LABORATORIO 2

Esercizio 1

Esercizio 1.1 : Implementazione metodi di EA,EI,CN per calore

Il problema a cui ci riferiamo è:

$$\begin{cases} u_t - \Delta u = f & x \in (-L, L) \quad t \in (0, T] \\ u(x, 0) = g(x) & t \in (0, T] \\ u(-L, t) = u_a(t) & x \in (-L, L) \\ u(L, t) = u_b(t) & x \in (-L, L) \end{cases}$$

Si parte con la discretizzazione in spazio su una griglia $x_i = -L + h\,i$ con $i = 0, \dots, N$ dove N è il numero di sottointervalli spaziali.

```
fprintf([ ...
'h = 2*L/N;\n', ...
'x = linspace(-L,L,N+1)'';\n']);
```

```
h = 2*L/N;
x = linspace(-L,L,N+1)';
```

Il problema semi discreto è quindi (nel caso di condizione al bordo di Dirichlet) quello di trovare N-1 funzioni che soddisfino

$$u_{xx}(x_i, t) = \frac{1}{h^2}(u(x_{i+1}, t) - 2u(x_i, t) + u(x_{i-1}, t)) + O(h^2).$$

$$\begin{cases} \dot{u_i}(t) - \frac{1}{h^2}\left(u_{i-1}(t) - 2u_i(t) + u_{i+1}(t)\right) = f(x_i, t), \quad i = 1, \dots, N-1, \\ u_0(t) = 0, \\ u_i(0) = u_0(x_i), \quad i = 1, N+1. \end{cases}$$

Che possiamo rappresentare in forma matriciale usando

$$A_h = \frac{1}{h^2}\begin{bmatrix} 2 & -1 & 0 & \cdots & 0 \\ -1 & 2 & -1 & \ddots & \vdots \\ 0 & -1 & 2 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & -1 \\ 0 & \cdots & 0 & -1 & 2 \end{bmatrix}, \quad F(t) = \begin{bmatrix} f(x_1, t) \\ \vdots \\ f(x_{N-1}, t) \end{bmatrix}.$$

$$\dot{U}(t) + A_h U(t) = F(t), \qquad U(0) = \{ u_i(0) \}_{i=1}^{N}.$$

```
fprintf([ ...
'%% Creo la matrice A_h\n', ...
'A = spdiags([-1 2 -1], [-1 0 1], N-1, N-1);\n', ...
'A = (1/h^2)*A;\n']);
```

```
% Creo la matrice A_h
A = spdiags([-1 2 -1], [-1 0 1], N-1, N-1);
A = (1/h^2)*A;
```

A questo punto si procede con la discretizzazione temporale tramite i passi temporali $t_n = \tau\, n \quad n = 0, \dots, K$

```
fprintf([ ...
'tau = T/K;\n', ...
't = linspace(0, T, K+1)'';\n\n', ...
'%% Inizializzazione della matrice soluzione U\n', ...
'U = zeros(N+1, K+1);\n\n', ...
'%% Condizione iniziale\n', ...
'U(:, 1) = u0(x);\n\n', ...
'%% Matrice identità sparsa\n', ...
'I = speye(N-1, N-1);\n\n', ...
'%% Condizioni al bordo\n', ...
'U(1, :)   = ua(t);\n', ...
'U(end, :) = ub(t);\n']);
```

```
tau = T/K;
t = linspace(0, T, K+1)';

% Inizializzazione della matrice soluzione U
U = zeros(N+1, K+1);

% Condizione iniziale
U(:, 1) = u0(x);

% Matrice identità sparsa
I = speye(N-1, N-1);

% Condizioni al bordo
U(1, :)   = ua(t);
U(end, :) = ub(t);
```

