Problema 2.1

Problema 2.1. Si consideri la funzione $\sqrt{x}$.

(a) Sia $p(x)$ il polinomio d'interpolazione di $\sqrt{x}$ sui nodi

$$x_0 = 0, \quad x_1 = \frac{1}{64}, \quad x_2 = \frac{4}{64}, \quad x_3 = \frac{9}{64}, \quad x_4 = \frac{16}{64}, \quad x_5 = \frac{25}{64}, \quad x_6 = \frac{36}{64}, \quad x_7 = \frac{49}{64}, \quad x_8 = 1.$$

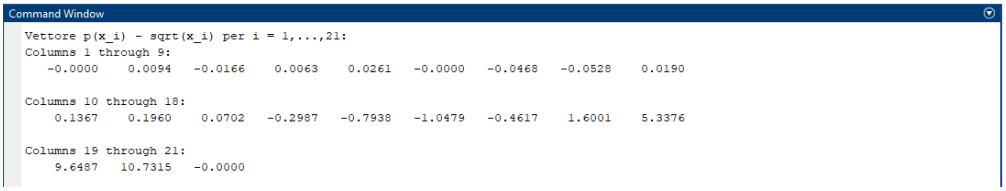
Calcolare il vettore (colonna)

$$\left[p(\zeta_1) - \sqrt{\zeta_1} \qquad p(\zeta_2) - \sqrt{\zeta_2} \qquad \cdots \qquad p(\zeta_{21}) - \sqrt{\zeta_{21}} \right]^T$$

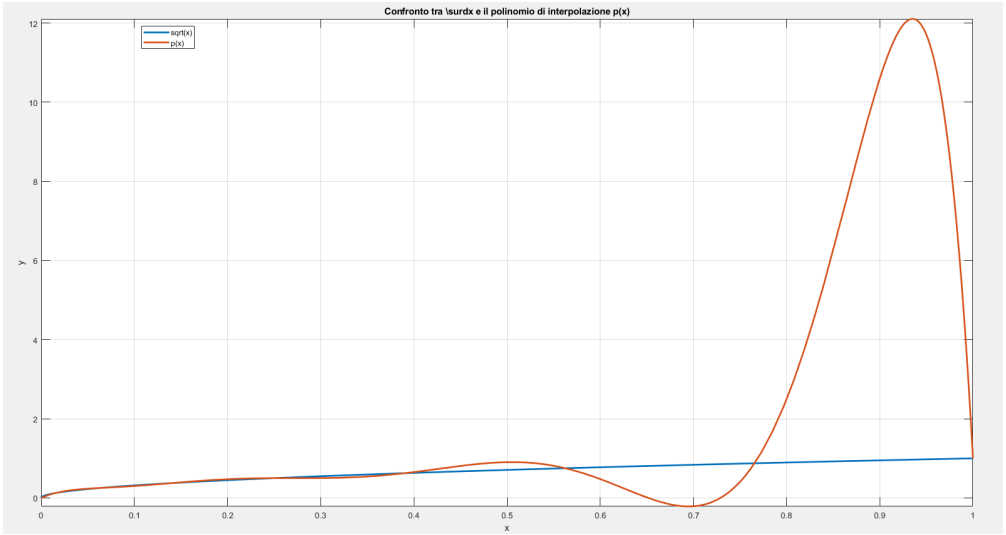
dove $\zeta_i = \frac{i-1}{20}$ per $i = 1, \ldots, 21$, e osservare in che modo varia la differenza $p(\zeta_i) - \sqrt{\zeta_i}$ al variare di i da 1 a 21.

(b) Tracciare il grafico di $\sqrt{x}$ e di $p(x)$ sull'intervallo $[0, 1]$, ponendo i due grafici su un'unica figura e inserendo una legenda che ci dica qual è la funzione $\sqrt{x}$ e qual è il polinomio $p(x)$.

Soluzione punto (a)



Soluzione punto (b)



