Analisi Matematica
Esercitazioni con MATLAB

Nicola Cavallini
nicola_cavallini@yahoo.it

Andrea Corli
crl@unife.it

21 novembre 2009

Indice

Introduzione

I piani di studi delle lauree specialistiche in Ingegneria Civile, Ingegneria per l'Ambiente e il Territorio, Ingegneria Meccanica e Ingegneria dei Materiali presso l'Università di Ferrara prevedono un corso avanzato di Analisi Matematica. Lo scopo di questi corsi è da un lato quello di colmare alcune lacune presenti nei corsi di Analisi Matematica della laurea triennale, dall'altro quello di introdurre alcuni fondamentali strumenti matematici per l'ingegneria. Tra questi compaiono, ad esempio, le serie di funzioni e l'analisi di Fourier, con applicazioni alle equazioni differenziali ordinarie e alle derivate parziali.

Grazie ad un progetto di tutorato finanziato dall'Università di Ferrara, nell'anno accademico 2004-05 è stato possibile affiancare a tali corsi, di svolgimento tradizionale, un corso libero di implementazione su computer degli argomenti trattati. Questo fascicolo è il risultato di tale esperimento. Si tratta pertanto di note che danno una rapida introduzione al software impiegato per un suo utilizzo funzionale ai corsi di cui sopra. Così, ad esempio, le equazioni differenziali sono trattate qui con le tecniche analitiche classiche, e nessun metodo numerico efficace è impiegato per la loro risoluzione. Molti aspetti inerenti ai precedenti corsi di matematica della laurea triennale sono stati tuttavia brevemente sviluppati (un po' di matrici, successioni, serie, grafici di funzioni, curve). Non si è rinunciato invece a utilizzare i cicli di base della programmazione (`for`, `if`, `while`) che, anzi, sono stati dettagliatamente introdotti.

Una sessantina di pagine non vogliono essere un corso organico di programmazione [5] né un saggio di programmazione applicata all'Analisi Matematica [6]; ancor meno, assolutamente meno, vogliono essere un libro di Analisi Numerica. L'idea che sta alla base di queste pagine, e delle notti insonni necessarie per scriverle, è che uno studente (mammifero comprensibilmente afflitto da poca voglia di leggere un libro di didattica) possa sfogliare queste misere carte, vedere una figura che lo interessa e domandarsi se non gli possa venire utile riprodurla, magari con finalità che esulano totalmente dall'Analisi. D'altro canto, questa è anche la presentazione dei vantaggi che un linguaggio di programmazione offre rispetto ad altri arnesi (dall'inglese *tools*), più adatti alla lista della spesa che al calcolo scientifico.

Il software utilizzato è MATLAB©, uno standard in ambiente ingegneristico. Poiché nessuna conoscenza di programmazione o di conoscenza del programma veniva richiesta, le note che seguono introducono i comandi principali di MATLAB e al tempo stesso sviluppano la parte teorica trattata nel corso. Un software analogo a MATLAB ma completamente gratuito è il francese SCILAB, scaricabile a `http://www.scilab.org/`. Per la redazione si è usato LaTeX, da anni il software di scrittura per eccellenza in ambiente scientifico; una versione libera (la stessa usata per queste note) è scaricabile a `http://www.miktex.org/`.

Imprecisioni ed errori di battitura sono inevitabili (e giustificabili) in una prima redazione come questa; gli autori si sono doverosamente impegnati per limitare eventuali altri errori (di matematica e di programmazione, meno giustificabili) ma ringraziano comunque anticipatamente quanti volessero segnalarli.

Ferrara, 15.2.2006

Nicola Cavallini, Andrea Corli

Capitolo 1

Matrici e un po' di grafica

1.1 MATLAB e le Matrici

La parola MATLAB è un acronimo per *Matrix Laboratory* perché tutto quello che il software usa e capisce sono matrici. Matrici, vettori e scalari sono memorizzati nel *workspace* come `array` bidimensionali (tabelle di numeri). Negli esercizi che seguono saranno manipolate alcune matrici per prendere confidenza con i primi comandi di base.

1.1.1 Matrici

A riga di comando digitare:

```
A = [1,2,3;4,5,6;7,8,9]        con e senza ";" in chiusura
B = A([1,2],[1,2,3])           una sottomatrice...
C = A([1:2],[1:end])           un'altra sottomatrice?
                               vediamo...
Test = C == B                  sorpresa!
                               riproviamo...
C(:,3)=0                       cambiamo un po'
Test = C == B                  qualcosa è cambiato
[righeB,colonneB]=size(B)      si ottengono due scalari
v = size(B)                    si ottiene un vettore
```

1.1.2 Calcoli con le Matrici

Ancora a riga di comando:

```
clear                          una bella spazzolata
A = [1,2,3;4,5,6]              si riparte
B = [7 8; 9 10 ;11 12])        una colonna
C = A*B                        prodotto righe × colonne
D = C^2                        il quadrato di una matrice
B = B'                         B = B^Trasposto
C = A*B                        c'è qualche problema??
C = A.*B                       prodotto elemento × elemento
D = C^2                        c'è qualche problema??
D = C.^2                       così va meglio
```

1.1.3 Matrici Speciali

L'ultima volta a riga di comando:

```
clear                  una bella spazzolata
A = [1,2,3;4,5,6]      si riparte
B = ones(3,2)          oppure...
B = ones(size(A'))     funzioni annidate
```

```
C = A*B              il prodotto righe × colonne ...
C = B*A              ... non è commutativo
C = A.*B'            questo è commutativo!
B = eye(3)           la matrice identità
C = A*B              prodotto righe × colonne
B = zeros(size(A))   una matrice di zeri
C = A.*B             un'ecatombe!
```

Esercizio 1.1 Il comando `det(A)` calcola il determinante della matrice (quadrata) A. Il comando `inv(A)` ne calcola l'inversa. Provarli.

1.1.4 Un Vettore delle Ascisse

Per generare un vettore delle ascisse MATLAB offre due comandi fra loro duali. Il primo:

`x = [`*inizio*`:`*passo*`:`*fine*`]`

permette di specificare punto iniziale, punto finale e passo. Se l'indicazione del passo è omessa, essa viene assunta uguale a uno:

`x = [`*inizio*`:`*fine*`]`

Il secondo permette di spaziare *linearmente* gli elementi specificandone il numero:

`x = linspace(`*inizio*`,`*fine*`,`*numero*`)`

1.2 Grafica 2D: Successioni e Serie

La grafica, oltre alla sintassi matriciale, è il punto di forza di MATLAB. Muoveremo ora i primi passi partendo da una semplice successione geometrica:

$$a_n = q^n = \left(\frac{1}{2}\right)^n .$$

Per prima cosa è necessario definire i valori dell'asse delle ascisse, che per una successione sono i numeri naturali (dovrebbe essere già chiaro quale sia la soluzione migliore per definire questi valori...):

```
n = [1:10];
```

Nel nostro caso dieci valori in ascissa sono sufficienti; il punto e virgola "`;`" alla fine dell'espressione serve per impedire la visualizzazione dell'operazione (provare a toglierlo!). Si definisce poi la base della potenza:

```
q = 1/2;
```

Infine, semplicemente utilizzando la sintassi matriciale, è possibile calcolare la successione per i valori prescritti:

```
an = q.^n;
```

Ottenere il grafico di questa successione è banale: è sufficiente utilizzare il comando `plot`, che di default unisce tutti i punti con una spezzata:

```
plot(n,an)
```

Qui il punto e virgola può essere omesso. Il risultato è un po' povero dal punto di vista grafico: per rappresentare una successione è più indicato un grafico a puntini `'o'`, magari rossi `'r'`:

```
plot(n,an,'or')
```

È poi indispensabile specificare le etichette degli assi (in inglese: label), un titolo e magari aggiungere una griglia per rendere il tutto più leggibile:

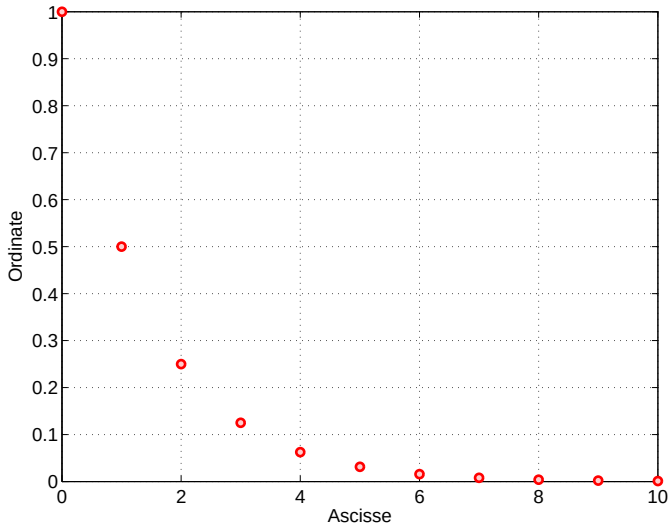


Figura 1.1: Successione geometrica di ragione 1/2.

```
grid on
xlabel('Ascisse')
ylabel('Ordinate')
title('Successione Geometrica')
```

Per passare da successione a serie è sufficiente applicare l'operatore somma:

$$s_n = \sum_{k=0}^{n} a_k = \sum_{k=0}^{n} q^k = \sum_{k=0}^{n} \left(\frac{1}{2}\right)^k .$$

A questo proposito MATLAB fornisce la funzione `cumsum`. Mentre la semplice funzione `sum` opera la *somma* degli elementi di un vettore, con `cumsum` si ottiene la somma *cumulativa*: è esattamente quanto serve per costruire la successione delle somme parziali di una successione. Tornando all'`m-file` precedente si pone:

```
sn = cumsum(an)
```

Per ottenere un secondo grafico è possibile aprire una seconda finestra grafica, `figure(2)` e ripetere i comandi di grafica, magari in blu `'b'`. Ricordiamo che per $|q| < 1$ si ha $\lim_{n\to\infty} \sum_{k=0}^{n} q^n = \frac{1}{1-q} = 2$. Il risultato è in Figura 1.2.

Riguardando il codice appena scritto ci si accorgerà che tutti i caratteri compresi fra apici sono visualizzati in rosso e sono detti stringhe: mentre gli array sono delle tabelle di numeri, le stringhe sono delle sequenze di caratteri.

Esercizio 1.2 Ripetere l'esempio $a_n = q^n$ con la successione $a_n = 1/n$. È noto che la serie armonica $\sum_{n=1}^{\infty} 1/n$ è divergente; cercare graficamente un N tale che $\sum_{n=1}^{N} 1/n > 10$. Disegnare anche la successione $a_n = \log n$ nello stessa finestra; per questo si usi la sintassi

```
plot(n,sn)
hold on
plot(n,log(n))
hold off
```

Cosa si nota? Si può provare che $\lim_{n\to\infty} \left(\sum_{k=1}^{n} 1/k - \log n\right) = \gamma = 0.5772\ldots$; la costante γ è detta costante di Eulero-Mascheroni.

Esercizio 1.3 Come rappresentare successioni con termini di segno alterno, ad esempio $(-1)^n$? Si ricordi che $\cos(n\pi) = \ldots$.

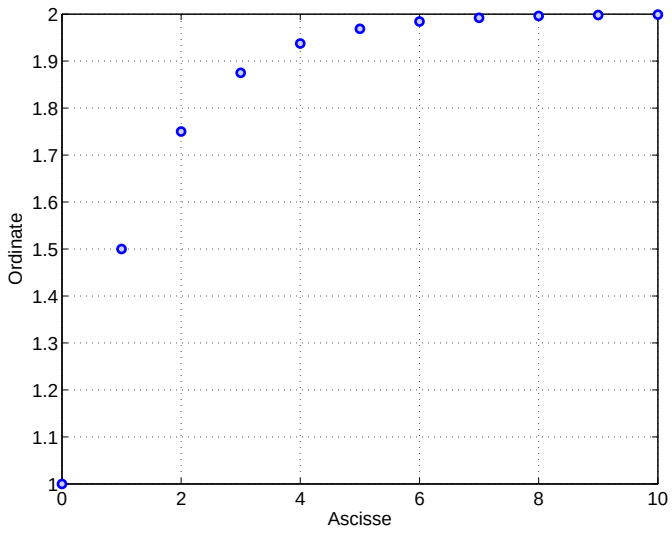


Figura 1.2: Serie geometrica di ragione 1/2.

Capitolo 2

Sviluppi di Taylor

In questo capitolo si approfondisce la grafica 2D e 3D studiando in particolare gli sviluppi di Taylor di funzioni di una o due variabili reali.

2.1 Funzioni di una Variabile Reale e Griglie di Calcolo

Il polinomio di Taylor di ordine n e centro x_0 di una funzione f di una variabile reale è

$$T_{n,x_0}(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \ldots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n \,.$$

Lo sviluppo di Taylor con il resto nella forma di Lagrange si scrive allora

$$f(x) = T_{n,x_0}(x) + \frac{f^{(n+1)}(c)}{(n+1)!}(x - x_0)^{n+1}$$

dove c è un punto compreso tra x e x_0. L'espressione dello sviluppo con il resto nella forma di Peano è infine

$$f(x) = T_{n,x_0}(x) + o\left((x - x_0)^n\right).$$

Concentriamoci ora sullo sviluppo di MacLaurin di ordine m di una funzione f:

$$f(x) = T_{m,0}(x) + o(x^m)\,.$$

Per avere contemporaneamente tutti gli sviluppi fino ad un certo ordine m è opportuno costruire una GRIGLIA di punti in cui per ogni riga sia ripetuto il vettore x (valori delle ascisse) e per ogni colonna il vettore n (ordini dello sviluppo). In sintesi si vuole passare da due vettori

$$x = \begin{pmatrix} x_1 & x_2 & x_3 & \ldots & x_k \end{pmatrix}, \quad n = \begin{pmatrix} 1 \\ 2 \\ \vdots \\ m \end{pmatrix}$$

a due matrici

$$X = \begin{pmatrix} x_1 & x_2 & x_3 & \ldots & x_k \\ x_1 & x_2 & x_3 & \ldots & x_k \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1 & x_2 & x_3 & \ldots & x_k \end{pmatrix}, \quad N = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 2 & 2 & 2 & \ldots & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ m & m & m & \ldots & m \end{pmatrix}.$$

Fissiamo ad esempio $k = 100$, $m = 5$ e consideriamo l'intervallo $[-\pi, \pi]$. Da quanto spiegato nei paragrafi precedenti dovrebbe essere chiaro come partire con l'`m-file`:

```
clear                              un repulisti per evitare inconvenienti
x = linspace(-pi, pi, 100);        mi interessano solo i punti iniziale e finale
n = [1:5]';                        qui voglio una colonna da 1 a 5 passo 1
```

Creare un griglia è altrettanto semplice: è sufficiente specificare che si intende creare una griglia a maglia rettangolare (in inglese: mesh) → `meshgrid`:

```
[X,N] = meshgrid(x,n);
```

Fatto questo è sufficiente valutare i termini del polinomio di MacLaurin sui punti specificati e sommare fra loro le righe della matrice risultante.

Per illustrare questa costruzione consideriamo la funzione $f(x) = \sin x$ nell'intervallo $[-\pi, \pi]$. Si ha:

$$\sin x \sim x - \frac{x^3}{3!} + \frac{x^5}{5!} \,. \tag{2.1}$$

In questo caso lo sviluppo comprende solo le potenze dispari e questo aggiunge un'ulteriore difficoltà. Occupiamoci in primo luogo del termine generale

$$\frac{x^n}{n!}$$

che implementiamo come

```
NUM = X.^N
DEN = cumprod(N)
Tn = NUM ./ DEN
```

ATTENZIONE! Il comando `cumprod` effettua il prodotto cumulativo degli elementi della matrice N, colonna per colonna. In questo caso il vettore n parte da 1 (si veda (2.1)), dunque questa procedura è efficace. Se n fosse partito da zero (ad esempio, se avessimo considerato la funzione esponenziale o la funzione coseno, vedi Esercizio 2.1), allora la matrice N avrebbe avuto una prima riga di zeri: la brutale applicazione del comando `cumprod` avrebbe generato una matrice di zeri. Per ovviare a ciò si propone la seguente sintassi:

```
Nfactorial = cumprod(N + (N==0))
```

In questo modo l'argomento `cumprod` ha le prime due righe identicamente uguali a 1; questo equivale alla definizione $0! = 1$.

Poiché lo sviluppo contiene soltanto termini con potenze dispari, della matrice `Tn` prendiamo le sole righe dispari "`1:2:end`", e tutte le colonne "`:`":

```
Tn = Tn(1:2:end,:);
```

Tutte le righe pari invece devono essere moltiplicate per -1, per rispettare l'alternanza di segno imposta dal termine $(-1)^n$:

```
Tn(2:2:end,:) = -1 * Tn(2:2:end,:);
```

Ora non resta che sommare membro a membro. A questo proposito potrebbe essere utilizzato il comando `sum`, facendo dunque la somma per colonne. In questo caso è più interessante fare una somma cumulativa (`cumsum`) per confrontare fra loro i diversi termini:

```
Tn = cumsum(Tn);
```

Restano soltanto elaborazioni grafiche. Poiché il comando `plot` precedentemente esposto "plotta" le matrici per colonne, è immediato ottenere una rappresentazione di quanto calcolato:

```
plot(x,Tn', 'b',...                    % T_n deve essere trasposto
 'LineWidth',2)                        % spessore della linea
xlabel( 'Ascisse', 'FontSize',14)      % etichetta, dimensione del testo
ylabel( 'Ordinate', 'FontSize',14)     % etichetta, dimensione del testo
hold on                                % "tieni duro"
plot(x,sin(x)), 'r', 'LineWith',2)     % la funzione sin x
```

Sul grafico così costruito le etichette hanno una dimensione diversa da quella degli assi; è necessario "settare" (`set`), gli assi correnti (gca = get current axes):

```
set(gca, 'FontSize',14)
```

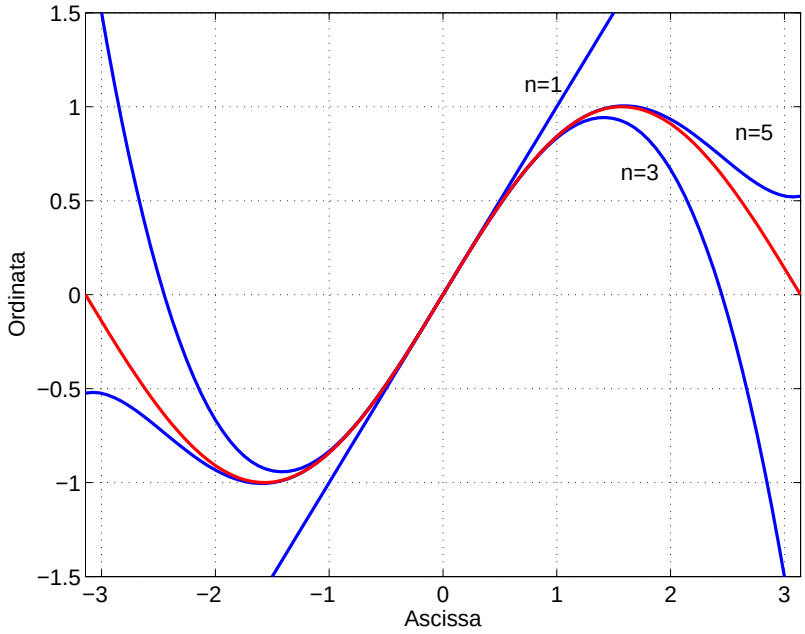


Figura 2.1: Sviluppi di MacLaurin della funzione $\sin x$.

