Serie 7 - Soluzione
Equazioni Differenziali Ordinarie

Esercizio 1.1

1. *Si approssimi $f'(0)$ tramite gli schemi alle differenze finite in avanti, all'indietro e centrate, utilizzando un passo $h = 0.4, 0.2, 0.1, 0.05, 0.025, 0.0125$. Per ciascuno dei tre schemi si calcoli l'errore commesso e se ne visualizzi l'andamento in funzione del passo h su un grafico in scala logaritmica su entrambu gli assi; verificare che ci sia accordo con i risultati teorici..*

   Tramite i seguenti comandi Matlab® si ottiene il grafico in Figura 1.

```
f = @( x ) exp( - x.^2 ) .* sin( 2 * x + 1 );
df = @( x ) 2 * ( cos( 2 * x + 1 ) - x .* sin( 2 * x + 1 ) ) .* exp( - x.^2 );

xb = 0;     dfxb = df( xb );
h_v = 0.4 * 2.^( - [ 0 : 5 ] );
dfa_v = [];     dfi_v = [];     dfc_v = [];
for h = h_v
   dfa_v = [ dfa_v, ( f( xb + h ) - f( xb ) ) / h ];
   dfi_v = [ dfi_v, ( f( xb ) - f( xb - h ) ) / h ];
   dfc_v = [ dfc_v, ( f( xb + h ) - f( xb - h ) ) / ( 2 * h ) ];
end
Err_dfa_v = abs( dfa_v - dfxb );
Err_dfi_v = abs( dfi_v - dfxb );
Err_dfc_v = abs( dfc_v - dfxb );

loglog( h_v, Err_dfa_v, '-ob', h_v, Err_dfi_v, '-xr',  h_v, Err_dfc_v, '-sg', ...
        h_v, h_v, '--k', h_v, h_v.^2, '-.k' );
xlabel( 'h [log]' ); ylabel( 'Err [log]' )
legend( 'DF avanti', 'DF indietro', 'DF centrate', 'h', 'h^2', 'Location', 'Best' );
grid on
```

   Per esempio, le approssimazioni di $f'(0)$ tramite le differenze finite in avanti per i valori di h indicati sono

```
dfa_v =
    -0.0290    0.5267    0.8129    0.9502    1.0164    1.0488
```

   mentre i corrispondenti errori sono:

```
Err_dfa_v =
    1.1096    0.5539    0.2677    0.1304    0.0642    0.0318
```

   Sappiamo dalla teoria che, se $f \in C^2(I_{\overline{x}})$ con $I_{\overline{x}}$ intorno di $\overline{x}$, allora i metodi delle differenze finite in avanti e all'indietro sono accurati di ordine $p = 1$ (gli errori corrispondenti sono proporzionali ad h); se invece $f \in C^3(I_{\overline{x}})$, allora il metodo delle differenze centrate

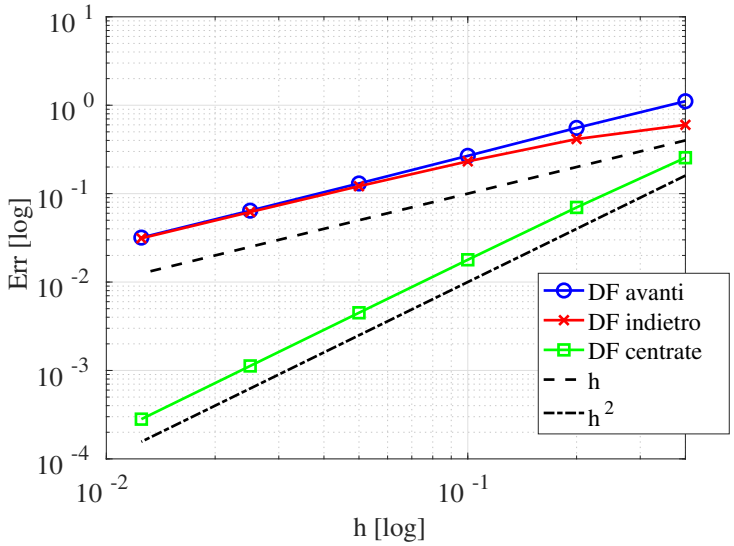


Figura 1: Differenze finite per l'approssimazione di $f'(0)$. Errori vs. h in scala logaritmica su entrambi gli assi

è accurato di ordine $p = 2$ (l'errore corrispondente è proporzionale a h^2). Osserviamo che $f \in C^\infty(\mathbb{R})$. Il grafico di Figura 1 evidenzia come, per h sufficientemente piccolo, gli andamenti degli errori associati ai metodi delle differenze finite in avanti e all'indietro risultino paralleli, in scala logaritmica su entrambi gli assi, alla retta (h, h): confermiamo dunque graficamente che tali metodi sono accurati di ordine $p = 1$. Invece, il metodo delle differenze finite centrate esibisce un andamento dell'errore parallelo a quello della retta (h, h^2) in scala log–log; dunque è anche confermato l'ordine di convergenza $p = 2$ previsto dalla teoria per tale schema.

2. *Si ripeta il punto 1 per l'approssimazione di $f''(0)$ tramite lo schema delle differenze finite centrate.*

   Tramite i seguenti comandi Matlab® si ottiene il grafico in Figura 2.

```
d2f = @( x ) - 2 * ( 3 * sin( 2 * x + 1 ) + 4 * x .* cos( 2 * x + 1 ) ...
                   - 2 * x.^2 .* sin( 2 * x + 1 ) ) .* exp( - x.^2 );
d2fxb = d2f( xb );

d2fc_v = [];
for h = h_v
   d2fc_v = [ d2fc_v, ( f( xb + h ) - 2 * f( xb ) + f( xb - h ) ) / ( h^2 ) ];
end
Err_d2fc_v = abs( d2fc_v - d2fxb );

loglog( h_v, Err_d2fc_v, '-sb', h_v, h_v, '--k', h_v, h_v.^2, '-.k' );
xlabel( 'h [log]' ); ylabel( 'Err [log]' )
legend( 'DF centrate', 'h', 'h^2', 'Location', 'Best' );
grid on
```

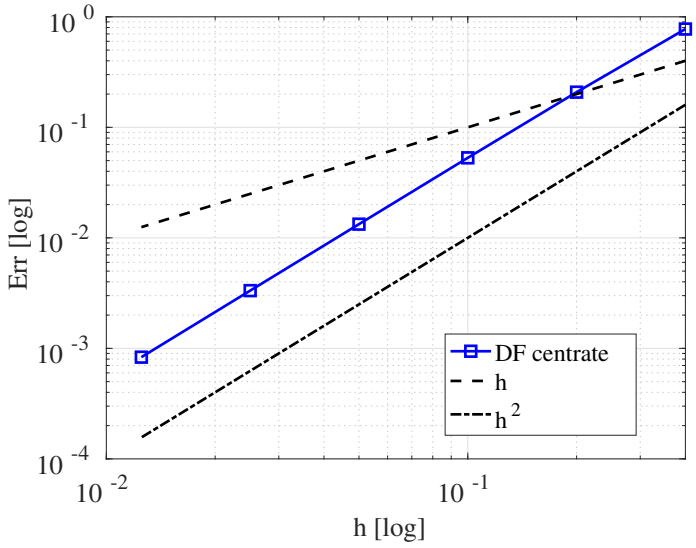


Figura 2: Differenze finite centrate per l'approssimazione di $f''(0)$. Errori vs. h in scala logaritmica su entrambi gli assi

Dalla teoria sappiamo che, se $f \in C^4(I_{\overline{x}})$, allora il metodo delle differenze centrate per l'approssimazione di $f''(\overline{x})$ è accurato di ordine $p = 2$ (l'errore corrispondente è proporzionale a h^2). Dalla Figura 2 osserviamo che l'andamento dell'errore è parallelo a quello della retta (h, h^2) in scala log–log; dunque è confermato l'ordine di convergenza $p = 2$ previsto dalla teoria per tale schema.

Esercizio 2.1

1. *Scrivere la funzione Matlab® `eulero_avanti.m` che implementa il metodo di Eulero in avanti per la risoluzione di un generico problema di Cauchy*

   Una possibile implementazione è la seguente:

```
function [t_h,u_h]=eulero_avanti(f,t_max,y_0,h)

% [t_h,u_h]=eulero_avanti(f,t_max,y_0,h)
%
% Risolve il problema di Cauchy
%
% y'=f(t, y)
% y(0)=y_0
%
% utilizzando il metodo di Eulero in Avanti
% (Eulero Esplicito) : u^(n+1)=u^n+h*f^n
%
% Input:
% -> f: function che descrive il problema di Cauchy
%       (dichiarata ad esempio tramite inline o @).
%       Deve ricevere in ingresso due argomenti: f=f(t,y)
```

```
% -> t_max: l'istante finale dell' intervallo temporale di soluzione
%                 (l'istante iniziale e' t_0=0)
% -> y_0: il dato iniziale del problema di Cauchy
% -> delta_t: l'ampiezza del passo di discretizzazione temporale.
%
% Output:
% -> t_h: vettore degli istanti in cui si calcola la soluzione discreta
% -> u_h: la soluzione discreta calcolata nei nodi temporali t_h

% vettore degli istanti in cui risolvo la edo
t0=0;
t_h=t0:h:t_max;

% inizializzo il vettore che conterra' la soluzione discreta
N_istanti=length(t_h);
u_h=zeros(1,N_istanti);

% ciclo iterativo che calcola u_(n+1)=u_n+h*f_n
u_h(1)=y_0;

for it=2:N_istanti
    u_old=u_h(it-1);
    u_h(it)=u_old+h*f(t_h(it-1),u_old);
end
```

2. *Utilizzando la funzione appena scritta al punto 1, risolvere il problema di Cauchy con la funzione $f(t,y)$ con il metodo di Eulero in avanti, scegliendo $\lambda = 2.4$, $t_0 = 0$, $t_{max} = 3$, $y_0 = 0.1$ per i valori del passo temporale $h_1 = 0.05$ e $h_2 = 0.01$. Rappresentare sullo stesso grafico le soluzioni numeriche ottenute e confrontarle con la soluzione esatta.*

   La soluzione si ottiene con i seguenti comandi Matlab®

```
y0=0.1;
lambda=2.4;
tmax=3;
t_plot=0:0.01:tmax;
y=@(x) y0*exp(lambda*x);
f= @(t,y) lambda*y;
h=0.05;
[EA_t_h,EA_u_h]=eulero_avanti(f,tmax,y0,h);
h=0.01;
[EA_t_h_fine,EA_u_h_fine]=eulero_avanti(f,tmax,y0,h);
plot(t_plot,y(t_plot),'k','LineWidth',2);
hold on
plot(EA_t_h,EA_u_h,'o','MarkerSize',4);
plot(EA_t_h_fine,EA_u_h_fine,'or','MarkerSize',4);
leg=legend('soluzione analitica','EA con h=0.05','EA con h=0.01',2);
set(leg,'FontSize',16)
```

   Il grafico è mostrato in Figura 3.

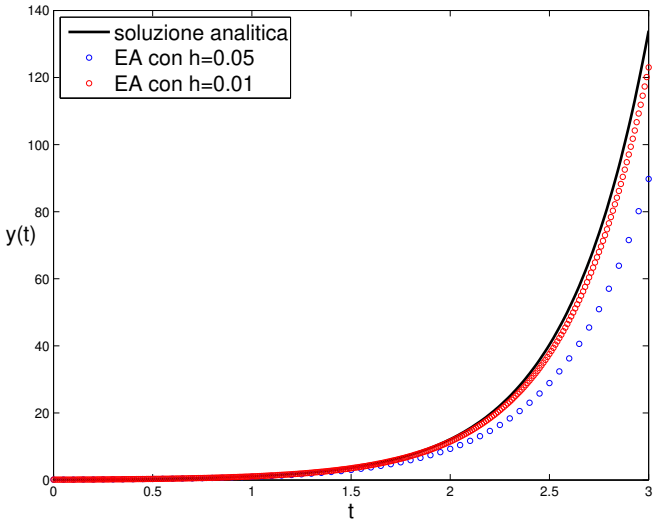


Figura 3: Confronto tra la soluzione esatta del problema di Cauchy e le soluzioni numeriche ottenute con il metodo di Eulero in avanti, con passi di discretizzazione $h_1 = 0.05$ e $h_2 = 0.01$.

3. *Scrivere la funzione Matlab® `eulero_indietro.m` che implementa il metodo di Eulero all'indietro per la risoluzione di un generico problema di Cauchy.*

   Una possibile implementazione è la seguente:

```
function [t_h,u_h,iter_pf]=eulero_indietro(f,t_max,y_0,h)

% [t_h,u_h,iter_pf]=eulero_indietro(f,t_max,y_0,h)
% Risolve il problema di Cauchy
%
% y'=f(t, y)
% y(0)=y_0
%
% utilizzando il metodo di Eulero Indietro
% (Eulero Implicito) : u^{n+1}=u^n+h*f^{n+1}.
% Per ciascun istante temporale l'equazione per u^(n+1)
% (a priori non lineare) viene risolta con un metodo di punto fisso:
%  u=u^n+delta_t*f(t^{n+1},u), utilizzando la funzione di iterazione
%
% phi(x)=u^n + h *f(t^{n+1},x)
%
% la condizione per la convergenza del metodo di punto fisso ( |phi(x)'|<1)
% e' h*|\partial_x f(t^{n+1},x)| < 1
% ed e' garantita se h e' sufficientemente piccolo.
%
% Input:
% -> f: function che descrive il problema di Cauchy
%       (dichiarata ad esempio tramite inline o @)
%       deve ricevere in ingresso due argomenti: f=f(t,y)
% -> t_max: l'istante finale dell' intervallo temporale di soluzione
%                 (l'istante iniziale e' t_0=0)
```

```
% -> y_0: il dato iniziale del problema di Cauchy
% -> h: l'ampiezza del passo di discretizzazione temporale.
%
% Output:
% -> t_h: vettore degli istanti in cui si calcola la soluzione discreta
% -> u_h: la soluzione discreta calcolata nei nodi temporali t_h
% -> iter_pf: vettore che contiene il numero di iterazioni
%                di punto fisso utilizzate per risolvere l'equazione
%                non lineare ad ogni istante temporale.

% vettore degli istanti in cui risolvo la ode
t0=0;
t_h=t0:h:t_max;

% inizializzo il vettore che conterra' la soluzione discreta
N_istanti=length(t_h);
u_h=zeros(1,N_istanti);

% ciclo iterativo che calcola u_(n+1)=u_n+h*f_(n+1) .
% Ad ogni iterazione temporale devo eseguire delle sottoiterazioni
% di punto fisso per il calcolo di u_(n+1):
%
% u_(n+1)^(k+1) = u_n + h * f_( t_(n+1) , u_(n+1)^k ).
u_h(1)=y_0;

% parametri per le iterazioni di punto fisso
N_max=100;
toll=1e-5;
iter_pf=zeros(1,N_istanti);

for it=2:N_istanti

    % preparo le variabili per le sottoiterazioni
    u_old=u_h(it-1);
    t_pf=t_h(it);

    phi=@(u) u_old + h * f( t_pf, u );

    % sottoiterazioni
    [u_pf, it_pf] = ptofis(u_old, phi, N_max, toll);
    u_h(it)=u_pf(end);

    % tengo traccia dei valori di it_pf per valutare la convergenza
    % delle iterazioni di punto fisso
    iter_pf(it)=it_pf;

end
```

4. *Utilizzando la funzione appena scritta al punto 3, risolvere il problema di Cauchy con il metodo di Eulero all'indietro, scegliendo* $\lambda = 2.4$, $t_0 = 0$, $t_{max} = 3$, $y_0 = 0.1$ *per i valori del passo* $h_1 = 0.05$ *e* $h_2 = 0.01$. *Rappresentare sullo stesso grafico le soluzioni numeriche ottenute e confrontarle con la soluzione esatta.*

   La soluzione si ottiene con i seguenti comandi:

```
h=0.05;
```

```
[EI_t_h,EI_u_h,iter_pf]=eulero_indietro(f,tmax,y0,h);
h=0.01;
[EI_t_h_fine,EI_u_h_fine,iter_pf_fine]=eulero_indietro(f,tmax,y0,h);
plot(t_plot,y(t_plot),'k','LineWidth',2);
hold on
plot(EI_t_h,EI_u_h,'o','MarkerSize',4);
plot(EI_t_h_fine,EI_u_h_fine,'or','MarkerSize',4);
leg=legend('soluzione analitica','EI con h=0.05','EI con h=0.01',2);
set(leg,'FontSize',16)
```

Il grafico è mostrato in Figura 4.