Codice Finale

```
clear; clc; close all;
%-------------------------------------------------------------------------
% 1) DEFINIZIONE DEI NODI DI INTERPOLAZIONE E DELLA FUNZIONE DA INTERPOLARE
```

```
%-------------------------------------------------------------------------
% I nodi sono i valori: 0, 1/64, 4/64, 9/64, 16/64, 25/64, 36/64, 49/64, 64/64
x_nodes = [0, 1/64, 4/64, 9/64, 16/64, 25/64, 36/64, 49/64, 64/64]; % [cite:
2]
% Funzione di cui vogliamo fare l'interpolazione
y_nodes = sqrt(x_nodes); % [cite: 3]
%-------------------------------------------------------------------------
% 2) COSTRUZIONE DEL POLINOMIO DI INTERPOLAZIONE p(x)
%-------------------------------------------------------------------------
% La valutazione del polinomio avverrà direttamente tramite ValPol.
%-------------------------------------------------------------------------
% 3) CALCOLO DEL VETTORE DELLE DIFFERENZE p(x_i) - sqrt(x_i) PER 21 PUNTI
%-------------------------------------------------------------------------
% Creiamo 21 punti equispaziati tra 0 e 1
N = 21; % [cite: 5]
x_eval = linspace(0, 1, N); % suddivisione di [0,1] in 21 punti [cite: 6]
% Valutiamo il polinomio interpolante nei 21 punti usando ValPol
% È necessario che ValPol.m sia accessibile (stessa cartella o nel path)
p_eval = ValPol(x_nodes, y_nodes, x_eval);
% Calcoliamo la funzione sqrt(x) negli stessi 21 punti
f_eval = sqrt(x_eval); % [cite: 7]
% Vettore delle differenze: p(x_i) - sqrt(x_i)
diff_vector = p_eval - f_eval; % [cite: 8]
% Visualizziamo il vettore delle differenze in colonna
disp('Vettore p(x_i) - sqrt(x_i) per i = 1,...,21 (calcolato con ValPol):'); %
[cite: 9]
% Stampiamo i valori in tre blocchi separati
disp('Columns 1 through 9:'); % [cite: 10]
disp(diff_vector(1:9)); % [cite: 10]
disp('Columns 10 through 18:'); % [cite: 10]
disp(diff_vector(10:18)); % [cite: 10]
disp('Columns 19 through 21:'); % [cite: 11]
disp(diff_vector(19:21)); % [cite: 11]
%-------------------------------------------------------------------------
% 4) GRAFICI: CONFRONTO TRA sqrt(x) E p(x)
%-------------------------------------------------------------------------
figure;
% Tracciamo sqrt(x)
fplot(@(x) sqrt(x), [0, 1], 'LineWidth', 2); % [cite: 12]
hold on; grid on;
% Tracciamo p(x) usando ValPol
% ValPol necessita dei nodi (x_nodes, y_nodes) e dei punti di valutazione (x)
fplot(@(x_dynamic) ValPol(x_nodes, y_nodes, x_dynamic), [0, 1], 'LineWidth',
2);
% Aggiungiamo titolo, legenda e assi
title('Confronto tra $\sqrt{x}$ e il polinomio di interpolazione p(x) (con
```

```
ValPol)', ...
'Interpreter', 'latex');
xlabel('x'); ylabel('y'); % [cite: 14]
legend('$\sqrt{x}$', 'p(x) (ValPol)', 'Location','best', 'Interpreter',
'latex'); % [cite: 14]
```

Problema 2.2

Problema 2.2. Si consideri la funzione

$$f(x) = \mathrm{e}^x.$$

Per ogni intero $n \geq 1$ indichiamo con I_n la formula dei trapezi di ordine n per approssimare

$$I = \int_0^1 f(x)\mathrm{d}x = 1.7182818284590...$$

[1]Si dice in tal caso che α è un *punto fisso* della funzione $g(x)$ in $[a, b]$, perché la funzione $g(x)$ "lascia fisso" α essendo $g(\alpha) = \alpha$.

(a) Per ogni fissato $\varepsilon > 0$ determinare un $n = n(\varepsilon)$ tale che $|I - I_n| \leq \varepsilon$.
(b) Costruire una tabella che riporti vicino ad ogni $\varepsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$:
- il numero $n(\varepsilon)$;
- il valore I_n per $n = n(\varepsilon)$;
- il valore esatto I (in modo da confrontarlo con I_n);
- l'errore $|I - I_n|$ (che deve essere $\leq \varepsilon$).

(c) Calcolare le approssimazioni di I ottenute con le formule dei trapezi I_2, I_4, I_8, I_{16} e confrontarle con il valore esatto I.
(d) Sia $p(x)$ il polinomio d'interpolazione dei valori I_2, I_4, I_8, I_{16} sui nodi h_2^2, h_4^2, h_8^2, h_{16}^2, dove $h_2 = \frac{1}{2}$, $h_4 = \frac{1}{4}$, $h_8 = \frac{1}{8}$, $h_{16} = \frac{1}{16}$ sono i passi di discretizzazione relativi alle formule dei trapezi I_2, I_4, I_8, I_{16} rispettivamente. Calcolare $p(0)$ e confrontare I_2, I_4, I_8, I_{16}, $p(0)$ con il valore esatto I. Che cosa si nota?