Uso una discretizzazione in avanti della derivata prima in t_n (**EA)** :

$$\partial_t U(t_n) \approx \frac{U(t_{n+1}) - U(t_n)}{\tau}$$

Applicandola al problema semi discreto otteniamo il seguente schema

$$\frac{U^{n+1} - U^n}{\tau} + A\,U^n = F\left(t_n\right).$$

Riarrangiando questa espressione e procedendo analogamente per gli altri schemi temporali otteniamo

- EA : $u^{k+1} = (I - \Delta t\, A)\, u^k + \Delta t\, f^k,$
- EI : $(I + \Delta t\, A)\, u^{k+1} = u^k + \Delta t\, f^{k+1}$
- CN : $\left(I + \frac{\Delta t}{2} A\right) u^{k+1} = \left(I - \frac{\Delta t}{2} A\right) u^k + \Delta t\, (f^k + f^{k+1})$

Implemento quindi il codice per **EA**

```
fprintf("\n ------------- EA ----------------")
```

```
 ------------ EA ---------------
```

```
fprintf([ ...
'%% Ciclo iterativo\n', ...
'for k = 1:K\n', ...
'    %% Assemblaggio termine noto al t_{k}\n', ...
'    F = f(x(2:end-1), t(k));\n\n', ...
'    %% Correzione del termine noto con le condizioni al bordo\n', ...
'    F(1)   = F(1)   + ua(t(k))*(1/h^2);\n', ...
'    F(end) = F(end) + ub(t(k))*(1/h^2);\n\n', ...
'    %% Calcolo della soluzione al tempo t_{k+1}\n', ...
'    U(2:end-1, k+1) = tau*F + (I - tau*A)*U(2:end-1, k);\n', ...
'end\n']);
```

```
% Ciclo iterativo
for k = 1:K
    % Assemblaggio termine noto al t_{k}
    F = f(x(2:end-1), t(k));

    % Correzione del termine noto con le condizioni al bordo
    F(1)   = F(1)   + ua(t(k))*(1/h^2);
    F(end) = F(end) + ub(t(k))*(1/h^2);

    % Calcolo della soluzione al tempo t_{k+1}
    U(2:end-1, k+1) = tau*F + (I - tau*A)*U(2:end-1, k);
end
```

```
```

Implemento il codice per **EI**

```
fprintf("\n------------------- EI -------------")
```

```
------------------ EI -------------
```

```
fprintf([ ...
'%% Ciclo iterativo\n', ...
'for k = 1:K\n', ...
'    %% Assemblaggio termine noto al tempo t_{k+1}\n', ...
'    F = f(x(2:end-1), t(k+1));\n\n', ...
'    %% Correzione del termine noto con le condizioni al bordo\n', ...
'    F(1)   = F(1)   + ua(t(k+1))*(1/h^2);\n', ...
'    F(end) = F(end) + ub(t(k+1))*(1/h^2);\n\n', ...
'    %% Calcolo della soluzione al tempo t_{k+1}\n', ...
'    U(2:end-1, k+1) = (I + tau*A)\\(tau*F + U(2:end-1, k));\n', ...
'end\n']);
```

```
% Ciclo iterativo
for k = 1:K
    % Assemblaggio termine noto al tempo t_{k+1}
    F = f(x(2:end-1), t(k+1));

    % Correzione del termine noto con le condizioni al bordo
    F(1)   = F(1)   + ua(t(k+1))*(1/h^2);
```

```
    F(end) = F(end) + ub(t(k+1))*(1/h^2);

    % Calcolo della soluzione al tempo t_{k+1}
    U(2:end-1, k+1) = (I + tau*A)\(tau*F + U(2:end-1, k));
end
```

