ESERCITAZIONE MATLAB 7

1 Integrazione numerica

1. I metodi dei trapezi e di Simpson compositi per la approssimazione di

$$I(f) = \int_a^b f(x)dx \tag{1}$$

sono dati, rispettivamente, da

$$I_1^{(N)}(f) = \frac{h}{2}\left[f(x_0) + 2\left(\sum_{i=1}^{N-1} f(x_i)\right) + f(x_N)\right], \tag{2}$$

$$\begin{aligned} I_2^{(N)}(f) &= \frac{h}{3}\left\{f(x_0) + 4\left[f(x_1) + f(x_3) + \ldots + f(x_{N-1})\right]\right. \\ &\quad \left. +2\left[f(x_2) + f(x_4) + \ldots + f(x_{N-2})\right] + f(x_N)\right\}, \end{aligned} \tag{3}$$

dove

$$h = \frac{b-a}{N}, \qquad x_i = a + ih, \quad i = 0, \ldots, N,$$

e, nel caso del metodo di Simpson, N è pari. Scrivere due function matlab che implementano tali metodi e le si salvi con il nome `trapezi.m` e `simpson.m`.

2. Si ricorda che se $f(x)$ è sufficientemente regolare allora esiste $\eta \in [a, b]$ tale che

$$E_1^{(N)}(f) = I(f) - I_1^{(N)}(f) = -\frac{b-a}{12} f^{(2)}(\eta)\left(\frac{b-a}{N}\right)^2, \tag{4}$$

$$E_2^{(N)}(f) = I(f) - I_2^{(N)}(f) = -\frac{b-a}{180} f^{(4)}(\eta)\left(\frac{b-a}{N}\right)^4. \tag{5}$$

Si verifichi la validità di tali formule prendendo $f(x) = x^2$ per il metodo dei trapezi e $f(x) = x^4$ per il metodo di Simpson.
Suggerimento: $\frac{d^j x^j}{dx^j} = j!$ per ogni $j \geq 1$.

3. Si comprenda la logica del seguente script e si provi ad eseguirlo

```
f=@sin;
a=0;  b=pi;
If=2;
NN = 50:10:500;
for i=1:length(NN)
    It(i) = trapezi(f,a,b,NN(i));
    Is(i) = simpson(f,a,b,NN(i));
end
plot(log10(NN),log10(abs(It-If)),'b-x',log10(NN),log10(abs(Is-If)),'r-*')
ct=polyfit(log10(NN),log10(abs(It-If)),1);
cs=polyfit(log10(NN),log10(abs(Is-If)),1);
```

4. Si scriva e si esegua uno script analogo per la approssimazione di

$$I(f) = \int_{-1}^{1} \frac{1}{1+25x^2}\, dx.$$

2 Equazioni Differenziali

Matlab include diverse routine per la risoluzione di problemi ai valori iniziali per equazioni differenziali ordinarie. Fra di esse vi segnalo le seguenti due:

- `ode45`: implementa metodi Runge-Kutta espliciti di ordine 4 e 5;
- `ode15s`: implementa schemi *multistep* $A(\alpha)$-stabili di ordine variabile da 1 a 5.

Se usati nel modo più semplice la loro interfaccia è

```
[t,y] =  solver(odefun,tspan,y0)
```

dove `odefun` è l'*handle* della funzione $f(t, y)$ che definisce l'equazione differenziale, `tspan` $= [t_0, T]$ e `y0` è il valore iniziale del problema ossia $y(t_0)$.

In uscita forniscono la griglia di punti e le corrispondenti approssimazioni numeriche della soluzione. In particolare, la griglia viene generata dinamicamente cercando di determinare una approssimazione con errore relativo 10^{-3} ed errore assoluto 10^{-6} su ciascun punto.