Soluzione punto (a)

Sia $ f(x) = e^x $. Per il teorema sul resto della formula dei trapezi:

$$\left| \int_0^1 e^x \, dx - I_n \right| = \left| -\frac{f''(\mu)}{12} \left(\frac{1}{n} \right)^2 \right| = \left| \frac{f''(\mu)}{12n^2} \right| \quad (\mu \in [0,1])$$

Calcoliamo $f''(x)$:

$-f'(x) = e^x$
$-f''(x) = e^x$

$\forall x \in [0,1]: \quad f''(x) = |e^x| = e^x \leq e^1 = e$

Dunque,

$$\left| \int_0^1 e^x \, dx - I_n \right| = \left| \frac{f''(\mu)}{12n^2} \right| \leq \frac{e}{12n^2}$$

Imponiamo:

$$\frac{e}{12n^2} \leq \varepsilon \iff n \geq \sqrt{\frac{e}{12\varepsilon}} \Rightarrow n(\varepsilon) = \left\lceil \sqrt{\frac{e}{12\varepsilon}} \right\rceil$$

Dunque, se prendo $n \geq n(\varepsilon)$, allora sono sicuro che:

$$\left| \int_0^1 e^x \, dx - I_n \right| \leq \varepsilon$$

Soluzione punto (b)

```
>> tabella
Epsilon    n(epsilon)    I_n                 Errore
1.0e-01    2             1.7539310925        3.5649264006e-02
1.0e-02    5             1.7240056198        5.7237913237e-03
1.0e-03    16            1.7188411286        5.5930012095e-04
1.0e-04    48            1.7183439765        6.2148054069e-05
1.0e-05    151           1.7182881084        6.2799898122e-06
1.0e-06    476           1.7182824604        6.3197400291e-07
1.0e-07    1506          1.7182818916        6.3133985595e-08
1.0e-08    4760          1.7182818348        6.3197409528e-09
1.0e-09    15051         1.7182818291        6.3209260048e-10
1.0e-10    47595         1.7182818285        6.3191452071e-11
>>
```

Codice soluzione punto (b)

```
% Definizione della funzione f(x) = exp(x)
f = @(x) exp(x);

% Valore esatto dell'integrale I
I_exact = exp(1) - 1;

% Valori di epsilon
epsilon_values = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-
10];

% Preallocazione dei risultati
n_values = zeros(size(epsilon_values));
I_n_values = zeros(size(epsilon_values));
errors = zeros(size(epsilon_values));

% Calcolo di n(epsilon), I_n e dell'errore
for i = 1:length(epsilon_values)
    epsilon = epsilon_values(i);
    n_values(i) = ceil(sqrt(exp(1)/(12*epsilon))); % n(epsilon) arrotondato
verso l'alto
    n = n_values(i);
```

```
    % Calcolo di I_n con la funzione Trapezi
    I_n = Trapezi(0, 1, n, f); % Chiamata alla funzione Trapezi
    I_n_values(i) = I_n;

    % Calcolo dell'errore
    errors(i) = abs(I_exact - I_n);
end

% Tabella dei risultati
fprintf('%-10s %-12s %-20s %-15s\n', 'Epsilon', 'n(epsilon)', 'I_n',
'Errore');
for i = 1:length(epsilon_values)
    fprintf('%-10.1e %-12d %-20.10f %-15.10e\n', epsilon_values(i),
n_values(i), I_n_values(i), errors(i));
end

function [app] = Trapezi(a, b, n, f)
    % Formula dei trapezi
    r = 0;
    h = (b - a) / n;
    for j = 1:(n - 1)
        r = r + f(a + j * h);
    end
    app = (((f(a) + f(b)) / 2) + r) * h;
end
```

Soluzione punti (c) e (d)

```
>> problema2
Valore esatto I: 1.7182818285
Valori calcolati con i trapezi:
I2 = 1.7539310925, I4 = 1.7272219046, I8 = 1.7205185922, I16 = 1.7188411286
Valore di p(0): 1.7182818285

Confronto:
Errore |I2 - I| = 3.5649264006e-02
Errore |I4 - I| = 8.9400760985e-03
Errore |I8 - I| = 2.2367637053e-03
Errore |I16 - I| = 5.5930012095e-04
Errore |p(0) - I| = 1.3438139490e-12
>>
```

Codice soluzione punti (c) e (d)

```
% Definizione della funzione e dell'intervallo
f = @(x) exp(x);
a = 0; b = 1;

% Calcolo delle approssimazioni con la formula dei trapezi
```

```
I2 = Trapezi(a, b, 2, f);    % n = 2
I4 = Trapezi(a, b, 4, f);    % n = 4
I8 = Trapezi(a, b, 8, f);    % n = 8
I16 = Trapezi(a, b, 16, f);  % n = 16

% Nodi e valori per l'interpolazione
H = [1/2, 1/4, 1/8, 1/16].^2;  % Quadrati dei passi
I = [I2, I4, I8, I16];          % Approssimazioni corrispondenti

% Valutazione del polinomio interpolante con ValPol
T = 0; % Valutiamo il polinomio in x = 0
P0 = ValPol(H, I, T);

% Valore esatto dell'integrale
I_exact = exp(1) - 1;

% Stampa dei risultati
fprintf('Valore esatto I: %.10f\n', I_exact);
fprintf('Valori calcolati con i trapezi:\n');
fprintf('I2 = %.10f, I4 = %.10f, I8 = %.10f, I16 = %.10f\n', I2, I4, I8, I16);
fprintf('Valore di p(0): %.10f\n', P0);

% Confronto tra I2, I4, I8, I16, p(0) e il valore esatto I
fprintf('\nConfronto:\n');
fprintf('Errore |I2 - I| = %.10e\n', abs(I2 - I_exact));
fprintf('Errore |I4 - I| = %.10e\n', abs(I4 - I_exact));
fprintf('Errore |I8 - I| = %.10e\n', abs(I8 - I_exact));
fprintf('Errore |I16 - I| = %.10e\n', abs(I16 - I_exact));
fprintf('Errore |p(0) - I| = %.10e\n', abs(P0 - I_exact));
```

