EQUAZIONI DIFFERENZIALI ORDINARIE *

A. SOMMARIVA E M. VIANELLO†

Conoscenze richieste. Formula di Taylor. Risoluzione di equazioni nonlineari. Calcolo differenziale. Conoscenza di Matlab/Octave.

Conoscenze ottenute. Discretizzazione con Eulero esplicito ed Eulero implicito. Stabilità assoluta dei due metodi.

1. Definizione dei metodi di Eulero implicito ed esplicito. Si consideri il problema di Cauchy

$$\begin{cases} y'(x) = f(x, y(x)), \ x \geq x_0 \\ y(x_0) = y_0 \end{cases} \tag{1.1}$$

dove f è a valori in $\mathbb{R}^n$ e definita in un sottoinsieme Ω di $\mathbb{R} \times \mathbb{R}^n$, con $(x_0, y_0) \in \Omega$.

Si supponga che (1.1) abbia una e una sola soluzione. A tal proposito ricordiamo che definite le condizioni (cf. [6, p.327])

$$\exists \delta, \epsilon, , L > 0 : \ [x_0 - \delta, x_0 + \delta] \times \overline{B}_\epsilon(y_0) \subseteq \Omega, \tag{1.2}$$

$$|f(x, z_1) - f(x, z_2)| \leq L|z_1 - z_2|, \ \forall x \in [x_0 - \delta, x_0 + \delta], \forall z_1, z_2 \in \overline{B}_\epsilon(y_0) \tag{1.3}$$

si dimostra il seguente teorema di esistenza ed unicità (in piccolo)

TEOREMA 1.1. *Siano Ω un aperto di $\mathbb{R} \times \mathbb{R}^n$, f una funzione di Ω in $\mathbb{R}^n$ e (x_0, y_0) un punto di Ω. Se f è continua e valgono le condizioni in (2.1) allora esiste un intervallo aperto contenente x_0 nel quale è definita una e una sola soluzione del problema di Cauchy (1.1).*

Accanto a tale teorema esiste quello di esistenza ed unicità (in grande)

TEOREMA 1.2. *Siano Ω un aperto di $\mathbb{R} \times \mathbb{R}^n$, f una funzione di Ω in $\mathbb{R}^n$ e (x_0, y_0) un punto di Ω. Se f è continua e valgono le condizioni in (2.1) allora esiste un intervallo aperto contenente x_0 nel quale è definita una e una sola soluzione del problema di Cauchy (1.1).*

Due classici metodi per risolvere il problema differenziale (1.1) sono il metodo di *Eulero esplicito* ed il metodo di *Eulero implicito*.

Sia $\mathcal{I}(s_1, s_2)$ il più piccolo intervallo aperto contenente s_1 e s_2. Assumendo che la soluzione sia sufficientemente regolare, abbiamo dalla formula di Taylor per $x \approx x_s$, $c_1 \in \mathcal{I}(x, \overline{x})$ e (1.1)

$$y(x) = y(\overline{x}) + y'(\overline{x})(x - \overline{x}) + \frac{y''(c_1)(x - \overline{x})^2}{2} \tag{1.4}$$

$$\approx y(\overline{x}) + y'(\overline{x})(x - \overline{x}) \tag{1.5}$$

$$= y(\overline{x}) + f(\overline{x}, y(\overline{x}))(x - \overline{x}) \tag{1.6}$$

Di conseguenza se si desidera calcolare la soluzione nei punti $x_{k+1} = x_0 + kh = x_k + h$ con $k > 0$, ponendo $x = x_{k+1}$, $\overline{x} = x_k$ abbiamo

$$y(x_{k+1}) \approx y(x_k) + hf(x_k, y(x_k)). \tag{1.7}$$

*Ultima revisione: 4 marzo 2010.

†Dipartimento di Matematica Pura ed Applicata, Universitá degli Studi di Padova, via Trieste 63, 35121 Padova, Italia (alvise@euler.math.unipd.it, marcov@euler.math.unipd.it).