Questa procedura viene definita *Graphics Handle* (*Manipolazione di Oggetti Grafici*). Il risultato della fatica è in Figura 2.1.

Esercizio 2.1 Ripetere le elaborazioni proposte in questo paragrafo per le funzioni $f(x) = e^x$ (usare `Nfactorial` in `DEN`) e $f(x) = \cos x$ (solo le righe pari).

Esercizio 2.2 Impostare un grafico dell'*errore relativo*

$$E(x) = \left| \frac{f(x) - T_n(x)}{f(x)} \right|$$

commesso approssimando una generica funzione ($\sin x,\ \cos x,\ e^x \ldots$) con il suo sviluppo di MacLaurin (sovrapporre i grafici per diversi valori di n). Trovare graficamente il valore di n per il quale $\max_I E < 0.1$ (cioè l'errore è inferiore al 10%) su un dominio che può essere $I = [-\pi, \pi]$ per le funzioni trigonometriche e $I = [-3, 3]$ per le funzioni non trigonometriche.

2.2 Funzioni di Due Variabili Reali e Superfici

In questa sezione facciamo i primi passi nella grafica 3D studiando alcuni grafici di funzioni di due variabili reali, motivati come nella sezione precedente dagli sviluppi di Taylor.

2.2.1 Uno Sviluppo . . . Mancato

Mentre nel caso di funzioni di una variabile *la derivabilità assicura la continuità e l'esistenza della retta tangente*, per le funzioni di due o più variabili *la sola derivabilità non implica né la continuità né l'esistenza del piano tangente*. In Figura 2.2 è riportato il grafico della funzione:

$$f(x,y) = \begin{cases} \frac{xy}{x^2+y^2} & \text{se } (x,y) \neq (0,0) \\ 0 & \text{se } (x,y) = (0,0)\,. \end{cases}$$

Per tale funzione esistono e sono nulle le derivate parziali nell'origine, ma è chiaro che non è possibile individuare un piano tangente al grafico della funzione nel punto $(0,0,0)$. È questo dunque il caso di una funzione di due variabili che è derivabile ma per la quale non è possibile uno sviluppo al primo ordine.

Ottenere grafici di funzioni di due variabili è molto semplice:

```
clear                                  ancora una spazzolata
x=linspace(-1,1,100)                   asse delle x
y=linspace(-1,1,100)                   asse delle y
[x,y] = meshgrid(x,y)                  griglia di calcolo
z = x.*y ./ (x.^2 + y.^2)              la funzione in esame
                                       Attenzione! Assicurarsi che (0,0)
                                       non sia nella griglia
surf(x,y,z, 'FaceColor', 'white')      più semplice di così non si può!
xlabel( 'Asse X')                      etichetta X
ylabel( 'Asse Y')                      etichetta Y
zlabel( 'Asse Z')                      etichetta Z
```

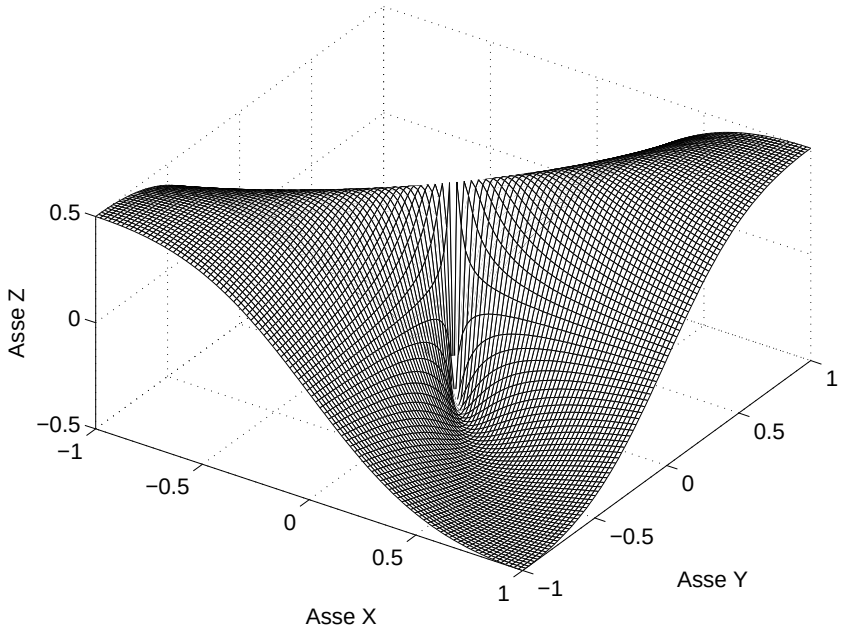


Figura 2.2: Una funzione derivabile nell'origine, ma non differenziabile.

2.2.2 Sviluppo al Secondo Ordine

Per le funzioni di una variabile l'esistenza della retta tangente è assicurata dall'esistenza della derivata, ovvero dalla relazione:

$$f(x_0+h) - f(x_0) = f'(x_0)h + o(h) \text{ per } h \to 0.$$

Generalizzando a funzioni di due variabili si ha:

$$\begin{aligned} & f(x_0+h, y_0+k) - f(x_0, y_0) \; = \\ & = \; \partial_x f(x_0,y_0)h + \partial_y f(x_0,y_0)k + o(\sqrt{h^2+k^2}) \quad \text{per } (h,k) \to (0,0)\,. \end{aligned} \tag{2.2}$$

Se la (2.2) è soddisfatta, cioè se la funzione è *differenziabile*, allora esiste il piano tangente. Se la funzione è ancor più che differenziablie, cioè di classe C^2, è possibile migliorare l'approssimazione utilizzando una quadrica tangente invece di un piano tangente.

Si consideri infatti la funzione $f : \mathbb{R}^2 \to \mathbb{R}$ definita in un intorno di (x_0, y_0), un versore (h, k) e supponiamo che il punto $(x_0 + th, y_0 + tk)$ appartenga ancora a tale intorno per t piccolo. Si consideri la funzione di una variabile:

$$g(t) = f(x_0 + th, y_0 + tk)$$

definita per $t \in [-\epsilon, \epsilon]$. Di questa funzione è possibile scrivere la formula di Maclaurin di ordine due:

$$g(t) = g(0) + g'(0)t + \frac{1}{2}g''(0)t^2 + o(t^2) \quad \text{per } t \to 0\,. \tag{2.3}$$

Applicando la formula di derivazione di una funzione composta si ha:

$$\begin{aligned} g'(t) &= f_x(x_0+th, y_0+tk)\cdot h + f_y(x_0+th, y_0+tk)\cdot k \\ g''(t) &= f_{xx}(x_0+th, y_0+tk)\cdot h^2 + 2\cdot f_{xy}(x_0+th, y_0+tk)\cdot hk + f_{yy}(x_0+th, y_0+tk)\cdot k^2\,. \end{aligned}$$

Infine per sostituzione e per $(h,k) = (\mathrm{d}x, \mathrm{d}y)$ si ha:

$$\begin{aligned} f(x_0+\mathrm{d}x, y_0+\mathrm{d}y) \sim\ & f(x_0,y_0) + \\ & f_x(x_0,y_0)\mathrm{d}x + f_y(x_0,y_0)\mathrm{d}y + \\ & \frac{1}{2}\cdot f_{xx}(x_0,y_0)\mathrm{d}x^2 + \frac{1}{2}\cdot f_{yy}(x_0,y_0)\mathrm{d}y^2 + \\ & f_{xy}(x_0,y_0)\cdot \mathrm{d}x\cdot \mathrm{d}y\,. \end{aligned} \tag{2.4}$$

Finite queste considerazioni teoriche passiamo all'elaborazione con MATLAB. Nella Sezione 2.1 si era implementata direttamente l'espressione dello sviluppo di MacLaurin di una funzione; verrà ora utilizzato il pacchetto simbolico di MATLAB in modo da calcolare analiticamente l'espressioni delle derivate parziali. A questo proposito è importante ricordare la distinzione fra **array** (tabelle di numeri) e **stringhe** (sequenze di caratteri).

La funzione presa in esame è:

$$f(x,y) = -\cos(x+y). \tag{2.5}$$

Per iniziare si valuta numericamente la funzione su un dominio rettangolare (è possibile valutare la funzione simbolicamente, ma questo potrebbe essere computazionalmente oneroso):

```
clear all                  una pulita a fondo
xo = 0.0;                  x_0
yo = 0.0;                  y_0
X = linspace(-6,6,100);    asse x
Y = linspace(-6,6,100);    asse y
[X,Y] = meshgrid(X,Y);     griglia di calcolo
Z = -cos(X+Y);             la funzione valutata numericamente
```

È poi necessario definire il dominio su cui è calcolato lo sviluppo:

```
x = linspace(-2,2,100)+xo;    un fazzoletto...
y = linspace(-2,2,100)+yo;    4 × 4...
[x,y] = meshgrid(x,y);        con la sua griglia
```

Per **dichiarare** un'espressione simbolica si usa il comando `sym`:

```
f = sym( '-cos(x+y)');    la funzione analitica
```

A riga di comando un controllo: `findsym(f)`. Segue il calcolo analitico delle derivate, utilizzando il comando `diff`:

```
fx = diff(f, 'x');      ∂_x f(x,y)
fy = diff(f, 'y');      ∂_y f(x,y)
fxx = diff(fx, 'x');    ∂_xx f(x,y)
fyy = diff(fy, 'y');    ∂_yy f(x,y)
fxy = diff(fx, 'y');    ∂_xy f(x,y)
```

Ora è necessario valutare la funzione e le sue derivate nel punto (x_0, y_0); a questo proposito è utile il comando `subs(Old,New)`:

```
fo = subs(f,{ 'x', 'y'},[xo, yo]);          sostituisci (x,y) con (x_0,y_0)
fxXo = subs(fx,{ 'x', 'y'},[xo,yo]);        ∂_x f(x_0,y_0)
fyXo = subs(fy,{ 'x', 'y'},[xo,yo]);        ∂_y f(x_0,y_0)
fxxXo = subs(fxx,{ 'x', 'y'},[xo,yo]);      ∂_xx f(x_0,y_0)
fyyXo = subs(fyy,{ 'x', 'y'},[xo,yo]);      ∂_yy f(x_0,y_0)
fxyXo = subs(fxy,{ 'x', 'y'},[xo,yo]);      ∂_xy f(x_0,y_0)
```

Conviene implementare separatamente tutti i termini dell'equazione (2.4):

```
dx = x-xo;                                  % dx = x - x_0
dy = y-yo;                                  % dy = y - y_0
lineare = fxXo * dx + fyXo * dy;            % f_x(x_0,y_0)dx + f_y(x_0,y_0)dy
ordineIIpure 1/2 * fxxXo * dx.^2 +...       % 1/2 f_xx(x_0,y_0)dx^2
1/2 * fyyXo * dy.^2;                        % 1/2 f_yy(x_0,y_0)dy^2
ordineIImista = fxyXo * dx .* dy            % f_xy(x_0,y_0) · dxdy
```

Ora è sufficiente mettere tutti i termini insieme:

```
Taylor = fo + lineare + ordineIIpure + ordineIImista;
```

Una volta sviluppati tutti i calcoli si passa alla elaborazione grafica, che sarà spettacolare!

```
surf(X,Y,Z, 'FaceColor', 'r', 'EdgeColor', 'none');
hold on
surf(x,y,Taylor, 'FaceColor', 'b', 'EdgeColor', 'none')
xlabel( 'Asse X', 'FontSize',14)
ylabel( 'Asse Y', 'FontSize',14)
zlabel( 'Asse Z', 'FontSize',14)
set(gca, 'FontSize',14)
camlight
```

Il tutto in Figura 2.3.

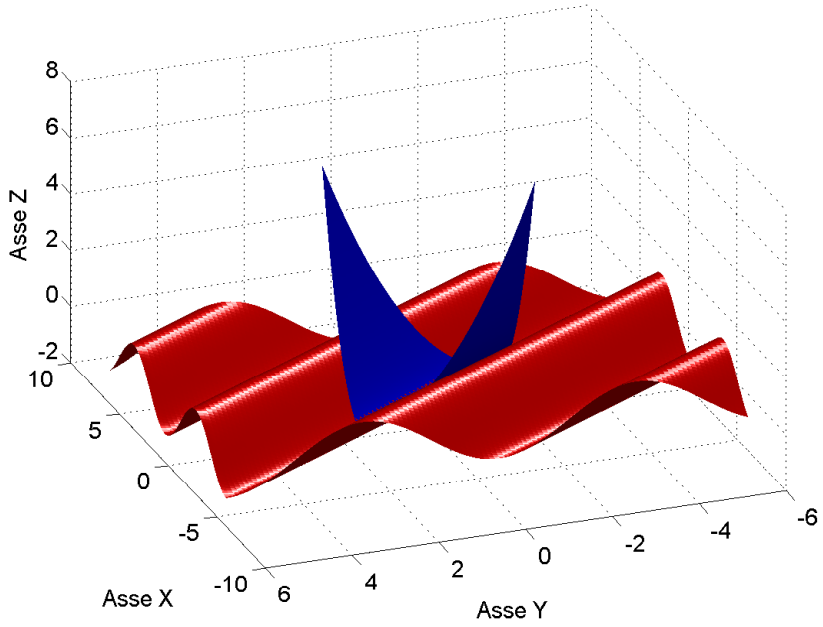


Figura 2.3: Sviluppo di Taylor per una funzione di due variabili.

Esercizio 2.3 Si esaminino gli sviluppi al primo e al secondo ordine di altre funzioni di due variabili. Si riportino quindi nella stessa finestra la funzione, il piano tangente e la quadrica tangente.

Esercizio 2.4 Si esaminino gli sviluppi del terzo ordine. Si tenga presente che il terzo ordine di approssimazione ha la forma:

$$\frac{f_{xxx}\mathrm{d}x^3 + 3f_{xxy}\mathrm{d}x^2\mathrm{d}y + 3f_{xyy}\mathrm{d}x\mathrm{d}y^2 + f_{yyy}\mathrm{d}y^3}{3!}.$$

Esercizio 2.5 Si studino gli sviluppi su un dominio circolare invece che rettangolare; si consideri ad esempio

$$f(\rho, \theta) = -\cos \rho \, .$$

Un possibile inizio potrebbe essere:

```
rho = linspace(...)
theta = linspace(...)
x = ...
y = ...
```

Capitolo 3

Successioni e Serie di Funzioni

Dal punto di vista numerico non cambia molto fra sviluppi di Taylor, serie di funzioni, serie di potenze, serie di Taylor. La procedura in questo caso potrà forse risultare ripetitiva, ma è utile a fissare le idee sull'elaborazione di grafici in ambiente MATLAB.

3.1 Serie di Funzioni

Si procede ora all'elaborazione dei grafici relativi a diverse successioni e serie di funzioni. Il caso particolare delle serie di potenze verrà considerato nella sezione seguente.

3.1.1 Una Serie di Funzioni

Consideriamo la serie di funzioni di termine generale $f_n(x) = \frac{1}{1+x^{2n}}$, cioè

$$\sum_{n=0}^{\infty} \frac{1}{1+x^{2n}}\,.$$

Come già osservato per gli sviluppi di Taylor, la somma parziale $s_n(x) = \sum_{k=0}^{n} \frac{1}{1+x^{2k}}$ dipende soltanto da n ed x. Si noti che le funzioni f_n sono pari e dunque nei disegni dei grafici ci si limita alla regione $x \geq 0$. Si inizi dunque col definire una griglia di calcolo:

```
X = linspace(0,4,100);
N = [0:5];
[x,n] = meshgrid(X,N);
```

Si valuti poi la successione di funzioni f_n sulla griglia di calcolo appena definita:

```
successione = 1./(1+x.^(2*n));
```

Per passare dalla successione di funzioni alla serie di funzioni è chiaramente sufficiente fare la somma. Siamo interessati alla somma dei termini per ogni valore di n, dunque al posto del comando `sum`, verrà impiegato il comando `cumsum`:

```
serie = cumsum(successione);
```

Ora la procedura di calcolo può considerarsi completata: non restano che le elaborazioni grafiche. In questo caso, come in molti altri nella pratica ingegneristica, è interessante affiancare nella stessa finestra grafica due rappresentazioni distinte. L'ambiente MATLAB consente di dividere una finestra grafica in righe e colonne ed affiancare in questo modo diversi grafici. Il comando in esame è `subplot(`*righe*`,`*colonne*`,`*numero del grafico*`)` dove i grafici sono numerati per righe. Nel nostro caso il codice può risultare:

```
subplot(2,1,1)
plot(X,successione,'LineWidth',2);
set(gca,'FontSize',14);
xlabel('Asse X,'FontSize',14)
```

```
ylabel('Asse Y','FontSize',14)
h = legend('n = 0','n = 1','n = 2','n = 3','n = 4','n = 5')
grid on
subplot(2,1,2)
plot(X,serie,'LineWidth',2);
set(gca,'FontSize',14);
xlabel('Asse X','FontSize',14)
ylabel('Asse Y','FontSize',14)
h = legend('n=0','n=1','n=2','n=3','n=4','n=5')
```

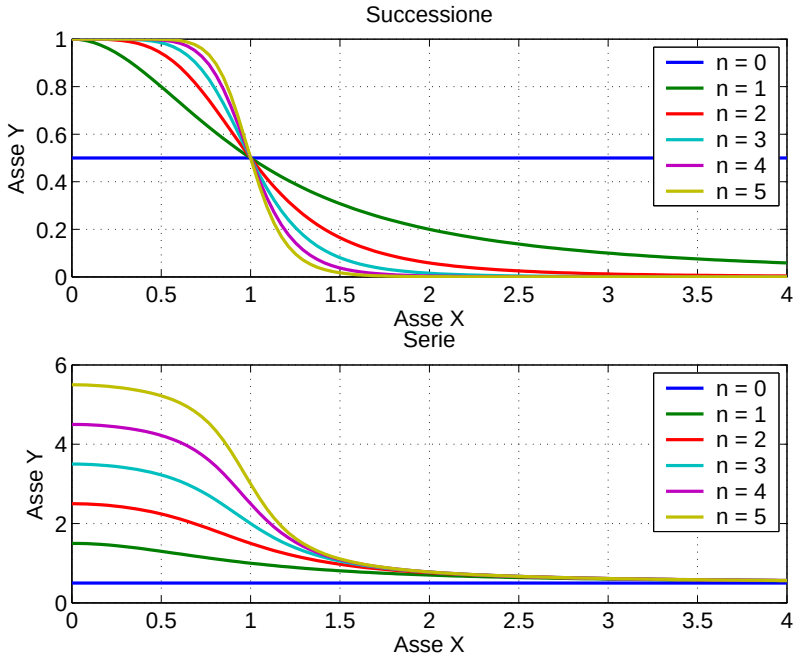


Figura 3.1: Rappresentazione della successione $f_n(x) = \frac{1}{1+x^{2n}}$ e della serie associata.

