

Abbiamo due input (a e b) e due output (q e r).

Per far questo abbiamo bisogno di un ciclo `while`.

- Creiamo la function. Nel file `divis.m` si crea la seguente function:

  ```
  %divis.m
  function [q,r] = divis(a,b)
  q = 1;                          %poiché a>=b
  r = a-b;
  while r >= b
   q = q+1;
   r = r-b;
  end
  ```

  Il procedimento è chiaro. Cominciamo col porre $q = 1$, in quanto $a \geq b$ e $r = a - b$. Se $r = a - b > b$, allora aumentiamo q di una unità; il nuovo resto sarà allora il precedente sottratto di b e si ripete il procedimento.

- Creiamo ora uno `script` che ci permetta di scegliere a *cw* il valore delle variabili di ingresso, restituire un errore se non è rispettata la condizione $a \geq b$ e poi di visualizzare quoziente e resto dell'operazione.

```
a = input('Inserire il valore di a:')
b = input('Inserire il valore di b:')
if a<b
    error('a deve essere maggiore di b!')
end
[q,r] = divis(a,b);
disp(['Il valore del quoziente è:',num2str(q)])
disp(['Il valore del resto è:',num2str(r)])
```

8.4.2 Applicazione: la legge dei gas perfetti

Come ulteriore esempio dell'uso di una `function` consideriamo l'equazione di stato dei gas perfetti, o brevemente la *legge dei gas perfetti*. Essa può essere scritta come

$$pV = m\bar{R}T.$$

Qui p è la pressione (in Pa), V il volume (in m^3), m la massa (in kg), T la temperatura (in K). Infine, $\bar{R}$ è la costante dei gas specifica (circa 286.7 J/(kg K) per l'aria), ovvero $\bar{R} = R/M$, dove R è la costante universale dei gas e M la massa molare.

Si crea inizialmente una `function` che consenta di calcolare il valore della pressione, noto il valore di tutte le altre variabili, e abbia, come secondo output, il valore della temperatura in K.

```
%gas_perfetti.m
function [p,TK]=gas_perfetti(m,T,V)
TK=T+273;
Rsp=286.7;
p=m.*Rsp.*TK./V;   %useremo input vettoriali; ecco perche' .* e ./
```

In un nuovo file.m è quindi possibile calcolare la pressione del gas (aria) per diversi valori del volume e della temperatura.

```
m=3;            %massa [kg]
T=[20 80 200];  %temperatura [°C]; ecco i dati vettoriali
V=[20:10:200];  %volume; ecco i dati vettoriali
%
for i=1:length(T)
    [p(i,:),TK(i)]=gas_perfetti(m,T(i),V);
end
%
plot(V,p)
grid on
xlabel('Volume [m^3]')
ylabel('Pressione [Pa]')
title('Legge dei gas perfetti')
legend('T=20°C','T=80°C','T=200°C') %o legend(num2str(T(1), ...)
```

Si noti il comando `p(i,:)`: nella riga `i` mettiamo tutti i valori dei volumi. Da notare l'utilizzo del ciclo `for` per memorizzare nelle righe della matrice p i valori della pressione per ciascuno dei tre valori di temperatura.

8.5 Esercizi

8.5.1 Mostrare la struttura della prima riga di un function M-file, chiamato pippo, che restituisce in uscita la variabile y date le variabili F e h.

[`function[y] = pippo(F,h)`]

8.5.2 Uno studente non brillantissimo non ha ben chiaro il grafico della funzione e^x e vuole studiarne gli zeri. Scrive allora `fzero(@exp, [0,1])`; che risposta ottiene? Rinuncia allora a specificare un intervallo e scrive solo `fzero(@exp,0)` e, miracolo!, ha trovato uno zero della funzione

esponenziale! O no? Convinto del suo risultato prova `fzero('1./x',1)` e trova pure uno zero della funzione $1/x$. Comincia a dubitare... Cosa c'è che non va?

8.5.3 Sappiamo (dovremmo sapere) che le funzioni $f(x) = \sin(x^2)$ e $g(x) = \sin(\sqrt{x})$ non sono periodiche. Plottare un grafico di f e g; calcolare di entrambe i primi cinque zeri positivi ($x_0 = 0 < x_1 < \ldots < x_5$) e calcolare la distanza di uno dal precedente ($x_1 - x_0, x_2 - x_1, \ldots, x_5 - x_4$).

8.5.4 Rappresentare le funzioni $f(x) = e^x$ e $g(x) = x + 3$, nell' intervallo $[0, 10]$. Per quale valore di x i grafici si intersecano? Rappresentarlo nel grafico insieme alle due funzioni.

8.5.5 Questo esercizio riguarda la risoluzione delle equazioni algebriche di secondo grado.

- Generare un vettore riga $v = [a\ b\ c]$ di tre numeri casuali compresi tra 0 e 10.
- Creare una function che accetti v in input e restituisca in output le soluzioni dell'equazione di secondo grado $ax^2 + bx + c = 0$. L'output darà le due soluzioni se il discriminante Δ è maggiore di zero, una se Δ è uguale a zero, o la stringa "'Delta negativo'" se Δ è minore di zero.
- Rappresentare in un grafico la funzione $f(x) = ax^2 + bx + c$ e i suoi zeri x_1 e x_2 nell'intervallo $[x_1 - 1, x_2 + 1]$.