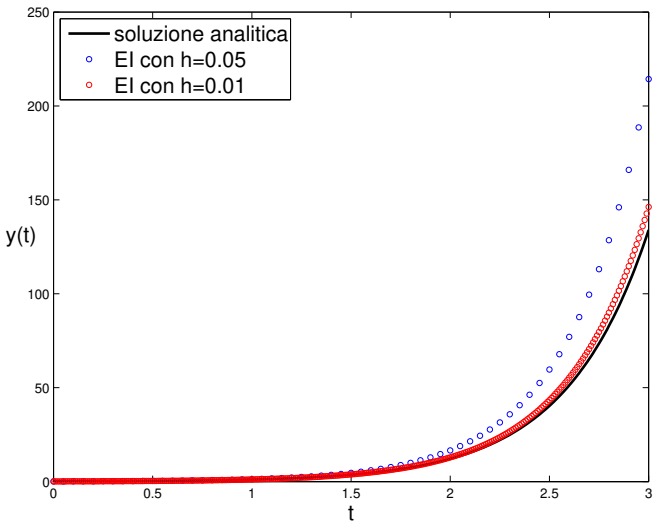


Figura 4: Confronto tra la soluzione esatta del problema di Cauchy e le soluzioni numeriche ottenute con il metodo di Eulero all'inidetro, con passi di discretizzazione $h_1 = 0.05$ e $h_2 = 0.01$.

5. *Riportare su un grafico l'andamento del numero di iterazioni impiegate dal metodo delle iterazioni di punto fisso per risolvere l'equazione non lineare con il metodo di Eulero all'indietro in funzione degli istanti temporali, scegliendo* $\lambda = 2.4$, $t_0 = 0$, $t_{max} = 3$, $y_0 = 0.1$ *e* $h_1 = 0.05$.

```
figure;
plot(EI_t_h,iter_pf,'o:','MarkerSize',4)
```

Il grafico è mostrato in Figura 14.

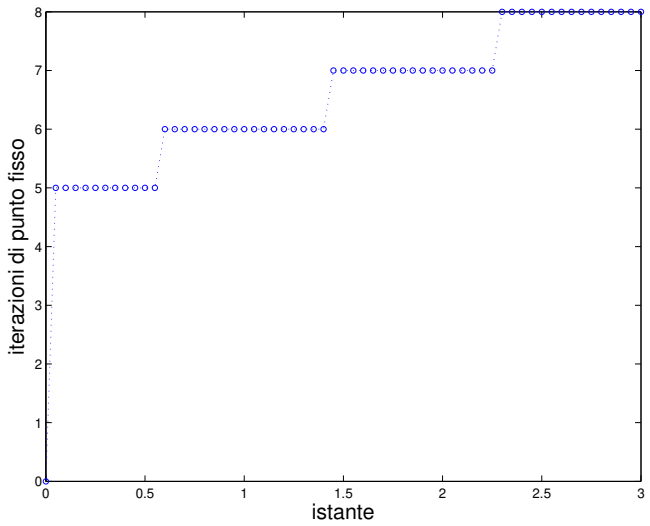


Figura 5: Numero di iterazioni di punto fisso necessarie per risolvere l'equazione non lineare in funzione degli istanti temporali.

6. *Definito* $e_h = \max_{n=0,1,\ldots,N_h} |y(t_n) - u_n|$, *il massimo modulo dell'errore compiuto approssimando la soluzione esatta* $y(t_n)$ *con la soluzione numerica* u_n, *riportare, su un grafico in scala logaritmica, l'andamento di* e_h *al variare di* h, *utilizzando i passi temporali* $h = 0.04, 0.02, 0.01, 0.005, 0.0025$ *per entrambi i metodi di Eulero in avanti e all'indietro. Si commenti il risultato ottenuto alla luce della teoria.*

   Il grafico degli errori si ottiene con i seguenti comandi:

```
N=5;
dimezz=2.^(1:N);
passi=0.08./dimezz;
for it=1:N
      [EA_t_h,EA_u_h]=eulero_avanti( f, tmax, y0, passi(it) );
      [EI_t_h,EI_u_h]=eulero_indietro( f, tmax, y0, passi(it) );
      y_h=y( 0 : passi(it) : tmax  );
      errore_EA(it)=max( abs (y_h-EA_u_h) );
      errore_EI(it)=max( abs (y_h-EI_u_h) );
end
figure;
loglog(passi,errore_EA,'-ob','LineWidth',2);
hold on
loglog(passi,errore_EI,'-or','LineWidth',2);
plot(passi,100*passi,'k')
plot(passi,(10*passi).^2,'k:')
leg=legend('errore EA','errore EI','ordine 1','ordine 2',4);
set(leg,'FontSize',16)
```

   Il grafico è mostrato in Figura 6 e conferma che entrambi i metodi convergono linearmente.

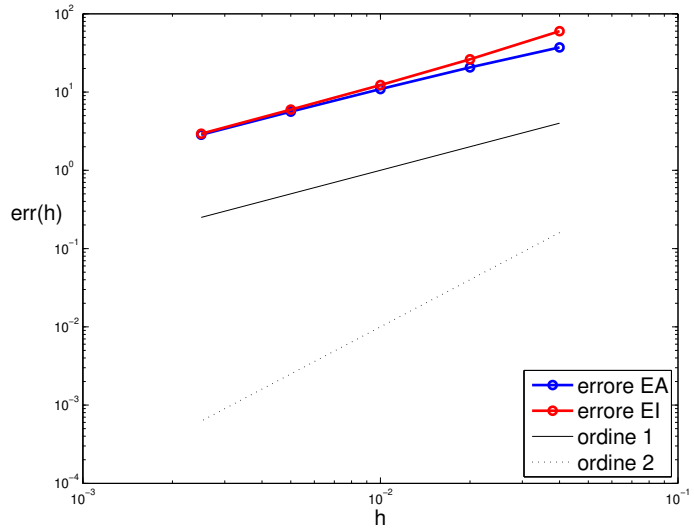


Figura 6: Il grafico mostra l'andamento degli errori per i metodi di Eulero in avanti e all'indetro al diminuire di h. Si osserva che entrambi sono metodi del primo ordine.

Esercizio 2.2

Si veda il file `es22.m` allegato.

Esercizio 2.3

Si consideri il seguente problema modello:

$$\begin{cases} y'(t) = \lambda y(t) & t > t_0 \\ y(t_0) = y_0 \end{cases} \tag{1}$$

con $\lambda = -42$ e $y_0 = 2$.

1. *Riportare il grafico della soluzione esatta $y(t)$.*

   La soluzione esatta è :

   $$y(t) = y_0\, e^{\lambda t} = 2\, e^{-42t} \qquad \text{per } t \geq 0.$$

   Con i comandi:

```
lambda=-42
t_max=1;
t_plot=0:0.001:t_max;
y0=2;
y=@(t) y0*exp(lambda*t);
plot(t_plot,y(t_plot),'k','LineWidth',2);
```

   si ottiene il grafico in Figura 7

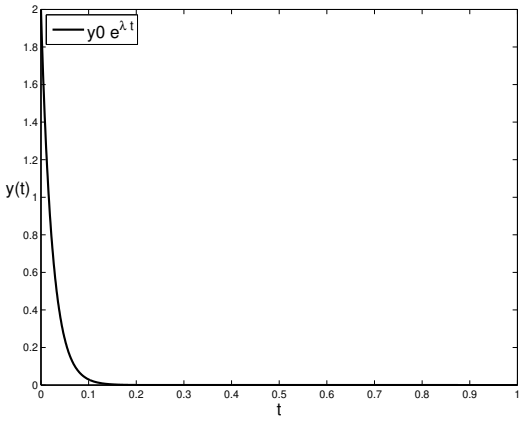


Figura 7: Grafico della soluzione esatta $y(t) = 2\,e^{-42t}$

2. *Usando la funzione* `eulero_avanti`*, calcolare la soluzione numerica del problema (1) utilizzando il metodo di Eulero in avanti con i seguenti dati:* $t_{max} = 1$*, e* $h = 0.05$*. Visualizzare sullo stesso grafico la soluzione esatta e la soluzione numerica. Ripetere la stessa procedura utilizzando il metodo di Eulero all'indietro con metodo di Newton per la soluzione dell'equazione non lineare di avanzamento temporale (funzione* `eulero_indietro_newton`*). Cosa si osserva? Valutare se il valore* λh *appartiente alla regione di assoluta stabilità dei due metodi.*

   Il valore di λh è $\lambda h = -2.1$, esterno alla regione di assoluta stabilità del metodo di Eulero in avanti. Di conseguenza tale metodo non produce una soluzione stabile, come si osserva dal grafico in Figura8(a).

   Utilizzando il metodo di Eulero all'indietro si ottiene invece una soluzione stabile, poiché il valore $\lambda h = -2.1$ appartiene alla regione di assoluta stabilità del metodo. Si ottiene quindi il grafico in Figura8(b).

   I comandi per utilizzare le function `eulero_avanti` ed `eulero_indietro_newton` sono:

```
h=0.05;
df_dy = @(t, y) lambda;
%%%%% EULERO IN AVANTI %%%%%%
[t_h,u_h]=eulero_avanti(f,t_max,y0,h);

%%%%% EULERO ALL'INDIETRO %%%%%%

[EI_t_h,EI_u_h]=eulero_indietro_newton(f,df_dy,t_max,y0,h);
```

   Ci chiediamo ora perchè sia necessario fornire alla funzione `eulero_indietro_newton` la derivata **parziale** di f rispetto a y (definita in `df_dy` nel codice riportato sopra). Il motivo di ciò è che ad ogni passo del metodo di Eulero all'indietro l'equazione da risolvere è

$$u_{n+1} = u_n + hf\left(t_{n+1}, u_{n+1}\right) .$$

Il tempo t_{n+1} è però noto a priori, quindi definendo la funzione f^* attraverso la relazione

$$f^*(y) = f(t_{n+1}, y)$$

possiamo riformulare il passo u_{n+1} del metodo di Eulero come

$$u_{n+1} = u_n + hf^*(u_{n+1}) \ .$$

Questa equazione può essere a sua volta riscritta nella forma

$$F(u_{n+1}) = 0 \,,$$

avendo definito F come

$$F(y) = y - u_n - hf^*(y) \ .$$

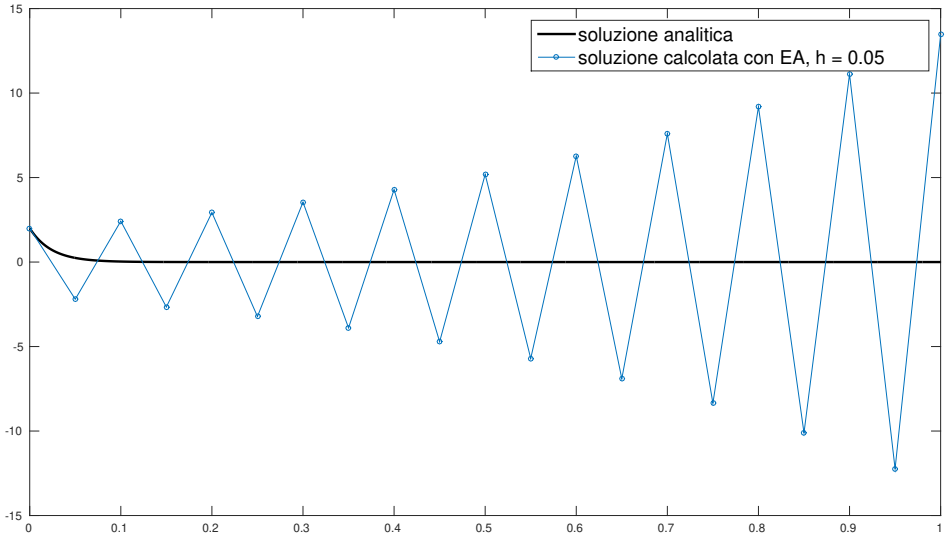
Risolvere un passo del metodo di Eulero in avanti corrisponde dunque a calcolare lo zero di $F(y)$. Una possibilità per fare ciò è usare il metodo di Newton: la derivata $F'(y)$ è

$$F'(y) = 1 - h(f^*)'(y) = 1 - h\frac{\partial f}{\partial y}(t_{n+1}, y) \,,$$

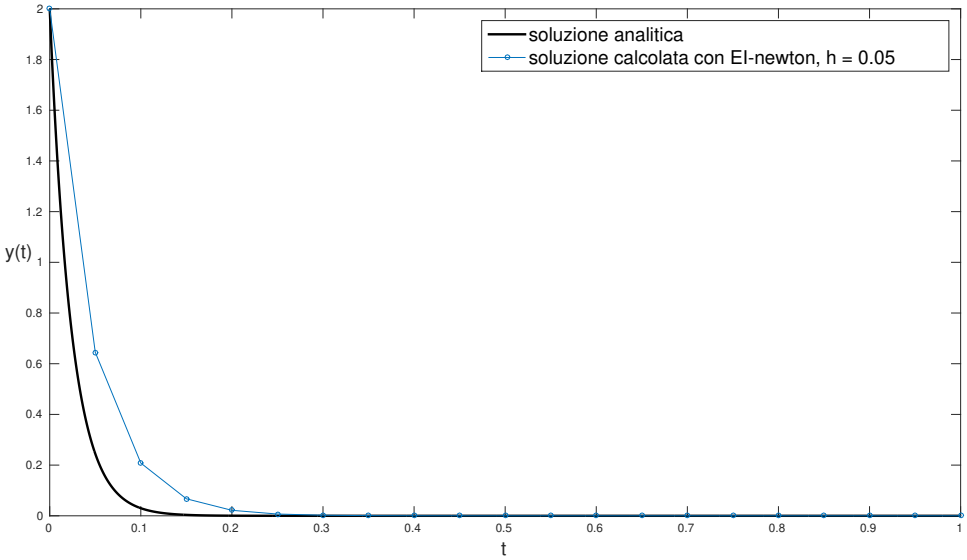
ed è per questo che è necessario fornire alla funzione `eulero_indietro_newton` la derivata della funzione che definisce l'equazione differenziale rispetto alla variable che rappresenta la soluzione dell'equazione stessa.

3. *Ripetere la stessa analisi del punto precedente usando* $h = 0.01$

   In questo caso, il valore di $h\lambda$ è $h\lambda = -0.42$, interno alla regione di stabilità di entrambi i metodi. Le soluzioni ottenute sono quindi stabili in tutti e due i casi, come si vede in Figura 9.