Dalla seconda finestra grafica in Figura 3.1 si può intuire graficamente quello che è possibile dimostrare analiticamente: la serie diverge per $|x| \leq 1$ e converge per $x > 1$. La convergenza totale ha luogo in $(-\infty, -a] \cup [a, +\infty)$ con $a > 1$.

3.1.2 Un'Altra Serie di Funzioni

Una seconda serie di funzioni può essere generata dalla successione di funzioni $f_n(x) = n^x \cdot x^n$, cioè:

$$\sum_{n=1}^{\infty} n^x \cdot x^n.$$

Il listato precedentemente steso è valido anche in questo caso salvo alcune semplici osservazioni:

- La serie questa volta parte da $n = 1$, dunque: `N = [1:5]`.
- Poiché le f_n non sono né pari né dispari si valuta la serie di funzioni su una porzione dell'asse delle ascisse diversa dalla precedente, ad esempio: `X = linspace(-1.5,1.5,100)`
- Naturalmente è poi necessario cambiare la successione di funzioni:
  `successione = n.^x .* x.^n;`

Il comando `axis([`$x_{min}, x_{max}, y_{min}, y_{max}$`])` rende l'elaborazione più leggibile, limitando la regione di piano rappresentata; si noti che l'argomento di `axis` è un vettore. In questo caso il grafico risulta sufficientemente chiaro per:

```
axis([-1.5 1.5 -1 2])
```

Il risultato è contenuto in Figura 3.2. La serie converge puntualmente in $I = [-1, 1)$ mentre la convergenza è totale per $I = [-r, r]$ con $r < 1$.

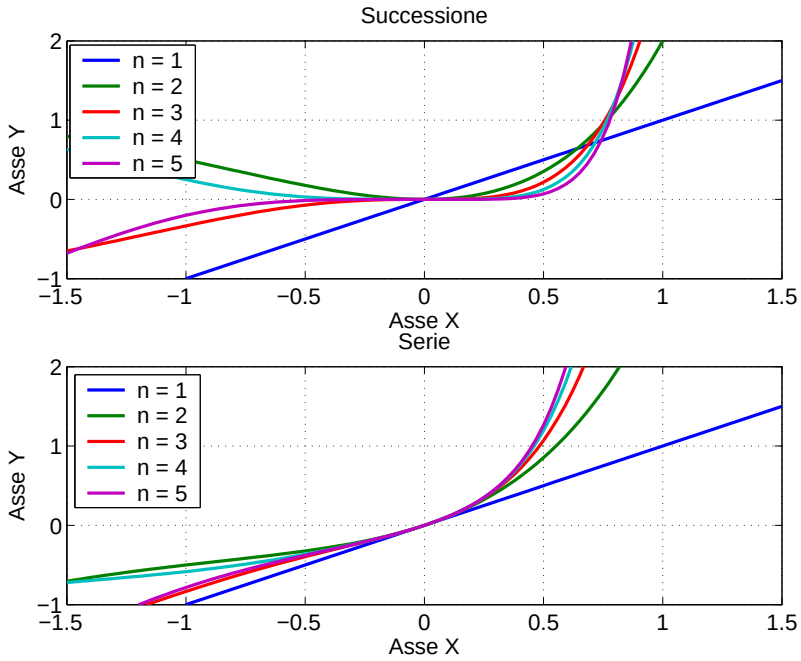


Figura 3.2: Successione $f_n(x) = n^x x^n$ e serie relativa.

3.1.3 Una Serie "Geometrica"

La serie di funzioni

$$\sum_{n=1}^{\infty} \left(\frac{x}{1+x^2} \right)^n$$

è una serie geometrica di ragione $\frac{x}{1+x^2} < 1$ per ogni $x \in \mathbb{R}$. Pertanto la serie converge su tutto l'asse reale e la sua somma vale $\frac{1+x^2}{1-x+x^2}$. La fase di calcolo resta la medesima implementata al punto precedente (salvo cambiare le variabili `X` ed `N`), con l'accortezza di modificare l'argomento della serie:

```
successione = (x./(1+x.^2)).^n;
```

Per quello che riguarda l'elaborazioni, risulta particolarmente interessante realizzare un filmino dove all'aumentare di n si nota che la serie di funzioni converge alla funzione somma. Viene introdotto qui il comando `for`.

```
for cnt = 1:length(N)          da 1 fino alla lunghezza di N...
subplot(2,1,1)                 dividi la finestra grafica
plot(X,successione(cnt,:));    "plotta" la successione una riga alla volta
title('Successione')           dagli un titolo
xlabel('Asse X')               etichetta x
ylabel('Asse Y')               etichetta y
axis([-1.5 1.5 -.2 .2])        tieni fermi gli assi se no balla tutto!
grid on                        la griglia
subplot(2,1,2)                 nella seconda sottofinestra...
plot(X,serie(cnt,:));          "plotta" la serie una riga alla volta
hold on                        tieni duro!
plot(X,somma,'r')              "plotta" la somma
title('Serie')                 dagli un titolo
xlabel('Asse X')               etichetta x
```

```
ylabel('Asse Y')              etichetta y
axis([-1.5 1.5 -.5 2])        tieni fermi gli assi se no balla tutto!
hold off                      molla tutto che dopo si ricomincia!
grid on                       la griglia
pause(.1)                     dammi il tempo di guardare!
end                           fine dei giochi
```

In Figura 3.3 il risultato finale.

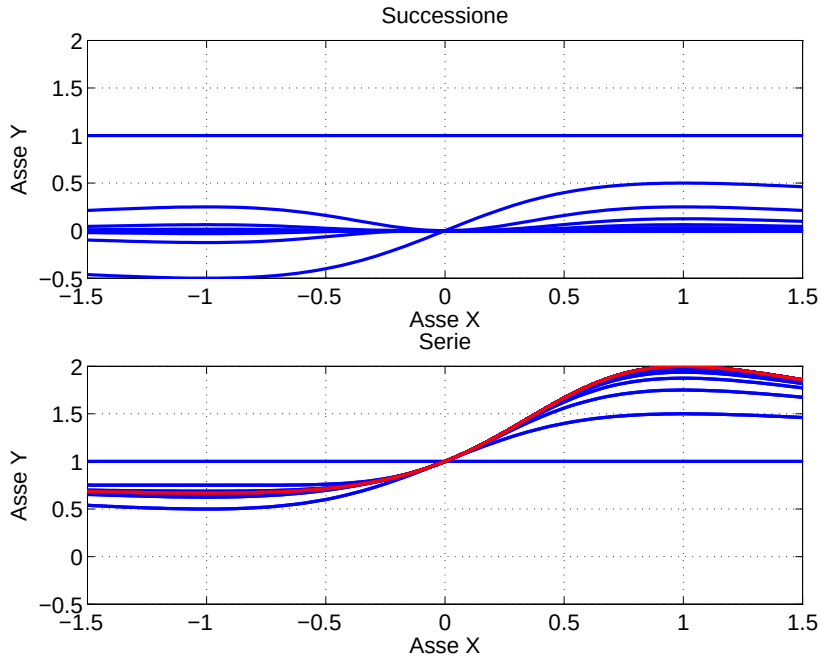


Figura 3.3: Successione $f_n(x) = \left(\frac{x}{1+x^2}\right)^n$, serie e funzione somma.

Esercizio 3.1 A volte è interessante rappresentare i grafici ad intervalli di 2 o 3 n. Per esempio per alcune serie di funzioni è interessante distinguere (per esempio con due colori diversi) i termini pari e quelli dispari; a questo proposito possono essere interessanti quelle serie che i cui termini "ballonzolano". Si analizzino in questo modo le serie (di potenze) $\sum_{n=0}^{\infty}(-1)^n x^n$ e $\sum_{n=0}^{\infty}(-1)^n x^{2n+1}/(n+1)!$ (che hanno per somme...).

3.2 Serie di Potenze

Una serie di potenze è una serie di funzioni della forma $\sum_{n=0}^{\infty} a_n(x-x_0)^n$, il cui raggio di convergenza R si può calcolare da $\frac{1}{R} = \lim_{n\to\infty} \sqrt[n]{|a_n|}$.

Si prenda in considerazione la serie di potenze:

$$\sum_{n=2}^{\infty} \frac{(x+1)^n}{\ln n}$$

dove $x_0 = -1$, il raggio di convergenza vale 1 poiché $\lim_{n\to\infty} 1/\sqrt[n]{\log n} = 1$ e l'insieme di convergenza è $I = [-2, 0)$. Vediamo ora come questo risultato possa essere rappresentato compiutamente in ambiente MATLAB; la parte di calcolo è analoga a quella dei paragrafi precedenti. Dato che l'insieme di convergenza è $I = [-2, 0)$, è opportuno porre:

```
X = linspace(-2.5,0.5,100);
```

Naturalmente è poi necessario cambiare l'espressione della successione di funzioni:

```
successione = 1./log(n).*(x+1).^n;
```

Completata la fase di calcolo si passa alle elaborazioni grafiche. In questo caso è interessante dividere la finestra grafica in tre parti. Nella prima si lascia la rappresentazione della successione $1/\sqrt[n]{a_n}$ per controllarne il limite; nelle altre due si lascia scorrere, in un filmino, la successione di funzioni e la serie di funzioni. In coda alla fase di calcolo si valuta la successione $a_n = 1/\sqrt[n]{\ln n}$;

```
an = (1./log(N)).^(1./N);
```

Passando alle elaborazioni, si parte dunque con la successione a_n:

```
subplot(3,1,1)                          dividi la finestra grafica
plot(N,plot(N,an,'b', 'LineWidth',2);   "plotta" la successione
title( 'Successione')                   dagli un titolo
xlabel( 'n')                            etichetta x
ylabel( 'an')                           etichetta y
grid on                                 la griglia
```

Ottenuto il primo grafico, per avere il filmino è sufficiente un ciclo `for`, esattamente come al paragrafo precedente:

```
for cnt = 1:length(N)                          da 1 fino alla lunghezza di N...
subplot(3,1,2)                                 secondo grafico
plot(X,successione(cnt,:), 'LineWidth',2);     la successione una riga alla volta
title( 'Successione di funzioni')              il titolo
xlabel( 'Asse X')                              etichetta x
ylabel( 'Asse Y')                              etichetta y
axis([-2.5 0.5 -1 10])                         tieni fermi gli assi
hold on                                        sovrapponiamo tutto
grid on                                        la griglia
subplot(3,1,3)                                 il terzo grafico
plot(X,serie(cnt,:), 'LineWidth',2);           la serie una riga alla volta
hold on                                        sovrapponiamo tutto
title( 'Serie')                                il titolo
xlabel( 'Asse X')                              etichetta x
ylabel( 'Asse Y')                              etichetta y
axis([-2.5 0.5 -1 10])                         evitiamo il mal di mare...
grid on                                        la griglia
box on                                         tutto in una bella scatolina
pause(1)                                       dammi il tempo di guardare!
if cnt == 1                                    al primo ciclo...
pause                                          il tempo di sistemare la finestra
end                                            fine dell'istruzione condizionale
end                                            fine della proiezione
```

Il risultato è in Figura 3.4.

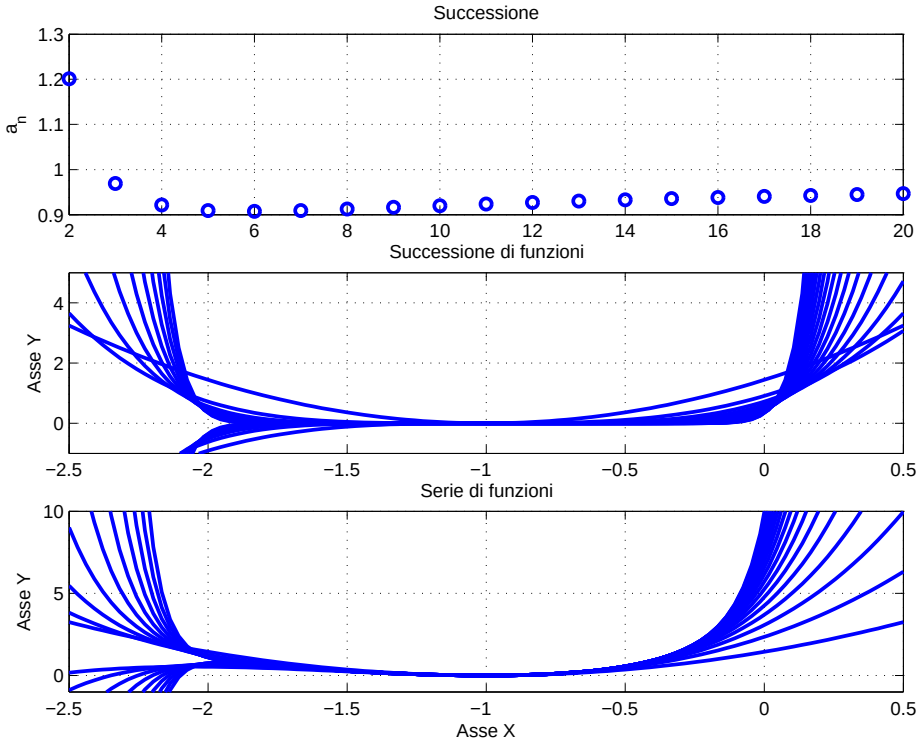


Figura 3.4: Serie di potenze $\sum_{n=2}^{\infty} \frac{(x+1)^n}{\ln n}$.

Capitolo 4

Equazioni Differenziali Ordinarie

Di seguito verranno utilizzati alcuni tools dell'ambiente MATLAB per la visualizzazione e la risoluzione (approssimata) di equazioni differenziali ordinarie (EDO in italiano, in inglese ODE). Si studieranno sia equazioni del primo che del secondo ordine, sistemi lineari, e si implementerà l'approssimazione di una soluzione di una equazione di secondo ordine tramite una serie di potenze troncata.

4.1 Campo direzionale di una EDO

Cominciamo col considerare l'equazione non lineare del primo ordine

$$y' = x\cos^2 y\,. \tag{4.1}$$

Se poniamo $f(x,y) = x\cos^2 y$ potremmo essere tentati di disegnare il grafico di f come nella Sezione 2.2; ciò è tuttavia di scarsa utilità. È invece più utile scrivere

$$\frac{\mathrm{d}y}{\mathrm{d}x} = x\cos^2 y$$

e rappresentare il campo vettoriale

$$(\mathrm{d}x, \mathrm{d}y) = (1, x\cos^2 y).$$

Per costruire il campo vettoriale in ambiente MATLAB partiamo ancora una volta dalla costruzione di una griglia di punti:

```
x = linspace(-5,5,30);
y = linspace(-5,5,30);
[X,Y] = meshgrid(x,y);
```

Sulla griglia si impone $\mathrm{d}x = 1$ e si valuta $\mathrm{d}y$:

```
dx = ones(size(X));
dy = Y.*cos(X);
```

Per visualizzare un campo vettoriale si utilizza il comando `quiver(x,y,Vx,Vy)`, dove `(x,y)`, rappresentano le componenti di posizione e `(Vx,Vy)` le componenti vettoriali. Nel caso in esame la sintassi ha la forma:

```
quiver(X,Y,dx,dy)
```

Il risultato in Figura 4.1.

4.2 Problema di Cauchy e Linee di Flusso

Consideriamo ancora l'equazione (4.1); la soluzione del problema di Cauchy:

$$\begin{cases} y'(x) = x\cos^2 y & \text{EDO} \\ y(0) = \pi/4 & \text{CI} \end{cases} \tag{4.2}$$

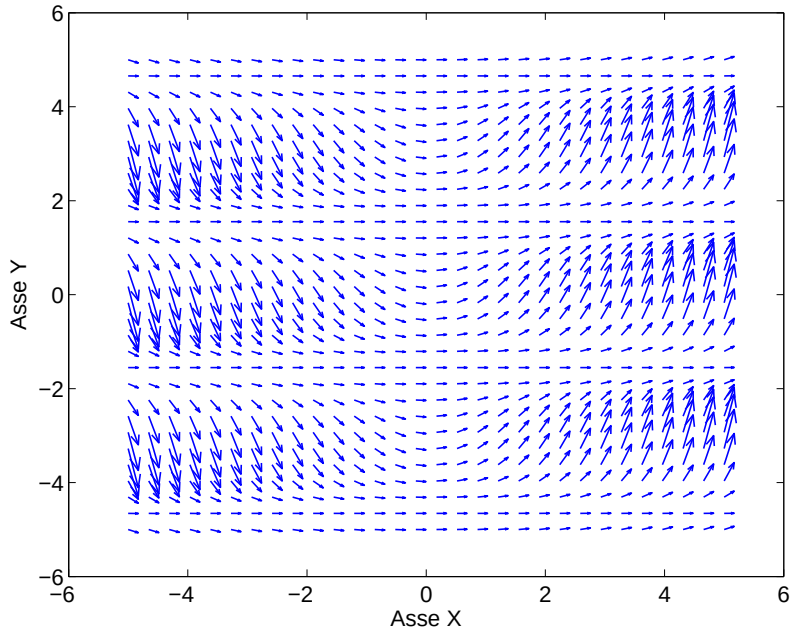


Figura 4.1: Campo differenziale per la EDO $y' = x\cos^2 y$.

è:

$$y(x) = \arctan\left(\frac{x^2}{2} + 1\right).$$

Tale soluzione è la *Linea di Flusso* passante per il punto $(0, \pi/4)$. Per linea di flusso di un campo vettoriale si intende una linea tangente in ogni punto al campo vettoriale; questo concetto è di uso comune in meccanica dei fluidi [7]. Per sovrapporre al campo di moto la soluzione analitica è sufficiente in primo luogo calcolare tale soluzione:

```
int = atan(x.^2./2 + 1);
```

e poi "plottarla":

```
hold on
plot(x,int, 'r')
```

Il comando `streamline(x,y,Vx,Vy,Sx,Sy)` consente l'elaborazione delle linee di flusso, dove le coppie `(x,y)` e `(Vx,Vy)` hanno il medesimo significato assunto nel comando `quiver`, mentre `(Sx,Sy)` rappresentano i vettori delle coordinate dei punti iniziali su cui sono calcolate le linee di flusso. Nel caso in esame si impone:

```
sy = pi./[3,4,5];
sx = zeros(size(sy))
S = streamline(X,Y,dx,dy,sx,sy)
set(S, 'Color', 'g')
```

Il risultato è riportato in Figura 4.2. Si noti che la soluzione analitica e la linea di flusso con punto iniziale $(0, \pi/4)$ coincidono con buona approssimazione: la non esatta coincidenza è dovuta alla discretizzazione degli assi.