8.5.6 Calcolare zeri, minimi e massimi della funzione $g(x) = x\sin(x)$ nell'intervallo $[-6, 1]$.

8.5.7 Creare una function che consenta di passare da gradi a radianti.

8.5.8 (Si veda [1]) Nell'Esercizio 1.4.8 abbiamo visto il comando `mean` che, nel caso di una matrice, restituisce un vettore le cui componenti sono le medie delle colonne. Creare una function che data una matrice ne calcoli il valore medio di tutte le sue componenti.

[Una matrice può essere pensata come un singolo lungo vettore... Oppure si può pensare di iterare il comando `mean`...]

8.5.9 (Si veda [1]) Creare una function che abbia come input un vettore di 10 elementi e restituisca in output il numero di elementi positivi del vettore.

[Un po' più difficile: che non restituisca solo il numero ma anche le componenti]

8.5.10 (Si veda [1]) Creare una function che calcoli il voto di laurea di uno studente del corso di laurea in Ingegneria Civile dell'Università di Legolandia (che prevede 19 esami), supponendo che faccia una ottima tesi di tipo sperimentale.

Il voto viene calcolato col seguente algoritmo: dall'elenco di tutti i voti (in trentesimi, 30 e lode vale 31), si toglie il peggiore (una sola volta) e si fa la media aritmetica dei rimanenti voti; il risultato si normalizza a 110, si aggiungono 6 punti per la tesi e si arrotonda per difetto. Ovviamente, se in questo modo si ottiene un voto finale che supera 110, il punteggio finale deve risultare 110.

8.5.11 Modificare leggermente lo script della Sezione 8.4.2 in modo da far variare la costante specifica dei gas invece, ad esempio, della temperatura o del volume.

8.6 Approfondimenti

8.6.1 Molte delle istruzioni e dei comandi che abbiamo impiegato fin qui sono definiti in MATLAB sotto forma di function. Analizzare ad esempio la struttura della function `linspace` (per visualizzarne il codice digitare a *cw* `edit linspace`).

8.6.2 Che differenza c'è tra il comando `fminbnd` e il comando `min`? Il primo consente lo studio dei minimi, locali o assoluti, come si è visto nella Sezione 8.3, il secondo considera solo il minimo assoluto. Infatti

```
a = linspace(-2,4);
b = polinomio(a);
min(b)
plot(a,b)
grid on
```

8.6.3 Bisogna fare attenzione nel calcolare in maniera ingenua gli zeri di una funzione utilizzando il vettore stesso della funzione. Ad esempio, i comandi

```
x=linspace(0,10);
f=x-5;
find(f==0)
```

non trovano che f si annulla (`linspace` divide in 100 intervalli: vedere a *cw*quali sono i punti in questione). Invece

```
x=0:0.1:10;
f=x-5;
find(f==0)
```

trova lo zero. Naturalmente questo procedimento non rimpiazza il comando `fzero` (che usa un algoritmo). In particolare

```
x=0:0.1:10;
f=exp(x)-1;
find(f==0)
```

non dà alcun risultato.

Capitolo 9

Il calcolo simbolico

In questo capitolo introduciamo le variabili simboliche, uno strumento che può rivelarsi utile per manipolare espressioni matematiche complesse al fine di svilupparle o fattorizzarle.

9.1 Le variabili simboliche

Il *Symbolic Toolbox* di MATLAB definisce un nuovo tipo di oggetti in MATLAB, chiamati oggetti simbolici, la cui classe è *sym*. Oggetti di classi diverse che abbiamo già incontrato sono gli oggetti numerici (classe *double array*, che sta per "Double precision floating point number array", lo standard di MATLAB per vettori e matrici) e le stringhe di caratteri (classe *char array*). Provare a digitare `help symbolic` per un elenco delle funzioni simboliche che Matlab mette a disposizione. Al solito tutto questo è perlomeno molto vago; lo scopo però è chiaro: poter effettuare calcoli simbolici e non solo numerici.

Facciamo qualche esempio per chiarire il minimo di informazione che ci servirà in seguito. Tutte le seguenti righe di codice sono da eseguirsi a *cw*, in una unica sessione di lavoro.

```
pi                %costante numerica: 3.1416, valore numerico approssimato
p = sym('pi')     %costante simbolica: pi, il vero numero irrazionale
                  %notare come pi e p vengono memorizzate nel workspace
a = sin(pi/4)     %0.7071, numerico, ma...
b = sin(p/4)      %1/2*2^(1/2), simbolico
```

Chiaro? In altre parole, quando utilizziamo le costanti simboliche esse non vengono calcolate (o approssimate) numericamente ma impiegate come quando usualmente facciamo i calcoli algebrici. Come fare però per passare da una variabile simbolica al suo valore numerico? Si usa `double` come segue.

```
c = double(b)     %da variabile simbolica a numerica: 0.7071
class(b)          %che tipo di oggetto è s? sym
class(c)          %... e n? double
```

Se vogliamo creare più variabili simboliche, ad esempio `x` e `y`, usiamo il comando `syms x y`. Con questa notazione il comando `x = sym('x')` equivale a `syms x`. Nel caso di costanti simboliche, tuttavia, solo la prima espressione è lecita.

9.2 Manipolare espressioni

Utilizzando le variabili simboliche possiamo eseguire operazioni come di solito facciamo con carta e penna... qui tutto di nuovo a *cw*. La sintassi per le variabili simboliche non richiede il punto necessario per le operazioni di moltiplicazione, divisione e potenza di vettori, anche se questo rallenta un pò le operazioni di calcolo (si veda l'Approfondimento 9.6.2). Per semplicità, nel calcolo simbolico ometteremo l'uso del punto che indica le operazioni vettoriali.