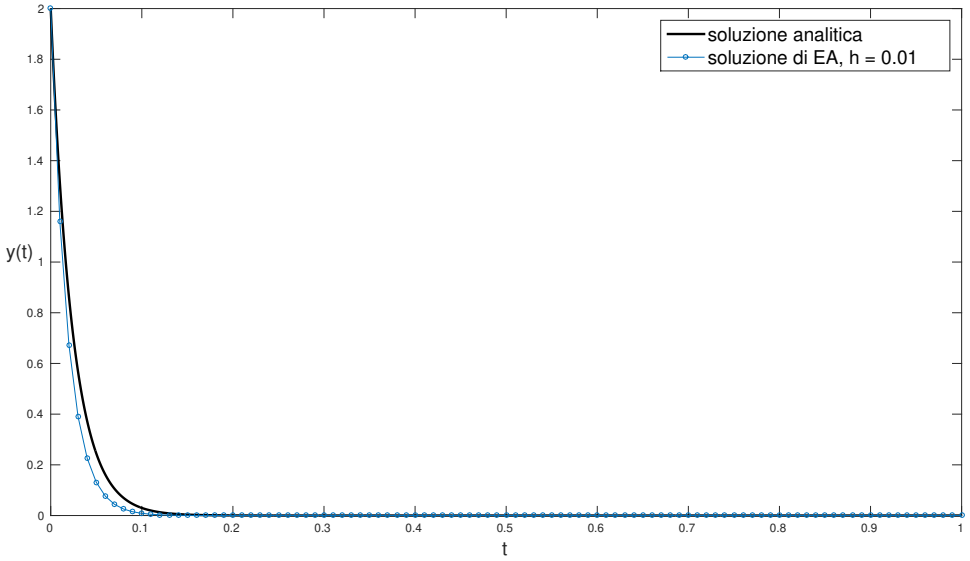


(a) Soluzione calcolata dal metodo di Eulero in avanti

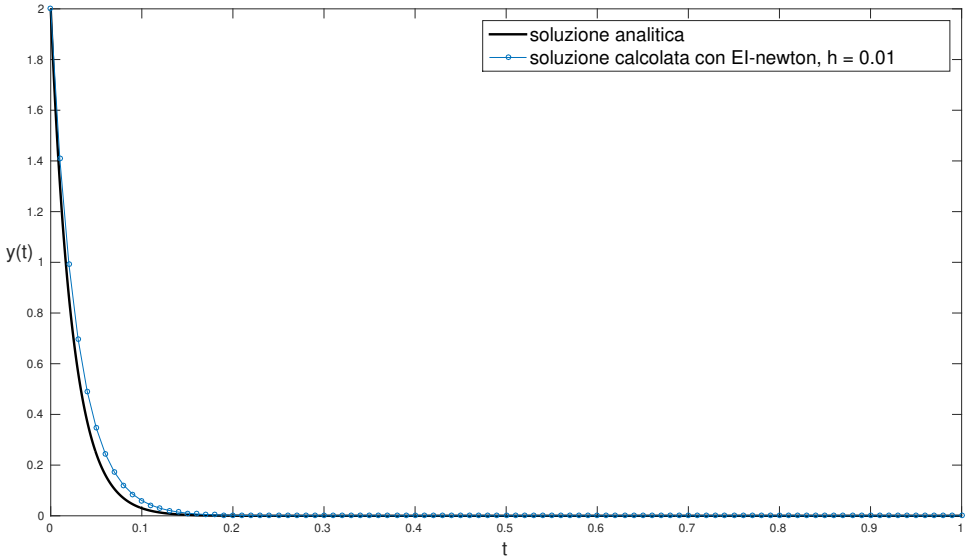


(b) Soluzione calcolata dal metodo di Eulero all'indietro

Figura 8: Soluzioni numeriche per $h = 0.05$



(a) Soluzione calcolata dal metodo di Eulero in avanti



(b) Soluzione calcolata dal metodo di Eulero all'indietro

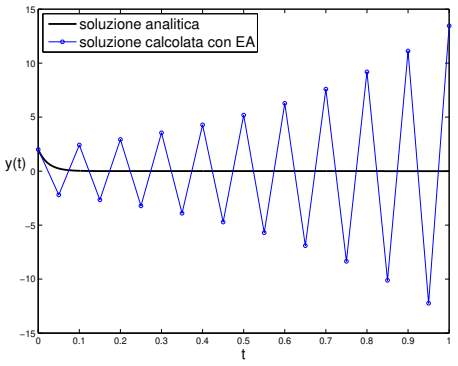
Figura 9: Soluzioni numeriche per $h = 0.01$

4. (*Importanza del solutore non lineare*)

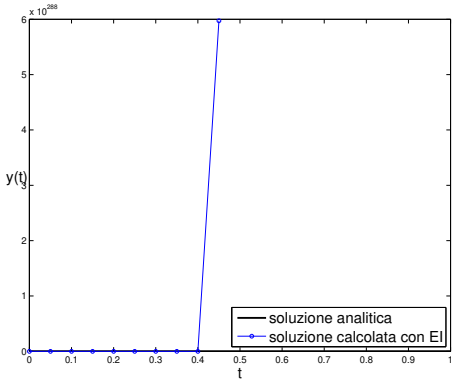
   (a) *Calcolare la soluzione numerica del problema (1) utilizzando il metodo di Eulero in avanti con i seguenti dati:* $t_{max} = 1$, *e* $h = 0.05$. *Visualizzare sullo stesso grafico la soluzione esatta e la soluzione numerica. Ripetere la stessa procedura utilizzando il metodo di Eulero all'indietro con metodo di punto fisso per la soluzione dell'equazione non lineare di avanzamento temporale (funzione* `eulero_indietro_pto_fisso`*). Cosa si osserva? Valutare se il valore* λh *appartiene alla regione di assoluta stabilità dei due metodi.*

   Il valore di λh è $\lambda h = -2.1$, esterno alla regione di assoluta stabilità del metodo (Figura10(c)). Di conseguenza il metodo di Eulero in avanti non produce una soluzione stabile, come si osserva dal grafico in Figura10(a).

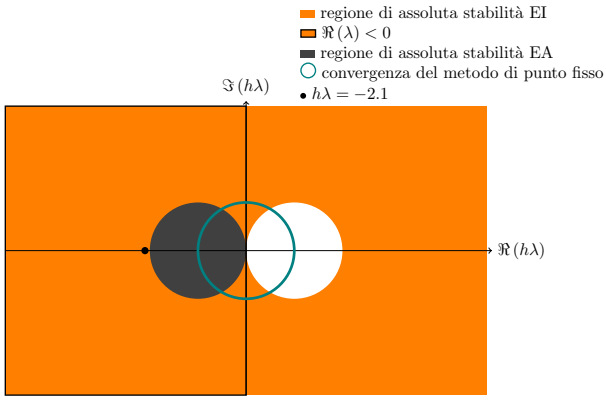
   Anche risolvendo con il metodo di Eulero all'indietro non si ottiene una soluzione stabile. Questo perchè il valore $\lambda h = -2.1$ appartiene alla regione di assoluta stabilità del metodo, ma non è compreso nella regione di convergenza del metodo di punto fisso. Si ottiene quindi il grafico in Figura10(b).



(a) Soluzione calcolata dal metodo di Eulero in avanti



(b) Soluzione calcolata dal metodo di Eulero all'indietro

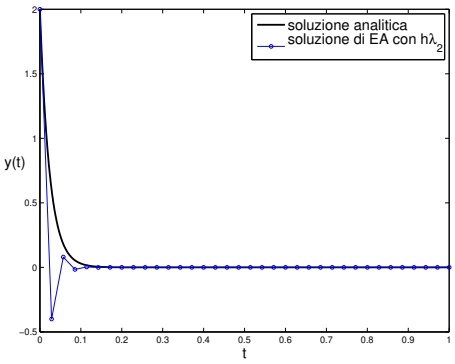


(c) Posizione di $\lambda h = -2.1$ nel piano complesso

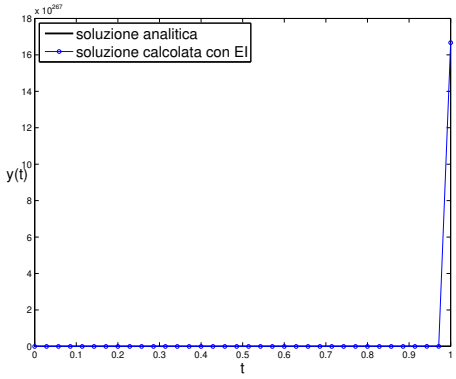
Figura 10: Soluzioni numeriche per $\lambda h = -2.1$

   (b) *Ripetere la stessa analisi del punto precedente usando prima* $h = 0.03$ *e poi* $h = 0.01$

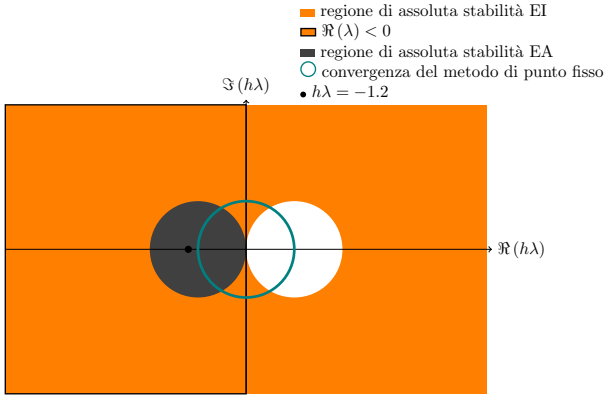
Nel primo caso $\lambda\hat{h} = -1.2$ è all'interno della regione di assoluta stabilità di entrambi i metodi, e tuttavia è ancora esterno alla regione di convergenza del metodo di punto fisso (Figura11(c)): questo significa che solo il metodo di Eulero in avanti produrrà risultati stabili. Ciò è confermato dai grafici riportati in Figura 11.



(a) Soluzione calcolata dal metodo di Eulero in avanti



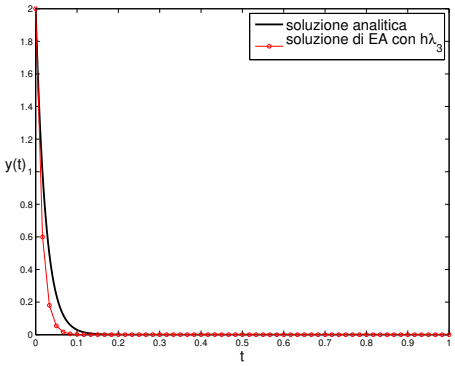
(b) Soluzione calcolata dal metodo di Eulero all'indietro



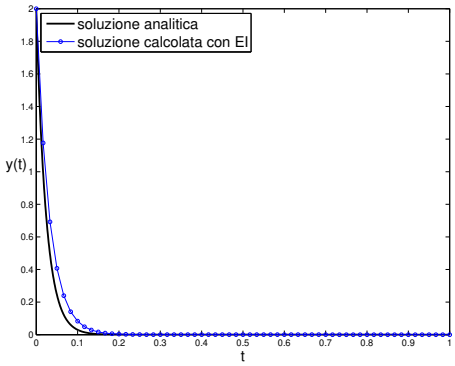
(c) Posizione di $\lambda h = -1.2$ nel piano complesso

Figura 11: Soluzioni numeriche per $\lambda h = -1.2$

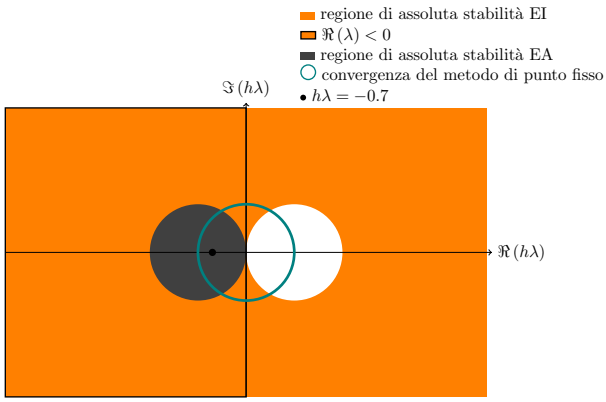
Nel secondo caso $\lambda\tilde{h} = -0.7$ è interno sia alla regione di assoluta stabilità di entrambi i metodi che alla regione di convergenza del metodo di punto fisso (Figura12(c)). In questo caso quindi anche il metodo di Eulero all'indietro produrrà risultati stabili, come si vede in Figura 12.



(a) Soluzione calcolata dal metodo di Eulero in avanti



(b) Soluzione calcolata dal metodo di Eulero all'indietro



(c) Posizione di $\lambda h = -0.7$ nel piano complesso

Figura 12: Soluzioni numeriche per $\lambda h = -0.7$

Esercizio 2.4

Si consideri il generico problema di Cauchy

$$\begin{cases} y'(t) = f(t,y) & t_0 < t \leq t_{max} \\ y(0) = y_0 \end{cases} \tag{2}$$

1. *Scrivere la function* `Crank_Nicolson.m` *che implementa il metodo di Crank-Nicolson per la risoluzione del problema di Cauchy (2) utilizzando la function* `ptofis.m` *per risolvere l'equazione non lineare ad ogni passo temporale del metodo. Si scelgano, per la function* `ptofis.m` *una tolleranza pari a* 10^{-5} *e un numero massimo di iterazioni pari a* 100.
   La funzione `Crank_Nicolson.m` *riceve in ingresso:*

   - *la funzione* $f(t,y)$ *definita come anonymous function (tramite il comando* `@`*). Si noti che* f *è una funzione di due variabili;*
   - *l'estremo* t_{max} *dell'intervallo di definizione;*
   - *il dato iniziale* y_0*;*
   - *il passo di avanzamento temporale* h*.*

   Si suppone che l'istante iniziale sia $t_0 = 0$*. La funzione* `Crank_Nicolson.m` *restituisce il vettore* `t_h` *degli istanti temporali discreti e il vettore* `u_h` *dei valori della corrispondente soluzione approssimata. Inoltre fornisce in uscita il vettore* `iter_pf` *che contiene il numero delle iterazioni effettuate dal metodo di punto fisso ad ogni passo temporale per risolvere l'equazione non lineare.*

   Una possibile implementazione della function è la seguente:

```
function [t_h,u_h,iter_pf]=Crank_Nicolson(f,t_max,y_0,h)

%[t_h,u_h,iter_pf]=Crank_Nicolson(f,t_max,y_0,h)
%
% Risolve il problema di Cauchy
%
% y'=f(t, y)
% y(0)=y_0
%
% utilizzando il metodo di Crank-Nicolson : u^{n+1}=u^n+1/2*h*( f^n +f^{n+1}).
% Per ciascun istante temporale l'equazione per u^(n+1)(a priori non lineare) viene
% risolta con un metodo di punto fisso: u=u^n+1/2*h*( f^n + f(t^{n+1},u) ),
% cioe' tramite la funzione di iterazione
%
% phi(x)=u^n+1/2*h*( f^n + f(t^{n+1},x) )
%
% la condizione per la convergenza del metodo di punto fisso ( |phi(x)'|<1) diventa:
%
% h*|\partial_x f(t^{n+1},x)| < 1
%
% ed e' garantita se h e' sufficientemente piccolo.
%
% Input:
% -> f: function che descrive il problema di Cauchy (dichiarata ad esempio tramite
%           inline o @) deve ricevere in ingresso due argomenti: f=f(t,y)
```

```
% -> t_max: l'istante finale dell' intervallo temporale di soluzione
%                 (l'istante iniziale e' t_0=0)
% -> y_0: il dato iniziale del problema di cauchy: y(t=0)=dato_iniziale
% -> h: l'ampiezza del passo di discretizzazione temporale.
%
% Output:
% -> t_h: vettore contenente gli istanti in cui si calcola la soluzione discreta
% -> u_h: la soluzione discreta calcolata nei nodi temporali t_h
% -> iter_pf: vettore che contiene il numero di iterazioni di punto fisso utilizzate
%                       per risolvere l'equazione non lineare ad ogni istante temporale.



% vettore degli istanti in cui risolvo la edo

t0=0;

t_h=t0:h:t_max;

% inizializzo il vettore che conterra' la soluzione discreta

N_istanti=length(t_h);

u_h=zeros(1,N_istanti);

% ciclo iterativo che calcola u^{n+1}=u^n+1/2*h*( f^n +f^{n+1}) .
% Ad ogni iterazione temporale devo eseguire delle sottoiterazioni di punto
% fisso per il calcolo di u^{n+1}:
%
% u_(n+1)^(k+1) = u_n + h/2*( f^n + f_( t_(n+1) , u_(n+1)^k ) ).

u_h(1)=y_0;

% parametri per le iterazioni di punto fisso
N_max=100;
toll=1e-5;

iter_pf=zeros(1,N_istanti);



for it=2:N_istanti

    % preparo le variabili per le sottoiterazioni

    u_old=u_h(it-1);
    f_old=f(t_h(it-1),u_old);

    t_pf=t_h(it);

    phi=@(u) u_old + 0.5*h * ( f_old + f( t_pf, u ) ) ;

    % sottoiterazioni

    [u_pf, it_pf] = ptofis(u_old, phi, N_max, toll);

    u_h(it)=u_pf(end);
```

```
    % tengo traccia dei valori di it_pf per valutare la convergenza delle
    % iterazioni di punto fisso

    iter_pf(it)=it_pf;

end
```

2. *Utilizzando la function appena scritta, risolvere il problema (1) con il metodo di Crank-Nicolson, scegliendo* $\lambda = -42$, $t_{max} = 1$, $y_0 = 2$ *per il valore del passo* $h = 0.02$.