4.3 Intermezzo: Function `m-files`

Il software MATLAB consente di creare funzioni personalizzate, ... basta "dirglielo". Le nuove funzioni sono degli `m-file` esattamente come gli altri, solo che dichiarano di essere un nuova funzione (`function`) ed hanno lo stesso nome di file della funzione che definiscono. Con un esempio il tutto è più chiaro. Si scriva un `m-file` siffatto:

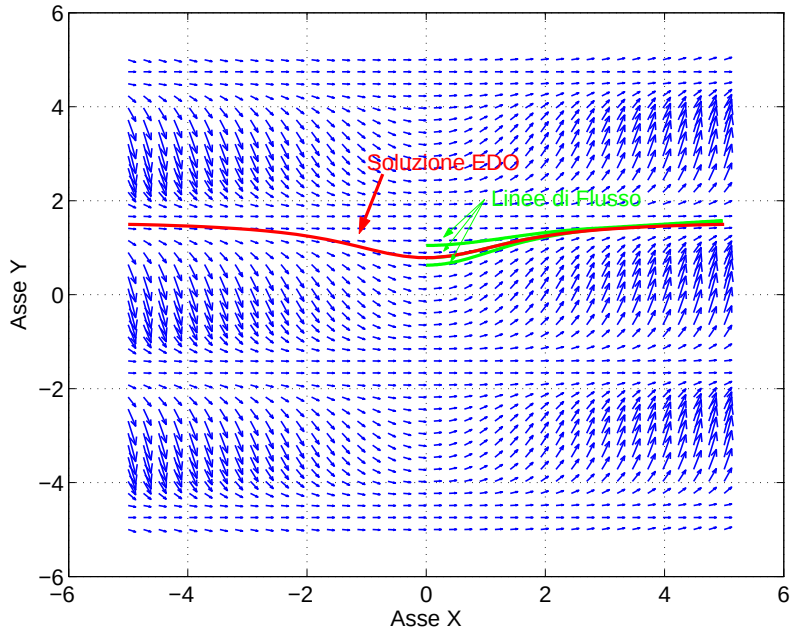


Figura 4.2: Campo differenziale, linee di flusso e soluzione del problema di Cauchy (4.2).

```
function y = pippo(x)
y = x.^2
```

lo si salvi con nome `pippo` e si presti attenzione al fatto che il linguaggio MATLAB è keysensitive. In un altro `m-file` di nome qualsiasi si implementi:

```
x = linspace(-2,2,100);
y = pippo(x)
plot(x,y)
```

4.4 Soluzione Numerica col Pacchetto `ode45`

Ritorniamo alle EDO considerando l'equazione non lineare del primo ordine

$$y' = \frac{x + 2 \cdot \cos x}{y^2}$$

e scrivendola in un `function-mfile`:

```
function Yprimo = edoI(x,y)
Yprimo = (x + 2*cos(x))./y.^2;
```

Dopodiché si utilizzi il comando `ode45` per la soluzione del problema di Cauchy:

$$\begin{cases} y'(x) = \frac{x+2\cdot\cos x}{y^2} & \text{EDO} \\ y(0) = 2 & \text{CI.} \end{cases} \tag{4.3}$$

La sintassi è:

```
[x,y]=ode45( 'edoI',[0 5],2);
```

dove

- la stringa `'edoI'` richiama la funzione appena definita;
- `[0 5]` è l'intervallo su cui l'equazione viene risolta;
- `2` è la condizione iniziale $y(0) = 2$.

La soluzione analitica del problema di Cauchy (4.3) è:

$$y(x) = \left(\frac{3}{2}x^2 + 6\sin x + 8\right)^{1/3}$$

che implementata ha la forma:

```
Y = (3*x.^2/2 + 6*sin(x)+8).^(1/3);
```

Riportando i grafici della soluzione analitica e di quella numerica

```
plot(x,y,'b',x,Y,'r');
```

si nota che i due grafici sono esattamente sovrapposti (Figura 4.3), a prova della coincidenza della soluzione numerica e di quella analitica.

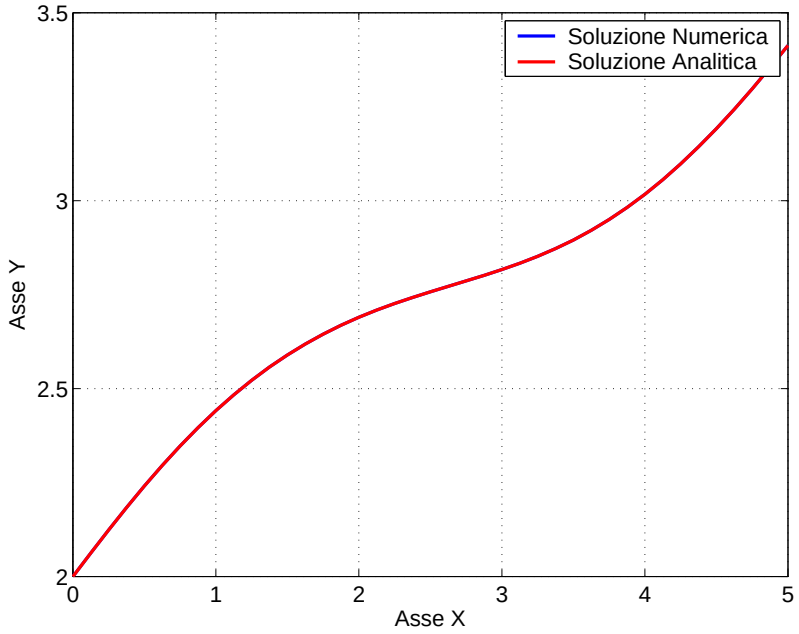


Figura 4.3: Soluzione analitica e numerica per il problema di Cauchy (4.3).

4.5 Sistemi di EDO

Si consideri il problema ai valori iniziali relativo al seguente sistema lineare:

$$\begin{cases} x'(t) = 2x(t) - 3y(t) & ; \ x(0) = 1 \\ y'(t) = 2x(t) + y(t) & ; \ y(0) = -1. \end{cases} \tag{4.4}$$

È noto che la soluzione (unica!) esiste per ogni $t \in \mathbb{R}$; la si vuole rappresentare per $0 \leq t \leq 1$. Se si pensa il problema in termini matriciali, allora la procedura è analoga a quella sviluppata nel paragrafo precedente. Si parte definendo in un `function m-file` il sistema di EDO:

```
function Xprime = SisEDO(t,X)
Xprime = [2*X(1) - 3*X(2);
          2*X(1) + X(2)];
```

Con la medesima sintassi presentata al paragrafo precedente la soluzione del problema ai valori iniziali viene approssimata numericamente utilizzando la funzione `ode45`:

```
tstart = 0; tend = 1
[t,X] = ode45('SisEDO', [tstart, tend], [1 -1])
```

Si noti che in questo caso le condizioni iniziali sono una sulla funzione $x(t)$ ed una sulla funzione $y(t)$. L'array `X` ha la forma:

$$\texttt{X} = \begin{pmatrix} x(t_1) & y(t_1) \\ x(t_2) & y(t_2) \\ \vdots & \vdots \\ x(t_{end}) & y(t_{end}) \end{pmatrix}.$$

Per rappresentare la soluzione del problema è possibile rappresentare in una prima sotto-finestra il piano xy, ottenendo dunque il sostegno della curva:

$$\begin{cases} x = x(t) \\ y = y(t). \end{cases}$$

In una seconda sotto-finestra si riportano invece i grafici delle funzioni $x(t)$ ed $y(t)$:

```
subplot(2,1,1);
plot(X(:,1),X(:,2),'LineWidth',2)
xlabel('Asse X')
ylabel('Asse Y')
grid on
subplot(2,1,2)
plot(t,X(:,1),t,X(:,2),'LineWidth',2)
xlabel('Asse X')
ylabel('Asse Y')
legend('x(t)','y(t)')
grid on
```

Il risultato finale è in Figura 4.4.

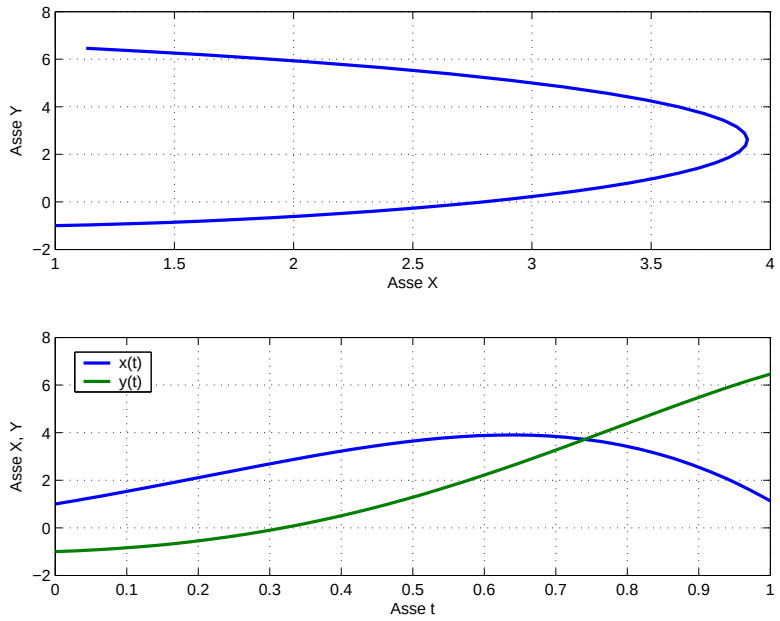


Figura 4.4: Rappresentazione della soluzione del problema ai valori iniziali (4.4) .

Esercizio 4.1 Considerare alcuni semplici sistemi differenziali lineari di tre equazioni in tre incognite $x(t)$, $y(t)$, $z(t)$. Procedere come sopra e disegnare il sostegno della curva soluzione nello spazio.

4.6 EDO di Ordine Superiore

Consideriamo brevemente in questa sezione le equazioni di secondo ordine, discutendone sia la soluzione numerica che una implementazione del noto procedimento teorico di integrazione per serie.

4.6.1 Soluzione Numerica per Riduzione a Sistema...

Consideriamo il seguente problema ai valori iniziali per una equazione lineare del secondo ordine:

$$\begin{cases} y'' - ty = 0 \\ y(0) = 0 \\ y'(0) = 1. \end{cases} \tag{4.5}$$

Il problema (4.5) può essere scritto come un problema ai valori iniziali per un sistema del primo ordine:

$$\begin{cases} x'(t) = t \cdot y(t) & ; \ y(0) = 0 \\ y'(t) = x(t) & ; \ x(0) = 1\,. \end{cases}$$

Questo sistema si risolve in maniera del tutto analoga a quanto visto nella Sezione 4.5. In primo luogo si scrive il `function m-file`:

```
function Xprime = higher(t,x)
Xprime =[t*X(2);
         X(1)];
```

Si tenga presente che `X(1)` $= x(t) = y'(t)$ e `X(2)` $= y(t)$. Si risolve quindi numericamente il problema:

```
[t,X] = ode45( 'higher', [0,5] , [1,0]);
```

4.6.2 ...e Via Serie di Potenze

L'equazione in (4.5) non è a coefficienti costanti e dunque la tecnica di integrazione basata sul polinomio caratteristico non si applica. Un modo classico di integrazione è tramite le serie di potenze ed è questo procedimento che si intende implementare ora.

Si può dimostrare che la soluzione di (4.5) può essere scritta come

$$y(t) = t + \sum_{n=1}^{\infty} \frac{t^{3n+1}}{(3 \cdot 4) \cdot (6 \cdot 7) \cdots 3n(3n+1)}\,.$$

Nelle elaborazioni precedenti le somme $\sum_{k=0}^{n} f_k$ di funzioni f_k erano sempre state valutate per un valore di n definito a priori. Ora invece vogliamo fermarci quando il massimo valore dell'*incremento relativo al passo n sul dominio $I = [0, 5]$* scende sotto l'uno per cento (la terminologia è imprecisa ma di largo uso in ingegneria). L'incremento relativo al passo n al tempo t è definito per $n \geq 1$ da

$$iR_n(t) = \frac{\frac{t^{3n+1}}{(3\cdot4)\cdot(6\cdot7)\ldots3n(3n+1)}}{t + \sum_{k=1}^{n} \frac{t^{3k+1}}{(3\cdot4)\cdot(6\cdot7)\ldots3k(3k+1)}}$$

e vogliamo dunque arrestare l'approssimazione a quell'n per cui $\max_I iR_n(t) < 0.01$. Si noti che nell'intervallo considerato tutti i termini della somma sono positivi; nel caso in cui non lo fossero stati la richiesta diventava $\max_I |iR_n(t)| < 0.01$.

La condizione che stiamo imponendo è quella tipica di un ciclo `while`: *mentre l'istruzione condizionale è soddisfatta, riesegui il procedimento che ti assegnerò, fine,* `end`. Si presti attenzione al fatto che quando non si ha ancora molta dimestichezza con questa istruzione è facile incorrere in un loop, vale a dire la condizione resta sempre soddisfatta ed il ciclo si ripete all'infinito.

Procedendo con l'implementazione è in primo luogo necessario definire i valori dell'ascissa t ed n, che per la prima iterazione è soltanto uguale ad uno:

```
T = linspace(0,5,100);
N(1) = 1;
```

L'incremento relativo, affinché non interferisca con l'istruzione condizionale `while`, è nella prima fase imposto uguale ad 1:

```
iR = 1
```

Ancora in fase preliminare viene definito un contatore `cnt` e si assegna immediatamente il primo termine della serie di potenze:

```
cnt = 1;
y(1,:) = T;
```

Le istruzioni cicliche ricalcano esattamente i codici eleborati per le serie di funzioni, con l'accortezza di aggiungere un elemento al vettore `N` ad ogni iterazione.

```
while iR >0.01                  mentre l'incremento supera l'1%
cnt = cnt +1;                   aggiorna il contatore
N(cnt) = cnt;                   aumenta di un elemento N
[tt,n] = meshgrid(T,N);         la griglia come ai paragrafi precedenti
an = 3*n .* (3*n+1);            3n(3n+1)
an = cumprod(an);               3·4·6·7...3n(3n+1)
sigma = (tt.^(3*n+1))./(an);    σ = t^(3n+1) / (3·4·6·7...3n(3n+1))
[r,c] = size(n);                righe e colonne di n
a = r+1;                        y avrà una riga in più di n
y(2:a,:)  = sigma;              ora ho la matrice degli incrementi ∀ grado
[iA,Id] = max(y(a,:));          ne prendo il massimo su t per l'ultimo n
%                               e ne memorizzo la posizione
y = cumsum(y);                  si sommano tutti gli incrementi ottenendo
%                               l'approssimazione della soluzione ∀n
iR = iA/(y(a,Id));              infine si calcola l'incremento relativo
end                             se iR<0.01 ci si ferma, altrimenti si riparte
```

Prima di passare all'elaborazione grafica si noti che l'incremento percentuale è sceso sotto l'1% per `cnt = 8`, sono dunque state necessarie sette iterazioni per ottenere $\max iR_8(t) = 0.0025$. Le elaborazioni grafiche prevedono di sovrapporre la soluzione numerica e l'approssimazione per serie così come riportato in Figura 4.5. La sintassi che permette di ottenere il grafico appena citato è:

```
plot(T,y, 'r', 'LineWidth',2)
hold on
plot(t,X(:,2), 'b', 'LineWidth',2)
axis([0,5,0,20])
xlabel( 'Asse X', 'FontSize',14)
ylabel( 'Asse Y', 'FontSize',14)
set(gca, 'FontSize',14))
grid on
```

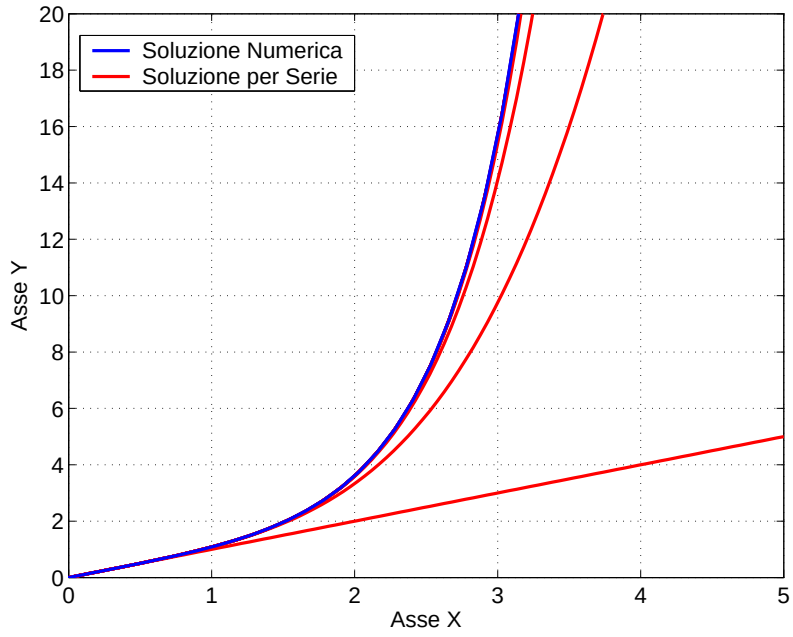


Figura 4.5: Soluzione numerica e la sua approssimazione per serie di potenze del problema (4.5).

Capitolo 5

Serie di Fourier

Le serie di Fourier permettono di approssimare una funzione T-periodica mediante polinomi trigonometrici. Infatti se f è abbastanza regolare allora

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} a_k \cdot \cos(\omega k x) + b_k \cdot \sin(\omega k x) \tag{5.1}$$

dove $\omega = \frac{2\pi}{T}$. I coefficienti della serie valgono:

$$\frac{a_0}{2} = \frac{1}{T}\int_{-T/2}^{T/2} f(x)\ \mathrm{d}x \tag{5.2}$$

$$a_k = \frac{2}{T}\int_{-T/2}^{T/2} f(x)\cdot\cos(\omega k x)\ \mathrm{d}x \tag{5.3}$$

$$b_k = \frac{2}{T}\int_{-T/2}^{T/2} f(x)\cdot\sin(\omega k x)\ \mathrm{d}x\,. \tag{5.4}$$

5.1 Una Funzione Dispari e la Regola dei Trapezi

Si prenda in considerazione la funzione:

$$f(x) = x \quad \text{per } x \in [-\pi, \pi)$$

e prolungata per periodicità su tutto $\mathbb{R}$. Si intende calcolare la somma parziale di ordine n della sua serie di Fourier nell'intervallo $[-\pi, \pi)$, valutando numericamente i coefficienti a_k e b_k. Come sempre si parte definendo l'ascissa x e valutando la funzione $f(x)$:

```
x = linspace(-pi,pi,100);
fx = x;
```

Nel caso in esame $T = 2\pi$, e dunque le (5.2), (5.3), (5.4) diventano:

$$\frac{a_0}{2} = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\ \mathrm{d}x, \quad a_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cdot\cos(kx)\ \mathrm{d}x, \quad b_k = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cdot\sin(kx)\ \mathrm{d}x\,.$$

Si osservi che i coefficienti a_k e b_k dipendono da x e k; è dunque opportuno creare una griglia di calcolo, come abbiamo fatto nei paragrafi precedenti:

```
k = [1:7]
[X,K] = meshgrid(x,k);
[FX,K] = meshgrid(fx,k);
```

Dal momento che è già stata valutata la funzione $f(x)$, e per dare una maggiore generalità al listato, è opportuno creare una griglia $f(x), k$. Per valutare numericamente gli integrali sarà utilizzato il comando `trapz(`*dominio*`,`*funzione integranda*`)`. È importante tener presente che il comando `trapz` svolge l'integrale numerico per colonne; nel nostro caso pertanto la matrice che rappresenta la

funzione integranda dovrà essere trasposta. Il comando `trapz` approssima la funzione con una spezzata e somma le aree dei singoli trapezi soggiacenti. Senza entrare nel dettaglio dell'algoritmo (l'area di un trapezio è data da base maggiore per base minore diviso due...) si può scrivere, con riferimento alla Figura 5.1:

$$A_j = (f(x_{x+1}) + f(x_i)) \cdot \frac{\Delta x}{2}\,, \qquad A = \sum_j A_j\,.$$

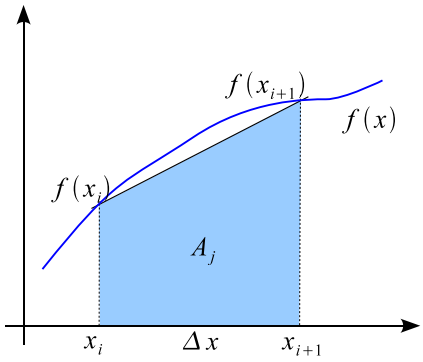


Figura 5.1: L'algoritmo `trapz` per il calcolo approssimato di integrali.