Il comando `expand` sviluppa le potenze, ma si usa anche con espressioni trigonometriche, esponenziali e logaritmiche.

```
syms x y
b = (x+y)^3
```

```
expand(b)              %voilà, il binomio è servito
%
s = sin(x+y)
expand(s)              %formula di addizione per il seno
%
expand(sin(2*x))       %...duplicazione;
p = exp(x+y)
expand(p)              %proprietà delle potenze
```

Il comando `factor` fattorizza gli elementi dell'espressione. Tra l'altro, se gli diamo in pasto un numero, ci restituisce gentilmente la sua fattorizzazione in numeri primi.

```
t = x^2+5*x+6          %oppure t = x.^2+5*x+6, notare il punto
factor(t)              %scrive il trinomio in fattori
%
factor(56)             %fattorizza in primi
%
m = (x^2-y^2)/(x^4-4*x^2*y^2)
pretty(m)              %pretty: visualizzazione "umana" a cw
                       %utile per controllare se l'input e' corretto
f = factor(a)
pretty(f)
```

Il comando `collect` raccoglie i coefficienti a fattore.

```
syms x y
p = x^2*(3*y -1) + y^2*(x-y)
collect(p,x)                %rispetto a x
collect(p,y)                %rispetto a y
```

Il comando `simplify` semplifica le espressioni

```
c = sin(x)^2+cos(x)^2
simplify(c)
```

Vediamo ora se riusciamo a fregare MATLAB. Noi sappiamo (dovremmo sapere) che $\sqrt{x^2} = |x|$.

```
simplify(sqrt(x^2))    %csgn? E' il segno di x ...
                       %... come numero complesso! (Analisi II)
syms x real            %ci ha fregato lui: dovevamo dirgli che x e' reale...
simplify(sqrt(x^2))    %ok
```

Inoltre

```
a = sym('14641')       %14641=11^4
l = log(a)
simplify(l)
```

Il comando `solve` permette di trovare gli zeri di una espressione simbolica.

```
syms a b c x
p = a*x^2+b*x+c
solve(p)                          %risolve simbolicamente p=0
```

9.3 Grafici e limiti

Le variabili simboliche possono essere utilizzate anche nell'ambito di funzioni di una variabile reale.

Il comando per rappresentare il grafico di una funzione simbolica `fplot` (nelle vecchie versioni di MATLAB è `ezplot`, acronimo di *Easy to use function plotter*). Per default questo comando rappresenta il grafico nell'intervallo $[-5, 5]$; per rappresentarlo in altri intervalli si veda l'esempio qui sotto.

Ripartiamo a *cw* con un polinomio.

```
syms a b c x
p = a*x^2+b*x+c
p1 = subs(p,[a b c],[2 4 3])    %comando subs introdotto sopra
fplot(p1)                       %grafico di una funzione simbolica
```

Puliamo tutto e ripartiamo con un altro esempio.

```
syms x
y = sin(x)/x
subs(y,x,[-3/2*pi:pi:3/2*pi])   %calcoliamo alcuni valori
fplot(y)
clf                             %puliamo la finestra grafica
                                %(clear current figure)
fplot(y,[0,2*pi])              %[xmin,xmax]
                                %passiamo ai limiti!
limit(y,x,0)                    %limite di y per x che tende a 0
%
f = 1/x
limit(f,x,0)                    %NaN, non esiste!
limit(f,x,0,'left')             %-Inf, limite sinistro
limit(f,x,0,'right')            %Inf, limite destro
limit(f,x,-inf)                 %0, ovviamente
```

Facciamo un ultimo esempio, utilizzando uno script che disegna un grafico e mostra a *cw* il valore di un limite. Notare che la funzione simbolica compare per default come titolo della figura.

```
syms x
y = sin(1/(log(x)));
ezplot(y,[0,3])                                 %grafico in [0,3]
grid on
%                                               %limite in 0+
l = limit(y,x,0,'right');
L = double(l);
disp(['Il limite per x che tende a 0^{+} è:',num2str(L)])
```

9.4 Complementi

9.4.1 Convergenza delle serie numeriche

Utilizzando il calcolo simbolico è possibile, in taluni (pochi) casi, calcolare il valore a cui converge una serie. A questo scopo si usa il comando `symsum`. Di nuovo tutto a *cw* in un'unica sessione.

Partiamo come al solito con la serie geometrica. Attenzione: nelle versioni meno recenti di MATLAB si può riscontrare qualche differenza con quanto riportato sotto.

```
syms q n                    %definiamo due variabili simboliche
s = symsum(q^n,n,0,inf)     %sommatoria simbolica
```

La risposta, ovvero

```
s =
piecewise([1 <= q, Inf], [abs(q) < 1, -1/(q - 1)])
```

ci ridà quello che sapevamo: la serie converge se $|q| < 1$ a $\frac{1}{1-q}$, diverge a $+\infty$ se $q \geq 1$. Vediamo qualche valore esplicito: per questo usiamo il comando `subs(s,old,new)` che rimpiazza `old` con `new` nell'espressione simbolica `s`.

```
subs(s,q,0.5)               %e va bene
subs(s,q,2)                 %pure bene
subs(s,q,-2)                %NaN, corretto, serie indeterminata
```

Consideriamo allora vari casi come segue.

```
sym('3')
s = symsum(3^n,n,0,inf)         %Inf!
%
sym('1/2')
s = symsum((1/2)^n,n,0,inf)     %2, ok.
```