Il metodo di Eulero esplicito consiste nell'approssimare $y(x_{k+1})$ con y_{k+1} definito da

$$y_{k+1} = y_k + hf(x_k, y_k), \tag{1.8}$$

ove $y_0 = y(x_0)$.

Similmente al caso di Eulero esplicito, se poniamo invece $x = x_k$, $\overline{x} = x_{k+1}$ abbiamo

$$y(x_k) \approx y(x_{k+1}) + f(x_{k+1}, y(x_{k+1}))(x_k - x_{k+1}), \tag{1.9}$$

e quindi

$$y(x_{k+1}) \approx y(x_k) + hf(x_{k+1}, y(x_{k+1})). \tag{1.10}$$

Il metodo di Eulero implicito consiste nell'approssimare $y(x_{k+1})$ con y_{k+1} definito da

$$y_{k+1} = y_k + hf(x_{k+1}, y_{k+1}), \tag{1.11}$$

ove $y_0 = y(x_0)$.

Evidentemente ad ogni iterazione si richiede di risolvere un'equazione nonlineare nella variabile z del tipo $z = y_k + hf(x_{k+1}, z)$, la cui soluzione può essere calcolata utilizzando ad esempio col metodo di Newton.

2. I metodi di Eulero in Matlab. Utilizzando l'help di Matlab, si nota subito che esistono delle routines predefinite per risolvere equazioni differenziali ordinarie. Infatti

```
>> help funfun
  Function functions and ODE solvers.

...

  Differential equation solvers.
  Initial value problem solvers for ODEs. (If unsure about stiffness,
  try ODE45 first, then ODE15S.)
    ode45     - Solve non-stiff differential equations, medium order
                method.
    ode23     - Solve non-stiff differential equations, low order
                method.
    ode113    - Solve non-stiff differential equations, variable order
                method.
    ode23t    - Solve moderately stiff ODEs and DAEs Index 1, trape-
                zoidal rule.
    ode15s    - Solve stiff ODEs and DAEs Index 1, variable order
                method.
    ode23s    - Solve stiff differential equations, low order method.
    ode23tb   - Solve stiff differential equations, low order method.

 ...

>>
```

Tra queste non sono incluse le routines dei metodi di Eulero esplicito ed implicito che d'altra parte non sono difficili da programmare. Dei codici in merito si trovano presso il sito

web [4], osservando che *forward Euler* e *backward Euler* corrispondono rispettivamente ai metodi di Eulero esplicito ed implicito. Tale software è gentilmente fornito gratuitamente dagli autori. Per quanto detto `euler_for` corrisponde al metodo di Eulero esplicito mentre `euler_back` al metodo di Eulero implicito.

2.1. Codice di Eulero esplicito. Di seguito scriviamo il codice Matlab che implementa il metodo di Eulero esplicito, offerto da K. Atkinson, W. Han e D. Steward. Quali inputs vengono richiesti il punto iniziale t_0 e il valore y_0 della soluzione, il punto finale t_end, il passo h e la funzione f via la variabile `fcn`.

```
function [t,y] = euler_for(t0,y0,t_end,h,fcn)
%-----------------------------------------------------
% MATLAB CODE TAKEN FROM
% K. Atkinson, W. Han e D. Steward,
% Numerical Solution of Ordinary Differential Equations,
% programs for the book, John Wiley and Sons, 2009,
% http://www.math.uiowa.edu/NumericalAnalysisODE/
%-----------------------------------------------------
% function [t,y]=euler_for(t0,y0,t_end,h,fcn)
%-----------------------------------------------------
% Solve the initial value problem
%    y' = f(t,y),  t0 <= t <= b,  y(t0)=y0
% Use Euler's method with a stepsize of h.  The user must
% supply a program to define the right side function of the
% differential equation. Use some name, say deriv, and a first
% line of the form
%   function ans=deriv(t,y)
% A sample call would be
%   [t,z]=eulercls(t0,z0,b,delta,'deriv')
%-----------------------------------------------------
% Output:
% The routine euler_for will return two vectors, t and y.
% The vector t will contain the node points
%   t(1)=t0, t(j)=t0+(j-1)*h, j=1,2,...,n
% with
%   t(n) <= t_end,  t(n)+h > t_end
% The vector y will contain the estimates of the solution Y
% at the node points in t.
%-----------------------------------------------------

n = fix((t_end-t0)/h)+1;
t = linspace(t0,t0+(n-1)*h,n)';
y = zeros(n,1);
y(1) = y0;
for i = 2:n
  y(i) = y(i-1) + h*feval(fcn,t(i-1),y(i-1));
end

end % euler_for
```