Implemento il codice per **CN**

```
fprintf("\n--------------- CN ---------- \n")
```

```
--------------- CN ----------
```

```
fprintf([ ...
'%% Ciclo iterativo\n', ...
'for k = 1:K\n', ...
'    %% Assemblaggio termine noto ai tempi t_{k} e t_{k+1}\n', ...
'    FA = f(x(2:end-1), t(k));\n', ...
'    FI = f(x(2:end-1), t(k+1));\n\n', ...
'    %% Correzione del termine noto con le condizioni al bordo\n', ...
'    FA(1)   = FA(1)   + ua(t(k))*(1/h^2);\n', ...
'    FA(end) = FA(end) + ub(t(k))*(1/h^2);\n\n', ...
'    FI(1)   = FI(1)   + ua(t(k+1))*(1/h^2);\n', ...
'    FI(end) = FI(end) + ub(t(k+1))*(1/h^2);\n\n', ...
'    %% Calcolo della soluzione al tempo t_{k+1}\n', ...
'    U(2:end-1, k+1) = (I + 0.5*tau*A)\\(0.5*tau*FA + 0.5*tau*FI + (I -
0.5*tau*A)*U(2:end-1, k));\n', ...
'end\n']);
```

```
% Ciclo iterativo
for k = 1:K
    % Assemblaggio termine noto ai tempi t_{k} e t_{k+1}
    FA = f(x(2:end-1), t(k));
    FI = f(x(2:end-1), t(k+1));

    % Correzione del termine noto con le condizioni al bordo
    FA(1)   = FA(1)   + ua(t(k))*(1/h^2);
    FA(end) = FA(end) + ub(t(k))*(1/h^2);

    FI(1)   = FI(1)   + ua(t(k+1))*(1/h^2);
    FI(end) = FI(end) + ub(t(k+1))*(1/h^2);

    % Calcolo della soluzione al tempo t_{k+1}
    U(2:end-1, k+1) = (I + 0.5*tau*A)\(0.5*tau*FA + 0.5*tau*FI + (I - 0.5*tau*A)*U(2:end-1, k));
end
```

Esercizio 2: caso N=35 e K= 50 e K= 200

Implementiamo il codice e poi commentiamo

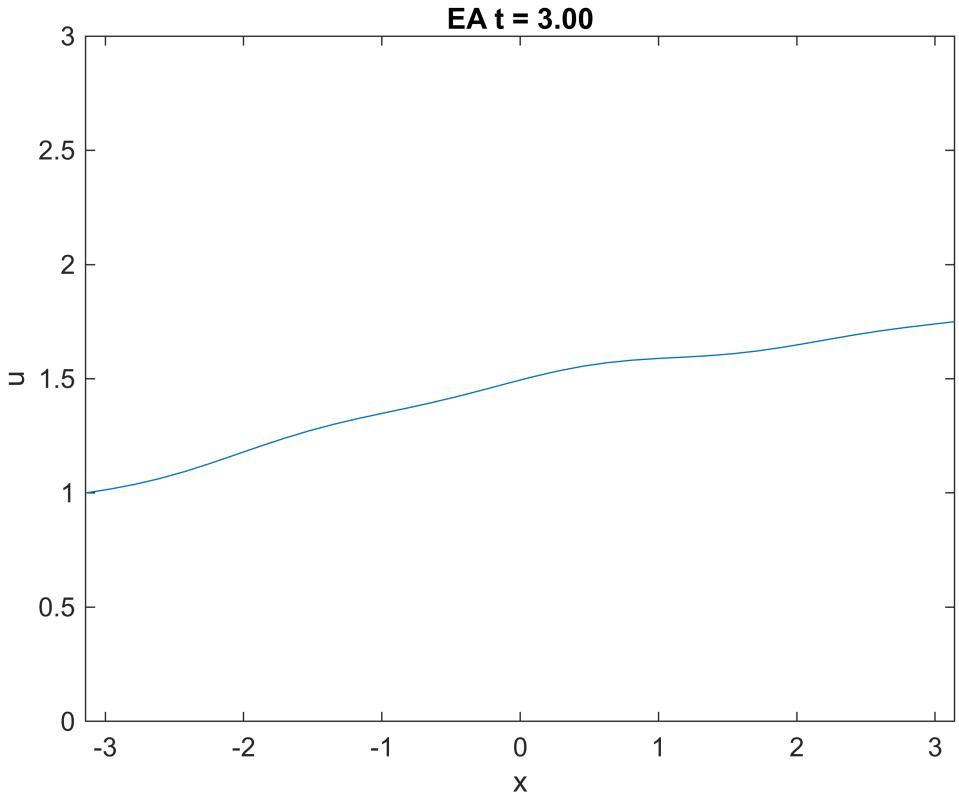
```
% Definisco i dati
T = 3; L = pi;
f = @(x,t) (2*sin(3*x)-1).*exp(-t) + (x+pi)./(2*pi*(1+t).^2);
u0 = @(x) 2 + cos(x) + 0.25*sin(3*x);
ua = @(t) 1;
```

```
ub = @(t) 1 + t./(1+t);

uex = @(x,t) (1+cos(x)+0.25*sin(3*x)).*exp(-t) + 1 + (t./(1+t)).*(x+pi)/(2*pi) ;

% Dati per la discretizzazione
N = 35; K =50;

% Simulo
[x,t,uEA] = calore_EA(L,N,T,K,ua,ub,f,u0);
[x,t,uEI] = calore_EI(L,N,T,K,ua,ub,f,u0);
[x,t,uCN] = calore_CN(L,N,T,K,ua,ub,f,u0);
```