Possiamo adesso verificare cosa sappiamo sulle serie armoniche generalizzate.

```
a = symsum(1/n,n,1,inf)        %serie armonica, divergente!
r = symsum(1/n^2,n,1,inf)      %chi l'avrebbe detto? (Analisi II)
s = symsum(1/sqrt(n),n,1,inf)
```

9.5 Esercizi

9.5.1 Calcolare la somma della serie $\sum_{n=1}^{\infty} \frac{(-1)^n}{n}$, già studiata numericamente nella Sezione 4.3.
$[-\ln 2]$

9.5.2 Calcolare simbolicamente i seguenti limiti e verificare il risultato tracciando il grafico della funzione corrispondente:

- $\lim_{x\to+\infty} \left(\sqrt{x^3-1} - \sqrt{x}\right)$
- $\lim_{x\to 0^-} \frac{x}{|x|}$
- $\lim_{x\to 0^-} e^{1/x}$.

9.5.3 Calcolare il valore di convergenza della serie a termini di segno variabile $\sum_{n=1}^{\infty}(-1)^n \frac{1}{n}$ e confrontarlo con il valore ottenibile numericamente mediante l'osservazione del grafico delle somme parziali.

9.6 Approfondimenti

9.6.1 Notare bene la differenza tra i seguenti comandi:

```
a=exp(1)
b=sym(e)
c=sym('e')
```

9.6.2 (Tic-toc) La sequenza di comandi `tic-toc`, introdotta nell'Approfondimento 9.6.2, può essere utile per capire quanto, nel simbolico, la notazione vettoriale incrementi la velocità di esecuzione. Ad esempio, si confronti

```
syms x
tic                                     %via
fplot('x^3 - exp(-x^2) + sin(x^2)')   %notazione simbolica
tempolento = toc                        %stop!
```

con

```
tic                                     %via
fplot('x.^3 - exp(-x.^2) + sin(x.^2)')%notazione vettoriale
temposvelto = toc                       %stop!
```

Capitolo 10

Derivate, numeriche e simboliche

Utilizzando MATLAB è possibile calcolare la derivata di una qualsiasi funzione, sia numericamente che ricorrendo al calcolo simbolico.

10.1 La derivata numerica

Tutto parte dalla definizione di derivata:

$$Df(x) = \lim_{y \to x} \frac{f(x) - f(y)}{x - y}. \tag{10.1}$$

Un rapido controllo mostra che, a parte le variabili diverse dal solito, è proprio la derivata.

10.1.1 Difficoltà numeriche (e un'idea dell'Analisi Numerica)

Da un punto di vista numerico sorgono due problemi che ora descriviamo e che cerchiamo di risolvere in modo intuitivo. La trattazione rigorosa di questi argomenti è l'argomento dell'Analisi Numerica.

Primo, l'operazione di limite non ha senso; pertanto dobbiamo cercare una qualche espressione che approssimi $Df(x)$. Secondo, sia $x = (x_1, x_2, \ldots, x_n)$ che $Df = \Big(Df(x_1), Df(x_2), \ldots, Df(x_n)\Big)$ sono vettori (usiamo la notazione ad indici per chiarezza).

L'idea più semplice potrebbe essere quella di approssimare la generica componente

$$Df_i = Df(x_i), \quad i = 1, 2, \ldots, n$$

con

$$D\tilde{f}_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}, \quad i = 2, \ldots, n.$$

Si noti che il vettore Df ha n componenti mentre $D\tilde{f}$ ne ha $n-1$. Dobbiamo rinunciare ad una componente, che non è un grave problema se n è abbastanza grande. Inoltre, si noti che $\frac{f(x_i)-f(x_{i-1})}{x_i-x_{i-1}}$ può essere preso anche come una approssimazione di Df_{i-1}. In conclusione, approssimeremo le $n-1$ coppie (stiamo pensando al grafico...)

$$\big(x_1, Df(x_1)\big), \ldots, \big(x_{n-1}, Df(x_{n-1})\big)$$

con le $n-1$ coppie

$$\left(x_1, \frac{f(x_2) - f(x_1)}{x_2 - x_1}\right), \ldots, \left(x_{n-1}, \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}\right).$$

In altri termini,

$$Df(x_i) \simeq \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}. \tag{10.2}$$

Se in particolare il vettore x è formato da componenti equispaziate, con $x_{i+1} - x_i = \Delta$, allora la (10.2) si può anche scrivere

$$Df(x_i) \simeq \frac{f(x_i + \Delta) - f(x_i)}{\Delta}. \tag{10.3}$$

10.1.2 Implementazione

La derivazione numerica in MATLAB è realizzata utilizzando la funzione `diff(x)` che calcola le differenze tra valori adiacenti del vettore `x`, generando un nuovo vettore con un valore in meno (ovviamente). In simboli (usiamo di nuovo la notazione ad indici per chiarezza)

$$x = \Big(x_1, x_2, \dots x_n\Big) \quad \Rightarrow \quad \mathrm{diff}(x) = \Big(x_2 - x_1, x_3 - x_2, \dots, x_n - x_{n-1}\Big).$$

Per familiarizzarci col comando `diff` partiamo a *cw*.

```
x = [1 4 2 6]
diff(x)                          %[3 -2 4], ovvio no?
```