Consideriamo quale test, il problema di Cauchy (cf. [2, p.404])

$$\begin{cases} y'(x) = \lambda y(x) + (1-\lambda)\cos(x) - (1+\lambda)\sin(x), \ x \geq 0 \\ y(0) = 1 \end{cases} \tag{2.1}$$

la cui soluzione è $\overline{y}(x) = \sin(x) + \cos(x)$.

```
>> % lambda = -1, h= 0.5.
>> fcn=inline('2*cos(x)-0*sin(x)-y');
>> t0=1; y0=1; t_end=10; h=0.5;
>> [t,y] = euler_for(t0,y0,t_end,h,fcn);
>> y_sol=sin(t)+cos(t);

>> [t y y_sol abs(y-y_sol) abs(y-y_sol)./abs(y_sol)]

ans =

    1.0000    1.0000    1.3818    0.3818    0.2763
    1.5000    1.0403    1.0682    0.0279    0.0261
    2.0000    0.5909    0.4932    0.0977    0.1982
    2.5000   -0.1207   -0.2027    0.0820    0.4044
    3.0000   -0.8615   -0.8489    0.0126    0.0149
    3.5000   -1.4207   -1.2872    0.1335    0.1037
    4.0000   -1.6468   -1.4104    0.2364    0.1676
    4.5000   -1.4771   -1.1883    0.2887    0.2430
    5.0000   -0.9493   -0.6753    0.2741    0.4059
    5.5000   -0.1910    0.0031    0.1941   62.0331
    6.0000    0.6132    0.6808    0.0676    0.0993
    6.5000    1.2668    1.1917    0.0750    0.0630
    7.0000    1.6100    1.4109    0.1991    0.1411
    7.5000    1.5589    1.2846    0.2742    0.2135
    8.0000    1.1261    0.8439    0.2822    0.3344
    8.5000    0.4175    0.1965    0.2211    1.1251
    9.0000   -0.3932   -0.4990    0.1058    0.2120
    9.5000   -1.1078   -1.0723    0.0354    0.0330
   10.0000   -1.5510   -1.3831    0.1680    0.1214
```

Nelle colonne dei risultati, la prima è l'ascissa in cui viene approssimata la soluzione, la seconda il valore approssimato dal metodo di Eulero esplicito, la terza la soluzione corretta. Infine nelle ultime due colonne vengono forniti gli errori assoluti e relativi compiuti dal metodo di Eulero esplicito.