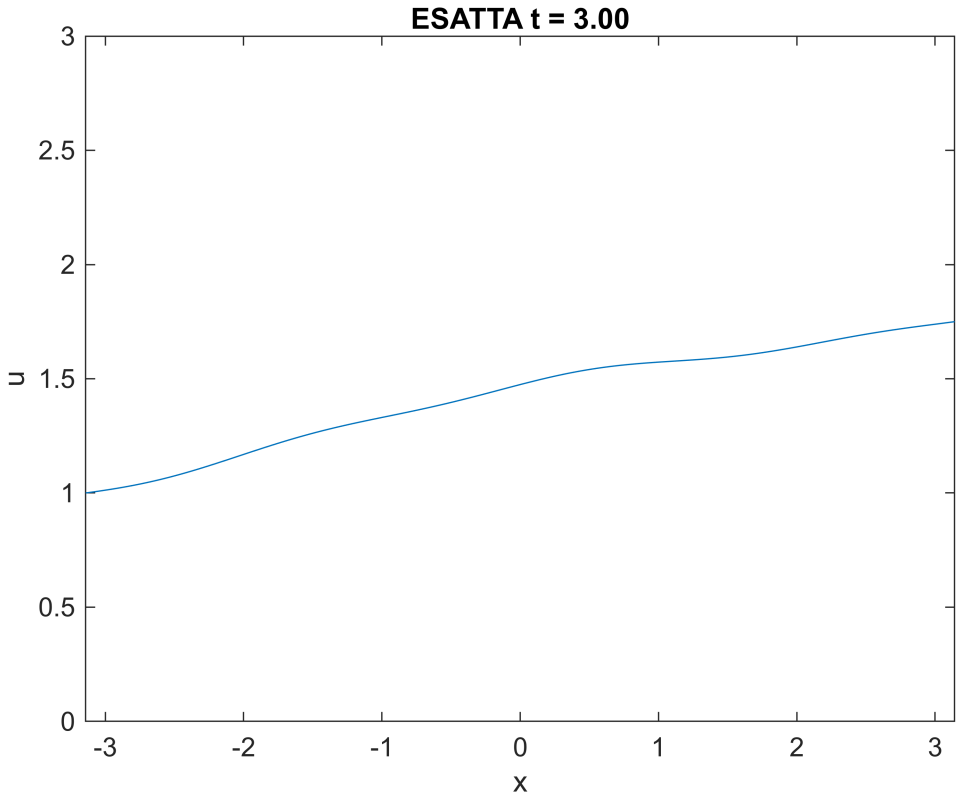
Vediamo ora i diversi modi di plottare

PLOT DINAMICO

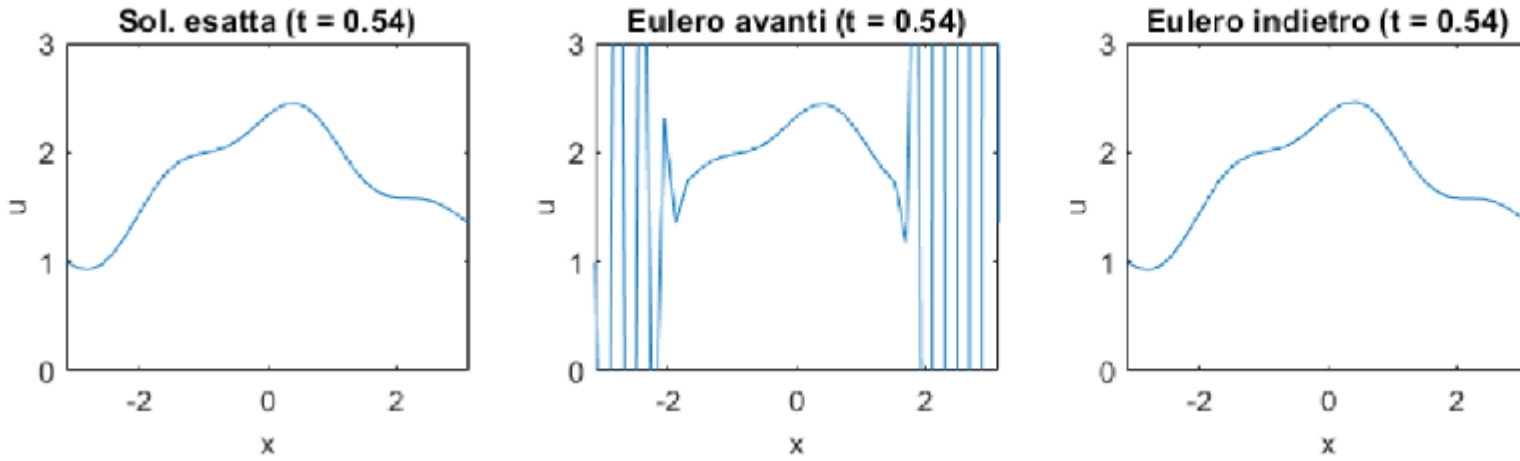
```
% --- Animazione (plot dinamico) ---
for k = 1:K+1
    % Soluzione al tempo t_k
    plot(x, uEI(:, k));

    % Titolo interattivo + fix grafici
    title(sprintf('EA t = %.2f', t(k)));
    xlabel('x');
    ylabel('u');
    axis([-L, L, 0, 3]);

    pause(1.0/24.0);
end
```




```
space = linspace(-L, L, 200);
time = linspace(0, T, 100);
[xx, tt] = ndgrid(space, time);
uexsim = uex(xx,tt);
% --- Animazione (plot dinamico) ---
for k = 1:length(time)
    plot(space, uexsim(:, k));

    title(sprintf('ESATTA t = %.2f', time(k)));
    xlabel('x');
    ylabel('u');
    axis([-L, L, 0, 3]);

    pause(1.0/24.0);
end
```



In figura che segue si vede una istantea nel caso K = 50:



In questo caso EA è instabile in quando non è richiesta la condizione di stabilità