E ora possiamo partire con uno script in cui confrontiamo la derivata esatta della funzione $f(x) = \sin x$ con quella ottenuta numericamente.

```
n = 50;                                  %n piccolo per vedere la differenza
x = linspace(0,2*pi,n);                  %vettore di n valori
f = sin(x);                              %vettore di n valori
Dfex = cos(x);                           %derivata esatta
plot(x,f,'k',x,Dfex,'b')
grid on
%
hold on                                  %arriva la derivata numerica
dx = diff(x);                            %vettore di n-1 valori!
df = diff(f);                            %vettore di n-1 valori!
Dfapp = df./dx;                          %derivata numerica
plot(x(1:n-1),Dfapp,'r')
legend('f','Df esatta','Df approssimata',3)
axis([0 2*pi min(f) max(f)])
hold off
title('Grafici di f(x) = sin(x) e della sua derivata')
%
err = max(abs(Dfex(1:n-1)-Dfapp));       %calcolo dell'errore
disp(['L''errore massimo è: ' num2str(err)])
```

Notare il doppio apice per rendere l'apostrofo (altrimenti MATLAB capisce che è un apice).

10.2 La derivata simbolica

La derivata di una funzione può anche essere calcolata simbolicamente, facendo uso delle funzioni definite nel *toolbox* simbolico di Matlab. Il calcolo della derivata simbolica di una funzione è particolarmente vantaggioso per funzioni di elevata complessità. Il comando è, di nuovo, `diff`. Notare l'uso diverso di tale comando nel numerico e nel simbolico!

```
syms x
f = log((x^2+4)/(x^2-4));
disp('Data f: ')
pretty(f)
Df = diff(f);
sDf = simplify(Df);
disp('Df è: ')
pretty(sDf)
```

Il comando `diff` funziona anche in presenza di più variabili simboliche. In tal caso bisogna specificare rispetto a quale variabile si deriva. Si possono inoltre calcolare simbolicamente le derivate seconde, terze...

```
syms x y
f = x^y;
pretty(f)                    %si ricorda con Pretty Woman...
%
Dfx = diff(f,'x');           %derivo rispetto ad x
sDfx = simplify(Dfx);
disp('D_xf: ')
```

```
pretty(sDfx)
%
Dfy = diff(f,'y');          %derivo rispetto ad y
sDfy = simplify(Dfy);
disp('D_yf: ')
pretty(sDfy)
%
D2fx = diff(f,'x',2);       %derivata seconda rispetto ad x
sD2fx = simplify(D2fx);
disp('D2_xf: ')
pretty(sD2fx)
```

Infine un esempio riassuntivo. La funzione è

$$f(x) = \frac{a}{5 + 4\cos(ax)}$$

e dipende dal parametro a. Lo script ci chiede dapprima di inserire il valore di a (e questo lo sappiamo fare). Poi rappresentiamo graficamente f, la sua derivata f' e calcoliamo il $\lim_{x\to 0} f(x)$.

```
syms a x
f = a/(5+4*cos(a*x));
i = input('Inserire un valore numerico per a:')
y = subs(f,a,i);
figure('Name','Grafico di una funzione simbolica')
ezplot(y,[-6,6])
grid on
%
l = limit(y,x,0);
disp('Il limite per x che tende a 0 è:')
double(l)
%
Dy = diff(y);
disp('La derivata prima della funzione f è:')
pretty(Dy)
figure('Name','Grafico della derivata prima')
ezplot(Dy,[-6,6])
grid on
```

In merito allo script precedente, interrogare MATLAB sul limite per x che tende a $+\infty$...

10.3 Complementi

10.3.1 La derivata seconda

Nel comando `diff(x,n)` il secondo argomento `n`, facoltativo se `n` vale 1, specifica il numero di volte in cui la funzione deve essere applicata ricorsivamente. Ad esempio, `diff(x,2) = diff(diff(x))`:

$$\begin{aligned} \mathrm{diff}(x,2) &= \Big(x_3 - x_2 - \big(x_2 - x_1\big), \ldots, x_n - x_{n-1} - \big(x_{n-1} - x_{n-2}\big)\Big) \\ &= \Big(x_3 - 2x_2 + x_1, \ldots, x_n - 2x_{n-1} + x_{n-2}\Big). \end{aligned}$$

Ad esempio,

```
x = [1 4 2 6]
diff(x)                     %[3 -2 4]
diff(x,2)                   %[-5 6]
```

E' questo il comando che serve per il calcolo numerico della derivata seconda (derivata della derivata prima...). Si noti che conviene fare un'altra approssimazione. Infatti, se applichiamo (10.3)

$$\begin{aligned} D^2 f(x_i) &\simeq \frac{Df(x_i+\Delta) - Df(x_i)}{\Delta} \\ &\simeq \frac{Df(x_{i+1}+\Delta) - Df(x_{i+1})}{\Delta} - \frac{Df(x_i+\Delta) - Df(x_i)}{\Delta} \\ &= \frac{Df(x_i+2\Delta) - 2f(x_i+\Delta) + f(x_i)}{\Delta^2}. \end{aligned}$$

A numeratore compare proprio la componente i del vettore $\mathrm{diff}(f,2)$.