2.2. Codice di Eulero implicito. Di seguito scriviamo il codice Matlab che implementa il metodo di Eulero esplicito, offerto da K. Atkinson, W. Han e D. Steward. Quali inputs vengono richiesti il punto iniziale t_0 e il valore y_0 della soluzione, il punto finale t_end, il passo h, la funzione f via la variabile `fcn` e `tol` la tolleranza richiesta dal metodo iterativo che risolve ad ogni iterazione l'equazione nonlineare richiesta dal metodo.

```
function [t,y] = euler_back(t0,y0,t_end,h,fcn,tol)

%-----------------------------------------------------
% MATLAB CODE TAKEN FROM
% K. Atkinson, W. Han e D. Steward,
% Numerical Solution of Ordinary Differential Equations,
% programs for the book, John Wiley and Sons, 2009,
% http://www.math.uiowa.edu/NumericalAnalysisODE/
%-----------------------------------------------------
% function [t,y] = euler_back(t0,y0,t_end,h,fcn,tol)
```

```
%-----------------------------------------------------
% Solve the initial value problem
%    y' = f(t,y),  t0 <= t <= b,  y(t0)=y0
% Use the backward Euler method with a stepsize
%  of h.
% The user must supply an m-file to define the
% derivative f, with some name,
% say 'deriv.m', and a first line of the form
%   function ans=deriv(t,y)
% tol is the user supplied bound on the difference
% between successive values of the backward Euler
% iteration.
% A sample call would be
%   [t,z]=euler_back(t0,z0,b,delta,'deriv',1e-6)
%-----------------------------------------------------
% Output:
% The routine euler_back will return two vectors,
% t and y.
% The vector t will contain the node points
%   t(1)=t0, t(j)=t0+(j-1)*h, j=1,2,...,N
% with
%   t(N) <= t_end,  t(N)+h > t_end
% The vector y will contain the estimates of the
% solution Y at the node points in t.
%-----------------------------------------------------

% Initialize.
n = fix((t_end-t0)/h) + 1;
t = linspace(t0,t0+(n-1)*h,n)';
y = zeros(n,1);
y(1) = y0;
% advancing
for i=2:n
% forward Euler estimate
  yt1 = y(i-1) + h*feval(fcn,t(i-1),y(i-1));
% one-point iteration
  count = 0;
  diff = 1;
  while diff > tol && count < 10
    yt2 = y(i-1) + h*feval(fcn,t(i),yt1);
    diff = abs(yt2-yt1);
    yt1 = yt2;
    count = count +1;
  end
  if count >= 10
   disp('Not converging after 10 steps at t = ')
   fprintf('%5.2f\n', t(i))
  end
  y(i) = yt2;
end

end % euler_back
```

Come prima, onsideriamo quale test il problema di Cauchy

$$\begin{cases} y'(x) = \lambda y(x) + (1-\lambda)\cos(x) - (1+\lambda)\sin(x), \ x \geq 0 \\ y(0) = 1 \end{cases} \tag{2.2}$$

la cui soluzione è $\overline{y}(x) = \sin(x) + \cos(x)$.

```
>> fcn=inline('2*cos(x)-0*sin(x)-y');
>> t0=1; y0=1; t_end=10; h=0.5; tol=10^(-2);
>> [t,y] = euler_back(t0,y0,t_end,h,fcn,tol);
>> [t y y_sol abs(y-y_sol) abs(y-y_sol)./abs(y_sol)]

ans =

   1.0000e+00   1.0000e+00   1.3818e+00   3.8177e-01   2.7629e-01
   1.5000e+00   7.1127e-01   1.0682e+00   3.5696e-01   3.3416e-01
   2.0000e+00   1.9496e-01   4.9315e-01   2.9819e-01   6.0467e-01
   2.5000e+00  -4.0679e-01  -2.0267e-01   2.0412e-01   1.0072e+00
   3.0000e+00  -9.2890e-01  -8.4887e-01   8.0027e-02   9.4274e-02
   3.5000e+00  -1.2469e+00  -1.2872e+00   4.0374e-02   3.1365e-02
   4.0000e+00  -1.2647e+00  -1.4104e+00   1.4573e-01   1.0332e-01
   4.5000e+00  -9.8131e-01  -1.1883e+00   2.0701e-01   1.7420e-01
   5.0000e+00  -4.6325e-01  -6.7526e-01   2.1201e-01   3.1396e-01
   5.5000e+00   1.6187e-01   3.1294e-03   1.5874e-01   5.0724e+01
   6.0000e+00   7.5062e-01   6.8075e-01   6.9869e-02   1.0263e-01
   6.5000e+00   1.1543e+00   1.1917e+00   3.7359e-02   3.1349e-02
   7.0000e+00   1.2700e+00   1.4109e+00   1.4092e-01   9.9881e-02
   7.5000e+00   1.0753e+00   1.2846e+00   2.0933e-01   1.6295e-01
   8.0000e+00   6.1780e-01   8.4386e-01   2.2605e-01   2.6788e-01
   8.5000e+00   1.2917e-02   1.9648e-01   1.8356e-01   9.3426e-01
   9.0000e+00  -6.0044e-01  -4.9901e-01   1.0143e-01   2.0325e-01
   9.5000e+00  -1.0674e+00  -1.0723e+00   4.9650e-03   4.6302e-03
   1.0000e+01  -1.2689e+00  -1.3831e+00   1.1417e-01   8.2547e-02

>>
```