Problema 2.3

Problema 2.3. Consideriamo la funzione $f(x) = \frac{1}{x \log x}$ e indichiamo rispettivamente con I_n e S_n la formula dei trapezi e di Cavalieri-Simpson di ordine n per approssimare $I = \int_2^5 f(x)\mathrm{d}x$.

(a) Calcolare I prima manualmente e poi con la funzione simbolica `int` di MATLAB.

(b) Costruire una tabella che riporti vicino ad ogni valore di

$$n = 5, 10, 20, 40, 80, 160, 320, 640, 1280, 2560$$

sia le approssimazioni di I ottenute con I_n e S_n sia i relativi errori $|I_n - I|$ e $|S_n - I|$. Quale delle formule I_n e S_n converge più velocemente al valore esatto I al crescere di n?

Soluzione punto (a)

Sia:

$$I = \int_2^5 \frac{1}{x \log x}\, dx$$

Con:

$$y = \log x \Rightarrow dy = \frac{1}{x}\, dx$$

Allora:

$$I = \int_{\log 2}^{\log 5} \frac{dy}{y} = \log|y| + C\Big|_{\log 2}^{\log 5} = \log(\log 5) - \log(\log 2)$$

$$\approx 0.84261360$$

Soluzione punto (b)

```
>> problema4
n        I_n             S_n             |I_n - I|       |S_n - I|
-------------------------------------------------------------------
5        8.667092e-01    8.426174e-01    2.431127e-02    2.195180e-04
10       8.486404e-01    8.424135e-01    6.242456e-03    1.557515e-05
20       8.439702e-01    8.423989e-01    1.572295e-03    1.011061e-06
40       8.427917e-01    8.423980e-01    3.938322e-04    6.382981e-08
80       8.424964e-01    8.423979e-01    9.850591e-05    3.999568e-09
160      8.424225e-01    8.423979e-01    2.462948e-05    2.501337e-10
320      8.424041e-01    8.423979e-01    6.157557e-06    1.563594e-11
640      8.423995e-01    8.423979e-01    1.539401e-06    9.768852e-13
1280     8.423983e-01    8.423979e-01    3.848510e-07    6.228351e-14
2560     8.423980e-01    8.423979e-01    9.621279e-08    3.663736e-15
fx >>
```

Codice soluzione punto (b)

```
% Definizione della funzione f(x)
f = @(x) 1 ./ (x .* log(x));

% Intervallo di integrazione
a = 2;
b = 5;

% Calcolare il valore esatto dell'integrale I
I_exact = integral(f, a, b);

% Vettore dei valori di n
n_values = [5, 10, 20, 40, 80, 160, 320, 640, 1280, 2560];

% Inizializzare vettori per gli errori
error_trapezi = zeros(size(n_values));
error_simpson = zeros(size(n_values));
```

```
% Creiamo la tabella
fprintf('%-8s %-15s %-15s %-15s %-15s\n', 'n', 'I_n', 'S_n', '|I_n - I|',
'|S_n - I|');
fprintf('%s\n', repmat('-', 1, 68));

for i = 1:length(n_values)
    n = n_values(i);

    % Approssimazione tramite la formula dei trapezi
    I_n = Trapezi(a, b, n, f);

    % Approssimazione tramite la formula di Cavalieri-Simpson
    S_n = CavaSimp(a, b, f, n);

    % Calcolare gli errori assoluti
    error_trapezi(i) = abs(I_n - I_exact);
    error_simpson(i) = abs(S_n - I_exact);

    % Visualizzare i risultati
    fprintf('%-8d %-15.6e %-15.6e %-15.6e %-15.6e\n', n, I_n, S_n,
error_trapezi(i), error_simpson(i));
end
```

Problema 2.4

Problema 2.4. Si consideri il sistema lineare $A\mathbf{x} = \mathbf{b}$ dove $\mathbf{b} = [1, 0, -2, 0]^T$ e

$$A = \begin{bmatrix} 1 & -\frac{1}{4} & \frac{1}{3} & 0 \\ -1 & 2 & 0 & \frac{1}{2} \\ 2 & 1 & 3 & -\frac{1}{3} \\ -1 & -2 & -4 & 7 \end{bmatrix}.$$