È possibile specificare alcune opzioni di integrazione passando in input un quarto parametro da settare facendo uso del comando `odeset`. Per entrambe le due routine definiremo le seguenti opzioni:

- 'RelTol': tolleranza per l'errore relativo;
- 'AbsTol': tolleranza per l'errore assoluto;
- 'Stats': chiederemo di visualizzare le statistiche di integrazione (numero di passi di integrazione, numero di valutazioni di funzione, etc.);
- 'Outputfcn': chiederemo di tracciare il grafico della soluzione numerica mano a mano che viene calcolata.

Per la routine `ode15s`, che implementa metodi impliciti, setteremo anche l'opzione 'Jacobian' specificando una funzione che calcola

$$\frac{\partial f(t,y)}{\partial y}.$$

3 Soluzioni

3.1 Integrazione Numerica

1. Il codice che implementa il metodo dei trapezi composito è il seguente

```
function I = trapezi(f,a,b,N)

% I = trapezi(f,a,b,N)
%
% Usa la formula dei trapezi composita
% per calcolare una approssimazione dell'integrale
% tra a e b di f suddividendo [a,b] in
% N sottointervalli.

h=(b-a)/N;
x=linspace(a,b,N+1);

fx=feval(f,x);

I = h*(0.5*(fx(1) + fx(N+1)) + sum(fx(2:N)));
```

   mentre quello che implementa Simpson composito è

```
function I = simpson(f,a,b,N)

% I = simpson(f,a,b,N)
%
% Usa la formula di Simpson composita
% per calcolare una approssimazione dell'integrale
% tra a e b di f suddividendo [a,b] in
% N sottointervalli.

if (mod(N,2)~=0),
    error('Il numero di sottointervalli deve essere pari')
end

h=(b-a)/N;
x=linspace(a,b,N+1);

fx=feval(f,x);

I = (1/3)*h*(fx(1) + 4*sum(fx(2:2:N)) + 2*sum(fx(3:2:N-1)) + fx(N+1));
```

2. Dato che $\frac{d^2x^2}{dx^2} = 2$ e $\frac{d^4x^4}{dx^4} = 4! = 24$, si ha

$$E_1^{(N)}(x^2) \;=\; I(x^2) - I_1^{(N)}(x^2) = \frac{b^3-a^3}{3} - I_1^{(N)}(x^2) = -\frac{(b-a)^3}{6N^2},$$

$$E_2^{(N)}(x^4) \;=\; I(x^4) - I_2^{(N)}(x^4) = \frac{b^5-a^5}{5} - I_2^{(N)}(x^2) = -\frac{2(b-a)^5}{15N^4}.$$

Con le seguenti istruzioni, tenendo conto dell'aritmetica finita, si controlla che tali uguaglianze sono verificate per il caso $[a,b] = [0,2]$ e $N = 10$

```
>> a=0; b=2; N=10;
>> f1=@(x) x.^2;
>> I1=trapezi(f1,a,b,N);
>> E1=(b^3-a^3)/3 - I1;
>> E1 + (b-a)^3/(6*N^2)
ans =

  -3.0705e-16

>> f2=@(x) x.^4;
>> I2=simpson(f2,a,b,N);
>> E2=(b^5-a^5)/5 - I2;
>> E2 + 2*(b-a)^5/(15*N^4)
ans =

  -1.0188e-15
```