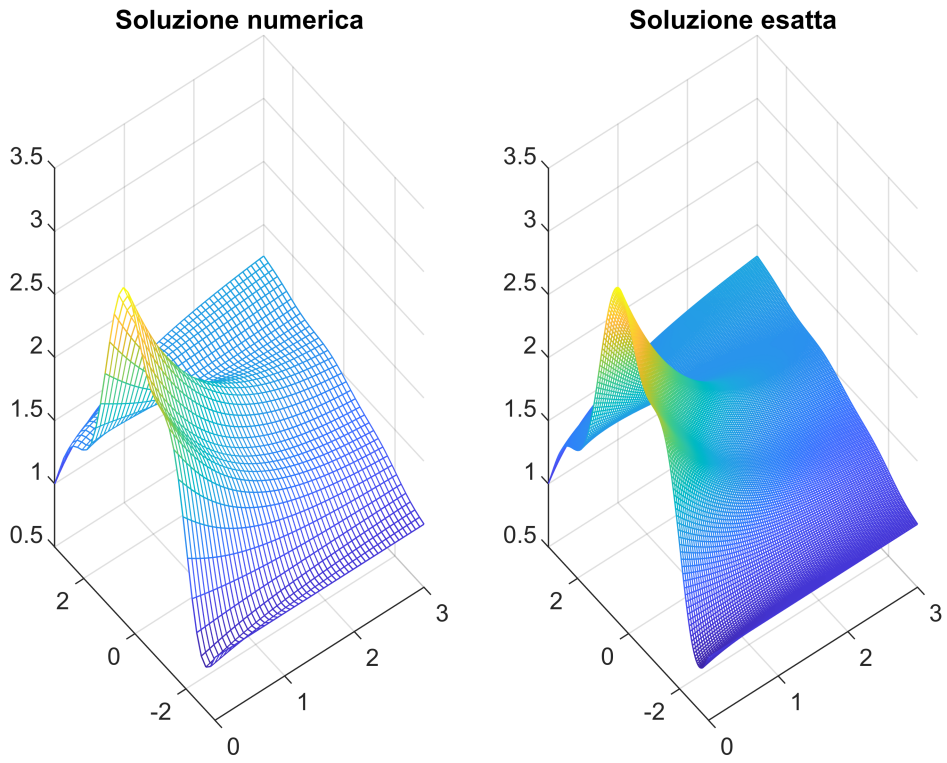
$\tau < \frac{h^2}{2} \quad \tau = 0.06 \quad \frac{h^2}{2} = 0.016$

Nel caso K = 200 invece si ha che $\tau = 0.015$ soddisfando così la condizione di stabilità.

PLOT 3D

```
%--- Grafico 3D (plot statico sul dominio spazio-tempo) ---
```

```
% Soluzione numerica
figure
ax1 = subplot(1,2,1);
mesh(t, x, uCN)
title("Soluzione numerica")

% Soluzione esatta
ax2 = subplot(1,2,2);
space = linspace(-L, L, 200);
time = linspace(0, T, 100);
[xx, tt] = ndgrid(space, time);
mesh(tt, xx, uex(xx, tt));
title("Soluzione esatta")

hlink = linkprop([ax1,ax2],{'CameraPosition','CameraUpVector'});
rotate3d on
```



Esercizio 1.3 Calcolo errori

Usiamo le seguenti definizioni di errore

$$e_1 = \max_k \max_n |u_n^k - u(x_n, t_k)|$$

$$e_2 = \max_k \sqrt{h \sum_n (u_n^k - u(x_n, t_k))^2}$$

```
% Valutazione dell'ordine di convergenza p (risp. errori e1 ed e2)
M = 4;
e1s = zeros(M, 1);
e2s = zeros(M, 1);

% Ciclo sui dimezzamenti
N = 25;
K = 35;
for i = 1:M
    [x, t, u] = calore_EI(L, N, T, K, ua, ub, f, u0);
    h = 2*L/N;

    [xx,tt] = ndgrid(x,t);
    uEX = uex(xx,tt);

    % Calcolo errori
    e1s(i) = max(max(abs(u-uEX)));
    e2s(i) = max((sqrt(h*sum((u-uEX).^2))));

    % Dimezzamento
    K = 2*K;
    N = 2*N;
end


% Stima dell'ordine di convergenza
p1 = log2(e1s(1:end-1)./e1s(2:end)) ;
p2 = log2(e2s(1:end-1)./e2s(2:end)) ;

fprintf("\np1 =\n")
```

```
p1 =
```

```
disp(p1)
```

```
    1.0552
    1.0162
    1.0046
```

```
fprintf("\np2 =\n")
```

```
p2 =
```