(a) Sia G_ω la matrice d'iterazione del metodo SOR con parametro di rilassamento $\omega > 0$ per risolvere il sistema dato. Tracciare con MATLAB il grafico della funzione $\omega \mapsto \rho(G_\omega)$ per $\omega \in (0, 2]$ e determinare il valore $\omega_{\text{opt}} \in \{\frac{i}{m} : i = 1, \dots, 2m\}$ che minimizza $\rho(G_\omega)$ nel caso $m = 1000$.
(b) Calcolare $\rho(G_\omega)$ nel caso $\omega = 1$ e $\omega = \omega_{\text{opt}}$.
(c) Riportare in una tabella:
- le prime 10 iterazioni del metodo di Gauss-Seidel (classico) per risolvere il sistema dato, partendo dal vettore d'innesco $\mathbf{x}^{(0)} = [0, 0, 0, 0]^T$;
- le prime 10 iterazioni del metodo SOR con parametro di rilassamento ω_{opt} per risolvere il sistema dato, partendo dal vettore d'innesco $\mathbf{x}^{(0)} = [0, 0, 0, 0]^T$.

Confrontare le iterazioni con la soluzione esatta $\mathbf{x}$ del sistema dato calcolando in particolare la norma ∞ della differenza fra le iterazioni e la soluzione: quale dei due metodi converge più velocemente alla soluzione esatta?

Punto (a) - Tracciare $(\omega \mapsto \rho(G_\omega))$ e determinare (ω_{opt})

Obiettivo:
Studiare come cambia il raggio spettrale $(\rho(G_\omega))$ al variare di (ω), per trovare il valore ottimale che minimizza $(\rho(G_\omega))$.

Procedura:

- Costruiamo la matrice di iterazione:

  $$G_\omega = M^{-1}N$$

  dove

  $$M = \frac{1}{\omega}D - E, \quad N = \frac{1-\omega}{\omega}D + F$$

  e (D), (E), (F) sono le parti diagonale, inferiore e superiore di (A).
- Per ogni $\omega \in (0, 2)$ calcoliamo $\rho(G_\omega)$, il massimo valore assoluto degli autovalori di G_ω.
- Individuiamo ω_{opt} come il valore di ω che minimizza $\rho(G_\omega)$.
- Tracciamo il grafico $\omega \mapsto \rho(G_\omega)$ per visualizzare il comportamento.

Codice

```
% Dati del problema
A = [1, -1/4, 1/3, 0;
    -1, 2, 0, 1/2;
     2, 1, 3, -1/3;
    -1, -2, -4, 7];
b = [1; 0; -2; 0];
m = 1000;
x0 = zeros(4,1);
epsilon = 1e-10;
Nmax = 10000;

% Decomposizione
D = diag(diag(A));
E = -tril(A, -1);
F = -triu(A, 1);

% Variazione di omega
omega_values = linspace(0.01, 2, 2*m);
rho_values = zeros(size(omega_values));

for i = 1:length(omega_values)
    omega = omega_values(i);
    M = (1/omega)*D - E;
    N = ((1-omega)/omega)*D + F;
```