   La soluzione si ottiene con i seguenti comandi:

```
y0=2;
lambda=-42;
tmax=1;
t_plot=0:0.01:tmax;
y=@(x) y0*exp(lambda*x);
f= @(t,y) lambda*y;
h=0.02;
[CN_t_h,CN_u_h,iter_pf]=Crank_Nicolson(f,tmax,y0,h);
plot(t_plot,y(t_plot),'k','LineWidth',2);
hold on
plot(CN_t_h,CN_u_h,'o','MarkerSize',4);
legend('soluzione analitica','CN con h=0.02',2);
```

   Il grafico è mostrato in Figura 13.

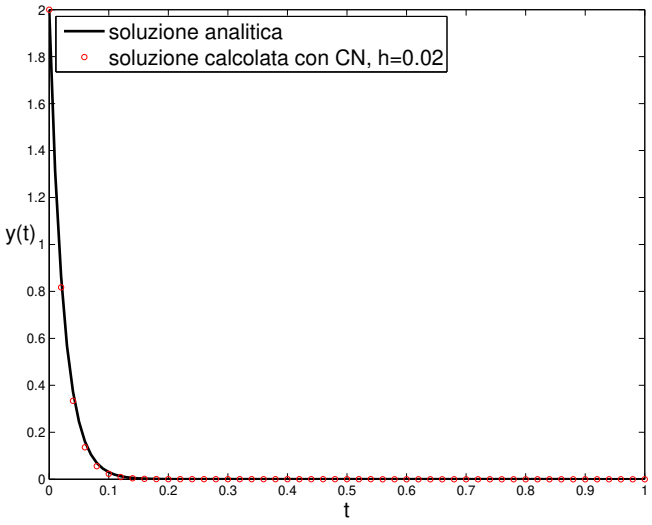


Figura 13: Confronto tra la soluzione esatta del problema (1) e la soluzione numerica ottenuta con il metodo di Crank-Nicolson, con passo di discretizzazione $h = 0.02$.

3. *Riportare su un grafico il numero di iterazioni impiegate dal metodo di punto fisso per risolvere l'equazione non lineare ad ogni passo temporale del metodo di Crank-Nicolson, scegliendo* $\lambda = -42$, $t_{max} = 1$, $y_0 = 2$ *e* $h = 0.02$.

```
figure;
plot(iter_pf,'o:','MarkerSize',4)
```

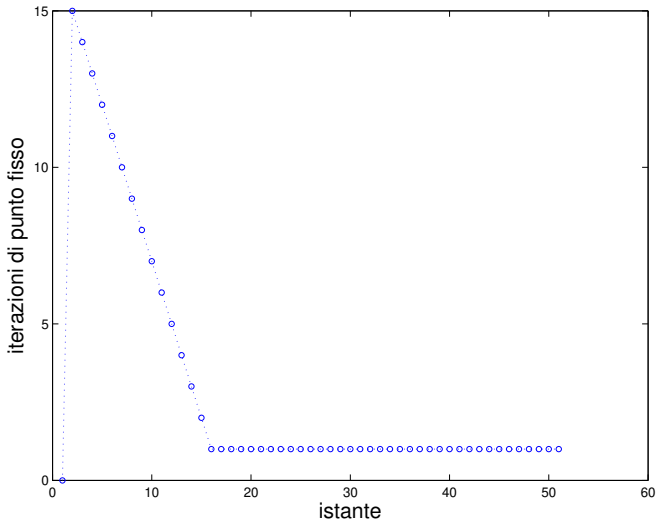


Figura 14: Numero di iterazioni di punto fisso effettuate dal metodo di Crank-Nicolson in funzione del tempo.

4. *Scrivere la function* `Heun.m` *che implementa il metodo di Heun per la risoluzione del problema di Cauchy (1). La funzione* `Heun.m` *riceve in ingresso:*

   - *la funzione* $f(t, y)$ *definita come anonymous function (tramite il comando* `@`*). Si noti che* f *è una funzione di due variabili;*
   - *l'estremo* t_{max} *dell'intervallo di definizione;*
   - *il dato iniziale* y_0*;*
   - *il passo di avanzamento temporale* h*.*

   Si suppone che l'istante iniziale sia $t_0 = 0$*. La funzione* `Heun.m` *restituisce il vettore* `t_h` *degli istanti temporali discreti e il vettore* `u_h` *dei valori della corrispondente soluzione approssimata.* Una possibile implementazione della function è la seguente:

```
function [t_h,u_h]=Heun(f,t_max,y_0,h)

%[t_h,u_h]=Heun(f,t_max,y_0,h)
%
% Risolve il problema di Cauchy
%
% y'=f(t, y)
% y(0)=y_0
%
% utilizzando il metodo di Heun : u_{n+1}=u_n+1/2*h*( f(t_n,u_n) +f(t_{n+1},u_n+h f(t_n,u_n)))
%
% Input:
% -> f: function che descrive il problema di Cauchy (dichiarata ad esempio tramite
```

```
%           inline o @) deve ricevere in ingresso due argomenti: f=f(t,y)
% -> t_max: l'istante finale dell' intervallo temporale di soluzione
%                 (l'istante iniziale e' t_0=0)
% -> y_0: il dato iniziale del problema di cauchy: y(t=0)=dato_iniziale
% -> h: l'ampiezza del passo di discretizzazione temporale.
%
% Output:
% -> t_h: vettore contenente gli istanti in cui si calcola la soluzione discreta
% -> u_h: la soluzione discreta calcolata nei nodi temporali t_h

% vettore degli istanti in cui risolvo la edo

t0=0;

t_h=t0:h:t_max;

% inizializzo il vettore che conterra' la soluzione discreta

N_istanti=length(t_h);

u_h=zeros(1,N_istanti);

u_h(1)=y_0;

for it=2 : N_istanti

    u_h(it) = u_h(it-1) + h/2 * ( f( t_h(it-1), u_h(it-1) ) + ...
                                  f( t_h(it), u_h(it-1) + h * f( t_h(it-1), u_h(it-1) ) ) );
end
```

5. *Utilizzando la function appena scritta, risolvere il problema (1) con il metodo di Heun, scegliendo* $\lambda = -42$, $t_{max} = 1$, $y_0 = 2$ *per il valore del passo* $h = 0.02$.

   Consideriamoi seguenti comandi:

```
y0=2;
lambda=-42;
tmax=1;
t_plot=0:0.01:tmax;
y=@(x) y0*exp(lambda*x);
f= @(t,y) lambda*y;
h=0.02;
[H_t_h,H_u_h]=Heun(f,tmax,y0,h);
plot(t_plot,y(t_plot),'k','LineWidth',2);
hold on
plot(H_t_h,H_u_h,'o','MarkerSize',4);
legend('soluzione analitica','H con h=0.02',2);
```

   Il grafico è mostrato in Figura 15.

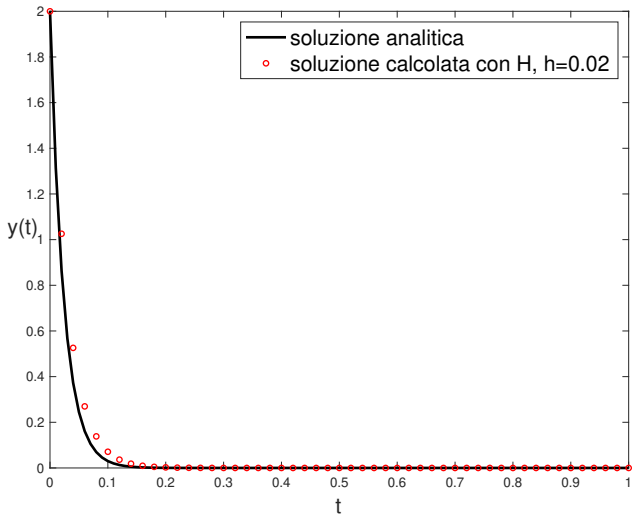


Figura 15: Confronto tra la soluzione esatta del problema (1) e la soluzione numerica ottenuta con il metodo di Heun, con passo di discretizzazione $h = 0.02$.

6. *Definiamo* $e_h = \max_{n\in[0,1,\ldots,N_h]} |y(t_n) - u_n|$, *il massimo del modulo dell'errore compiuto approssimando la soluzione esatta* $y(t_n)$ *con la soluzione numerica* u_n. *Riportare, su un grafico in scala logaritmica, l'andamento di* e_h, *ottenuto utilizzando i metodi di Eulero all'indietro (funzione* `eulero_indietro_pto_fisso`*), Crank-Nicolson, Heun, il metodo di Runge-Kutta di ordine 4 (si usi la function* `Runge_Kutta_4.m` *fornita), al variare di* h, *utilizzando i passi temporali* $h = [0.02, 0.01, 0.005, 0.0025, 0.00125]$. *Confrontare gli ordini di convergenza ottenuti con quelli previsti dalla teoria.*

```
N=5;
dimezz=2.^(1:N);
passi=0.04./dimezz;

for it=1:N
      [EI_t_h, EI_u_h] = eulero_indietro_pto_fisso( f, tmax, y0, passi(it) );
      [CN_t_h, CN_u_h] = Crank_Nicolson( f, tmax, y0, passi(it) );
      [H_t_h, H_u_h] = Heun( f, tmax, y0, passi(it) );
      [RK_t_h, RK_u_h] = Runge_Kutta_4(f, tmax, y0, passi(it));
      y_h=y( 0 : passi(it) : tmax  );
      errore_EI(it) = max( abs (y_h-EI_u_h) );
      errore_CN(it) = max( abs (y_h-CN_u_h) );
      errore_H(it) = max( abs (y_h-H_u_h) );
      errore_RK(it) = max( abs (y_h-RK_u_h) );
end

figure;

loglog(passi,errore_CN,'-o','LineWidth',2);
hold on
loglog(passi,errore_H,'-o','LineWidth',2);
loglog(passi,errore_EI,'-o','LineWidth',2);
```

```
loglog(passi,errore_RK,'-o','LineWidth',2);
plot(passi,passi,'k')
plot(passi,(passi).^2,'k--')
plot(passi,(passi).^4,'k-.')
grid on

xlabel('h','FontSize',16);
lab=ylabel('err(h)','FontSize',16,'Rotation',0);
set(lab,'Position',[0.0007 0.2])

leg=legend('errore CN','errore H', 'errore EI','errore RK','ordine 1','ordine 2','ordine 4');
```

Il grafico è mostrato in Figura 16. Come da teoria, si trova che il metodo di Eulero indietro è un metodo del primo ordine, Crank-Nicolson e Heun del secondo e Runge-Kutta-4 del quarto. Infatti gli andamenti degli errori sono paralleli in questo grafico in scala logaritmica su entrambi gli assi a quelli delle rette del tipo (h, h^p), dove $p = 1, 2$ e 4 sono rispettivamente gli ordini di convergenza dei metodi.

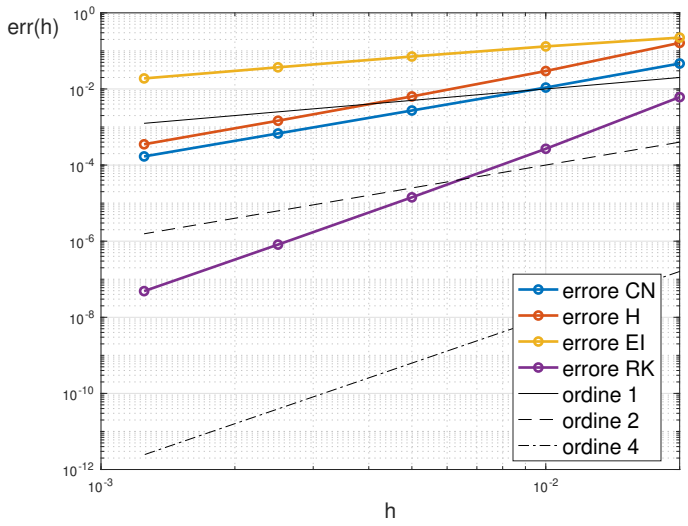


Figura 16: Andamento degli errori per i metodi di Eulero all'indietro, Crank-Nicolson e Heun al variare di h; grafico in scala log-log.

Esercizio 3.1

1. *Dopo aver scritto l'equazione differenziale di ordine 2 sotto forma di un sistema di due equazioni differenziali di ordine 1, scrivere la* `function` `mms` *che restituisce la valutazione dei termini di destra del sistema.*

   L'equazione differenziale di ordine 2 può essere riscritta come un sistema di due equazioni differenziali nel seguente modo:

   $$\begin{cases} y_1'(t) = y_2(t) & t \in (t_0, t_f), \\ y_2'(t) = -\gamma y_2(t) - \omega^2 y_1(t) & t \in (t_0, t_f), \\ y_1(t_0) = 2, \quad y_2(t_0) = 0. \end{cases}$$

   Ovvero abbiamo

   $$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) & t \in (t_0, t_f), \\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases}$$

   dove $\mathbf{y} = (y_1, y_2)^T$, $\mathbf{y}_0 = (2, 0)^T$ e $\mathbf{f} : (t_0, t_f) \times \mathbb{R}^2 \to \mathbb{R}^2$ assume l'espressione seguente:

   $$\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} y_2 \\ -\gamma\, y_2 - \omega^2\, y_1 \end{bmatrix}.$$

   La `function` `mms` che restituisce la valutazione dei termini di destra del sistema, ovvero $\mathbf{f}(t, \mathbf{y})$, è allora

```
function fn = mms(t,y)
gamma = 0.1;
omega2 = 1;
[n,m] = size(y);
fn = zeros(n,m);
fn(1) = y(2);
fn(2) = -gamma*y(2) - omega2*y(1);
return
```

2. *Risolvere il problema nell'intervallo di tempo* $(0, 100)$ *con le condizioni iniziali* $x(0) = 2$, $x'(0) = 0$ *usando la* `function` `ode45` *built-in di Matlab®* . *Verificare graficamente che la soluzione del problema è costituita da un'oscillazione armonica di frequenza* $\omega_1/2\pi$ *con* $\omega_1 = \sqrt{4\omega^2 - \gamma^2}/2$ *smorzata nel tempo.*

   Consideriamo i seguenti comandi Matlab® :

```
[t,u] = ode45('mms',[0 100],[2 0]);
plot(t,u(:,1))
```

   Il grafico della posizione della massa in funzione del tempo è riportato in Figura 17.