Torniamo alla funzione definita sopra. Poiché f è dispari i coefficienti a_k sono nulli per ogni $k = 0, 1, \ldots$. Per impostare una procedura generale li calcoliamo ugualmente in modo numerico; non dovremo aspettarci che siano nulli, a causa dell'approssimazione del calcolo.

Valutiamo dunque a_0:

```
ao = 1/pi * trapz(x,fx);
```

Per calcolare gli a_k è necessario utilizzare la griglia di calcolo creata in precedenza ed utilizzare `trapz` in senso matriciale:

```
ak = FX .* cos(K .* X);
ak = trapz(x,ak');
ak = ak./pi;
```

Si osservi che gli a_k calcolati in questo modo sono molto prossimi a zero. Per il calcolo dei b_k la procedura è del tutto analoga:

```
bk = FX .* sin(K .* X);
bk = trapz(x,bk');
bk = bk./pi;
```

Per valutare ora la somma della serie è necessario ricostruire la griglia di calcolo, dal momento che `trapz` ha operato per le colonne di `FX'`. Dunque:

```
[FX,AK] = meshgrid(fx,ak);
[FX,BK] = meshgrid(fx,bk);
```

Non resta che costruire la matrice `SF`, dove per ogni riga si ha la k-esima approssimazione della funzione f:

```
SF = ao/2 * ones(size(x));
SF(2:(length(k)+1),:) = AK .*cos(K.*X) + BK .* sin(K.*X);
SF = cumsum(SF);
```

In questo caso l'elaborazione migliore sembra essere un filmino (si noti che ora estendiamo la f e la sua approssimazione fuori dall'intervallo $[-\pi, \pi,)$:

```
[r,c] = size(SF);                          Le dimensioni di SF
for cnt = 1:r                              per ciascuna riga...
p = plot(x,fx,'b',x,SF(cnt,:),'r',...      rappresenta la sua f(x) e la SF...
  x-2*pi,fx,'b',x-2*pi,SF(cnt,:),'r',...   anche per il periodo precedente...
  x+2*pi,fx,'b',x+2*pi,SF(cnt,:),'r');     e per il successivo
set(p,'LineWidth',2)                       spessore della linea
grid on                                    griglia
set(gca,'FontSize',14)                     dimensione font assi
xlabel('Asse X','FontSize',14)             asse x
ylabel('Asse Y','FontSize',14)             asse y
pause(1)                                   un secondo di pausa
end                                        fine dei giochi
```

La Figura 5.2 dà il risultato finale.

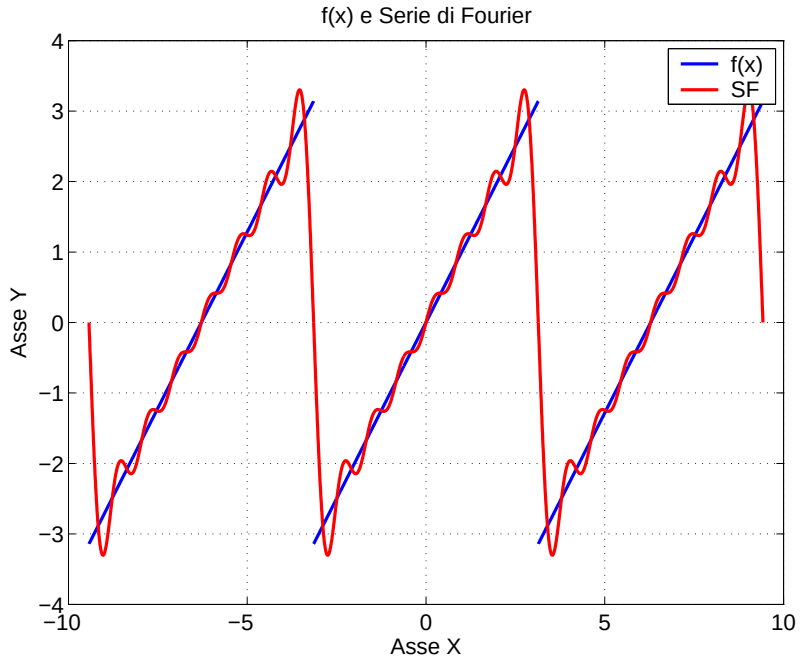


Figura 5.2: Sviluppo in serie di Fourier di una funzione dispari.

5.2 Istruzioni Logiche e Rappresentazione nel Piano delle Frequenze

Rappresentare una funzione nel piano delle frequenze significa rappresentare le ampiezze a_k e b_k dello sviluppo (5.1) in funzione delle frequenze ωk. Diversamente dal caso considerato nella Sezione 5.1 si prende ora in considerazione una funzione né pari né dispari, in modo che i coefficienti a_k e b_k non siano identicamente nulli.

Si consideri la funzione f definita in $[-\pi, \pi)$ da

$$f(x) = \begin{cases} x^2 & \text{se } x \in [-\pi, 0) \\ x & \text{se } x \in [0, \pi) \end{cases} \tag{5.5}$$

e prolungata per periodicità a tutto $\mathbb{R}$.

I valori delle ascisse sono definiti in maniera identica a quanto fatto in precedenza:

```
linspace(-pi,pi,100);
```

Per brevità intenderemo nel seguito come "dominio" l'intervallo $[-\pi, \pi)$; vedremo poi come estendere tutto quanto a $\mathbb{R}$. Il passo successivo è il calcolo di f. Si noti che l'espressione analitica di

f cambia a seconda che l'ascissa sia positiva o negativa. Per il calcolo di f è dunque necessario ricorrere ad *istruzioni logiche*; questo in ambiente MATLAB è piuttosto semplice.

Si digiti a riga di comando:

```
A = x>0
```

Ci si accorgerà che il vettore `A` è nullo dove l'istruzione logica non è verificata (per gli x negativi o nulli), mentre vale 1 laddove l'istruzione è verificata (per gli x positivi). Per implementare la funzione in esame è sufficiente scrivere:

```
fx = x.^2 .* (x<0) + x .* (x>=0)
```

Si noti che con la scrittura precedente abbiamo:

- valutato x^2 su tutto il dominio;
- moltiplicato x^2 per 1 sulle $x < 0$ e per 0 sulle $x > 0$;
- moltiplicato x per 1 sulle $x \geq 0$ e per 0 sulle $x < 0$;
- sommato infine i due risultati.

Si ricordi che in ambiente MATLAB le operazioni sono svolte *nell'ordine definito per le espressioni algebriche* (potenze, prodotti e divisioni, somme). La funzione f è 2π periodica, dunque $\omega = 1$. La procedura di calcolo è del tutto analoga al caso precedente:

```
k  = [1:7];                          si definisce k
[X,K] = meshgrid(x,k);               griglia x-k
[FX,K] = meshgrid(fx,k);             griglia f(x)-x
ao = 1/pi * trapz(x,fx);             calcolo a_0, trapz su di un vettore
ak = FX .* cos(K .* X);              funzione integranda
ak = trapz(x,ak');                   integrale
ak = ak./pi;                         termine di sporcizia
bk = FX .* sin(K .* X);              stessa...
bk = trapz(x,bk');                   ...filastrocca...
bk = bk./pi;                         ...sui b_k
[FX,AK] = meshgrid(fx,ak);           griglia f(x) - a_k
[FX,BK] = meshgrid(fx,bk);           griglia f(x) - b_k
SF = ao/2 * ones(size(x));           termine a_0/2
SF(2:(length(k)+1),:) = ...          successione di...
  AK .*cos(K.*X) + BK .* sin(K.*X);  ...polinomi trigonometrici
SF = sum(SF);                        serie di Fourier!
```

Si osservi che non a caso questa volta è stato utilizzato il comando `sum`: non si è voluto sovrapporre sul medesimo grafico i diversi gradi di approssimazione, in quanto il risultato sarebbe stato piuttosto confuso. Si vuole ora creare una finestra grafica con due sotto-finestre: nella prima abbiamo il grafico di f e la sua approssimazione, nella seconda rappresentiamo le ampiezze in funzione delle frequenze. Per dividere la finestra grafica sarà nuovamente utilizzato il comando `subplot`, mentre per realizzare un diagramma a barre verticali è possibile utilizzare il comando `bar`. Anche `bar` lavora per colonne, dunque si avrà `bar([ak;bk]')`. Il listato utilizzato per elaborare la Figura 5.3 (si noti che ora estendiamo la f e la sua approssimazione fuori dall'intervallo $[-\pi, \pi,)$ è:

```
subplot(2,1,1)                       due righe ed una colonna
p = plot(x,fx,'b',x,SF,'r',...       tre ...
  x-2*pi,fx,'b',x-2*pi,SF,'r',...    ...pezzi...
  x+2*pi,fx,'b',x+2*pi,SF,'r');      ...di funzione
set(p,'LineWidth',2)                 spessore della linea
grid on                              griglia
set(gca,'FontSize',14)               font per gli assi
title('f(x) e Serie di Fourier',...  titolo
  'FontSize',14)                     font per il titolo
xlabel('Asse X','FontSize',14)       asse x
```

```
ylabel('Asse Y','FontSize',14)          asse y
legend('f(x)','SF')                     legenda
subplot(2,1,2)                          seconda finestra
bar([ak;bk]')                           un diagramma a barre, semplicissimo!
grid on                                 griglia
set(gca,'FontSize',14)                  font degli assi
title('Piano delle Frequenze',...       titolo
  FontSize',14)                         font del titolo
xlabel('k','FontSize',14)               asse delle frequenze
ylabel('a_k, b_k','FontSize',14)        asse delle ampiezze
legend('a_k','b_k')                     legenda
```

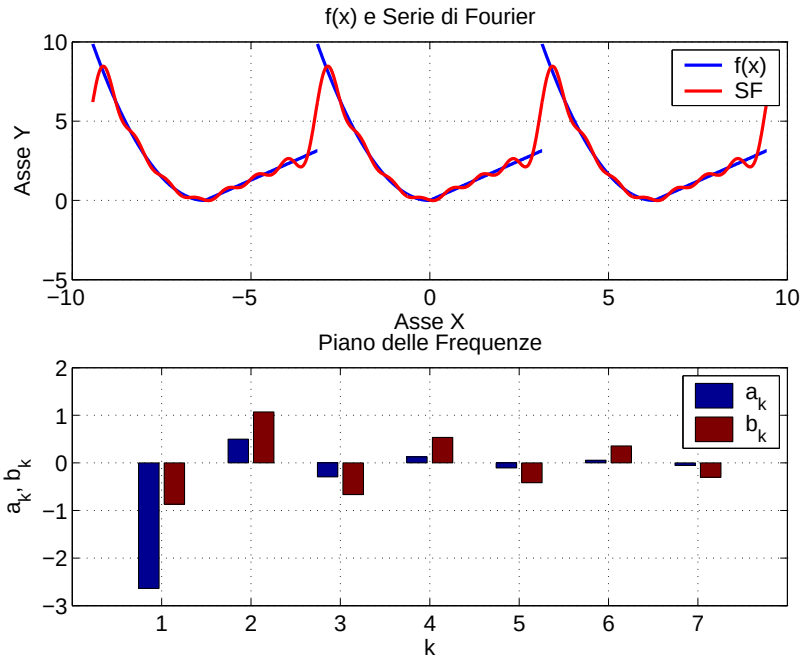


Figura 5.3: La funzione (5.5), la sua approssimazione con polinomi trigonometrici, i coefficienti nel piano delle frequenze.

Esercizio 5.1 Nella rappresentazione nel piano delle frequenze per semplicità non si è riportato il termine di media $a_0/2$. Modificare il listato per includervelo.

Esercizio 5.2 Molto spesso in ambiente ingegneristico non si riportano nel piano delle frequenze tanto le ampiezze a_k, b_k quanto i loro valori assoluti. Modificare il listato precedente per rappresentare $|a_k|$, $|b_k|$.

Esercizio 5.3 Negli esempi precedenti siamo partiti da una funzione (che possiamo pensare rappresentare un segnale temporale) e ne abbiamo fatto l'analisi in frequenza. Si provi a fare il contrario: si considerino due successioni $\{a_n\}$, $\{b_n\}$ infinitesime (Riemann-Lebesgue!) e si costruisca, adoperando un numero finito di termini delle successioni, una approssimazione della funzione che ha a_n e b_n come coefficienti di Fourier.

5.3 Il Fenomeno di Gibbs

Esaminiamo in questa sezione il *fenomeno di Gibbs*: l'approssimazione di una funzione discontinua con un polinomio trigonometrico (una funzione continua!) produce un errore che all'aumentare dell'approssimazione si localizza sempre più vicino al punto di discontinuità, ma non diminuisce in

valore assoluto e resta asintoticamente pari a circa il 10% del valore assoluto del salto della funzione nel punto di discontinuità. Tutto questo viene implementato ora utilizzando l'errore relativo

$$E_n(x) = \frac{\left| f(x) - \left[\frac{a_0}{2} + \sum_{k=1}^{k=n} \left(a_k \cdot \cos(\omega k x) + b_k \cdot \sin(\omega k x) \right) \right] \right|}{f(x)}.$$

Laddove la funzione è nulla l'errore, per come è definito, potrebbe essere "infinito". In ambiente MATLAB questo non dà alcun problema, in quanto produce il risultato `Inf` che è ignorato in fase di elaborazione. Si vuole ottenere un'elaborazione dove in una finestra grafica superiore sia rappresentata la funzione con la sua approssimazione, mentre nella finestra inferiore sia invece rappresentato l'errore $E_n(x)$.

Per illustrare il fenomeno di Gibbs consideriamo la funzione pari "onda quadra" definita in $[-\pi, \pi)$ da

$$f(x) = \begin{cases} 1 & \text{se } x \in [-\pi/2, \pi/2] \\ 0 & \text{se } x \in [0, -\pi/2) \cup (\pi/2, \pi) \end{cases} \tag{5.6}$$

e poi prolungata per periodicità a $\mathbb{R}$. Per ottenere il grafico di f e la sua approssimazione di Fourier è sufficiente sostituire nel codice sviluppato nella Sezione 5.2 l'espressione della funzione:

```
fx = 1 .* (x>=-pi/2 & x<=pi/2);
```

In Figura 5.4 è riportato il grafico di f e una sua approssimazione; abbiamo rappresentato anche i coefficienti: i b_k sono tutti nulli così come gli a_k relativi ad indici pari. Nel caso in esame il salto di f nei punti di discontinuità vale 1 e dunque ci attendiamo un errore vicino a tali punti di circa 0.1.

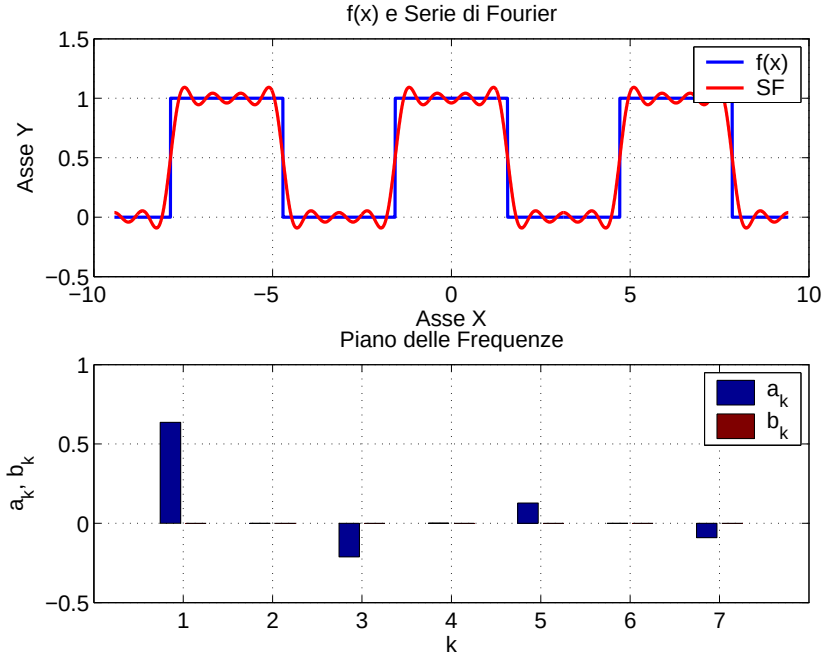


Figura 5.4: Approssimazione in serie di Fourier di una funzione pari.

Per l'analisi del fenomeno di Gibbs modifichiamo come segue il codice elaborato precedentemente. In coda alla fase di calcolo, calcoliamo l'errore $E_n(x)$:

```
En = abs((SF-fx))./fx;
```

Limitiamo poi gli assi della prima finestra grafica `subplot(2,1,1)`:

```
axis([0,1.6,.5,1.2])
```

Nella seconda finestra `subplot(2,1,2)`:

```
plot(x,En,'g','LineWidth',2)              il grafico
axis([0,1.6,0,.1])                        limiti degli assi
grid on                                   la solita griglia
set(gca,'FontSize',14)                    i soliti font
title('Errore Relativo','FontSize',14)    un titolo
xlabel('Asse X','FontSize',14)            etichetta x
ylabel('Asse Y','FontSize',14)            etichetta y
```

In Figura 5.5 ci si è spinti fino ad $n = 100$. Per un'approssimazione di questo livello si noti come l'errore relativo sia inferiore all'1% per $x < 1.25$, mentre aumenti fino al 10% in prossimità della discontinuità. Inoltre la Figura 5.6 rappresenta il fenomeno di Gibbs per diversi ordini di approssimazione; si noti come l'errore relativo sia quasi costante al variare di k, per k abbastanza grande.