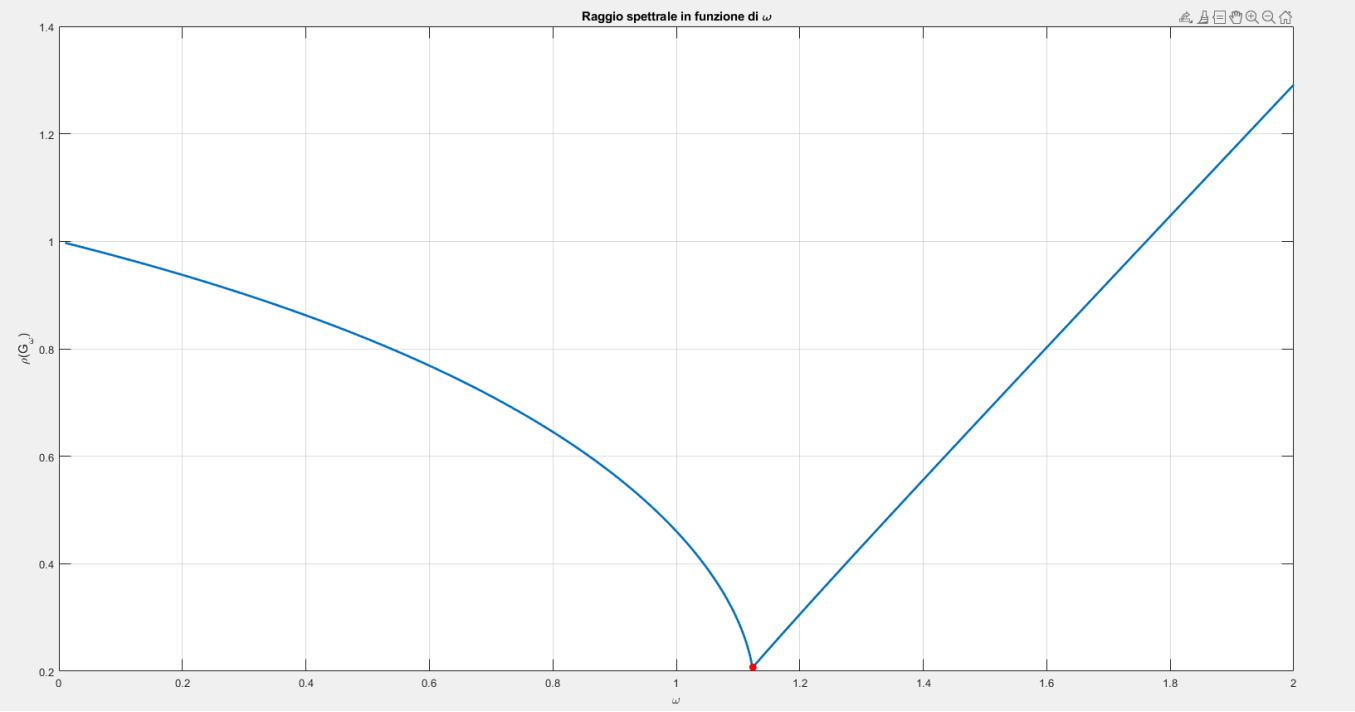
```
    G_omega = M \ N;    % Matrice di iterazione
    rho_values(i) = max(abs(eig(G_omega))); % raggio spettrale
end

% Trova omega ottimale
[rho_min, idx_min] = min(rho_values);
omega_opt = omega_values(idx_min);

% Plot
figure;
plot(omega_values, rho_values, 'LineWidth', 2);
xlabel('\omega');
ylabel('\rho(G_\omega)');
title('Raggio spettrale in funzione di \omega');
grid on;
hold on;
plot(omega_opt, rho_min, 'ro', 'MarkerFaceColor','r');
legend('\rho(G_\omega)', 'omega_{opt}');
hold off;

fprintf('Omega ottimale: %.4f\n', omega_opt);
fprintf('Rho minimo: %.4f\n', rho_min);
```

Grafico della funzione



Risultato

$\omega\ ottimale$: 1.1240
$\rho\ minimo$: 0.2075

Punto (b) - Calcolare $\rho(G_\omega)$ per $\omega = 1$ e $\omega = \omega_{\text{opt}}$

Obiettivo:
Confrontare il raggio spettrale:

- Quando $\omega = 1$ (corrispondente al metodo Gauss-Seidel).
- Quando $\omega = \omega_{\text{opt}}$ (ottimale).

Procedura:

- Calcolare G_ω e $\rho(G_\omega)$ nei due casi.
- Verificare che $\rho(G_{\omega_{\text{opt}}}) < \rho(G_1)$, ossia che il metodo ottimizzato converge più velocemente.

Codice

```
% Per omega = 1 (Gauss-Seidel)
omega1 = 1;
M1 = (1/omega1)*D - E;
N1 = ((1-omega1)/omega1)*D + F;
G1 = M1 \ N1;
rho1 = max(abs(eig(G1)));

fprintf('Rho(G) per omega = 1: %.4f\n', rho1);

% Per omega = omega_opt (già calcolato sopra)
Mopt = (1/omega_opt)*D - E;
Nopt = ((1-omega_opt)/omega_opt)*D + F;
Gopt = Mopt \ Nopt;
rho_opt = max(abs(eig(Gopt)));

fprintf('Rho(G) per omega_opt: %.4f\n', rho_opt);
```

Risultato

$\rho(G_\omega)\ per\ \omega = 1$: 0.4604
$\rho(G_\omega)\ per\ \omega_{opt}$: 0.2075

Punto (c) - Prime 10 iterazioni: confronto tra Gauss-Seidel e SOR con ω_{opt}

Obiettivo:
Osservare praticamente la velocità di convergenza dei due metodi in 10 iterazioni.

Procedura:

- Metodo Gauss-Seidel: risolvo

$$x^{(k+1)} = (D - E)^{-1}(Fx^{(k)} + b)$$

- Metodo SOR con ω_{opt}: risolvo

$$x^{(k+1)} = M^{-1}(Nx^{(k)} + b)$$

- Calcolare la norma $\|x^{(k)} - x\|_2$ ad ogni iterazione.
- Confrontare graficamente l'andamento dell'errore.