Nelle colonne dei risultati, la prima è l'ascissa in cui viene approssimata la soluzione, la seconda il valore approssimato dal metodo di Eulero implicito, la terza la soluzione corretta. Infine nelle ultime due colonne vengono forniti gli errori assoluti e relativi compiuti dal metodo di Eulero implicito.

Una variante che utilizza il metodo delle secanti per risolvere l'equazione nonlineare (e non il metodo di punto fisso come in `euler_back`) è il seguente codice

```
function [t,y] = euler_back_secants(t0,y0,t_end,h,fcn,maxit,tol)

%-----------------------------------------------------
% MATLAB CODE TAKEN FROM
% K. Atkinson, W. Han e D. Steward,
% Numerical Solution of Ordinary Differential Equations,
% programs for the book, John Wiley and Sons, 2009,
% http://www.math.uiowa.edu/NumericalAnalysisODE/
%-----------------------------------------------------
```

```
% Solve the initial value problem
%    y' = f(t,y),  t0 <= t <= b,  y(t0)=y0
% Use the backward Euler method with a stepsize of h.  The user must
% supply an m-file to define the derivative f, with some name,
% say 'deriv.m', and a first line of the form
%   function ans=deriv(t,y)
% tol is the user supplied bound on the difference between successive
% values of the backward Euler iteration.
% Maxit is the maximum number of iterations performed
% by the secants method.
% A sample call would be
%   [t,z]=euler_back_secants(t0,z0,b,delta,'deriv',maxit,1e-6)
%-------------------------------------------------------
% Output:
% The routine euler_back will return two vectors, t and y.
% The vector t will contain the node points
%   t(1)=t0, t(j)=t0+(j-1)*h, j=1,2,...,N
% with
%   t(N) <= t_end,  t(N)+h > t_end
% The vector y will contain the estimates of the solution Y
% at the node points in t.
%-------------------------------------------------------

% Initialize.
n = fix((t_end-t0)/h) + 1;
t = linspace(t0,t0+(n-1)*h,n)';
y = zeros(n,1);
y(1) = y0;
% advancing
for i=2:n
    % forward Euler estimate
    yt1 = y(i-1) + h*feval(fcn,t(i-1),y(i-1));

    % fixed point iteration
    yt2 = y(i-1) + h*feval(fcn,t(i),yt1);

    % secant iteration
    k=1;
    yy(1)=min(yt1,yt2); yy(2)=max(yt1,yt2);
    step_err(1)=yy(2)-yy(1);

    while (abs(step_err(k)) > tol) & (k < maxit)

        k=k+1;

        fx_old=yy(k-1)-y(i-1)-h*feval(fcn,t(i),yy(k-1));
        fx_new=yy(k)-y(i-1)-h*feval(fcn,t(i),yy(k));
        dfx=(fx_new-fx_old)/(yy(k)-yy(k-1));

        step_err_k=-fx_new/dfx;
        step_err=[step_err; step_err_k];

        yy(k+1)=yy(k)+step_err(end);
```

```
    end

    step_err=[];

    if k == maxit
    fprintf('\n \t Results may be not correct.');
    end

    y(i) = yy(end);
end
```