Studiamo ora la derivata prima e seconda della funzione $f(x) = x\sin(x^2)$.

```
x = linspace(-2,2,100);
f = x.*sin(x.^2);
%
subplot(3,1,1)
plot(x,f,'k')
title('La funzione f(x) = x sin(x^2)')
grid on
%                                    %calcolo della derivata prima
dx = diff(x);                        %vettore di 99 valori!
df = diff(f);
Dfapp = df./dx;
L=length(Dfapp);
%
subplot(3,1,2)
plot(x(1:end-1),Dfapp,'r')
grid on
title('Df')
hold on
%                                   %calcolo della derivata seconda
dx2 = dx.^2;
df2 = diff(f,2);                    %vettore di 98 valori!
D2fapp = df2./dx2(1:end-1);
L2 = length(D2fapp);
subplot(3,1,3)
plot(x(1:end-2),D2fapp,'g')
grid on
title('D^2f')
```

Si noti come i grafici di f' e f'' terminino prima di quello di f...

10.4 Esercizi

10.4.1 Cosa cambia se si sostituisce `plot(x(1:n-1),Dfapp,'r')` con `plot(x(2:n),Dfapp,'r')` nello script della Sezione 10.1.2? Si provi inoltre a modificare il valore della variabile `n`; come cambia l'errore commesso?

10.4.2 Usando il *toolbox* simbolico, tracciare il grafico delle seguenti funzioni, assieme a quello delle loro derivate numeriche prime e seconde:

- $f(x) = |\sin(x)|^x$
- $f(x) = 2^{\sqrt{x}} + x^{\sqrt{x}}$
- $f(x) = e^{\sin(x)} + \sin(e^x)$.

10.4.3 (Teorema di Lagrange) Creare uno script che, data una funzione f definita in $[a,b]$: crea la funzione g della dimostrazione del teorema di Lagrange; plotta i grafici di f, g e della retta passante per $(a,f(a))$, $(b,f(b))$; calcola g'; trova uno zero x_0 di g'; disegna il grafico della retta tangente al grafico di f per $(x_0,f(x_0))$.

10.5 Approfondimenti

10.5.1 Il simbolico è fragile. Notare infatti come il codice ingenuo

```
syms x
e=exp(1)
f = e^(-(x^2))
```

lo distrugga, mentre il codice corretto

```
syms x
f = exp(-(x^2))
```

lo preservi.

Capitolo 11

Integrazione numerica e simbolica

In MATLAB è possibile calcolare l'integrale di una funzione sia numericamente che simbolicamente. In questo capitolo li facciamo vedere entrambi.

11.1 Integrazione numerica con il metodo dei trapezi (un metodo di Analisi Numerica...)

Esistono numerosi metodi per il calcolo approssimato di integrali, e qui si entra nel campo dell'Analisi Numerica. Operativamente, in MATLAB Il comando `trapz` è forse quello più semplice per quanto riguarda l'integrazione. Per calcolare

$$\int_0^1 x^2\,dx = \frac{1}{3}$$

si fa così:

```
x = linspace(0,1);
f = x.^2;
trapz(x,f)              %0.3334 a cw;
```

Cosa fa il comando `trapz`? Usa il metodo dei trapezi (o di Newton-Cotes) per il calcolo approssimato dell'integrale. L'idea è di approssimare l'area del sottografico di una funzione $f : [a, b] \to \mathbb{R}$ con quella del trapezio rettangolo avente per altezza il segmento $[a, b]$ e basi i segmenti che congiungono i punti $(a, 0)$ con $(a, f(a))$ e $(b, 0)$ con $(b, f(b))$: si veda la figura 11.1.

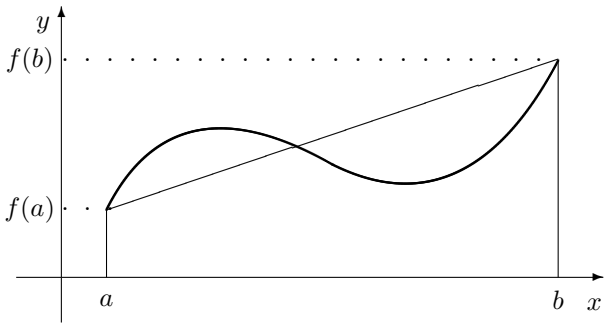

Figura 11.1: Il metodo dei trapezi.

In altre parole,

$$\int_a^b f(x)\,dx \simeq (b-a)\frac{f(a)+f(b)}{2}. \tag{11.1}$$

Naturalmente si sta intendendo area nel senso di "area col segno": le regioni sotto l'asse x danno contribuiti negativi.

Poiché l'approssimazione (11.1) sarebbe troppo rozza, si divide l'intervallo $[a, b]$ in n sottointervalli di estremi $a = x_0 < x_1 < \ldots < x_n = b$, ad esempio con $x_{i+1} - x_i = \Delta$. Si applica quindi l'approssimazione di sopra in ogni sottointervallo. Pertanto

$$\begin{aligned}\int_a^b f(x)\,dx &= \sum_{i=0}^{n}\int_{x_i}^{x_{i+1}} f(x)\,dx \\ &\simeq \frac{\Delta}{2}\sum_{i=0}^{n}\Big(f(x_i)+f(x_{i+1})\Big) \\ &= \frac{\Delta}{2}\Big(f(x_0)+2f(x_1)+\ldots+2f(x_{n-1})+f(x_n)\Big).\end{aligned}$$

A titolo di esempio, implementiamo direttamente questo metodo e poi confrontiamo il risultato con quello ottenuto usando semplicemente il comando `trapz`. Creiamo la `function` seguente:

```
%funt.m
function y = funt(x)
y = x.^2.*(sin(x)).^2;
```

Quindi, creiamo uno script:

```
a = 0;
b = 10;
n = 100;
dx = (b-a)/n;
x = [a:dx:b];
f = funt(x);
plot(x,f)
grid on
hold on
                                        %met. trap. implementato direttamente:
A = dx*(f(1)+2*sum(f(2:end-1))+f(end))/2;
Atrapz = trapz(x,f);                    %usando trapz
disp(['A= ',num2str(A)])
disp(['Atrapz= ',num2str(Atrapz)])
```

Chiarito tutto questo, a scanso di equivoci precisiamo che se si desidera integrare numericamente una funzione non troppo complicata, si usa direttamente il comando `trapz`. Fine.