3. eseguendo lo script si utilizzano i metodi dei trapezi e di Simpson compositi per la approssimazione di
$$\int_0^{\pi} \sin(x)dx = 2,$$
con $N = 50, 60, 70, \ldots, 500$. In maggior dettaglio, alla variabile `If` viene assegnato il valore esatto dell'integrale mentre alle componenti dei vettori `It` e `Is` si assegnano le approssimazioni ottenute applicando trapezi e Simpson compositi, rispettivamente, per i vari valori di N.
L'istruzione `plot` presente nello script traccia poi i grafici di $\log_{10}\left(\left|E_1^{(N)}(\sin)\right|\right)$ e di $\log_{10}\left(\left|E_2^{(N)}(\sin)\right|\right)$ verso $\log_{10}(N)$. Le curve che si ottengono sono di tipo lineare (si veda Figura 1). Infatti, da (4)-(5), applicando le proprietà dei logaritmi, si deduce che
$$\begin{aligned}
\log_{10}\left(\left|E_1^{(N)}(\sin)\right|\right) &= \log_{10}\left(\frac{(b-a)^3}{12}\sin(\eta)\right) - 2\log_{10}(N),\\
\log_{10}\left(\left|E_2^{(N)}(\sin)\right|\right) &= \log_{10}\left(\frac{(b-a)^5}{180}\sin(\eta)\right) - 4\log_{10}(N).
\end{aligned}$$
Al fine di determinare sperimentalmente i coefficienti angolari di tali rette, che dovrebbero essere circa -2 e -4, rispettivamente, viene infine utilizzato il comando `polyfit` che implementa l'approssimazione polinomiale di funzioni ai minimi quadrati nel discreto. In particolare, avendo specificato uno come

terzo parametro di input a `polyfit`, si chiede di approssimare i dati con un polinomio di primo grado. I vettori `ct` e `cs` che si ottengono in tal modo sono i seguenti

```
>> ct
ct =

  -2.0000e+00   2.1619e-01

>> cs
cs =

  -4.0001e+00   3.4654e-02
```

Considerando che la loro prima componente contiene il coefficiente angolare della corrispondente retta, i risultati ottenuti sono in accordo con la teoria.

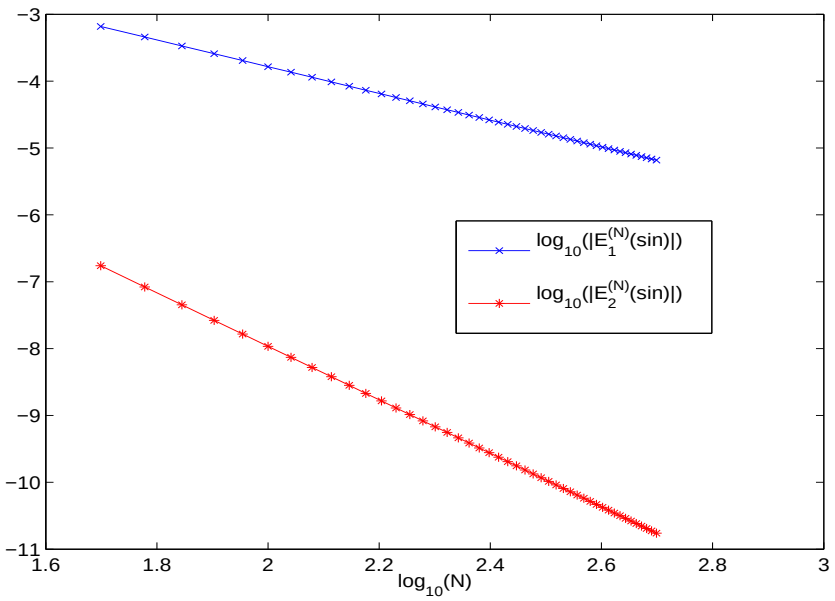

Figura 1: Errori nella approssimazione di $\int_0^\pi \sin(x)dx$ applicando i metodi dei trapezi e di Simpson compositi.

3.2 Equazioni Differenziali Ordinarie

Il primo problema test considerato è il seguente

$$\begin{cases} y'(t) &= \lambda(y - \cos(t)) - \sin(t), \quad t \in [0, \pi], \\ y(0) &= \cos(0) \end{cases} \tag{6}$$