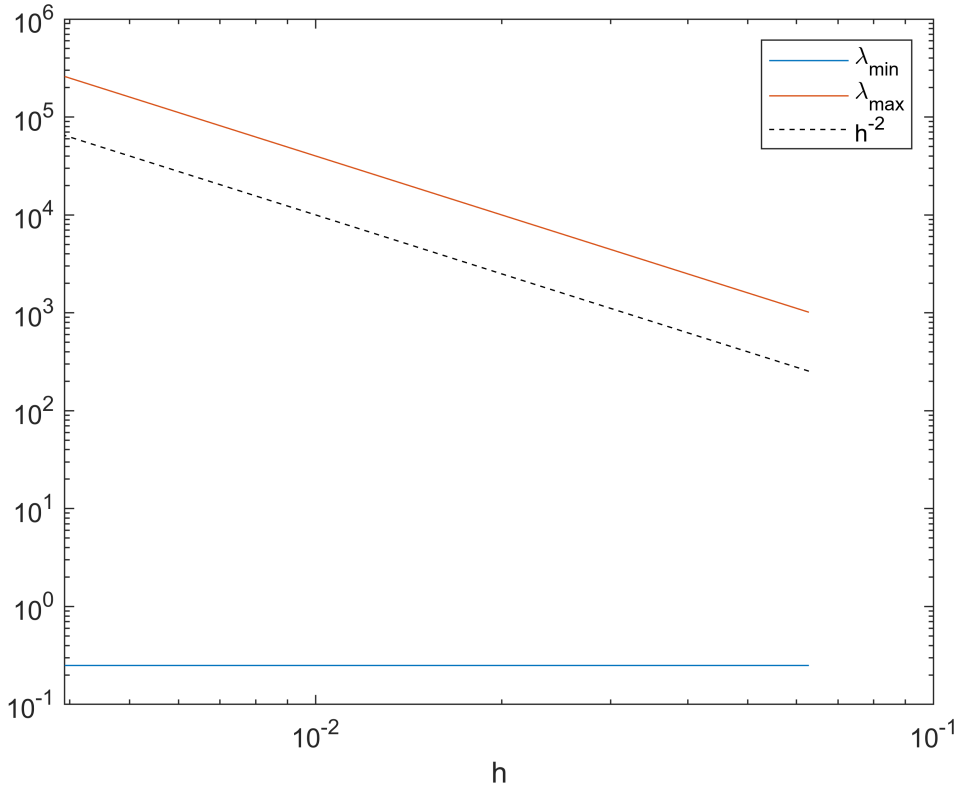
```
disp(p2)
```

```
    1.0266
    1.0132
    1.0065
```

Si ottengono gli ordini di convergenza che ci si aspetta

Esercizio 1.4 : ANDAMENTO AUTOVALORI

```
L = pi;
N = 100;
M = 5;

h = zeros(M, 1);
lambdamin = zeros(M, 1);
lambdamax = zeros(M, 1);

for i = 1:M
    % Costruzione in formato sparso della matrice A
    h(i) = 2*L/N;
    A = (1/h(i)^2) * spdiags([-1 2 -1], [-1 0 1], N-1, N-1);
    N = 2*N;
    % Calcolo degli autovalori minimo e massimo
    lambdas = eig(A);
    lambdamin(i) = min(lambdas);
    lambdamax(i) = max(lambdas);
end
figure
loglog(h, lambdamin);
hold on
loglog(h, lambdamax);
loglog(h, h.^-2, '--k');
xlabel('h')
legend('\lambda_{min}','\lambda_{max}','h^{-2}')
```



Si vede che l'autovalore minimo è costante mentre quello massimo si comporta come h^{-2} . Inoltre sono tutti strettamente positivi coerentemente col fatto che A è simmetrica definita positiva