3. *Aggiungere al sistema la forzante esterna* $F_{ext}(t) = m\, A_0 \sin(\omega_f t)$, *con* $A_0 = 0.5$ *e* $\omega_f = 0.5$, *ottenendo il seguente bilancio delle forze* $F_k + F_c + F_{ext} = mx''$. *Verificare*

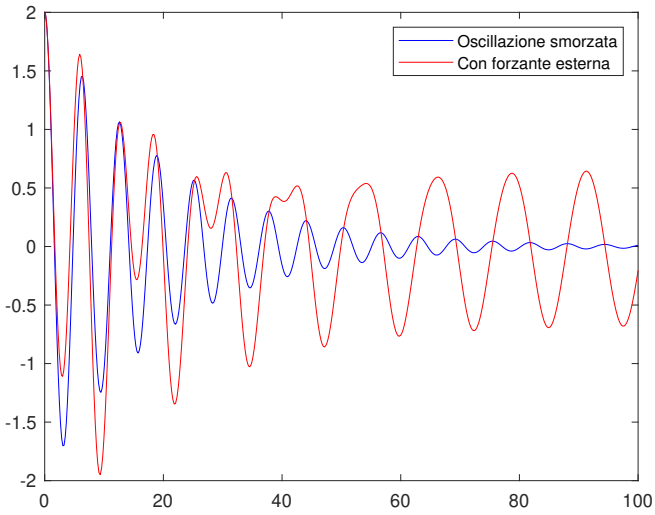


Figura 17: Esercizio 1. Andamento nel tempo del sistema massa-molla-smorzatore, con e senza forzante esterna.

graficamente che, a regime, la soluzione del problema è un'oscillazione armonica di frequenza $\omega_f/2\pi$. Il sistema in questo caso diventa:

$$\begin{cases} y_1' = y_2 \\ y_2' = -\gamma y_2 - \omega^2 y_1 + A_0 \sin(\omega_f t) \end{cases}$$

e i termini di destra vengono valutati attraverso la `function mms_forz`:

```
function fn = mms_forz(t,y)
gamma = 0.1;
omega2 = 1;
A = 0.5;
wf = 0.5;
[n,m] = size(y);
fn = zeros(n,m);
fn(1) = y(2);
fn(2) = -gamma*y(2) - omega2*y(1) + A*sin(wf*t);
return
```

La soluzione si ottiene con comandi analoghi al caso precedente; il confronto tra la soluzione ottenuta con e senza la forzante è riportato in Figura 17.

```
[tf,uf] = ode45('mms_forz',[0 100],[2 0]);
plot(t,u(:,1),'b',tf,uf(:,1),'r')
legend('Oscillazione smorzata','Con forzante esterna')
```

Esercizio 3.2

1. *Si riscriva il sistema di EDO del secondo ordine come un sistema di EDO del primo ordine di dimensione m.*

   Il sistema di 2 equazioni differenziali di ordine 2 può essere riscritto come un sistema di $m = 4$ equazioni differenziali del primo ordine, assumendo $y_1(t) = r_x(t)$, $y_2(t) = r'_x(t)$, $y_3(t) = r_y(t)$, $y_4(t) = r'_y(t)$, da cui

   $$\begin{aligned}
   &y'_1(t) = y_2(t) && t \in (t_0, t_f),\\
   &y'_2(t) = -4\pi^2 \frac{y_1(t)}{[(y_1(t))^2+(y_3(t))^2]^{3/2}} && t \in (t_0, t_f),\\
   &y'_3(t) = y_4(t) && t \in (t_0, t_f),\\
   &y'_4(t) = -4\pi^2 \frac{y_3(t)}{[(y_1(t))^2+(y_3(t))^2]^{3/2}} && t \in (t_0, t_f),\\
   &y_1(t_0) = r_{0,x}, \quad y_2(t_0) = v_{0,x}, \quad y_3(t_0) = r_{0,y}, \quad y_4(t_0) = v_{0,y}.
   \end{aligned}$$

   Ovvero abbiamo

   $$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) & t \in (t_0, t_f),\\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases}$$

   dove $\mathbf{y} = (y_1, y_2, y_3, y_4)^T$, $\mathbf{y}_0 = (r_{0,x}, v_{0,x}, r_{0,y}, v_{0,y})^T$ e $\mathbf{f} : (t_0, t_f) \times \mathbb{R}^4 \to \mathbb{R}^4$ assume l'espressione seguente:

   $$\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} y_2 \\ -4\pi^2 \frac{y_1}{[(y_1)^2+(y_3)^2]^{3/2}} \\ y_4 \\ -4\pi^2 \frac{y_3}{[(y_1)^2+(y_3)^2]^{3/2}} \end{bmatrix}.$$

2. *Posti $\mathbf{r}_0 = (1, 0)^T$ e $\mathbf{v}_0 = (0, -5.1)^T$, si risolva il problema nell'intervallo di tempo $(0, 5)$ usando opportunamente la `function ode45` di Matlab® . Si rappresentino l'andamento della posizione $\mathbf{r}(t)$ approssimata vs. t e l'orbita del pianeta nel piano. Si usi la funzione come segue `[t, u] = ode45( ..., options );` avendo assegnato in precedenza `options = odeset('reltol',1e-6);`.*

   Iniziamo scrivendo la funzione `twobody.m`

```
function f = twobody( t, y )
%
% TWO BODY Function - Sun (assumed fixed in space) - Earth
% y( 1 ) = x_e Earth
% y( 2 ) = dx_e/dt Earth
% y( 3 ) = y_e Earth
% y( 4 ) = dy_e/dt Earth
%
% Dimensionless eqs. UA units
%
```

```
[ n, m ] = size( y );
f = zeros( n, m );

Den = ( y( 1 )^2 + y( 3 )^2 )^( 3 / 2 );

f( 1 ) = y( 2 );
f( 2 ) = 4 * pi^2 * ( - y( 1 ) / Den );
f( 3 ) = y( 4 );
f( 4 ) = 4 * pi^2 * ( - y( 3 ) / Den );

return
```

Eseguiamo ora i seguenti comandi Matlab® :

```
t0 = 0; tf = 5;
y0 = [ 1 0 0 -5.1 ]';
options = odeset('reltol' ,1e-6);
[ tv, uv ] = ode45( 'twobody', [ t0 tf ], y0, options );
uv = uv';
figure(1);
plot( tv, uv( 1, : ), '--k', tv, uv( 3, : ), '--m', 'LineWidth', 2 );
axis( [ t0 tf -2 2 ] ); axis tight
xlabel('t'); ylabel('x, y'); grid on

figure( 2 )
plot( 0, 0, 'xk', uv( 1, 1 ), uv( 3, 1 ), 'sb', ...
      uv( 1, : ), uv( 3, : ), '-b', 'LineWidth', 2, 'MarkerSize', 10 );
xlabel('x'); ylabel('y'); grid on
axis equal
axis( [ -2 2 -2 2 ] );
```

Otteniamo le soluzioni riportate in Figura 18 e l'orbita indicata in Figura 19. Dalla dimensione della matrice `uv` notiamo che il metodo adattivo ha ottenuto la soluzione con 1137 tempi discreti e un passo h variabile, come possiamo osservare dalla Figura 20 tramite i seguenti comandi Matlab® :

```
plot( tv(1:end-1), tv(2:end)-tv(1:end-1), 'LineWidth', 2);
xlabel( 't'); ylabel('h');  grid on
```

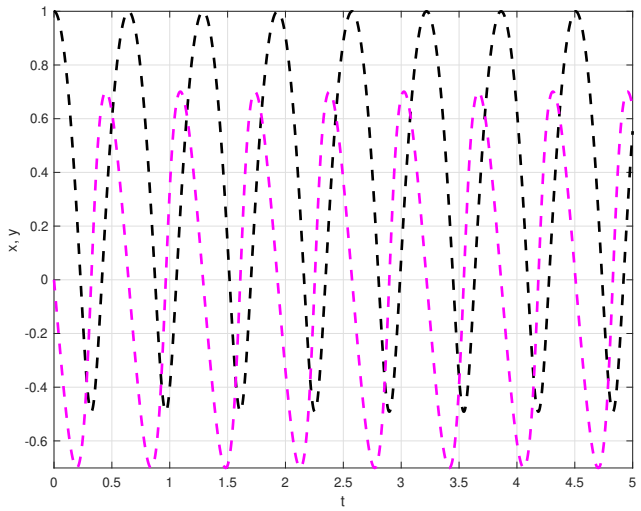


Figura 18: Esercizio 2.2. Soluzioni approssimate $y_1(t) = r_x(t)$ e $y_3(t) = r_y(t)$ vs. t ottenute tramite il metodo `ode45`

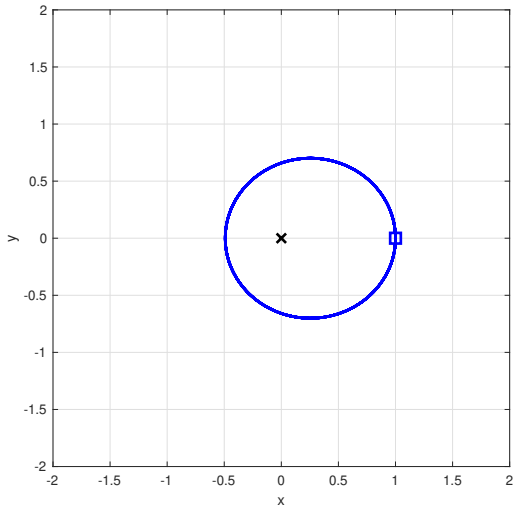


Figura 19: Esercizio 2.2. Orbita corrispondente alle soluzioni approssimate $y_1(t) = r_x(t)$ e $y_3(t) = r_y(t)$ ottenute tramite il metodo `ode45`

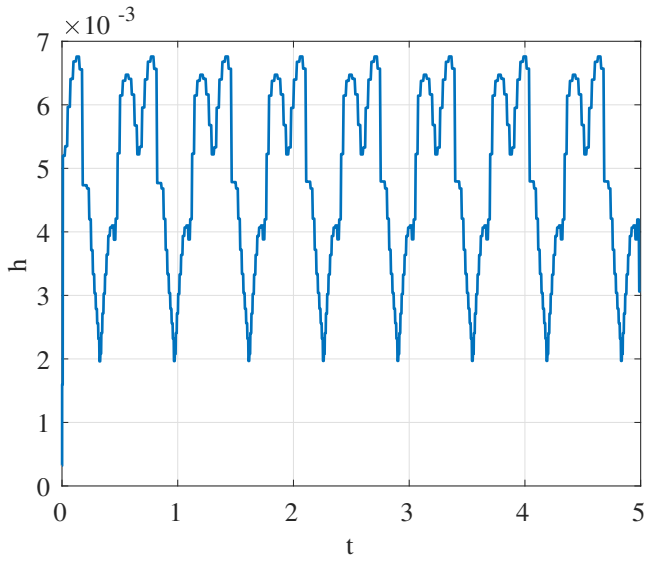


Figura 20: Esercizio 2.2. Passo h vs. t_n per il metodo adattivo `ode45`

3. *Si scriva una* `function` *Matlab® `eulero_avanti_sistemi.m` che implementi il metodo di Eulero in avanti per l'approssimazione di un sistema di EDO del primo ordine. L'intestazione della funzione sarà:*

```
[ t, u ] = eulero_avanti_sistemi( @twobody, [t0 tf], y0, Nh )
```

Si consideri ad esempio la seguente implementazione:

```
function [ t, u ] = eulero_avanti_sistemi( fun, tv, y0, Nh )
h = ( tv( end ) - tv( 1 ) ) / Nh;
u = zeros( size( y0, 1 ), Nh + 1 );
t = linspace( tv( 1 ), tv( end ), Nh + 1 );
u( :, 1 ) = y0;
for n = 1 : Nh
    u( :, n + 1 ) = u( :, n ) + h * fun( t( n ), u( :, n ) );
end
return
```

4. *Si utilizzi la funzione Matlab® `eulero_avanti_sistemi.m` per risolvere il problema con i dati precedentemente indicati e per valori del passo $h = 10^{-5}$ ($Nh = 5 \cdot 10^5$); si rappresenti la soluzione numerica ottenuta. Si ripeta per $h = 10^{-3}$ e si commenti il risultato ottenuto.* Tramite i seguenti comandi Matlab® otteniamo il risultato delle Figure 21 e l'orbita indicata in Figura 22. In questo caso la soluzione approssimata è stata ottenuta con un numero di passi pari a 500001, molto maggiore che con il metodo adattivo `ode45`.

```
h = 1e-5;
Nh = round( ( tf - t0 ) / h );
```

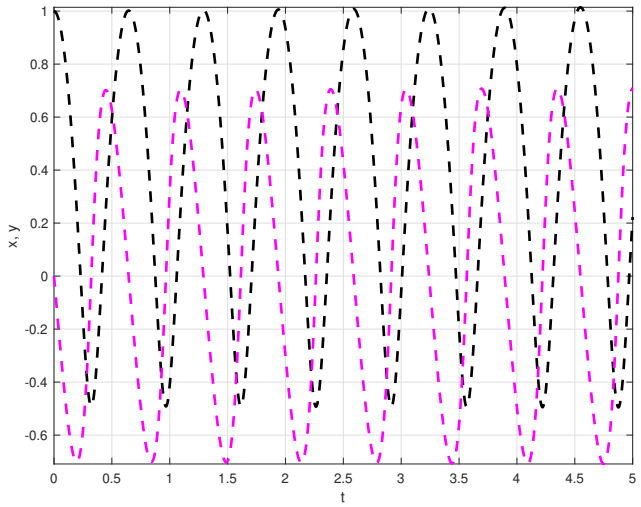


Figura 21: Esercizio 2.4. Soluzioni approssimate $y_1(t) = r_x(t)$ e $y_3(t) = r_y(t)$ vs. t ottenute tramite il metodo di Eulero in avanti

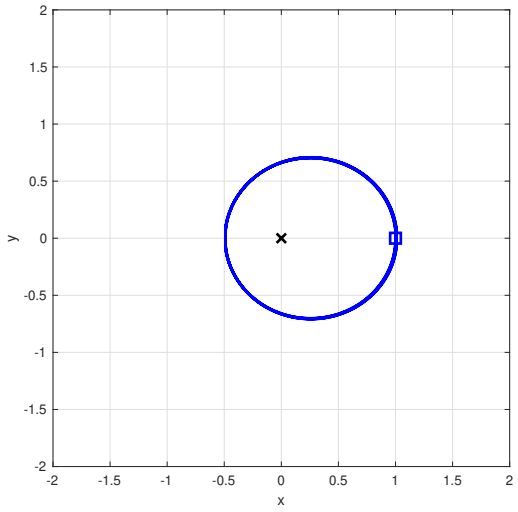


Figura 22: Esercizio 2.4. Orbita corrispondente alle soluzioni approssimate $y_1(t) = r_x(t)$ e $y_3(t) = r_y(t)$ ottenute tramite il metodo di Eulero in avanti

```
[ tEA, uEA ] = eulero_avanti_sistemi( @twobody, [ t0 tf ], y0, Nh );

figure( 3 )
plot( tEA, uEA( 1, : ), '--k', tEA, uEA( 3, : ), '--m', 'LineWidth', 2 );
axis( [ t0 tf -2 2 ] ); axis tight
xlabel('t'); ylabel('x, y'); grid on
```

```
figure( 4 )
plot( 0, 0, 'xk', uEA( 1, 1 ), uEA( 3, 1 ), 'sb', ...
      uEA( 1, : ), uEA( 3, : ), '-b', 'LineWidth', 2, 'MarkerSize', 10 );
xlabel('x'); ylabel('y'); grid on
axis equal
axis( [ -2 2 -2 2 ] );
```

Ripetendo ora i comandi precedenti con $h = 10^{-3}$ otteniamo il risultato di Figura 23. Risulta evidente come tale soluzione approssimata non sia adeguata per la rappresentazione del fenomeno fisico in questione. Dato che l'intervallo di tempo $[t_0, t_f]$ è limitato, stiamo osservando un'istanza del problema della *zero–stabilità* del metodo; questa è controllabile scegliendo un valore del passo h "sufficientemente" piccolo per il metodo di Eulero in avanti.