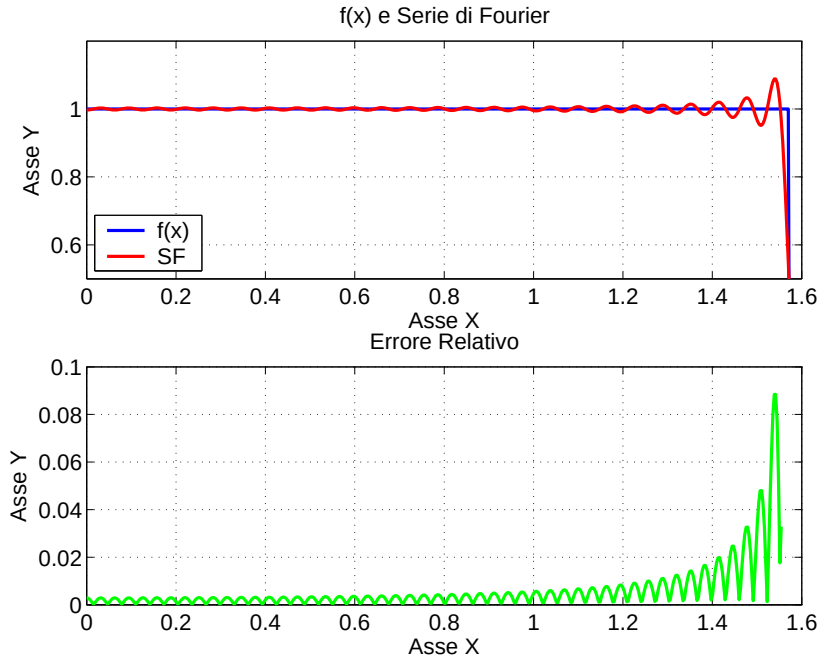


Figura 5.5: La funzione (5.6), il suo sviluppo all'ordine 100 e l'errore relativo.

Esercizio 5.4 Riprodurre la Figura 5.6, partendo dal listato appena scritto è piuttosto semplice senza fare ricorso ad alcun ciclo `for`; si ricordi bene che `sum` ha un parente piuttosto stretto...

5.4 Un `function m-file` per il calcolo dei coefficienti di Fourier

Il codice scritto nella Sezione 5.1 è particolarmente flessibile in quanto è infatti sufficiente cambiare l'espressione della funzione f per calcolarne numericamente i coefficienti e avere una approssimazione di Fourier. Per conferire al listato una maggiore generalità lo si può cambiare in un `function m-file` e, già che ci siamo, implementarlo per una generica funzione T-periodica.

Se si vuole passare da un codice valido per una funzione 2π-periodica ad una funzione di periodo generico sono necessari i seguenti cambiamenti all'interno dello script:

```
T = 10;                          T = 10
omega = 2*pi ./ T;               ω = 2π/T
x = linspace(-pi,pi,1000);       → x = linspace(-T/2,T/2,1000);
...                              ...
ak = FX.*cos(K.*X);              → ak = FX.*cos(omega.* K.* X)
ak = trapz(x,ak');               resta uguale a se stesso
```

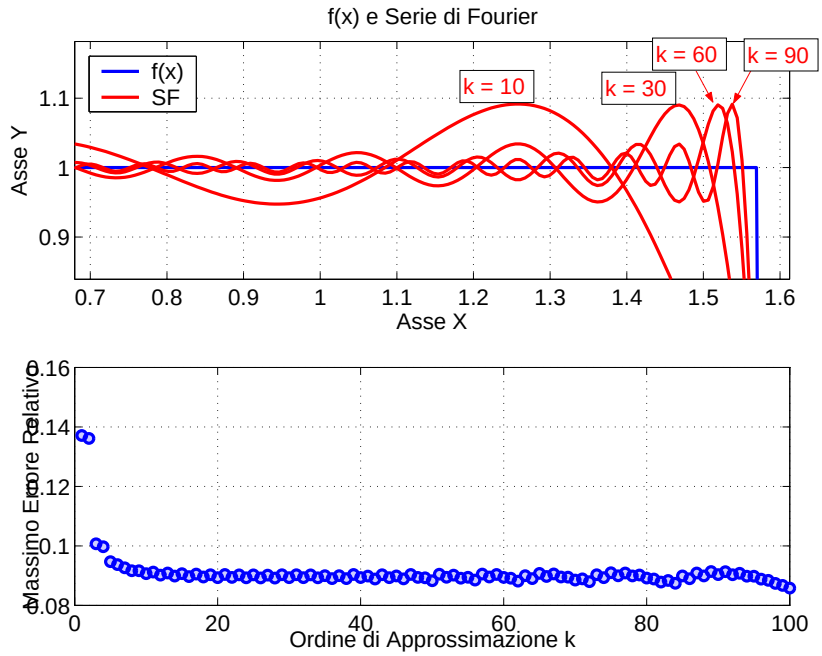


Figura 5.6: Rappresentazione del fenomeno di Gibbs per diversi ordini di approssimazione.

```
ak = ak./pi;                    → ak = 2 .* ak ./ T;
bk = FX.*sin(K.*X);             → bk = FX.*sin(omega.* K.* X)
bk = trapz(x,bk');              resta uguale a se stesso
bk = bk./pi;                    → bk = 2 .* bk ./ T;
```

Infine deve essere corretta la serie di Fourier:

```
SF(2:(length(k)+1),:) = AK .*cos(K.*X) + BK .* sin(K.*X);
```

diventa:

```
SF(2:(length(k)+1),:) = ...
AK .*cos(omega * K.*X) + BK .* sin(omega * K.*X);
```

A questo punto rieseguendo lo script si noterà che non sarà cambiato nulla salvo il periodo della funzione $f(x)$.

Ora da uno `script file`, si vuole passare ad un `function m-file`. In questo caso vogliamo essere particolarmente raffinati. Vogliamo avere come argomento della `function` soltanto il vettore delle ascisse `x` ed il vettore che definisce la funzione `fx`. Questo ha diversi vantaggi, fra tutti:

- rendere più leggibile il listato,
- controllare le variabili direttamente dal file principale, senza entrare in quelle che in modo colto sono definite *subroutine*.

La funzione avrà dunque l'aspetto:

```
[ak,bk] = CoeffsFourier(fx,x);
```

Resta da chiarire come fare per definire il valore del periodo T, la frequenza ω e l'ordine di approssimazione k all'interno della subroutine. Esistono naturalmente moltissimi modi, tutti allo stesso modo validi. In questo caso vogliamo definire le varibili `T`, `omega` e `k` come variabili *globali*. Il file principale e le sue subroutine hanno uno specifico insieme di variabili, ciascuno indipendente dagli altri. Per fare in modo che alcune variabili siano riconosciute ad ogni livello queste devono essere dichiarate globali, per farlo, in ambiente MATLAB, si utilizza il comando `global`. Oltre che essere *dichiarate* globali, esse devono essere anche *richiamate* nella subroutine.
Le prime righe della nostra `function` avranno dunque l'aspetto:

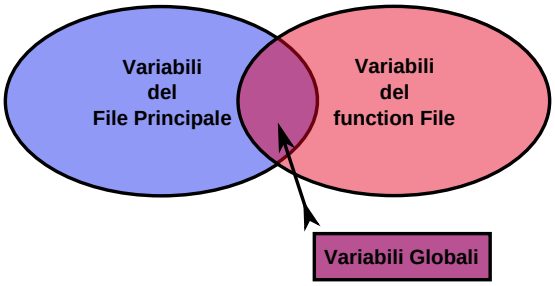


Figura 5.7: Variabili definite in un codice e nelle sue subroutine.

```
function [ao,ak,bk] = CoeffsFourier(fx,x);
global k T omega
```

In coda a queste è sufficiente un brutale *cut* & *paste* delle righe dove sono calcolati i coefficienti di Fourier:

```
[X,K] = meshgrid(x,k);              griglia x-k
[FX,K] = meshgrid(fx,k);            griglia f(x)-x
ao = 2/T * trapz(x,fx);             a_0
ak = FX .* cos(omega .* K .* X);    funzione integranda
ak = trapz(x,ak');                  integrale
ak = 2 .* ak ./ T;;                 a_k
bk = FX .* sin(omega .* K .* X);    stessa...
bk = trapz(x,bk');                  ...filastrocca...
bk = 2 .* bk./T;                    ...sui b_k
```

Infine per utilizzare la `function` appena creata è necessario dichiarare le variabili *globali*:

```
global k T omega
```

Si presti attenzione che quando il codice diventa complesso questa operazione è particolarmente delicata perché non tutte le subroutine richiamano le stesse variabili: l'operazione di *cut* & *paste* non sempre è efficiente e non ci tutela da errori. Infine è necessario un piccolo cambiamento nelle ultime righe:

```
[FX,BK] = meshgrid(fx,bk);    →   [X,BK] = meshgrid(x,bk);
```

Ora ci si può scatenare.

Capitolo 6

Equazioni alle Derivate Parziali

In questo capitolo consideriamo brevemente due equazioni alle derivate parziali: l'equazione di diffusione del calore e quella relativa alle oscillazioni di una membrana elastica. Nel primo caso la soluzione è approssimata da una serie di Fourier, nel secondo, tramite una riduzione dimensionale, fanno il loro ingresso le funzioni di Bessel.

6.1 L'Equazione del Calore

La diffusione del calore in una sbarra di lunghezza finita L può essere descritta da una *equazione alle derivate parziali* e dalle sue *condizioni iniziali ed al contorno*. L'equazione in esame è l'**Equazione di Fourier**[1] e il relativo problema si scrive

$$\begin{cases} \partial_t u - \alpha \cdot \partial_{xx} u = 0 & (t,x) \in (0,+\infty) \times [0,L] \quad \text{EQ} \\ u(t,0) = u(t,L) = 0 & t \in (0,+\infty) \quad \text{CB} \\ u(0,x) = h(x) & x \in [0,L] \quad \text{CI}\,. \end{cases} \tag{6.1}$$

Il modello fisico soggiacente a tale equazione è rappresentato schematicamente in Figura 6.1. Si noti che la temperatura agli estremi della sbarra è mantenuta nulla; di conseguenza ci si attende dal modello che la distribuzione iniziale di temperatura $h(x)$ evolva secondo una legge $u(t,x)$ che per conduzione tende ad annullarsi per $t \to +\infty$.

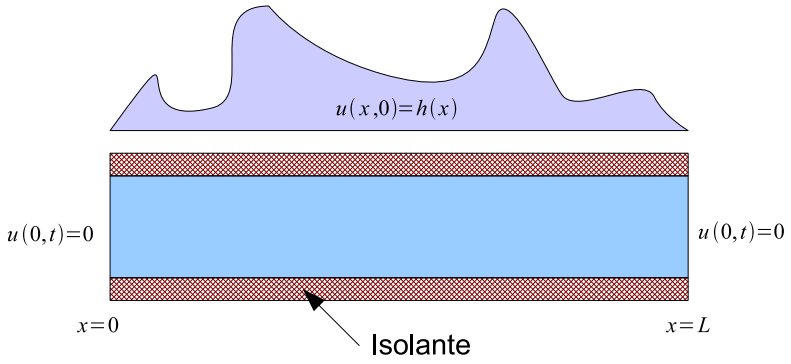


Figura 6.1: La fisica modellata dall'equazione di Fourier.

Di seguito si ripercorre brevemente la trattazione analitica per la soluzione dell'equazione di Fourier.

1. Si prolunga h in $[-L,0]$ in maniera dispari ed la si estende poi su tutto $\mathbb{R}$ per periodicità. Ne segue lo sviluppo in serie di Fourier:

$$h(x) = \sum_{k=1}^{\infty} b_k \sin\left(\frac{k\pi x}{L}\right), \qquad b_k = \frac{2}{L}\int_0^L \sin\left(\frac{k\pi x}{L}\right) \mathrm{d}x\,.$$

[1] Jean Baptiste Joseph Fourier nato nel 1768 ad Auxerre, morto nel 1830 a Parigi.

2. Posto
$$h_k(x) = b_k \sin\left(\frac{k\pi x}{L}\right)$$
la linearità[2] della (6.1) ci consente di affermare che se $u_k(t,x)$ è la soluzione del problema elementare
$$\begin{cases} \partial_t u_k - \alpha \cdot \partial_{xx} u_k = 0 \\ u_k(t,0) = u_k(t,L) = 0 \\ u_k(0,x) = h_k(x), \end{cases} \tag{6.2}$$
allora $u(t,x) = \sum_{k=1}^{\infty} u_k(t,x)$, ottenendo così una soluzione per serie di funzioni.

3. Si cerca la soluzione $u_k(t,x)$ di (6.2) per separazione di variabili:
$$u_k(t,x) = f_k(t) \cdot h_k(x). \tag{6.3}$$

Procediamo dunque con la soluzione del problema. Sostituendo la (6.3) in (6.2) si ottiene il problema ai valori iniziali per $f_k(t)$
$$\begin{cases} f_k' + \alpha k^2 f_k = 0 \\ f_k(0) = 1, \end{cases}$$
la cui soluzione è $f_k(t) = e^{-\alpha k^2 t}$. Si ottiene dunque
$$u(x,t) = \sum_{k=1}^{\infty} u_k(x,t) = \sum_{k=1}^{\infty} \underbrace{e^{-\alpha k^2 t}}_{f_k(t)} \cdot \underbrace{b_k \sin\left(\frac{k\pi x}{L}\right)}_{h_k(x)}. \tag{6.4}$$

6.1.1 Implementazione della Soluzione

In Tabella 6.1 sono riportate in maniera approssimativa[3] le caratteristiche termodinamiche di alcuni materiali utili per simulare il comportamento della sbarra. Il parametro α vale:
$$\alpha = \frac{\lambda}{\rho \cdot c_p}$$

Materiale	ρ	λ	c_p	α
	$\mathrm{kg/m^3}$	$\mathrm{W/m \cdot K}$	$\mathrm{J/m \cdot K}$	$\mathrm{m^2/s}$
gomma vulcanizzata	1100	0.13	2010	$5.88 \cdot 10^{-8}$
abete	420	0.14	2720	$1,23 \cdot 10^{-7}$
calcestruzzo	2300	1.4	880	$6.92 \cdot 10^{-7}$
acciaio	7854	60.5	434	$1,77 \cdot 10^{-5}$

Tabella 6.1: Proprietà di alcuni materiali.

Si consideri $L = 5$ m e si inizi un nuovo `m-file` dichiarando le variabili *globali*:

```
clear all                spazzola...
close all                ...e chiudi
global T omega k         variabili globali
L = 5;                   la sbarra
alpha = 1.23e-7;         o meglio, un tronco di abete
x = linspace(0,L,50);    non serve un'ascissa troppo raffinata
time = [0:1e+6:4e+6];    ci vuole un po' di tempo...più di 40 giorni
T = 2*L;                 il periodo
omega = 2*pi ./T;        ω = 2·π/T
k = [1:7];               l'ordine di approssimazione
```

[2] Il mondo non è lineare.
[3] In generale calore specifico, densità e conduttività dipendono dalla temperatura, ma se si tenesse conto di ciò il problema non sarebbe più lineare.

Si consideri poi una condizione iniziale semplice, cioè una parabola:

$$h(x) = x \cdot (L - x)$$

che implementata diventa:

```
hx = x .* (L-x);
```

Ora è necessario prolungare in maniera dispari il dominio. Per farlo si utilizza il comando `fliplr` che inverte (flip) da destra a sinistra gli elementi dell'array (left right):

```
xx = [fliplr(-x),x];
hhx = [fliplr(-hx),hx];
```

Si noti che è essenziale il segno meno (prolungamento dispari). Avendo creato una `function` il calcolo dei coefficienti di Fourier è immediato:

```
[ao,ak,bk] = CoeffsFourier(xx,hhx);
```

Si tenga presente che la (6.4) dipende da tre variabili (x, t, k); in questo caso dunque la griglia di calcolo è tridimensionale:

```
[X,TIME,K] = meshgrid(x,time,k);
[X,TIME,BK] = meshgrid(x,time,bk);
```

Ad ogni "strato" dell'array corrisponde un valore di k. Sulla griglia si valuta la soluzione:

```
Temperatura = BK .* sin( omega .* K .* X ) .*...
   exp( -alpha .* K.^2 .* TIME );
```

E come per le torte meglio riuscite, si fa la somma per strati:

```
Temperatura = sum(Temperatura,3);
```

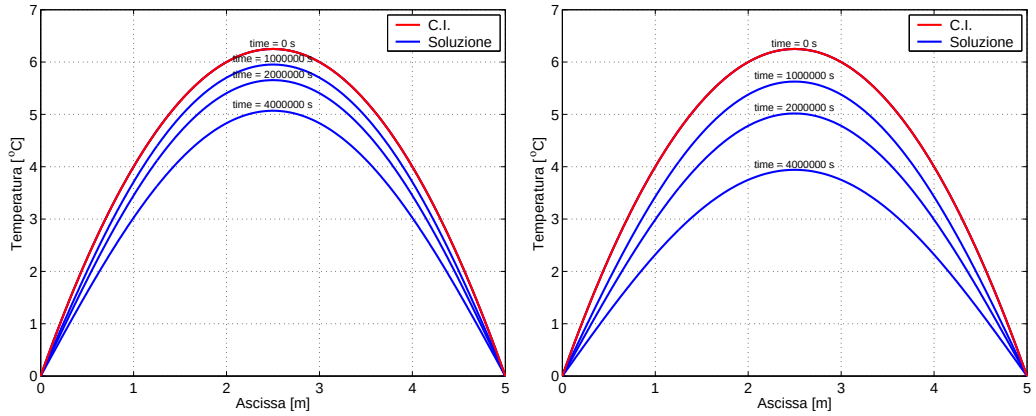
Il comando `sum(`*argomento*`,dim)` permette di specificare la dimensione secondo la quale fare la somma:

- righe → 1
- colonne → 2
- strati → 3

Infine non restano che le elaborazioni grafiche:

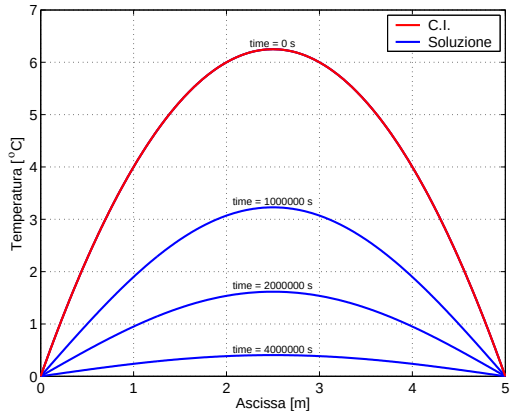
```
G = plot(x,Temperatura,'b',x,hx,'r');        condizione iniziane e soluzione
set(G, 'LineWidth', 2)                       spessore della linea
set(gca,'FontSize', 14)                      dimensione del font
xlabel('Ascissa [m]')                        ascissa
ylabel('Temperatura [ ^\oC]')                ordinata
title('Abete')                               l'albero
grid on                                      la griglia
for cnt = 1:length(time)                     ∀ istante
[temp,id] = max(Temperatura(cnt,:));         un'etichetta
text(x(id),temp,...                          posizionata sul massimo,
['time = ',num2str(time(cnt)),' s'],...      con segnato l'istante,
  'VerticalAlignment','bottom',...           allineata in basso
  'FontSize',14)                             e grande 14 punti
end                                          FINE
legend('Condizione Iniziale','Soluzione')    legenda
```

Il risultato è in Figura 6.1.1. Si noti la lentezza della diffusione della temperatura.