Codice

```
% Metodo Gauss-Seidel (omega = 1)
[x_GS, ~, ~] = metodo_SOR(A, b, 1, epsilon, x0, 10);

% Metodo SOR con omega_opt
[x_SORopt, ~, ~] = metodo_SOR(A, b, omega_opt, epsilon, x0, 10);

% Soluzione esatta
x_exact = A \ b;

% Iterazioni per confronto
X_GS = zeros(4,10);
X_SOR = zeros(4,10);

x_current = x0;
for k = 1:10
    [x_current, ~, ~] = metodo_SOR(A, b, 1, epsilon, x_current, 1);
    X_GS(:,k) = x_current;
end

x_current = x0;
for k = 1:10
    [x_current, ~, ~] = metodo_SOR(A, b, omega_opt, epsilon, x_current, 1);
    X_SOR(:,k) = x_current;
end

% errori
norm_GS = vecnorm(X_GS - x_exact);
norm_SOR = vecnorm(X_SOR - x_exact);
```

```
% Tabelle dei risultati
disp('Iterazioni Gauss-Seidel:');
disp(X_GS);

disp('Iterazioni SOR con omega_opt:');
disp(X_SOR);

% === Calcolo norma infinito ===
norminf_GS = zeros(1, 10);
norminf_SOR = zeros(1, 10);

for k = 1:10
    norminf_GS(k) = norm(X_GS(:,k) - x_exact, inf);
    norminf_SOR(k) = norm(X_SOR(:,k) - x_exact, inf);
end

% Grafico errori
figure;
semilogy(1:10, norminf_GS, '-o', 'LineWidth', 2);
hold on;
semilogy(1:10, norminf_SOR, '-x', 'LineWidth', 2);
xlabel('Numero di iterazioni');
ylabel('Norma errore ||x_k - x||_\infty');
legend('Gauss-Seidel', 'SOR \omega_{opt}');
title('Confronto convergenza tra Gauss-Seidel e SOR');
grid on;
hold off;
% Stampa norme infinito
disp('Norma infinito Gauss-Seidel:');
disp(norminf_GS);
disp('Norma infinito SOR con omega_opt:');
disp(norminf_SOR);
```

Tabelle

- **Iterazioni Gauss-Seidel**

Iterazione	1	2	3	4	5	6	7	8
x_1	1.0000	1.6250	1.9495	2.0982	2.1667	2.1983	2.2128	2.2195
x_2	5.000 $\cdot 10^{-1}$	9.554 $\cdot 10^{-1}$	1.1530	1.2443	1.2863	1.3056	1.3145	1.3186
x_3	-1.5000	-2.1319	-2.4299	-2.5670	-2.6301	-2.6591	-2.6725	-2.6787

Iterazione	1	2	3	4	5	6	7	8
x_4	-5.714 $\cdot 10^{-1}$	-7.132 $\cdot 10^{-1}$	-7.806 $\cdot 10^{-1}$	-8.116 $\cdot 10^{-1}$	-8.259 $\cdot 10^{-1}$	-8.324 $\cdot 10^{-1}$	-8.355 $\cdot 10^{-1}$	-8.369 $\cdot 10^{-1}$
$\|\|x^{(k)} - x\|\|_\infty$	1.2251	6.001 $\cdot 10^{-1}$	2.756 $\cdot 10^{-1}$	1.269 $\cdot 10^{-1}$	5.84 $\cdot 10^{-2}$	2.69 $\cdot 10^{-2}$	1.23 $\cdot 10^{-2}$	5.7 $\cdot 10^{-3}$

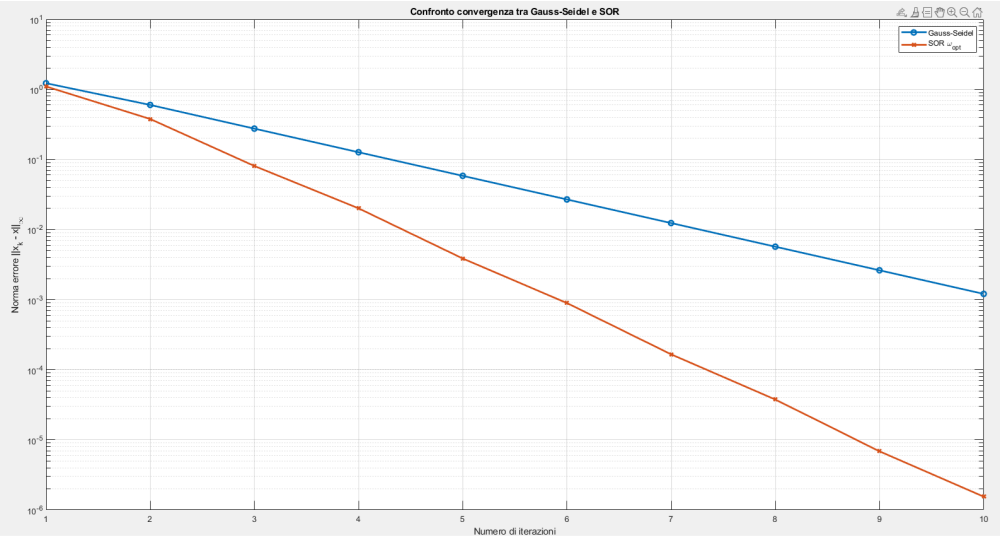
9	10
2.2225	2.2240
1.3205	1.3214
-2.6815	-2.6828
-8.375 $\cdot 10^{-1}$	-8.378 $\cdot 10^{-1}$
2.6 $\cdot 10^{-3}$	1.2 $\cdot 10^{-3}$