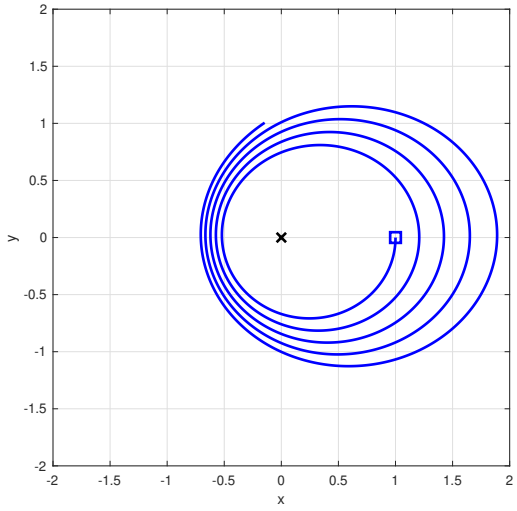


Figura 23: Esercizio 2.4. Orbita corrispondente alle soluzioni approssimate $y_1(t) = r_x(t)$ e $y_3(t) = r_y(t)$ ottenute tramite il metodo di Eulero in avanti

Esercizio 3.3

1. *Si riscriva il modello SEIR come un sistema di $m = 3$ EDO del primo ordine. Posti $N = 10^7$, $E_0 = 100$, $I_0 = 30$, $R_0 = 0$, $\beta = 0.7586$ giorni^{-1}, $\alpha^{-1} = 5.2$ giorni, $\gamma^{-1} = 2.9$ giorni, $t_0 = 0$ e $t_f = 300$ giorni, si risolva il problema tramite il metodo di Eulero in avanti con passo $h = 0.25$ giorni usando la funzione* `eulero_avanti_sistemi.m`. *Che percentuale della popolazione ($100\, R_\infty/N$) rientra nel gruppo dei rimossi al tempo $t = t_f$?*

   Poniamo $\mathbf{y} = (y_1, y_2, y_3)^T$, con $y_1 = S$, $y_2 = E$ e $y_3 = I$; inoltre $\mathbf{y}_0 = (S_0, E_0, I_0)^T$. Otteniamo dunque il sistema di EDO del primo ordine seguente:

$$\begin{cases} \mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t)) & t \in (t_0, t_f), \\ \mathbf{y}(t_0) = \mathbf{y}_0, \end{cases}$$

   dove $\mathbf{f} : (t_0, t_f) \times \mathbb{R}^3 \to \mathbb{R}^3$ assume l'espressione seguente:

$$\mathbf{f}(t, \mathbf{y}) = \begin{bmatrix} -\beta \frac{y_1\, y_3}{N} \\ \beta \frac{y_1\, y_3}{N} - \alpha\, y_2 \\ \alpha\, y_2 - \gamma\, y_3 \end{bmatrix}.$$

   Ricordiamo che $R(t) = N - \sum_{i=1}^{3} y_i(t)$ per $t \in [t_0, t_f]$. Scriviamo come segue la funzione `seir.m` che implementa $\mathbf{f}(t, \mathbf{y})$:

```
function f = seir( t, y )
%
% SEI-(R)
% y( 1 ) = S
% y( 2 ) = E
% y( 3 ) = I
N = 1e7;
beta = 0.7586;
alpha = 1 / 5.2;
gamma = 1 / 2.9;
[ n, m ] = size( y );
f = zeros( n, m );
f( 1 ) = - beta * y( 1 ) .* y( 3 ) / N;
f( 2 ) = beta * y( 1 ) .* y( 3 ) / N - alpha * y( 2 );
f( 3 ) = alpha * y( 2 ) - gamma * y( 3 );
return
```

   Consideriamo ora i seguenti comani Matlab® per ottenere il grafico di Figura 24:

```
t0 = 0; tf = 300;
N = 1e7;
E0 = 100;   I0 = 30;    S0 = N - E0 - I0;
y0 = [ S0 E0 I0 ]';
h = 0.25;
Nh = round( ( tf - t0 ) / h );
[ tEA, uEA ] = eulero_avanti_sistemi( @seir, [ t0 tf ], y0, Nh );
REA = N - uEA( 1, : ) - uEA( 2, : ) - uEA( 3, : );
```

```
figure( 1 )
plot( tEA, uEA( 1, : ), '-b', tEA, uEA( 2, : ), '-r', ...
      tEA, uEA( 3, : ), '-m', tEA, REA, '-k', 'LineWidth', 2 );
xlabel( 't [giorni]' );
ylabel( '# per compartimento');
legend( 'S', 'E', 'I', 'R', 'Location', 'Best' );
grid on
```

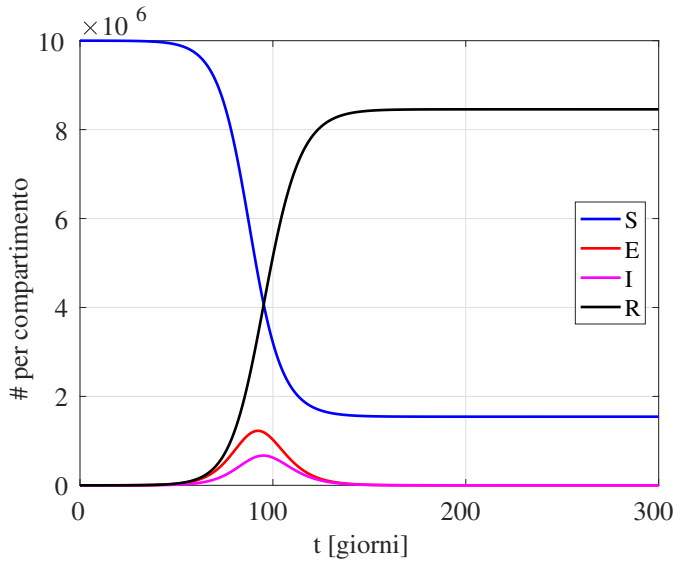


Figura 24: Esercizio 3.1. Evoluzione di $S(t)$, $E(t)$, $I(t)$ e $R(t)$ vs. t con $\beta = 0.7586$ giorni $^{-1}$; soluzione approssimata tramite il metodo di Eulero in avanti

Osserviamo come il numero di infettivi giornalieri $I(t)$ raggiunga il numero massimo all'incirca al giorno 95. Il valore limite dei rimossi $R(t)$ per $t = t_f$ corrisponde a `REA(end)`, da cui deduciamo che circa l'84.6% della popolazione sia stata coinvolta nella dinamica epidemiologica prima della rimozione (guarigione, isolamento, decesso, etc...). Osserviamo che in questo caso il numero di riproduzione di base è $r_0 = \frac{\beta}{\gamma} = 2.2$, ovvero ogni infezione genera in media 2.2 infetti secondari.

2. *Si risolva ora il problema assumendo che il valore β passi da* 0.7586 *a* 0.25 *giorni*$^{-1}$ *a partire dal giorno* $t = 80$ *a seguito dell'attuazione di una politica di isolamento della popolazione. Che percentuale della popolazione rientra nel gruppo dei rimossi al tempo* $t = t_f$*? Cosa succede se il valore di* β *risale a* 0.65 *giorni*$^{-1}$ *al giorno* $t = 120$*?* Ripetiamo il punto 1, modificando la funzione `seir.m` per esempio come segue:

```
function f = seir( t, y )
%
% SEI-(R)
% y( 1 ) = S
% y( 2 ) = E
% y( 3 ) = I
N = 1e7;
% beta = 0.7586;
beta = 0.7586 * ( t < 80 ) + 0.25 * ( t >= 80 );
% beta = 0.7586 * ( t < 80 ) + 0.25 * ( t >= 80 ) .* ( t < 120 ) + 0.65 * ( t >= 120 );
alpha = 1 / 5.2;
gamma = 1 / 2.9;
[ n, m ] = size( y );
f = zeros( n, m );
f( 1 ) = - beta * y( 1 ) .* y( 3 ) / N;
f( 2 ) = beta * y( 1 ) .* y( 3 ) / N - alpha * y( 2 );
f( 3 ) = alpha * y( 2 ) - gamma * y( 3 );
return
```

Otteniamo, eseguendo i comandi Matlab® visti in precedenza il risultato di Figura 25.

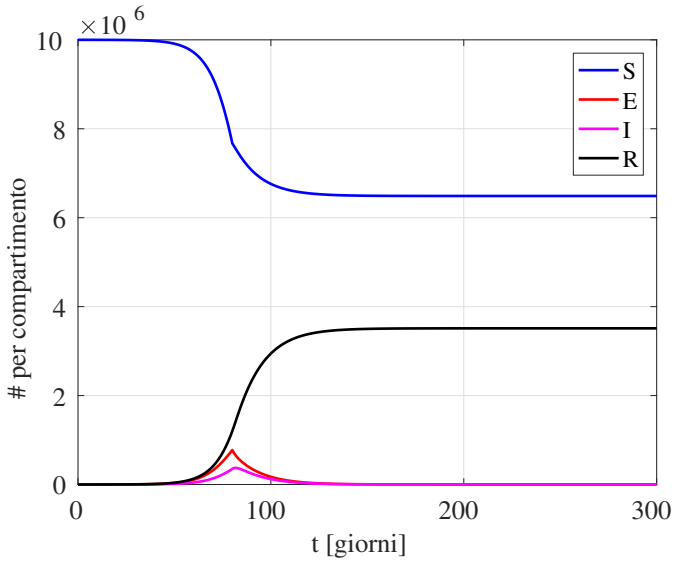


Figura 25: Esercizio 3.2. Evoluzione di $S(t)$, $E(t)$, $I(t)$ e $R(t)$ vs. t con $\beta = 0.7586$ giorni^{-1} per $t < 80$ giorni e $\beta = 0.25$ giorni^{-1} per $t \geq 80$; soluzione approssimata tramite il metodo di Eulero in avanti

Osserviamo come il numero di infettivi giornalieri $I(t)$ raggiunga il numero massimo all'incirca al giorno 82. Il valore limite dei rimossi $R(t)$ per $t = t_f$ corrisponde a circa

il 35.1% della popolazione. Osserviamo che in questo caso il numero di riproduzione di base diventa $r_0 = 0.725 < 1$ a partire dal giorno $t = 80$.

Modificando ulteriormente il file `seir.m` come

```
...
beta = 0.7586 * ( t < 80 ) + 0.25 * ( t >= 80 ) .* ( t < 120 ) + 0.65 * ( t >= 120 );
...
```

otteniamo il risultato di Figura 26, da cui si evince che a $t = t_f$ il numero dei rimossi corrisponde a circa il 59.2% della popolazione. Le curve degli infetti ed esposti presentano due massimi locali nell'intervallo di tempo in considerazione. A partire dal giorno 120, abbiamo $r_0 = 1.89 > 1$.

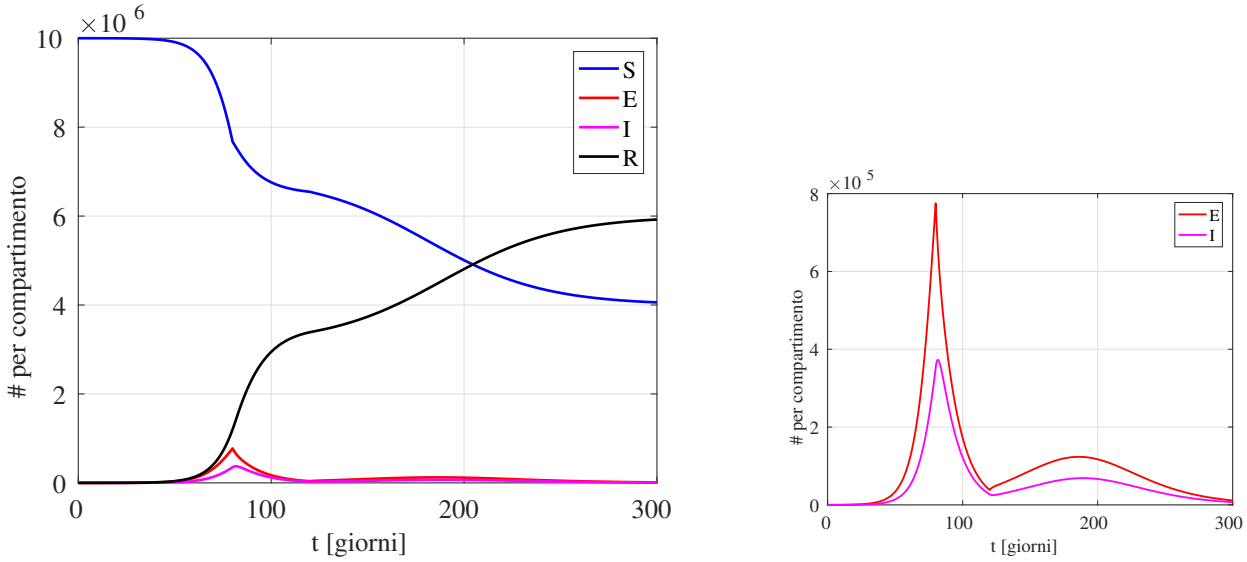


Figura 26: Esercizio 3.2. Evoluzione di $S(t)$, $E(t)$, $I(t)$ e $R(t)$ vs. t con $\beta = 0.7586$ giorni^{-1} per $t < 80$ giorni, $\beta = 0.25$ giorni^{-1} per $t \in [80, 120)$ e $\beta = 0.65$ giorni^{-1} per $t \geq 120$; soluzione approssimata tramite il metodo di Eulero in avanti

3. *Usando i dati di cui al punto 1, si studi l'assoluta stabilità del metodo di Eulero in avanti, sapendo che* $\lim_{t\to+\infty} S(t) = S_\infty = N - R_\infty$ *e* $\lim_{t\to+\infty} E(t) = \lim_{t\to+\infty} I(t) = 0$.

   Anche se il problema non è lineare in $\mathbf{y}$, ovvero $\mathbf{f}(t, \mathbf{y})$ non dipende linearmente da $\mathbf{y}$, possiamo ricondurre l'analisi di assoluta stabilità del metodo di Eulero in avanti agli autovalori di $\Lambda(\mathbf{y}(t)) = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(t, \mathbf{y}(t))$ per $t \in [t_0, t_f]$, immaginando di conoscere la soluzione analitica del problema $\mathbf{y}(t)$. Abbiamo:

$$\Lambda(\mathbf{y}) = \begin{bmatrix} -\beta \frac{y_3}{N} & 0 & -\beta \frac{y_1}{N} \\ \beta \frac{y_3}{N} & -\alpha & \beta \frac{y_1}{N} \\ 0 & \alpha & -\gamma \end{bmatrix}.$$

Dopo aver ottenuto numericamente un'approssimazione particolarmente accurata di $\mathbf{y}(t)$, studiamo gli autovalori $\lambda_i(t)$, di $\Lambda(\mathbf{y}(t))$ per $i = 1, 2, 3$. In particolare, per quanto concerne l'assoluta stabilità ci interessiamo alla loro parte reale negativa, almeno per t "sufficientemente" grande.

Sappiamo che il metodo di Eulero in avanti è assolutamente stabile se $(\lambda_i(t)\, h) \in \mathcal{A} = \{z \in \mathbb{C} : |1 + z| < 1\}$ per ogni $t \in [t_0, t_f]$. Usiamo i seguenti comandi Matlab® per calcolare gli autovalori e localizzarli nel piano complesso, assumendo che con $h = 0.25$ si ottenga una soluzione numerica "sufficientemente" accurata.

```
...
h = 0.25;
...
beta = 0.7586;  alpha = 1 / 5.2;  gamma = 1 / 2.9;

Lambda = @( it, uEA) [ - beta / N * uEA( 3, it ), 0, - beta / N * uEA( 1, it );
                 beta / N * uEA( 3, it ), - alpha,  beta / N * uEA( 1, it );
                 0, alpha, - gamma ];
for it = 1 : length( tEA )
    lambda_h_EA( :, it ) = h * eig( Lambda( it, uEA ) );
end

th = linspace( 0, 2 * pi, 301 );
plot( real( lambda_h_EA ), imag( lambda_h_EA ), 'x', ...
    cos( th ) - 1, sin( th ), '-k' );
xlabel( 'Re( lambda h)' );      ylabel( 'Im( lambda h)' );
grid on
axis( [ -2.5 0.5 -1.5 1.5 ] )
axis equal
```

Osserviamo dalle Figure 27 e 28 che, per $h = 0.25$, esistono alcuni autovalori $\lambda_i(t)$ tali per cui la quantità $(\lambda_i(t)h)$ risiede al di fuori della regione $\mathcal{A}$, almeno per alcuni tempi $t \in [t_0, t_f]$.