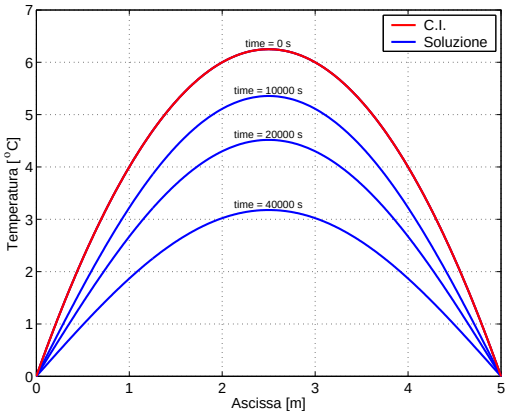


(a) Gomma vulcanizzata

(b) Abete



(c) Calcestruzzo



(d) Acciaio

Figura 6.2: Confronto fra le risposte fornita da diversi materiali. Per i primi tre materiali la curva più bassa si riferisce a circa 46 giorni, per l'acciaio a circa 11 ore.

6.1.2 Una Condizione Iniziale un Po' Strana...

Col listato predente si è creato un codice piuttosto potente, nel senso che l'unico limite alla raffinatezza della griglia è la potenza della macchina. Di seguito si prova a vedere cosa accade ad una barra di acciaio ponendo l'ordine di approssimazione a 100, ed implementando una condizione iniziale un po' strana (discontinua):

$$f(x) = \begin{cases} 0 & \text{se } x \in [0,1] \cup [2,3] \cup [4,5] \\ 1 & \text{se } x \in [1,2] \cup [3,4]\,. \end{cases} \tag{6.5}$$

Guardando la Figura 6.3 si possono fare diverse riflessioni:

- Avendo approssimato la soluzione con serie di Fourier troncate è naturale che per condizioni iniziali discontinue si noti il fenomeno di Gibbs.
- La temperatura ha immediatamente un aspetto regolare; tende dapprima ad avere un andamento uniforme e in seguito comincia, uniformemente, a decrescere a zero.

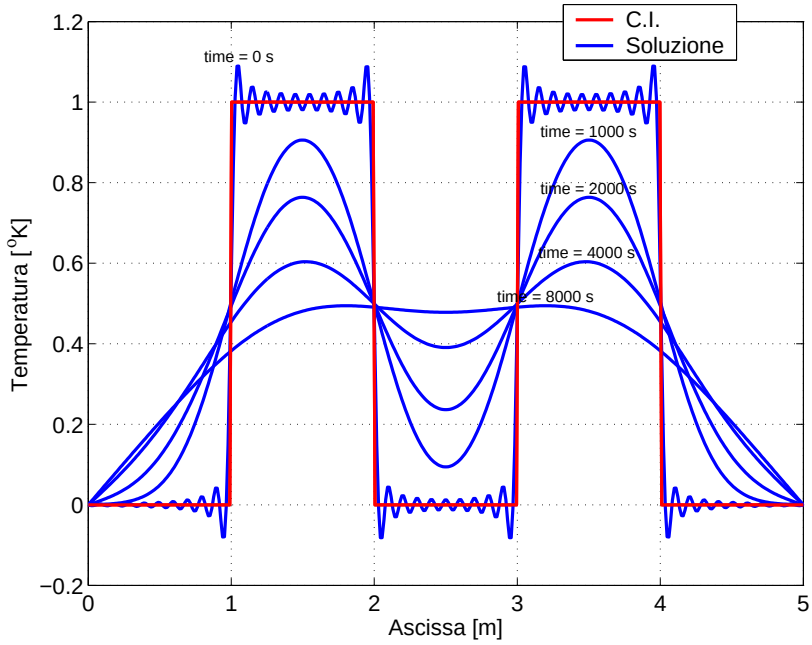


Figura 6.3: Soluzione dell'equazione di Fourier per una condizione iniziale discontinua.

6.2 La Membrana Vibrante a Simmetria Radiale

In questa sezione studiamo le oscillazioni di una membrana elastica circolare sotto una ipotesi di simmetria radiale del dato iniziale. Viene premesso un semplice codice per il calcolo degli zeri di una funzione che verrà utilizzato in seguito.

6.2.1 Preludio: un Function m-file per il Calcolo degli Zeri di una Funzione

Brutalmente viene riportato solo il listato, dal momento che questo codice non è né particolarmente originale (fa solo un'interpolazione lineare), né particolarmente fine dal punto di vista della programmazione:

```
function z=ZERI(J,x) for k=1:(length(J)-1)
    cerco=sign(J(k))*sign(J(k+1));
    if cerco<0%.....................Cambio di Segno
        DJ=abs(J(k))+abs(J(k+1));
```

```
        Dx=abs(x(k+1)-x(k));
        dJ=abs(J(k));
        dx=dJ/DJ*Dx;
        z(k)=x(k)+dx;
    else
        z(k)=0;
    end
end z=(nonzeros(z))';
```

6.2.2 Il problema

Ricordiamo brevemente come si pone il problema. L'equazione alle derivate parziali che regola le oscillazioni di una membrana elastica è

$$\partial_{tt} u - c^2 \Delta u = 0. \tag{6.6}$$

Assumeremo che la membrana sia circolare, di raggio R, e che inizialmente anche la sua posizione sia radiale; è naturale allora cercare una soluzione che presenti la stessa simmetria. L'operatore Laplaciano Δu in coordinate polari r-θ si scrive come:

$$\Delta u = \partial_{rr} u + \frac{1}{r}\partial_r u + \frac{1}{r^2}\partial_{\theta\theta} u\,. \tag{6.7}$$

Sostituendo l'espressione (6.7) nella (6.6) e tenendo conto dell'annullarsi di qualsiasi derivata rispetto a θ (simmetria radiale), si ottiene:

$$\partial_{tt} - c^2 \left(u\partial_{rr} u + \frac{1}{r}\partial_r u \right) = 0\,. \tag{6.8}$$

Il problema completo delle condizioni al contorno ed iniziali risulta quindi, nel caso di velocità iniziale nulla,

$$\begin{cases} \partial_{tt} u - c^2 \left(\partial_{rr} u + \frac{1}{r}\partial_r u\right) = 0 & \\ u(t,R) = 0 & \text{CB} \\ u(0,r) = h(r) \quad \partial_t u(0,r) = 0 & \text{CI.} \end{cases} \tag{6.9}$$

La soluzione si scrive come

$$u(t,r) = \sum_{n=1}^{\infty} a_n \cos(k_n c t) \cdot J_0(k_n r)\,. \tag{6.10}$$

Qui J_0 è la funzione di Bessel di ordine 0, $k_n = x_n/R$ dove gli x_n sono gli zeri positivi di J_0, le costanti a_n (*coefficienti di Bessel-Fourier* di h) sono definite da

$$a_n \quad = \quad \frac{2}{[J_1(k_n R)]^2} \int_0^1 h(r) r J_0(k_n r) \mathrm{d}r \tag{6.11}$$

dove $J_1 = -J_0'$ è la funzione di Bessel di ordine 1. Le costanti a_n rappresentano h nel senso che

$$h(r) = \sum_{n=1}^{\infty} a_n J_0(k_n r)\,. \tag{6.12}$$

Passiamo ora all'implementazione del codice. In primo luogo è necessario definire il raggio della membrana (che per semplicità prenderemo uguale a 1), i vettori distanza dall'origine e tempo, la celerità c (anch'essa uguale a 1):

```
raggio=1;
r=linspace(0,raggio,50);
time=linspace(0,20,200);
c = 1
```

Si definisce poi un'ascissa per la funzione di Bessel e la funzione di Bessel stessa, tramite il comando `besselj(`*ordine*`,`*ascissa*`)`:

```
mu=linspace(0,20,1000);
J=besselj(0,mu);
```

Utilizzando la function definita al paragrafo precedente sono poi calcolati gli zeri della funzione J_0:

```
mu_n = ZERI(J,mu);
k_n = mu_n./raggio;;
```

Si definisca la condizione iniziale, in questo caso una semplice parabola:

```
CI = (1+r).*(1-r);
```

Per calcolare gli a_k si ricorre ad una griglia di calcolo, come fatto per l'equazione del calore:

```
Ji = besselj(1,kappa_n);          definisco J_1(k_n)
[R,KN] = meshgrid(r , k_n);       raggio e k_n
[ic,JI] = meshgrid(CI , Ji););    CI e J_1(k_n)
B = R.*ic;                        spezzo il calcolo degli a_k, per comodità
A = 2 ./ (Ji.^2);                 il termine fuori integrale
C = besselj(0, (KN .* R) );       un pezzo di integrale
ARGint = B.*C;                    l'argomento dell'integrale
INT = trapz(r,ARGint');           l'integrale
A_n = A .* INT;                   gli a_n
```

Per calcolare la soluzione è poi necessario ricorrere ad una griglia tridimensionale:

```
[R,TIME,KN] = meshgrid(r,time, k_n);    raggio, tempo, k_n
[R,TIME,AN] = meshgrid(r,time, A_n);    raggio, tempo, a_n
COS = cos(c*KN.*TIME);                  un pezzo di soluzione
JO = besselj(0,KN.*R);                  l'altro pezzo
Un=AN.*COS.*JO;                         tutti i pezzi insieme
UrhoTime = sum(Un,3);                   tutti gli strati sommati
```

Per rappresentare correttamente un problema a simmetria assiale è bene costruire un sistema di coordinate polari:

```
theta = linspace(0,2*pi,length(r));   l'angolo θ
[Rho,Theta] = meshgrid(r,theta);;     raggio, θ
X = Rho.*cos(Theta);                  array delle x
Y = Rho.*sin(Theta);                  array delle y
```

Infine, una riga alla alla volta si procede col filmino:

```
for cnt=1:length(time)                          ∀ istante
[Uistant,PP]=meshgrid(UrhoTime(cnt,:),rho);     lo spostamento è costante con r
surf(X,Y,Uistant);                              La Pelle del Tamburo
camlight                                        una luce nel buio
shading('interp')                               ombreggia interpolando
axis([-1,1,-1,1,-.8,.7,-1.3,1.3])               tieni fermi gli assi
end                                             fine dei giochi
```

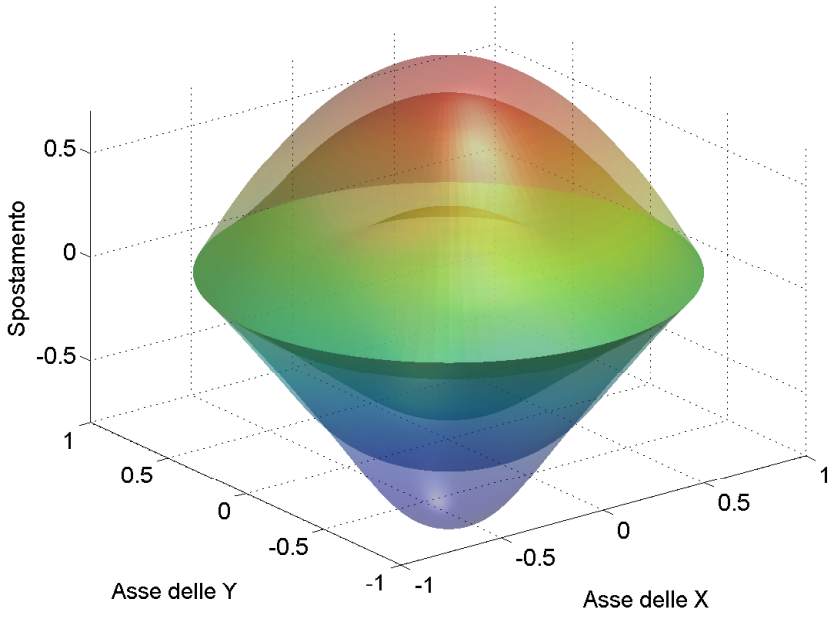


Figura 6.4: Rappresentazione per diversi istanti della soluzione dell'equazione del tamburo vibrante.

TAMBURO VIBRANTE

$$\partial_{tt}u - c^2\ \partial_{rr}u + \tfrac{1}{r}\partial_r u\ = 0$$
$$\text{CI}: \ u(0,r) = u_0(r), \quad \partial_t u(0,r) = 0$$

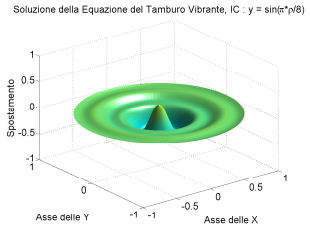
$$u = \sum_{n=1}^{\infty} a_n \cos(k_n c t) \cdot J_0(k_n r)$$

$$\text{CB}: J_0(k_n R) = 0$$

$$\text{CI}: u(r;0) = \sum_{n=1}^{\infty} a_n J_0(k_n r)$$

$$a_n = \frac{2}{[J_1(k_n R)]^2} \int_0^1 h(r) r J_0(k_n r)\mathrm{d}r$$

INTEGRAZIONE P.D.E. COMPLETATA



SVILUPPO DEL CODICE

$r =$ `linspace(...)`
$t =$ `linspace(...)`

$\mu = kr =$ `linspace(...)`
$J_0 =$ `besselj`$(0, \mu)$

$J_0(\mu_n) = 0 \Rightarrow \mu_n =$`ZERI`$(J_0, \mu)$
$k_n = \mu_n / R$

$J_1(k_n) =$ `besselj`$(1, k_n)$
`[R,KN]=meshgrid(`r`,`k_n`);`
`[ic,JI] = meshgrid(`u_0`,`$J_1(k_n)$`);`

`ARGint = R.*ic.* besselj(0,(KN.*R))`
`INT=trapz(ARGint,`r`);` $A_n =$ `2*INT./JI`^{2}

`[R,T,KN]=meshgrid(`r`,`t`,`k_n`)`
`[RHO,T,AN]=meshgrid(`r`,`t`,`A_n`)`

`Un=AN.*cos(c*T*KN).*besselj(0,KN.*R)`
U_{rt}`=sum(Un,3)`

$\theta =$`linspace(...)`
$(\theta, r) \rightarrow (x, y)$

```
for cnt=1:length(t)
[Ui,PP]=meshgrid(Urt(cnt,:),r)
surf(X,Y,Ui)
end
```

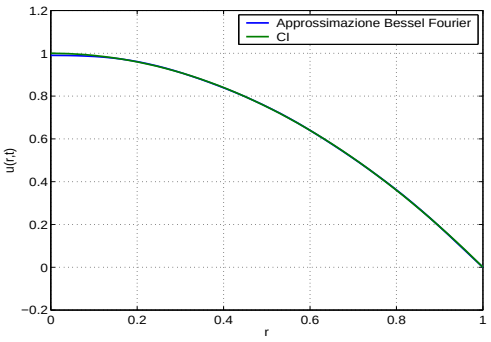
Figura 6.5: Diagramma di flusso per l'integrazione dell'equazione del tamburo vibrante

6.2.3 Approssimazione di Alcune Condizioni Iniziali

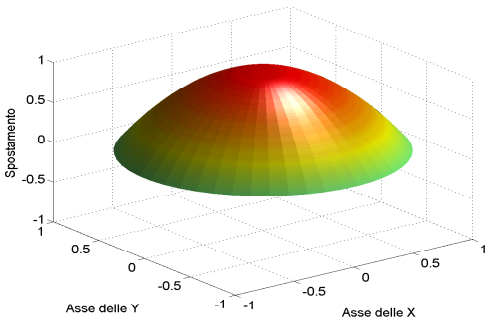
La costruzione di una soluzione al problema (6.9) effettuata nella sezione 6.2.2 prevede la decomposizione in serie di Bessel-Fourier del dato iniziale h, (6.12). In modo analogo a quanto fatto nel Capitolo 5 analizziamo brevemente in questa sezione tale approssimazione per alcune condizioni iniziali h. Questo ci permetterà di analizzare una approssimazione diversa da quelle per polinomi (serie di potenze) o polinomi trigonometrici (serie di Fourier) considerate precedentemente e contemporaneamente sarà un esempio di grafica per funzioni di due variabili reali. Consideriamo dapprima il caso $h(r) = 1 - r^2$. L'approssimazione per serie troncate di Bessel-Fourier è da considerarsi soddisfacente: la Figura 6.6(a) mostra infatti come la funzione e la sua approssimazione risultino sostanzialmente coincidenti. In Figura 6.6(b) il grafico dell'approssimazione di h nello spazio.

Il caso $h(r) = 1 - r^8$ è mostrato in Figura 6.6(c). Per quanto riguarda l'approssimazione si notino le oscillazioni per $r \to 0$.

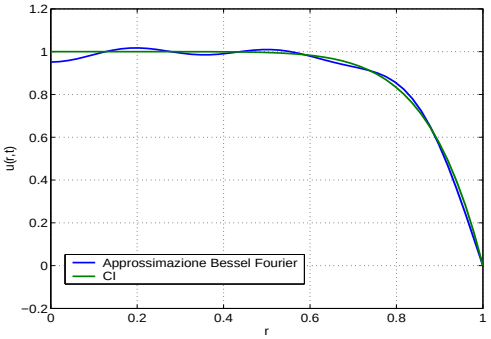
Infine il caso $h(r) = \cos(\frac{5\pi}{2}r)$ è visualizzato in Figura 6.6(e). La funzione h si annulla per $r = 1$ ed vale 1 in zero, dunque è sufficiente $n = 6$ per ottenere un'ottima concordanza fra $h(r)$ e la sua approssimazione.



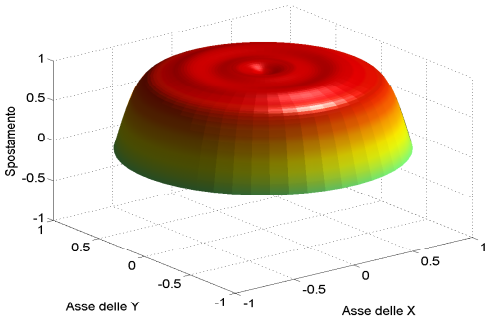
(a) Dettaglio $r \to h(r)$



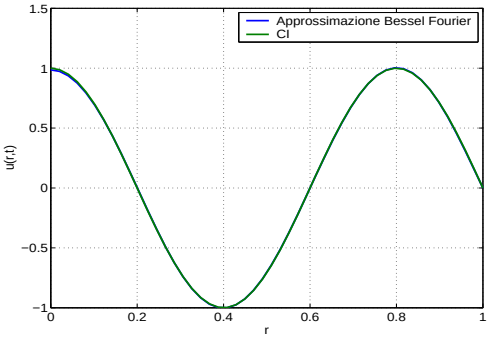
(b) L'approssimazione di $(x, y) \to h(x, y)$



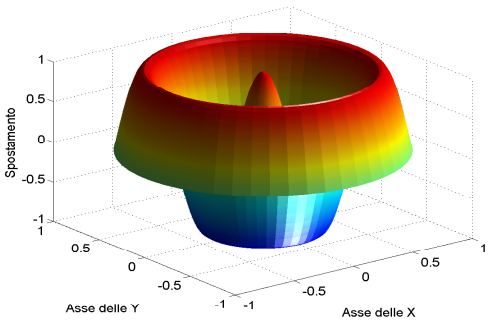
(c) Dettaglio $r \to h(r)$



(d) L'approssimazione di $(x, y) \to h(x, y)$



(e) Dettaglio $r \to h(r)$



(f) L'approssimazione di $(x, y) \to h(x, y)$

Figura 6.6: Alcune funzioni $h = h(r)$ ed i grafici delle loro approssimazioni nello spazio.