11.2 Integrazione simbolica

Utilizzando il *toolbox* simbolico di Matlab è possibile anche calcolare simbolicamente primitive di funzioni ed integrali definiti. Tutto quanto segue a *cw*; i comandi sono estremamente intuitivi.

Calcoliamo due integrali indefiniti (cioè due primitive)... ricordando che MATLAB, tra le infinite primitive $F + C$ di una funzione f definita in un intervallo, ci calcola quella in cui $C = 0$.

```
syms x
f = log(x)
int(f)                  %x*log(x)-x
%
syms n
p = x^n
int(p,x)                %specifichiamo rispetto a quale variabile integrare
```

Passiamo poi agli integrali definiti.

```
syms x
f = x.^2
int(f,0,1)
```

Infine, MATLAB è di aiuto anche per gli integrali generalizzati.

```
syms x
f = 1/x;
a = int(f,x,1,Inf)          %Inf
g = 1/x^2;
b = int(g,x,1,Inf)          %1
```

Si noti che nelle righe qui sopra potevamo omettere di specificare che integravamo rispetto a x, e scrivere più brevemente, ad esempio,

```
a = int(f,1,Inf)
```

11.3 Complementi

11.3.1 Ripasso: integrazione numerica tramite punti casuali

Nel seguente esempio ripassiamo la costruzione dell'integrale di Riemann di una funzione f definita nell'intervallo $[a, b]$. Lo scopo di questo esempio *non è* pertanto quello di fornire una tecnica efficiente di integrazione con MATLAB, quanto quello di rivedere numericamente quanto fatto teoricamente. L'integrazione numerica necessita di un solo comando, `trapz`, che è stato introdotto nella Sezione 11.1.

L'intervallo $[a, b]$ è diviso in n intervalli tramite $n + 1$ punti x_i, $i = 1, \ldots, n + 1$. In ogni sottointervallo $[x_i, x_{i+1}]$, $i = 1, \ldots, n$, si sceglie un punto c_i e si costruisce la somma di Riemann

$$\frac{b-a}{n} \sum_{i=1}^{n} f(c_i).$$

Approssimiamo quindi l'integrale con la somma di Riemann; in un caso seguiamo la costruzione di Riemann, scegliendo c_i come punti casuali. In altri due casi scegliamo i c_i rispettivamente come gli estremi sinistri e destri degli intervalli $[x_i, x_{i+1}]$.

```
a = 0;
b = 3;
n = 20;
dx = (b-a)/n;
x = [a:dx:b];
f = exp(x);
plot(x,f,'k','linewidth',2)
grid on
hold on
%
xleft = [a:dx:b-dx];                    %estremi sinistri degli interv.
xright = [a+dx:dx:b];                   %estremi destri degli interv.
xrand = xleft+rand(1,length(x)-1).*dx;  %punti casuali negli interv.
frand = exp(xrand);                     %f valutata sui punti casuali
Srand = sum(frand)*dx;                  %somma di Riemann:
bar(xleft+dx/2,frand,1,'edgecolor','r','facecolor','none')
%                                       %... grafico a barre
fleft = exp(xleft);                     %f valutata negli estr. sinistri
Sleft = sum(fleft)*dx;                  %somma di Riemann relativa
%
fright = exp(xright);                   %f valutata negli estr. destri
Sright = sum(fright)*dx;                %somma di Riemann relativa
%
disp(['Stima con punti casuali: ',num2str(Srand)])
disp(['Stima con estremi sinistri: ',num2str(Sleft)])
disp(['Stima con estremi destri: ',num2str(Sright)])
```

Abbiamo utilizzato nello script il comando `bar`. Con la sintassi `bar(x,y)` il comando disegna colonne alte `y(i)` centrate sopra i punti `x(i)`. Ecco perché abbiamo dovuto incrementare di `dx/2` le `x`. In riferimento al comando usato nello script, `1` specifica la larghezza delle colonne (default 0.8, un po' spaziate), `'edgecolor'` il colore dei lati delle colonne e `'facecolor'` il colore dell'interno delle colonne. Provare a cambiare questi parametri per rendersi conto di come funziona il comando.

Al crescere di n sia l'approssimazione con punti casuali (che ovviamente cambia ogni volta che si ripete la simulazione) che le altre due hanno valori molto vicini, che approssimano il valore esatto dell'integrale.

11.3.2 La funzione integrale

Ci occupiamo ora della funzione integrale. Data una funzione $f : [a, b] \to \mathbb{R}$ la funzione integrale F è definita in $[a, b]$ da

$$F(x) = \int_a^x f(x)\,dx. \tag{11.2}$$

Creiamo una `function` con due output che calcoli la funzione integrale utilizzando come punti di campionatura, nel calcolo dell'integrale, i punti medi di ogni sottointervallo.

```
%Fint.m
function [g,x] = Fint(fun,a,b,n)
fun = fcnchk(fun)
dx = (b-a)/n;
x = [a:dx:b];
xm = [a+dx/2:dx:b-dx/2];
y = fun(xm);
g = [0 cumsum(y)*dx];
```

A cosa serve il misterioso comando `fcnchk` (for "function check")? Senza l'utilizzo del comando `fcnchk` potremmo applicare il file `function` qui sopra solo a

- funzioni già predefinite in MATLAB (*built-in*, ad esempio `exp`, cioè e^x);
- funzioni definite in un altro M-file.