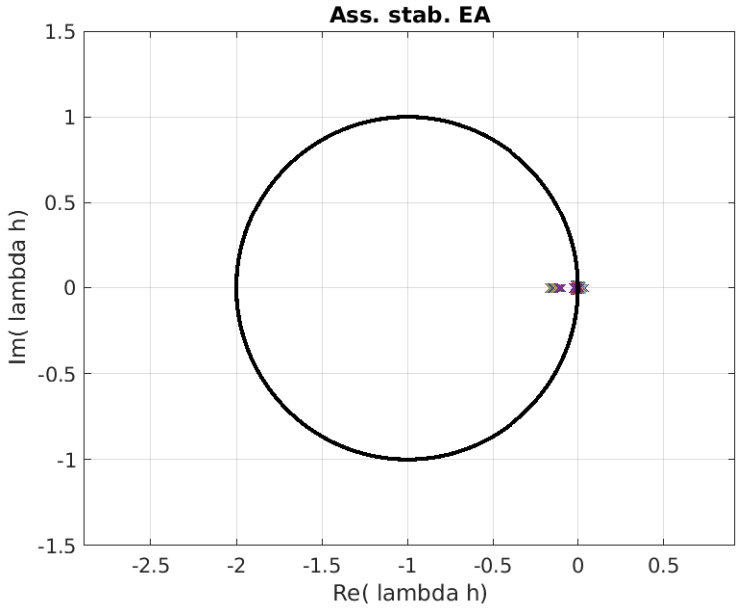


Figura 27: Esercizio 3.3. Regione di assoluta stabilità per il metodo (area inclusa nel cerchio) e valori $(\lambda_i(t)\, h)$ per $i = 1, 2, 3$ e $t \in [t_0, t_f]$ per $h = 0.25$

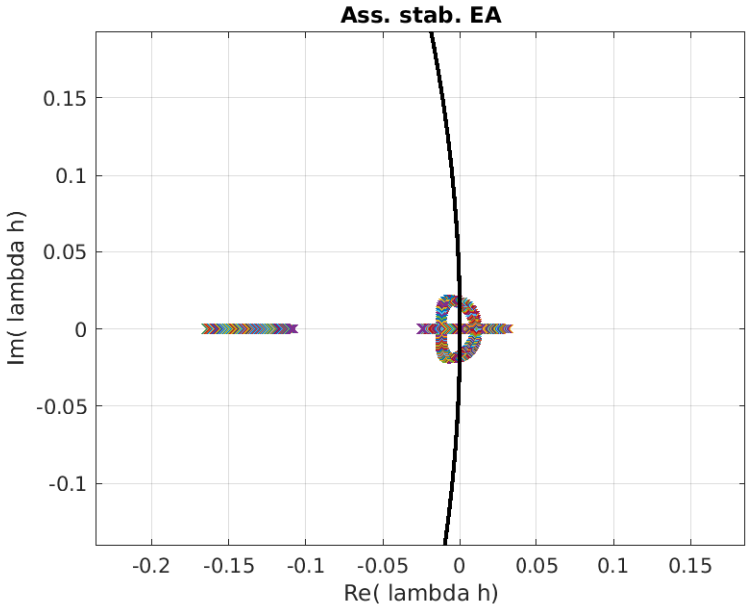


Figura 28: Esercizio 3.3. Regione di assoluta stabilità per il metodo (area inclusa nel cerchio) e valori $(\lambda_i(t)\, h)$ per $i = 1, 2, 3$ e $t \in [t_0, t_f]$ per $h = 0.25$ (dettaglio)

A prescindere dalle proprietà del metodo, è dunque interessante analizzare il comportamento degli autovalori $\lambda_i(t)$ per $i = 1, 2, 3$ al variare di $t \in [t_0, t_f]$, usando per esempio i seguenti comandi Matlab® (sempre assumendo che la soluzione numerica per $h = 0.25$ sia "sufficientemente" accurata da poter essere assunta come soluzione analitica per questa analisi).

```
subplot( 2, 1, 1 )
plot(tEA, real(lambda_h_EA(1, :))/h, ...
     tEA, real(lambda_h_EA(2, :))/h, ...
     tEA, real(lambda_h_EA(3, :))/h, 'LineWidth', 2);
grid on
xlabel( 't [giorni]' );
ylabel( 'Re( lambda )');
legend( 'Re( lambda_1 )', 'Re( lambda_2 )', 'Re( lambda_3 )' );
settings_4_plot_no_LaTeX
subplot( 2, 1, 2 )
plot(tEA, imag(lambda_h_EA(1, :))/h, ...
     tEA, imag(lambda_h_EA(2, :))/h, ...
     tEA, imag(lambda_h_EA(3, :))/h, 'LineWidth', 2);
grid on
xlabel( 't [giorni]' );
ylabel( 'Im( lambda )');
legend( 'Im( lambda_1 )', 'Im( lambda_2 )', 'Im( lambda_3 )' );
```

Deduciamo dalla Figura 29 che alcuni autovalori hanno parte reale positiva per $t <$ 88 giorni (circa), inoltre due autovalori sono complessi per $t \in (75, 116)$ giorni (circa). Infine, tutti gli autovalori sono reali e negativi per $t > 88$ giorni circa. Dunque non è possibile discutere dell'assoluta stabilità del metodo di Eulero in avanti per $t < 88$ giorni (esistendo un autovalore a parte reale positiva e dunque fuori da $\mathcal{A}$ per ogni $h > 0$). Va inoltre verificato che, anche per $t \in (88, 116)$ giorni sia possibile prendere $h > 0$ tale che $(\lambda_i(t)\, h) \in \mathcal{A} = \{z \in \mathbb{C} : |1 + z| < 1\}$ per $i = 1, 2, 3$, dato che in questa fase vi sono autovalori $\lambda_i(t)$ con parte reale negativa, ma con parte immaginaria non nulla. Consideriamo i seguenti comandi Matlab® per trovare il primo tempo $\bar{t} \in [t_0, t_f]$ tale che $(\lambda_i(t)\, h) \in \mathcal{A}$ per ogni $t \in [\bar{t}, t_f]$ a valore di $h > 0$ fissato ($h = 0.25$ in questo caso).

```
lambda_h_EA_region = max( sqrt( ( real( lambda_h_EA ) + 1 ).^2 ...
     + ( imag( lambda_h_EA ) ).^2 ), [], 1 );
ilambda_h = find( lambda_h_EA_region < 1, 1 );
t_bar = tEA( ilambda_h )

th = linspace( 0, 2 * pi, 301 );
plot( real( lambda_h_EA( :, ilambda_h + 1 : end ) ), ...
      imag( lambda_h_EA( :, ilambda_h + 1 : end ) ), 'x', ...
      cos( th ) - 1, sin( th ), '-k' );
xlabel( 'Re( lambda h)' );  ylabel( 'Im( lambda h)' );
grid on;
```

Si verifica che $\bar{t} = 88.25$ giorni; in particolare, abbiamo $(\lambda_i(t)\, h) \in \mathcal{A}$ per ogni $i = 1, 2, 3$ per $t > \bar{t}$ (Figura 30).

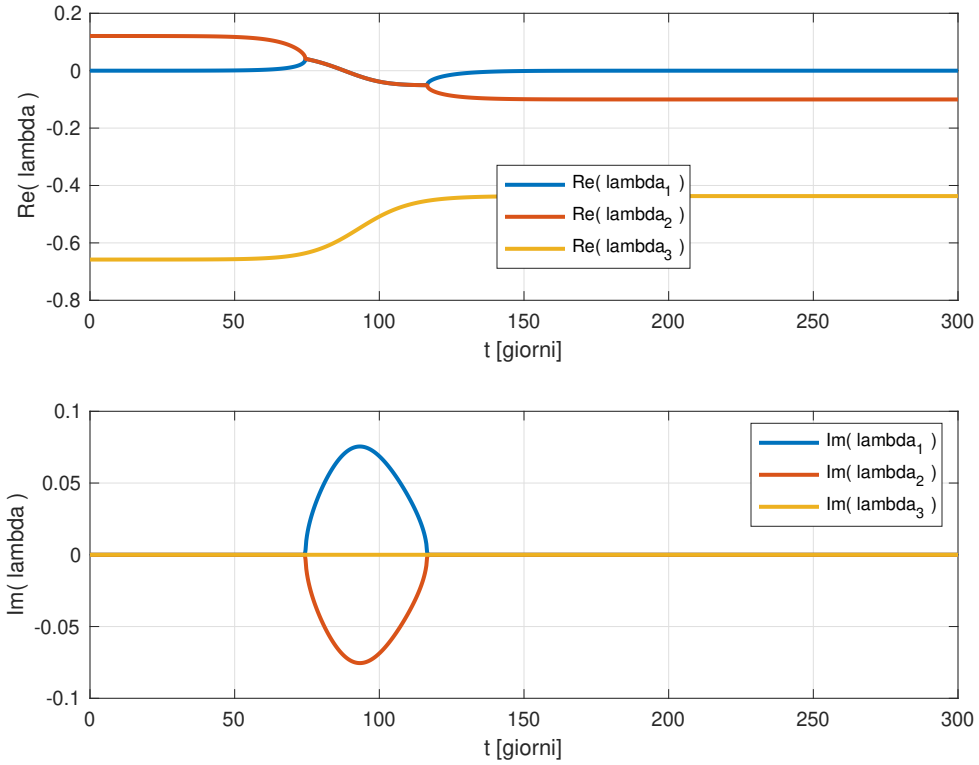


Figura 29: Esercizio 3.3. Evoluzione di $\mathcal{R}e\{\lambda_i(t)\}$ (in alto) e $\mathcal{I}m\{\lambda_i(t)\}$ (in basso) per $i = 1, 2, 3$ vs. t (risultato ottenuto per $h = 0.25$)

Se invece aumentiamo il passo h, e poniamo ad esempio $h = 4$, osserviamo in Figura 31 che la soluzione presenta significative oscillazioni, preludio di instabilità.

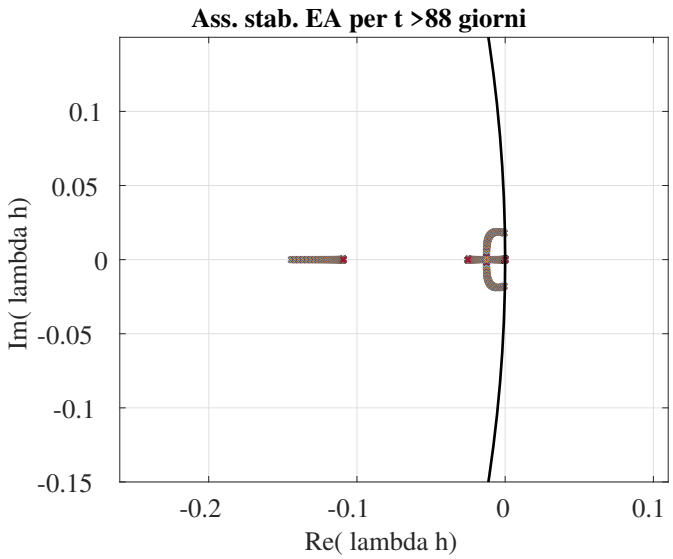


Figura 30: Esercizio 3.3. Regione di assoluta stabilità per il metodo (area inclusa nel cerchio) e valori $(\lambda_i(t)\, h)$ per $i = 1, 2, 3$ e $t \in [\bar{t}, t_f]$ per $h = 0.25$ (dettaglio)

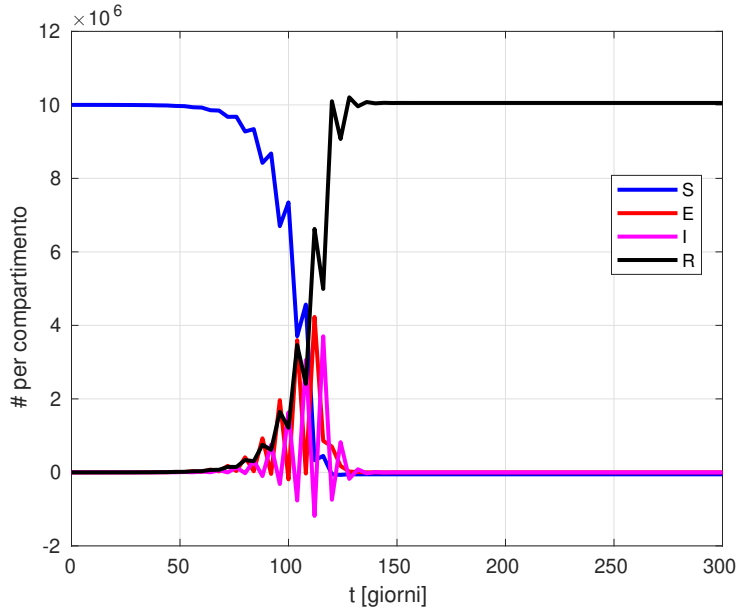


Figura 31: Esercizio 3.3. Evoluzione di $S(t)$, $E(t)$, $I(t)$ e $R(t)$ vs. t con $h = 4.0$; la soluzione presenta oscillazioni numeriche, preludio di instabilità.

Esercizio 4.1

1. *Si riscriva il problema come un sistema di Equazioni Differenziali Ordinarie del primo ordine.*

Data l'Equazione Differenziale del secondo ordine nell'incognita $x(t)$:

$$\begin{cases} x''(t) + 2\ x'(t) + 10\ x^2(t) = z(t) & t \in (0, 10), \\ x'(0) = 2, \\ x(0) = 0, \end{cases} \tag{3}$$

possiamo riscriverla come un sistema di Equazioni Differenziali Ordinarie del primo ordine introducendo il vettore soluzione $\mathbf{y}(t) \ : \ [0, t_f] \to \mathbb{R}^2$, con $t_f = 10$ e tale che $\mathbf{y}(t) = (x(t),\ w(t))^T$ con $w(t) = x'(t)$ per $t \in (0, t_f)$. Da cui si ricava che $w'(t) = x''(t)$. Sostituendo nell'Equazione (3) si ottiene

$$\begin{cases} x'(t) = w(t) & t \in (0, t_f), \\ w'(t) = -2\ w(t) - 10\ x^2(t) + z(t) & t \in (0, t_f), \\ x(0) = 0, \\ w(0) = 2. \end{cases}$$

Tale sistema può essere scritto nella forma compatta (4) definendo:

- $\dfrac{d\mathbf{y}}{dt}(t) = (x'(t), w'(t))^T$;
- $\mathbf{f}(\mathbf{y}) = (\ y_2,\ -2\ y_2 - 10\ {y_1}^2\ )^T$ dove $\mathbf{f} \ : \ \mathbb{R}^2 \to \mathbb{R}^2$.
  In particolare, $\mathbf{f}(x, w) = (\ w,\ -2\ w - 10\ x^2\ )^T$;
- $\mathbf{g}(t) = (0,\ z(t))^T$ con $\mathbf{g}(t) \ : \ (0, t_f) \to \mathbb{R}^2$;
- $\mathbf{y}_0 = (0,\ 2)^T$.

Si ottiene quindi:

$$\begin{cases} \dfrac{d\mathbf{y}}{dt}(t) = \mathbf{f}(\mathbf{y}) + \mathbf{g}(t) & t \in (0, t_f), \\ \mathbf{y}(0) = \mathbf{y}_0. \end{cases} \tag{4}$$

2. *Si approssimi il problema* (4) *tramite il metodo di Eulero in avanti usando opportunamente la funzione* Matlab® `eulero_avanti_sistemi.m` *con passo* $h = 10^{-2}$.

   I comandi Matlab® utilizzati sono riportati di seguito.
   Dopo aver definito i dati del problema:

```
tf = 10;

g = @(t) [ 0; exp(-t/2).*( 2*cos(t) - (7/2)*sin(t) ) + 40*exp(-t).*sin(t).^2];
f = @(y) [  y(2); -2*y(2) - 10*y(1)^2 ];

fun = @(t,y) g(t) + f(y);

y0 = [0; 2];
```

   Risolviamo con `eulero_avanti_sistemi.m`:

```
h = 1e-2;
Nh = round(tf / h);
tv = [0, tf];

[t, u] = eulero_avanti_sistemi(fun, tv, y0, Nh);

figure
plot(t,u(1,:),t,u(2,:),'LineWidth',2)
grid on
legend('$x(t)$','$x^{\prime}(t)$','Interpreter','latex','FontSize',14)
xlabel('$t$','Interpreter','latex','FontSize',14)
ylabel('$\mathbf{y}(t)$','Interpreter','latex','FontSize',14)
```

E plottiamo la soluzione numerica ottenuta, che riportiamo in Figura 32.