Capitolo 7

Trasformata di Fourier

La trasformata di Fourier $\hat{f}$ di una funzione $f : \mathbb{R} \to \mathbb{R}$ è la funzione $\hat{f} : \mathbb{R} \to \mathbb{C}$ definita da

$$\hat{f}(\nu) = \int_{-\infty}^{+\infty} f(x) \cdot e^{-i\nu x} \mathrm{d}x. \tag{7.1}$$

Questa definizione può cambiare leggermente a seconda dell'autore; ad esempio si può definire $\hat{f}(\nu) = \int_{-\infty}^{+\infty} f(x) \cdot e^{-i2\pi\nu x} \mathrm{d}x$. La definizione (7.1) è tuttavia quella utilizzata da MATLAB ed è dunque quella impiegata nel seguito.

7.1 Dalla Gaussiana alla Delta di Dirac

Una gaussiana di media nulla e deviazione standard σ ha l'espressione:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-x^2}{2\sigma^2}}. \tag{7.2}$$

Per $\sigma \to 0$ si ottiene la "funzione" *delta di Dirac* centrata in $x_0 = 0$:

$$\delta = \begin{cases} 0 & \text{se } x \neq 0 \\ \infty & \text{se } x = 0. \end{cases}$$

In questo paragrafo sarà utilizzato il toolbox simbolico di MATLAB per calcolare la trasformata di Fourier di una gaussiana e rappresentare cosa accade al diminuire di σ. Per fare questo è disponibile la funzione `fourier`. Per scrivere l'`m-file` si inizi come sempre cancellando e chiudendo tutto (`clear all`, `close all`); si definisca poi un vettore delle deviazioni standard:

```
s = ([3 1 .3 .001]);
hold on
```

Prima ancora di cominciare coi grafici è necessario utilizzare il comando `hold on` affinché ogni elaborazione sia sovrapposta alla precedente. Per valutare la trasformata di Fourier simbolicamente è questa volta necessario un ciclo `for`:

```
for cnt = 1:length(s);              ∀σ
f = sym('1/(s * sqrt(2*pi)) * ...   si definisca...
exp(-x^2 / (2*s^2))')               ...la f(x)
```

Come già fatto nei paragrafi precedenti abbiamo dichiarato una funzione simbolica utilizzando il comando `sym`. Di seguito è necessario sostituire alla *variabile simbolica* `'s'` il corrispondente valore dell'*array* `s(cnt)`; per farlo è sufficiente il comando `subs`.

```
f = subs(f,'s',s(cnt))    sostituisco a σ il suo valore
tf = fourier(f)           e calcolo la trasformata di Fourier
```

Il comando `ezplot(f,[Xmin,Xmax])` è ora utilizzato per rappresentare i grafici di funzioni simboliche. In questo caso la finestra grafica sarà divisa in due righe ed una colonna. Per le particolari impostazioni del comando `ezplot` questa volta è necessaria una costellazione di `hold on`.

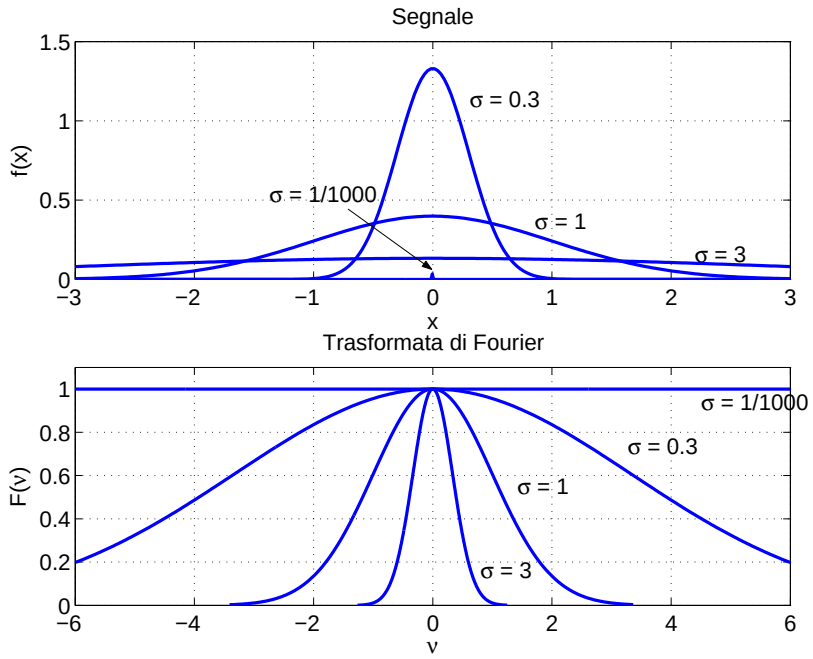


Figura 7.1: Rappresentazione di alcune gaussiane e delle rispettive trasformate di Fourier.

```
subplot(2,1,1)          la prima finestra
hold on                 comincia a tener duro!
ezplot(f,[-3,3])        rappresenta la f(x)
axis([-3,3,0,1.5])      limiti degli assi
subplot(2,1,2)          la seconda finestra
hold on                 tieni duro ancora!
ezplot(tf)              rappresenta la f̂(ω)
axis([-6,6,0,1])        fissiamo ancora gli assi
end                     FINE!
```

Il tutto è riportato in Figura 7.1. Si osservi, come già del resto noto, che la trasformata di Fourier di una gaussiana è ancora essenzialmente una gaussiana, e quanto più l'una è piccata tanto più l'altra è piatta; questo fenomeno è ben noto in meccanica quantistica, dove prende il nome di *principio di indeterminazione di Heisenberg*.

7.2 Fast Fourier Transform (FFT), un'Applicazione

Oltre alla valutazione simbolica della trasformata di Fourier vista sopra mostriamo ora l'utilizzo di un algoritmo numerico di grandissimo uso nelle applicazioni: la trasformata di Fourier veloce (Fast Fourier Transform, FFT). Lo stesso ambiente MATLAB propone l'utilizzo della FFT per il calcolo numerico della trasformata di Fourier. Premettiamo all'applicazione un semplice codice che verrà impiegato nel seguito.

7.2.1 Una Function per il Calcolo della Frequenza

Si intende creare una function per il calcolo delle frequenze relative a dati empirici campionati. Indichiamo con Δt l'intervallo temporale costante di campionamento, con N il numero di campioni. Le frequenze vanno dalla frequenza relativa all'intero periodo di campionamento

$$\omega_1 = \frac{1}{N \cdot \Delta t}$$

alla frequenza di Nyquist ω_{max}, pari alla metà della frequenza di campionamento,

$$\omega_{max} = \frac{1}{2\Delta t},$$

intervallate sempre da ω_{max}:

```
function f=frequency(Dt,N)      definizione della function
nyquist=1/(2*Dt);               ω_max
f=[1:N/2]/(N/2)*nyquist;        da ω_1 a ω_max con intervallo ω_max
```

La scelta della frequenza di Nyquist è necessaria per la corretta ricostruzione del segnale.

7.2.2 Analisi in Frequenza di una Richiesta Idrica

Il segnale che si intende indagare è la portata d'acqua richiesta da un centro abitato. I dati in questione sono memorizzati all'interno di un `m-file` chiamato RichiesteIdriche.m, che può essere caricato col comando `load`:

```
load 'RichiesteIdriche'
```

I dati in questo file sono campionati ad intervalli di mezz'ora e ad ogni riga corrisponde un giorno, per un intero mese di 31 giorni. Abbiamo dunque $24 \times 2 \times 31 = 1488$ dati. Da questo vogliamo ricondurci ad un unico vettore rappresentante il segnale:

```
RichiesteIdriche = RichiesteIdriche';    una colonna ∀ giorno
Segnale = RichiesteIdriche(:);           appendo ogni giorno sotto l'altro
```

Conoscendo l'intervallo Δt di campionamento è possibile ricostruire l'asse temporale:

```
Dt=.5;
N=length(Segnale);
Tempo = [Dt:Dt:Dt*N];
```

Per costruire l'asse delle frequenze si ricorre alla `function` creata nella Sezione 7.2.1:

```
frequenza=frequency(Dt,N);
```

Dopodiché è sufficiente utilizzare l'algoritmo precompilato `fft` per calcolare le ampiezze:

```
Fourier=fft(Segnale);              i coefficienti di Fourier
Ampiezza=abs(Furier(1:N/2));       le ampiezze
```

Una volta ottenuto il valore delle ampiezze si può passare agli elaborati:

```
subplot(2,1,1)                                  la prima finestra grafica
plot(Tempo,Portata);                            il grafico
xlabel('tempo [ore]')                           ascisse
label('Portata [l/s]')                          ordinate
subplot(2,1,2)                                  la seconda finestra
ghost = stem(frequenza,Ampiezza,'b');           spettro (ghost) delle ampiezze
grid on                                         la griglia
set(ghost,'MarkerSize',0.1,'LineWidth',1.5)     testa e larghezza
xlabel('Frequenza')                             frequenza
ylabel('Ampiezza')                              ampiezza
axis([0 0.2 0 1e+5])                            gli assi
```

Il risultato è riportato in Figura 7.2.

Si osservi come Fourier duecento anni fa abbia elaborato un potentissimo strumento per la descrizione dei fenomeni periodici o assimilabili a questi. Analizzando il fenomeno nello spazio delle frequenze si osserva infatti come da qualcosa di apparentemente rarefatto si sia in grado di interpretare le abitudini quotidiane di una popolazione. Riportando le frequenze principali si notano (denominatori in ore):

- $1/744 \sim 0.001$: il periodo di campionamento;
- $1/24 \sim 0.041$: la giornata tipo;
- $1/12 \sim 0.083$: il ciclo sonno-veglia;
- $1/6 \sim 0.016$: i due pasti giornalieri.

Esercizio 7.1 In maniera analoga all'Esercizio 5.3 ricostruire un segnale temporale dai quattro dati relativi alle frequenze evidenziati in Figura 7.2. Confrontare quindi questo segnale con quello reale.

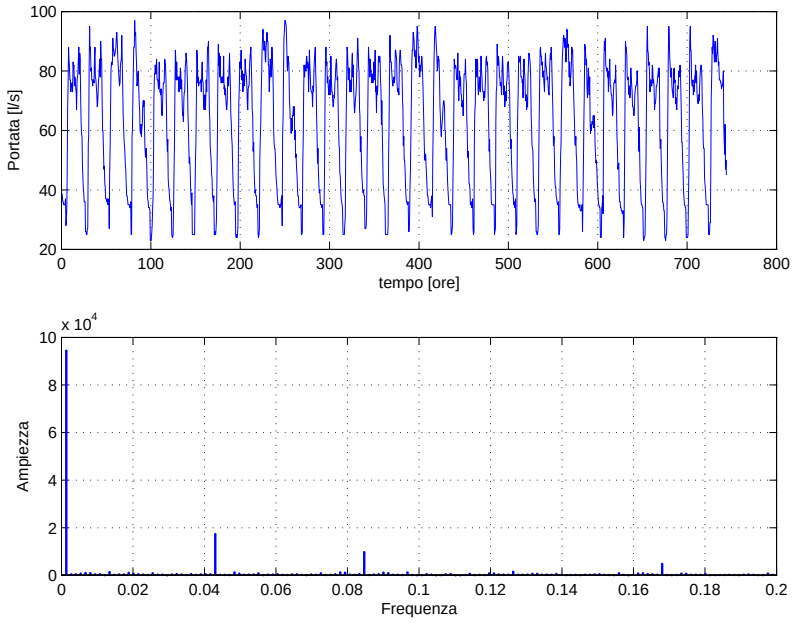


Figura 7.2: Segnale e decomposizione in frequenza per la portata uscente da un serbatoio.

7.3 Aliasing: lo Scorretto Campionamento di un Segnale

In questa sezione facciamo vedere come uno scorretto campionamento di un segnale (si pensi al dato della Sezione 7.2.2) possa portare ad una analisi in frequenza falsata. Il problema è di importanza fondamentale nel trattamento di dati; qui viene evidenziato in modo estremamente semplificato.

Pensiamo di avere un segnale di forma sinusoidale caratterizzato da una frequenza propria ω_S. Per fissare le idee si prenda in considerazione una funzione trigonometrica "seno"di periodo $\pi/6$, cioè $\sin(12x)$. Dunque la frequenza propria del segnale vale:

$$\omega_S = \frac{2\pi}{T} = 12 \text{ Hz}.$$

La frequenza massima campionabile dipende dall'intervallo di campionamento:

$$\omega_{max} = \frac{1}{2\Delta t}\,.$$

Per un corretto campionamento la frequenza di campionamento ω_C deve essere maggiore o uguale a ω_S, dunque:

$$\omega_C \geq \omega_S \;\Rightarrow\; \Delta t_C := \frac{1}{2\omega_C} \leq \frac{1}{2\omega_S}. \tag{7.3}$$

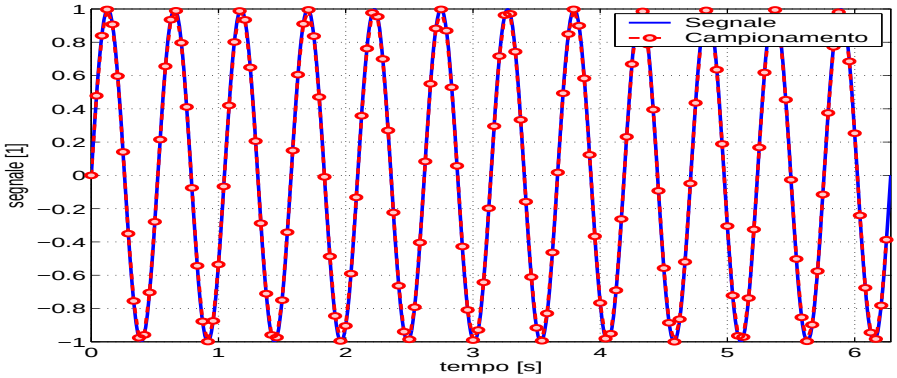
Per "campionare" un segnale lo interpoleremo linearmente con il comando `interp1(x,y,x1)` che accetta rispettivamente l'ascissa della funzione da interpolare, la funzione da interpolare e l'ascissa interpolante. Tenuto conto di ciò un listato potrebbe avere l'aspetto:

```
t  = linspace(0,2*pi,503);      503 un po strano...
omegaS = 12                     la frequenza del segnale
S = sin(omegaS * t);            il segnale
omegaC = 2                      una frequenza di campionamento molto bassa
t1 = [0:1/(2*omegaC):2*pi];     Δt_C = 1/2ω_C
C = interp1(t,S,t1);            il segnale campionato
```

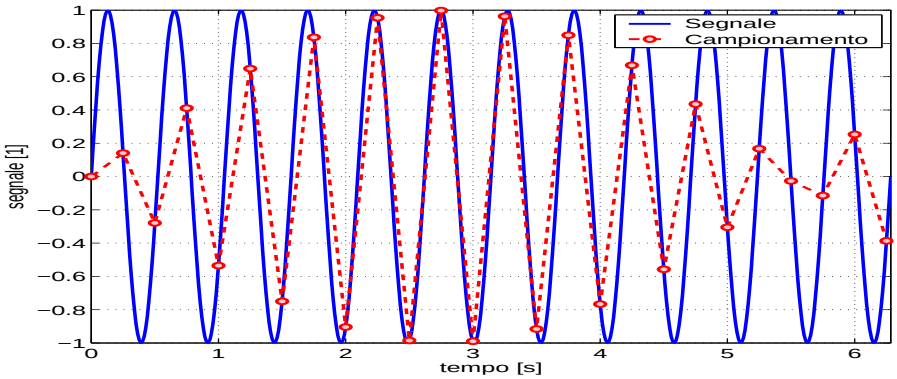
Una volta ottenuto il segnale campionato per visualizzare il problema è sufficiente sovrapporre il grafico del segnale e del campione:

```
plot(t,S,t1,C,'--ro')
```

In Figura 7.3 è rappresentato il confronto fra un segnale correttamente campionato, (a), ed una misura soggetta ad *Aliasing*, (b). Il fenomeno è analogo a quello per il quale i raggi di una bicicletta, ripresi da una videocamera, sembrano girare in direzione opposta al moto della ruota. Si noti infatti che il segnale campionato in (b) appare caratterizzato da una frequenza diversa dal segnale reale. Con un campionamento sbagliato siamo passati da un segnale caratterizzato da una sola frequenza, ad uno più complesso, che mette in evidenza due frequenze distinte, cioè 4π e $\pi/6$.



(a) Un campionamento corretto $\omega_C = \omega_S = 12$ Hz.



(b) Campionamento scorretto $\omega_C = 2$ Hz, *Aliasing*.

Figura 7.3: Campionamento corretto e non di un segnale.

Esercizio 7.2 In riferimento all'esempio della Sezione 7.2.2, per diversi valori della ω_C si ottenga un segnale campionato ed a tale campione si applichi la FFT. Cosa succede?

Indice analitico

Bibliografia

[1] L. C. Andrews. *Elementary Partial Differential Equations.* Academic Press, 1986.

[2] M. Bramanti, C. D. Pagani e S. Salsa. *Calcolo infinitesimale e algebra lineare.* Seconda edizione. Zanichelli, 2004.

[3] D. Citrini e G. Noseda. *Idraulica.* Seconda edizione. Casa Editrice Ambrosiana, 1987.

[4] M. P. Coleman. *Introduction to Partial Differential Equations with* MATLAB. Chapman and Hall/CRC, 2005.

[5] D. M. Etter e D. C. Kuncicky. *Introduzione a Matlab.* Apogeo, 2001.

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

[7] E. Marchi e A. Rubatta. *Meccanica dei Fluidi.* UTET, 2000.

[8] W. J. Palm III. *Matlab 7 per l'Ingegneria e le Scienze.* McGraw-Hill, 2004.

[9] J. C. Polking e D. Arnold. *Ordinary Differential Equations with Matlab.* Prentice-Hall, 1999.

[10] S. Salsa. *Equazioni a Derivate Parziali.* Metodi, Modelli e Applicazioni. Springer, 2004.

[11] S. Salsa e G. Verzini. *Equazioni a Derivate Parziali.* Complementi ed Esercizi. Springer, 2005.

[12] I. S. Sokolnikoff e R. M. Redheffer. *Mathematics of Physics and Modern Engineering.* McGraw-Hill, 1958.