la cui soluzione esatta è $y(t) = \cos(t)$ indipendentemente dal valore di λ. La seguente function consente di risolverlo numericamente usando la routine `ode45` per un certo valore di λ, una tolleranza per l'errore relativo pari a `rtol` ed una per l'errore assoluto pari a `atol`

```
function [t,y] = ode45_test(lam,rtol,atol)
% [t,y] = ode45_test(lam,rtol,atol)
%
% Risolve il problema ai valori iniziali
%
% y'(t) = lam*(y-cos(t)) - sin(t),  0<=t<=2*pi
%  y(0) = 1
%
% applicando la routine ode45 con tolleranza per l'errore relativo rtol
% e per l'errore assoluto atol.


phi  = @(t)  cos(t);
phi1 = @(t) -sin(t);

y0 = phi(0);

f=@(t,y) lam*(y-phi(t)) + phi1(t);

options = odeset('RelTol',rtol,'AbsTol',atol,'Stats','on','OutputFcn',@odeplot);

T = 2*pi;
[t,y] = ode45(f,[0 T],y0,options);
```

La seguente function ha uno scopo analogo alla precedente ma utilizza la routine `ode15s`

```
function [t,y] = ode15s_test(lam,rtol,atol)

% [t,y] = ode15s_test(lam,rtol,atol)
% Risolve il problema ai valori iniziali
%
% y'(t) = lam*(y-cos(t)) - sin(t),  0<=t<=2*pi
%  y(0) = 1
%
% applicando la routine ode15s con tolleranza per l'errore relativo rtol
% e per l'errore assoluto atol.

phi  = @(t)  cos(t);
phi1 = @(t) -sin(t);

y0 = phi(0);

f=@(t,y) lam*(y-phi(t)) + phi1(t);

options = odeset('RelTol',rtol,'AbsTol',atol,'Stats','on','OutputFcn',@odeplot);
options = odeset(options,'Jacobian',lam); % si aggiunge l'opzione della jacobiana
                                          % a quelle settate con il comando
                                          % precedente

T = 2*pi;
[t,y] = ode15s(f,[0 T],y0,options);
```

Al fine di valutare il costo computazionale sostenuto si può utilizzare il numero di passi di integrazione effettuti con successo. Tali valori sono stati riportati in Tabella 1. Come si può vedere nel caso della `ode45` il costo aumenta non soltanto se diminuiscono le tolleranze ma anche al crescere di $|\lambda|$. Lo stesso non accade invece applicando la `ode15s`. In questo caso, infatti, si ha che il costo computazionale è essenzialmente legato soltanto alla accuratezza richiesta per la soluzione numerica.

Il seguente problema

$$\begin{aligned} y_1'(t) &= -0.04y_1(t) + 10^4 y_2(t)y_3(t) \\ y_2'(t) &= 0.04y_1(t) - 10^4 y_2(t)y_3(t) - 3\cdot 10^7 y_2^2(t) \\ y_3'(t) &= 3\cdot 10^7 y_2^2(t) \\ y_1(0) &= 1 \\ y_2(0) &= 0 \\ y_3(0) &= 0 \end{aligned}$$

con $t \in [0, 10^{10}]$, è noto come Robertson e deriva dalla cinetica chimica. La differenza di ordine di grandezza nei coefficienti della equazione lo rendono fortemente *stiff*. È

Tabella 1: Numero di passi di integrazione per il problema (6) usando le routine `ode45` e `ode15s`.

λ	`ode45` rtol = atol			`ode15s` rtol = atol		
	1e − 3	1e − 6	1e − 9	1e − 3	1e − 6	1e − 9
-1	10	26	100	25	71	182
-100	191	462	1682	27	76	181
-5000	9464	9331	19586	28	68	186

un problema non lineare che non si sa risolvere per via analitica. Tuttavia si può dimostrare che se i valori iniziali sono non negativi allora

$$\lim_{t\to+\infty} y_1(t) = \lim_{t\to+\infty} y_2(t) = 0, \qquad \lim_{t\to+\infty} y_3(t) = y_1(0) + y_2(0) + y_3(0). \tag{7}$$