In particolare, dal momento che la funzione e^{-x^2}, della quale calcoliamo la primitiva qui sotto, non è predefinita in MATLAB, dovremmo creare un breve script in cui la definiamo: una perdita di tempo.

Con il comando `fcnchk`, invece, non solo possiamo utilizzare i due modi precedenti per definire una funzione, ma soprattutto la possiamo definire direttamente nell'argomento della `Fint`, come facciamo qui sotto.

```
a = 0;
b = 3;
n = 1000;
[F,x] = Fint('exp(-x.^2)',a,b,n);          %grazie a fcnchk!
f = exp(-x.^2);
plot(x,f,'k',x,F,'r','linewidth',2)
legend('f(x) = e^{-x^2}', 'F = primitiva')
grid on
```

11.3.3 Integrali ellittici e altri loro amici

Calcolando delle primitive con MATLAB senza un po' di cognizione ci possono essere delle sorprese. Ad esempio abbiamo studiato che la funzione e^{x^2} non ammette una primitiva esprimibile in termini di funzioni elementari. Vediamo cosa ci dice MATLAB...

```
syms x
f = exp(x^2);
i = int(f)             %1/2*i*pi^(1/2)*erf(i*x) ?
```

E qui arriva l'identità immaginaria i e la funzione erf che non è tra quelle che noi chiamiamo elementari. Proviamo con un'altra che come la precedente non ha una primitiva esprimibile in termini di funzioni elementari.

```
syms x
g = sin(x^2);
h = int(g)         %1/2*2^(1/2)*pi^(1/2)*FresnelS(2^(1/2)/pi^(1/2)*x)?
```

E qui arrivano gli integrali di Fresnel..., insomma, MATLAB conosce più matematica di quella che conosciamo noi.

Abbiamo inoltre visto che una classe particolarmente importante di integrali che sono risolubili solo numericamente sono gli integrali ellittici. L'integrale ellittico di prima specie, ad esempio, è

$$\int \frac{1}{\sqrt{1 - k^2 \sin^2 x}}\,dx\,,$$

con $k \in (0,1)$. Qui conviene creare uno script a causa dei molti comandi. Vediamo...

```
syms x
k = input('Inserire un valore compreso tra 0 ed 1:')
if k<0 | k>1                              %| = oppure
 error('k deve essere compreso tra 0 ed 1!')
end
f = 1/(sqrt(1-k^2*(sin(x))^2));
integr_indef = int(f)
integr_def = int(f,0,1)
```

Il risultato, a parte funzioni elementari, coinvolge anche un termine `EllipticF(sin(1),k)`. Non ci dilunghiamo su quest'argomento; è sufficiente ricordare che, all'occorrenza, MATLAB può dare anche un aiuto teorico. Naturalmente, il comando `trapz` permette di valutare subito l'integrale definito.

11.4 Esercizi

11.4.1 In riferimento alla Sezione 11.3.1, modificare lo script in modo da scegliere il punto medio di ogni sottointervallo come punto di campionamento.

11.4.2 Calcolare numericamente i seguenti integrali definiti:

- $\displaystyle\int_{-3}^{2} \ln(x^2-1)\,dx$
- $\displaystyle\int_{-5}^{-4} \frac{5-2x}{9-x^2}\,dx.$

11.4.3 Il solito Artemio è entusiasta del comando `trapz` che gli permette di calcolare numericamente una gran quantità di integrali definiti (che di solito non gli vengono mai). Ha visto a lezione che l'area sotto la parabola, nell'intervallo $[0,1]$, vale $1/3$. Ne vuole una conferma e scrive

```
x=linspace(0,1);
trapz(x.^2,x)
```

Cosa c'è che non va? Che area ha calcolato Artemio?

11.4.4 Calcolare una primitiva delle seguenti funzioni e tracciarne il grafico insieme a quello della funzione integranda:

- $f(x) = xe^{x^2}$
- $f(x) = \sin(x)\cos(x)$
- $f(x) = \dfrac{x+1}{x^2+x+1}.$

11.5 Approfondimenti

11.5.1 Uno studente un po' ingenuo vuole calcolare l'area sottesa dalla funzione x nell'intervallo $[0,1]$... e prepara lo script

```
x=0:0.01:1;
trapz(x)
```

Il risultato che trova lo stupisce! Il punto è che:

> Z = TRAPZ(Y) computes an approximation of the integral of Y via the trapezoidal method (with unit spacing). To compute the integral for spacing different from one, multiply Z by the spacing increment.

In altre parole, ha sommato cento volte lo stesso integrale!

Bibliografia

[1] A. Colesanti, S. Dolfi, E. Rubei e P. Salani. *Dispense di Laboratorio di Matematica I, a.a 2007-2008.* Dipartimento di Matematica, Università degli Studi di Firenze

[2] G. Jensen. *Using Matlab in Calculus.* Prentice Hall, 2000.

[3] W. J. Palm III. *MatLab Un'introduzione per gli ingegneri.* McGraw-Hill, 2011.

[4] A. Quarteroni e F. Saleri. *Calcolo scientifico. Esercizi e problemi risolti con MATLAB e Octave.* Springer, 2008.