Applicando tale implementazione di Eulero implicito abbiamo

```
>> fcn=inline('2*cos(x)+0*sin(x)-y');
>> t0=1; y0=1; t_end=10; h=0.5; maxit=100; tol=10^(-4);
>> [t,y] = euler_backn(t0,y0,t_end,h,fcn,maxit,tol);
>> y_sol=sin(t)+cos(t);
>> [t y y_sol abs(y-y_sol) abs(y-y_sol)./abs(y_sol)]

ans =

   1.0000e+00   1.0000e+00   1.3818e+00   3.8177e-01   2.7629e-01
   1.5000e+00   7.1382e-01   1.0682e+00   3.5441e-01   3.3177e-01
   2.0000e+00   1.9845e-01   4.9315e-01   2.9470e-01   5.9758e-01
   2.5000e+00  -4.0179e-01  -2.0267e-01   1.9912e-01   9.8249e-01
   3.0000e+00  -9.2786e-01  -8.4887e-01   7.8985e-02   9.3047e-02
   3.5000e+00  -1.2429e+00  -1.2872e+00   4.4363e-02   3.4464e-02
   4.0000e+00  -1.2643e+00  -1.4104e+00   1.4610e-01   1.0358e-01
   4.5000e+00  -9.8343e-01  -1.1883e+00   2.0490e-01   1.7243e-01
   5.0000e+00  -4.6651e-01  -6.7526e-01   2.0875e-01   3.0914e-01
   5.5000e+00   1.6144e-01   3.1294e-03   1.5831e-01   5.0587e+01
   6.0000e+00   7.4774e-01   6.8075e-01   6.6985e-02   9.8398e-02
   6.5000e+00   1.1496e+00   1.1917e+00   4.2156e-02   3.5374e-02
   7.0000e+00   1.2690e+00   1.4109e+00   1.4192e-01   1.0059e-01
   7.5000e+00   1.0771e+00   1.2846e+00   2.0757e-01   1.6158e-01
   8.0000e+00   6.2105e-01   8.4386e-01   2.2281e-01   2.6404e-01
   8.5000e+00   1.2690e-02   1.9648e-01   1.8379e-01   9.3541e-01
   9.0000e+00  -5.9896e-01  -4.9901e-01   9.9949e-02   2.0029e-01
   9.5000e+00  -1.0641e+00  -1.0723e+00   8.2349e-03   7.6795e-03
   1.0000e+01  -1.2688e+00  -1.3831e+00   1.1432e-01   8.2655e-02

>>
```

3. Assoluta stabilitá. Nell'ambito delle equazioni differenziali ordinarie, esistono vari criteri di stabilitá. Un classico problema è quello di vedere se un metodo è *assolutamente stabile*. Definito il problema di Cauchy

$$\begin{cases} y'(x) = \lambda y(x), \ x \geq 0 \\ y(0) = 1 \end{cases} \tag{3.1}$$

per un certo $\lambda \in \mathbb{C}$ con $\Re(\lambda) < 0$, visto che l'unica soluzione è $y(x) = \exp(\lambda x)$ si cerca di definire il passo h cosicchè il metodo numerico abbia lo stesso comportamento asintotico di

$\exp(\lambda x)$. Ricordiamo a tal proposito che dalla formula di Eulero, se $z = a + ib$ allora

$$\exp(z) = \exp(a) \cdot (\cos(b) + i \sin(b)).$$

Quindi se $\Re(\lambda) < 0$, visto che $|\cos(\Im(\lambda x)) + i \sin(\Im(\lambda x))| = 1$ e $x > 0$ abbiamo

$$|\exp(\lambda x)| = |\Re(\lambda x)||\cos(\Im(\lambda x)) + i \sin((\Im(\lambda x)))| = |\Re(\lambda x)| \to 0, \text{ per } x \to \infty.$$