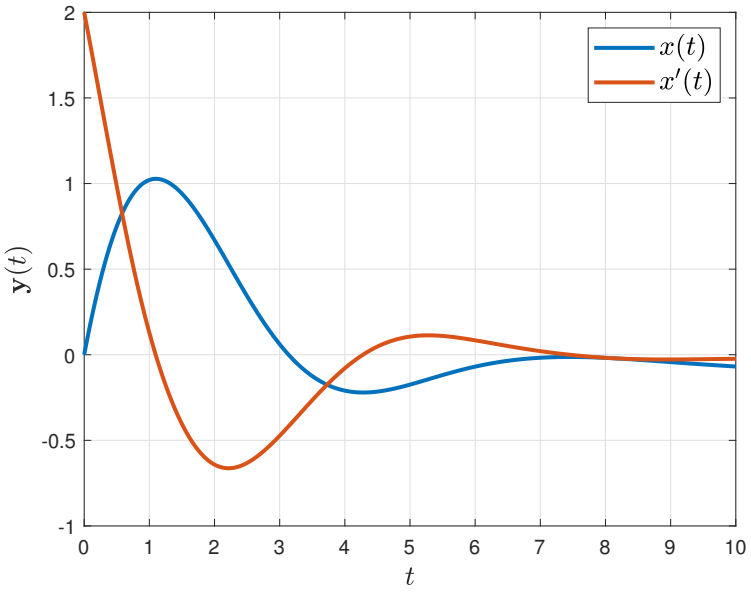


Figura 32: Esercizio 4.1. Evoluzione di $x(t)$ e $x'(t)$ vs. t ottenute con il metodo di Eulero in avanti.

- *Si valutino i valori delle approssimazioni di $x(t_1)$ e $x(t_f)$.*
  I valori approssimati di $x(t_1)$ e $x(t_f)$ sono:

```
u_1 = u(1,2)    % Approssimazione di x(t1) = x_1
u_Nh = u(1,end)    % Approssimazione di x(tf) = x_Nh

u_1 =
    0.0200

u_Nh =
   -0.0688
```

- *Sia u_n l'approssimazione di $x(t_n)$, si valuti il tempo discreto $t_m \geq 0$ tale per cui $|u_n| < 0.1$ per ogni $n \geq m$.*
  Risulta $t_m = 5.6800$, ottenuto tramite i comandi:

```
tm = [];
for k = 2:length(t)
    if( abs(u(1,k-1)) >= 0.1 && abs(u(1,k)) < 0.1 )
        tm = [tm t(k)];
    end
end

t_m = tm(end)

t_m =
    5.6800
```

3. *Dopo aver risposto al Punto 2) e sapendo che la soluzione esatta del problema* (3) *è $x(t) = 2\ e^{-t/2}\ sin(t)$, si calcolino gli errori $E_{h_i} = \max_{n=0,\ldots,N_{h_i}} |u_n - x(t_n)|$ ottenuti con il metodo di Eulero in avanti e corrispondenti ai passi $h_1 = 10^{-3}$, $h_2 = 5 \cdot 10^{-4}$, $h_3 = 2.5 \cdot 10^{-4}$ e $h_4 = 1.25 \cdot 10^{-4}$. Si riportino i valori E_{h_i} per $i = 1, \ldots, 4$.*

   Riportiamo di seguito i comandi Matlab® utilizzati per valutare gli errori ottenuti risolvendo il problema (4) con il metodo di Eulero in avanti per i quattro valori h_i:

$$E_{h_i} = \max_{n=0,\ldots,N_{h_i}} |u_n - x(t_n)| \qquad i = 1, \ldots, 4$$

```
x_ex = @(t) 2 * exp(-t/2) .* sin(t);

h = 1e-3 * [1 1/2 1/4 1/8];
Nh = round(tf ./ h);
Eh = zeros(1,4);

for i = 1:4
    [ti,ui] = eulero_avanti_sistemi( fun, tv, y0, Nh(i));
    Eh(i) = max( abs( ui(1,:) - x_ex(ti) ) );
end
```

   I valori degli errori ottenuti sono i seguenti:

```
Eh =
    0.0046    0.0023    0.0011    0.0006
```

   Coerentemente con quanto ci si aspetta, al diminuire del passo h l'errore del metodo si riduce da $E_{h_1} = 0.0046$ per $h_1 = 10^{-3}$ a $E_{h_4} = 0.0006$ per $h_4 = 1.25 \cdot 10^{-4}$.

4. *Si utilizzino gli errori E_{h_i} ottenuti al Punto 3) per stimare algebricamente l'ordine di convergenza p del metodo di Eulero in avanti. Si riportino i comandi* Matlab® , *si giustifichi la risposta data e la si motivi alla luce della teoria.*

Un metodo numerico per approssimare la soluzione di una Equazione Differenziale Ordinaria ha ordine di convergenza $p \geq 1$ rispetto al passo di discretizzazione h se l'errore

$$E_h = \max_{n=0,1,\ldots,N_h} e_n \leq C\ h^p,$$

con C costante indipendente da h. Per stimare numericamente il valore di p, si considerano gli errori associati a due passi di discretizzazione diversi e se ne prende il rapporto:

$$\frac{E_{h_1}}{E_{h_2}} \simeq \left(\frac{h_1}{h_2}\right)^p \Rightarrow p \simeq \frac{\log\frac{E_{h_1}}{E_{h_2}}}{\log\frac{h_1}{h_2}}$$

Si può dimostrare che l'ordine di convergenza del metodo di Eulero in avanti è pari a 1, vale cioè la stima dell'errore rispetto al passo di discretizzazione h:

$$E_h \simeq C\ h.$$

Nel caso in esame, i valori di h rispetto a cui abbiamo calcolato gli errori sono $h_1 = 10^{-3},\ h_2 = 5 \cdot 10^{-4},\ h_3 = 2.5 \cdot 10^{-4}$ e $h_4 = 1.25 \cdot 10^{-4}$. Gli errori E_{h_i} associati possono essere confrontati a due a due, a partire dagli errori salvati nel vettore `Eh`:

```
p = log(Eh(2:end) ./ Eh(1:end-1)) ./ log(h(2:end) ./ h(1:end-1))
```

Si ottengono gli ordini di convergenza `p` per le tre coppie di errori:

```
p =
    1.0141    1.0069    1.0034
```

Consideriamo come riferimento l'ultimo valore, perché associato ai passi h più piccoli. Concludiamo quindi che $p \simeq 1.0034$, coerentemente con quanto ci si aspetta dal risultato teorico. Infatti, si dimostra che il metodo di Eulero in avanti ha ordine di convergenza pari 1 se la soluzione del problema di Cauchy è una funzione $\mathbf{y} \in C^2([t_0, t_f])$, come si verifica nel caso in considerazione dato che $x \in C^\infty([t_0, +\infty])$.

5. *Con riferimento al sistema di Equazioni Differenziali Ordinarie* (4) *del Punto 1) e sapendo che la soluzione esatta $x(t)$ è data al Punto 3), per quali valori di $h > 0$ il metodo di Eulero in avanti risulta assolutamente stabile?*

   Dato il problema nella forma (4), definiamo il termine di destra del problema di Cauchy come $\mathbf{F}(t, \mathbf{y}(t)) = \mathbf{f}(\mathbf{y}(t)) + \mathbf{g}(t)$.

   Osserviamo che $\lim_{t+\infty} \mathbf{g}(t) = \mathbf{0}$. Inoltre, la soluzione esatta è tale per cui $\lim_{t+\infty} \mathbf{y}(t) = \mathbf{0}$.

   Calcoliamo la matrice Jacobiana $J(t)$ associata al termine $\mathbf{F}(t, \mathbf{y}(t))$ definito sopra:

$$J(t) = \frac{\partial \mathbf{F}}{\partial \mathbf{y}}(t, \mathbf{y}(t)) = \begin{bmatrix} 0 & 1 \\ -20\ y_1(t) & -2 \end{bmatrix}. \tag{5}$$

   Conoscendo la soluzione esatta, possiamo sostituire a $y_1(t)$ l'espressione $x(t) = 2\, e^{-t/2} \sin(t)$, ottenendo quindi:

$$J(t) = \begin{bmatrix} 0 & 1 \\ -40\ e^{-t/2} \sin(t) & -2 \end{bmatrix}$$

Indicati con $\lambda_1(t)$ e $\lambda_2(t)$ gli autovalori di $J(t)$, dalla teoria sappiamo che il metodo di Eulero in avanti è assolutamente stabile se

$$(\lambda_i(t)\ h)\ \in\ \mathcal{A} = \{z \in \mathbb{C} :\ |1+z| < 1\} \quad \forall t\ \in\ [0, t_f], \quad i = 1, 2. \tag{6}$$

con $\mathcal{A}$ regione di assoluta stabilità del metodo di Eulero in avanti.

Vogliamo valutare quale sia il range di valori del passo h che garantiscono che la condizione di assoluta stabilità (6) sia verificata.

Per il metodo di Eulero in avanti, nel caso $\lambda_i \in \mathbb{C}$, se $\mathcal{R}e\{\lambda_i\} < 0$, allora l'approssimazione con il metodo di Eulero in avanti è assolutamente stabile per $0 < h < h_{max}$ con

$$h_{max} = -\frac{2\ \mathcal{R}e\{\lambda_{max}\}}{|\lambda_{max}|^2},$$

dove λ_{max} è l'autovalore di modulo massimo.

A questo scopo, utilizziamo i comandi Matlab® seguenti con cui vengono calcolati gli autovalori della matrice Jacobiana ad ogni istante temporale (con vettore dei tempi `t = linspace(0,tf,Nh+1)`) e valutiamo il valore di $h_{max}(t)$ per ogni $t\ \in\ [0, t_f]$, salvato nel vettore `hk`:

```
J = @(t) [0 1; -20 * x_ex(t) -2];
lambdat = [];
ht = [];

for tk = t
    lambdak = eig(J(tk));
    lambdat = [lambdat, lambdak];
    hmax = - 2 * real(lambdak(2)) / abs(lambdak(2))^2;
    ht = [ht, hmax];
end

hmax = min(ht)


hmax =
    0.0972
```

Il valore che si ottiene è $\overline{h}_{max} = \min_{t\in[0,t_f]} h_{max}(t) = 0.0972$. Quindi l'assoluta stabilità del metodo di Eulero in avanti applicato al problema in esame è garantita per ogni $t \geq 0$ se

$$0 < h < \overline{h}_{max}.$$

6. *Si vuole ora applicare a un sistema di Equazioni Differenziali Ordinarie nella forma* (4) *il seguente metodo multipasso, scritto di seguito per un problema di Cauchy scalare con* $y'(t) = f(t, y(t))$ *per* $t \in (t_0, t_f)$ *e condizione iniziale* $y(t_0) = y_0$*:*

$$\begin{cases} u_{n+1} = u_n + \dfrac{3}{2}\ h\ f(t_n, u_n) - \dfrac{1}{2}\ h\ f(t_{n-1}, u_{n-1}) \quad \textit{per } n = 1, 2, \ldots, N_h - 1, \\ u_1 = u_0 + h\ f(t_0, u_0), \\ u_0 = y_0, \end{cases} \tag{7}$$

Si scriva un'opportuna funzione Matlab® *che implementi il metodo precedente per un sistema di EDO con il generico passo* $h > 0$. *Si utilizzi la funzione precedentemente implementata per risolvere il sistema di Equazioni Differenziali Ordinarie* (4) *di cui al Punto 1) usando il passo* $h = 10^{-2}$. *Si riportino i valori delle approssimazioni di* $x(t_1)$, $x(t_2)$ *e* $x(t_f)$.

Riportiamo di seguito il corpo della funzione `multipasso_sistemi.m` che implementa il metodo (7):

```
function [t,u] = multipasso_sistemi(fun, tv, y0, Nh)
% [t,u] = multipasso_sistemi(fun, tv, y0, Nh)
%
% Metodo multipasso per la soluzione di sistemi di equazioni differenziali
% ordinarie del primo ordine
%
% Parametri di ingresso:
%
% fun               funzione di t, u associata al problema di Cauchy
% tv                estremi di integrazione [t0, tf]
% y0                valore della soluzione al tempo iniziale t0 = 0
% Nh                numero di sotto-intervalli di tv
%
%
% Parametri di uscita:
%
% t                 vettore dei tempi in cui la soluzione viene calcolata
% u                 vettore delle soluzioni calcolate in ogni time-step t


% Calcolo del passo di discretizzazione
h = ( tv(end) - tv(1) ) / Nh;

% Inizializzazione dei parametri di output t e u
t = linspace( tv( 1 ), tv( end ), Nh + 1 );
u = zeros( size( y0, 1 ), Nh + 1 );

% Imposizione delle condizioni iniziali u(t0) e u(t1)
u(:,1) = y0;
u(:,2) = u(:,1) + h * fun( t(1) , u(:,1) );

% Ciclo su n per il calcolo u(t_{n+1}) con t2 <= t_{n+1} < = tf
for n = 2 : Nh
    u(:,n+1) = u(:,n) ...
        + 3/2 * h * fun( t(n) , u(:,n) ) ...
        - 1/2 * h * fun( t(n-1) , u(:,n-1) );
end


end
```

Risolviamo e riportiamo le iterate u_1, u_2 e u_{N_h}:

```
h = 1e-2;
Nh = round(tf / h);
tv = [0,tf];
```

```
[t_mp, u_mp] = multipasso_sistemi(fun, tv, y0, Nh);

figure
plot(t_mp,u_mp(1,:),t_mp,u_mp(2,:),'LineWidth',2)
grid on
legend('$x(t)$','$x^{\prime}(t)$','Interpreter','latex','FontSize',14)
xlabel('$t$','Interpreter','latex','FontSize',14)
ylabel('$\mathbf{y}(t)$','Interpreter','latex','FontSize',14)

u_1_mp = u_mp(1,2)   % Approssimazione di x(t1) = x_1
u_2_mp = u_mp(1,3)   % Approssimazione di x(t2) = x_2
u_Nh = u_mp(1,end) % Approssimazione di x(tf) = x_Nh


u_1_mp =
    0.0200

u_2_mp =
    0.0397

u_Nh_mp =
    -0.0073
```

7. *Dopo aver risposto al Punto 6), si ripetano i Punti 3) e 4) per stimare l'ordine di convergenza p atteso dal metodo. Si riportino, oltre ai comandi* Matlab® *usati, gli errori E_{h_i} e l'ordine p stimato.*

   Di seguito riportiamo i comandi Matlab® che risolvono il problema (4) con il metodo multipasso (7) implementato nella funzione `multipasso_sistemi.m` e calcolano errori e ordine di convergenza:

```
h = 1e-3 * [1 1/2 1/4 1/8];
Nh = round(tf ./ h);
Eh_mp = zeros(1,4);

for i = 1:4
    [ti_mp,ui_mp] = multipasso_sistemi( fun, tv, y0, Nh(i));
    Eh_mp(i) = max( abs( ui_mp(1,:) - x_ex(ti_mp) ) );
end

Eh_mp
p = log(Eh_mp(2:end) ./ Eh_mp(1:end-1)) ./ log(h(2:end) ./ h(1:end-1))
```

   I valori di E_{h_i} e p ottenuti applicando tale metodo sono:

```
Eh_mp =
    1.0e-05*
     0.2020    0.0505    0.0126    0.0032

p =
    1.9996    1.9998    1.9999
```

   Prendendo come riferimento l'ultimo valore di p ottenuto, corrispondente ai valori di h minori, la stima dell'ordine di convergenza è $p = 1.9999 \simeq 2$.

Possiamo concludere che utilizzando il metodo multipasso (7) si ha una convergenza di ordine più alto rispetto al metodo di Eulero in avanti che, al contrario, converge linearmente in h, come verificato al punto 4).