La routine `ode15s`, che implementa metodi $A(\alpha)$-stabili, può essere tranquillamente usata per risolvere questo problema e la soluzione che fornisce rispetta (7). Per verificare questo occorre utilizzare le seguenti due function che implementano la valutazione della funzione $f(t, y)$ che definisce l'equazione differenziale e la valutazione della sua matrice jacobiana

```
function f = rober(t,y)

f(1) = -0.04*y(1) + 1e4*y(2)*y(3);
f(2) =  0.04*y(1) - 1e4*y(2)*y(3) - 3e7*y(2)^2;
f(3) =                              3e7*y(2)^2;
f = f(:); % rende f vettore colonna
return

function Jf = jrober(t,y)

Jf = [-0.04    1e4*y(3)             1e4*y(2)
       0.04   -1e4*y(3)-6e7*y(2)  -1e4*y(2)
         0       6e7*y(2)             0];
return
```

ed il seguente driver per la `ode15s`

```
function [t,y] = ode15s_rober(rtol,atol)

% [t,y] = ode15s_rober(rtol,atol)

options = odeset('RelTol',rtol,'AbsTol',atol,'Stats','on','OutputFcn',@odeplot);
options = odeset(options,'Jacobian',@jrober);
```

```
y0 = [1 0 0]';

T = 1e10;

[t,y] = ode15s(@rober,[0 T],y0,options);
```

I risultati ottenuti con il seguente comando sono riportati in Figura 2.

```
>>  [t,y] = ode15s_rober(1e-6,1e-9);
```

Le corrispondenti statistiche di integrazione sono

```
604 successful steps
20 failed attempts
1092 function evaluations
10 partial derivatives
117 LU decompositions
1090 solutions of linear systems
```

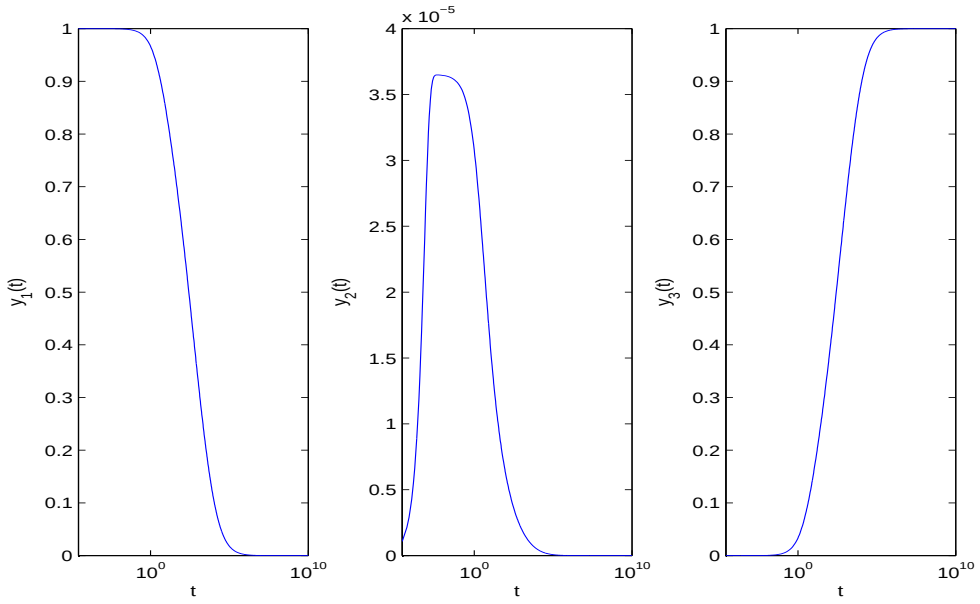

Figura 2: Soluzione numerica del problema Robertson utilizzando `ode15s`

Provando ad usare il seguente driver con le medesime tolleranze, invece, il problema risulta non risolvibile in tempi ragionevoli.

```
function [t,y] = ode45_rober(rtol,atol)

% [t,y] = ode45_rober(rtol,atol)

options = odeset('RelTol',rtol,'AbsTol',atol,'Stats','on','OutputFcn',@odeplot);

y0 = [1 0 0]';

T = 1e10;

[t,y] = ode45(@rober,[0 T],y0,options);
```