Visto il comportamento asintotico di $\exp(\lambda x)$, se $y_h(x_n)$ è l'approssimazione della soluzione in x_n fornita da un metodo numerico a passo h, si desidera sia $y_h(x_n) \to 0$ per $n \to +\infty$. Nel caso del metodo di Eulero esplicito, si dimostra che ciò accade se $|1 + h\lambda| < 1$ (cf. [2, p.395]), mentre per il metodo di Eulero implicito ciò si realizza se $|1 - h\lambda| > 1$ cioè qualsiasi sia h indipendentemente da λ con $\Re(\lambda) < 0$.

4. Esercizio.

1. Approssimare per $\lambda = -100$ il valore assunto dalla soluzione $\overline{y}$ del problema di Cauchy

$$\begin{cases} y'(x) = \lambda y(x), \ x \geq 0 \\ y(0) = 1 \end{cases} \tag{4.1}$$

   nel punto $x = 0.2$. A tal proposito si utilizzano i metodi di Eulero esplicito e Eulero implicito (nella versione con il metodo delle secanti) con passi $h = 0.1$, $h = 0.05$, $h = 0.02$, $h = 0.01$, $h = 0.001$. Al variare di h verificare quando $|1 + h\lambda| < 1$. Se eseguiti correttamente dovrebbero fornire quali risultati (cf. [2, p.396])

```
1.0000e-01   8.1000e+01   8.2645e-03
5.0000e-02   2.5600e+02   7.7160e-04
2.0000e-02   1.0000e+00   1.6933e-05
1.0000e-02   2.0612e-09   9.5161e-07
1.0000e-03   1.3556e-09   3.2046e-09
```

   in cui alla prima colonna si stabilisce il passo h, nella seconda si espongono i risultati ottenuti dal metodo di Eulero esplicito e nella terza quelli di Eulero implicito (nella versione con il metodo delle secanti).

2. Approssimare per $\lambda = -500$ del problema di Cauchy

$$\begin{cases} y'(x) = \lambda y(x), \ x \geq 0 \\ y(0) = 1 \end{cases} \tag{4.2}$$

   il valore assunto dalla soluzione $\overline{y}$ del problema di Cauchy nel punto $x = 0.2$. A tal proposito si utilizzano i metodi di Eulero esplicito e Eulero implicito (nella versione con il metodo delle secanti) con passi $h = 0.1$, $h = 0.05$, $h = 0.02$, $h = 0.01$, $h = 0.001$, $h = 0.0005$. Al variare di h verificare quando $|1 + h\lambda| < 1$. Il metodo di Eulero implicito per h piccoli può richiedere molto tempo di calcolo.
3. (Facoltativo) Aiutandosi con la routine di Eulero implicito (con il metodo delle secanti quale solutore nonlineare), scrivere una routine che esegua il *metodo di Crank-Nicolson* anche detto *metodo dei trapezi*

$$y_{n+1} = y_n + \frac{h}{2}\left(f(x_n, y_n) + f(x_{n+1}, y_{n+1})\right), \ n \geq 0, \ y_0 \text{ fissato.}$$

   Si eseguano quindi gli esercizi precedenti con tale metodo (e non con Eulero esplicito/implicito).

RIFERIMENTI BIBLIOGRAFICI

[1] K. Atkinson, *Introduction to Numerical Analysis*, Wiley, 1989.

[2] K. Atkinson and W. Han, *Elementary Numerical Analysis*, John Wiley and Sons, 2001.

[3] K. Atkinson and W. Han, *Theoretical Numerical Analysis*, Springer Verlag, 2001.

[4] K. Atkinson, W. Han e D. Steward, *Numerical Solution of Ordinary Differential Equations, programs for the book*, John Wiley and Sons, 2009, http://www.math.uiowa.edu/NumericalAnalysisODE/.

[5] G. Gilardi, *Analisi Due*, seconda edizione, Mc Graw Hill, 1996.