- **Iterazioni SOR con ω_{opt}**

Iterazione	1	2	3	4	5	6	7	8
x_1	1.1240	1.8470	2.1444	2.2051	2.2213	2.2243	2.2250	2.2251
x_2	6.316 $\cdot 10^{-1}$	1.1819	1.2829	1.3150	1.3203	1.3218	1.3220	1.3221
x_3	-1.8282	-2.4483	-2.6329	-2.6723	-2.6816	-2.6834	-2.6838	-2.6839
x_4	-7.908 $\cdot 10^{-1}$	-7.983 $\cdot 10^{-1}$	-8.358 $\cdot 10^{-1}$	-8.363 $\cdot 10^{-1}$	-8.380 $\cdot 10^{-1}$	-8.380 $\cdot 10^{-1}$	-8.380 $\cdot 10^{-1}$	-8.380 $\cdot 10^{-1}$
$\|\|x^{(k)} - x\|\|_\infty$	1.1012	3.781 $\cdot 10^{-1}$	8.08 $\cdot 10^{-2}$	2.01 $\cdot 10^{-2}$	3.8 $\cdot 10^{-3}$	8.9516 $\cdot 10^{-4}$	1.6618 $\cdot 10^{-4}$	3.7642 $\cdot 10^{-5}$

9	10
2.2252	2.2252
1.3221	1.3221
-2.6839	-2.6839
-8.380 $\cdot 10^{-1}$	-8.380 $\cdot 10^{-1}$
6.8686 $\cdot 10^{-6}$	1.5493 $\cdot 10^{-6}$

Confronto Convergenza



Conclusioni

Dal confronto risulta evidente che:

- Il metodo SOR con paramentro ottimale ω_{opt} converge molto piú velocemente rispetto al metodo Gauss-Seidel con $\omega = 1$
- $\rho(G_{\omega_{opt}})$ é inferiore a $\rho(G_1)$
- Giá dopo poche iterazioni, il metodo SOR ottimizzato presenta un errore significativamente minore

Codice Finale

```
clc;
clear;
close all;

% Dati del problema
A = [1, -1/4, 1/3, 0;
    -1, 2, 0, 1/2;
     2, 1, 3, -1/3;
    -1, -2, -4, 7];
b = [1; 0; -2; 0];
x0 = zeros(4,1);
epsilon = 1e-10;
Nmax = 1000;
m = 1000; % Per la discretizzazione di omega

% Decomposizione matrice
```

```
D = diag(diag(A));
E = -tril(A, -1);
F = -triu(A, 1);

% (a) Calcolo rho(G_omega) per omega in (0,2)
omega_values = linspace(0.01, 2, 2*m);
rho_values = zeros(size(omega_values));

for i = 1:length(omega_values)
    omega = omega_values(i);
    M = (1/omega)*D - E;
    N = ((1-omega)/omega)*D + F;
    G_omega = M \ N;
    rho_values(i) = max(abs(eig(G_omega)));
end

% Trova omega ottimale
[rho_min, idx_min] = min(rho_values);
omega_opt = omega_values(idx_min);

% Grafico
figure;
plot(omega_values, rho_values, 'LineWidth', 2);
xlabel('\omega');
ylabel('\rho(G_\omega)');
title('Raggio spettrale in funzione di \omega');
grid on;
hold on;
plot(omega_opt, rho_min, 'ro', 'MarkerFaceColor','r');
legend('\rho(G_\omega)', 'omega_{opt}');
hold off;

fprintf('Omega ottimale: %.4f\n', omega_opt);
fprintf('Raggio spettrale minimo: %.4f\n', rho_min);

% (b) Calcolo rho(G) per omega=1 e omega=omega_opt
% Omega = 1 (Gauss-Seidel)
M1 = (1/1)*D - E;
N1 = ((1-1)/1)*D + F;
G1 = M1 \ N1;
rho1 = max(abs(eig(G1)));
fprintf('Rho(G) per omega=1: %.4f\n', rho1);

% Omega = omega_opt
Mopt = (1/omega_opt)*D - E;
Nopt = ((1-omega_opt)/omega_opt)*D + F;
```

```
Gopt = Mopt \ Nopt;
rho_opt = max(abs(eig(Gopt)));
fprintf('Rho(G) per omega_opt: %.4f\n', rho_opt);

% (c) 10 iterazioni Gauss-Seidel e SOR(omega_opt)

% Metodo Gauss-Seidel
X_GS = zeros(4,10);
x_current = x0;
for k = 1:10
    [x_current, ~, ~] = metodo_SOR(A, b, 1, epsilon, x_current, 1);
    X_GS(:,k) = x_current;
end

% Metodo SOR con omega_opt
X_SOR = zeros(4,10);
x_current = x0;
for k = 1:10
    [x_current, ~, ~] = metodo_SOR(A, b, omega_opt, epsilon, x_current, 1);
    X_SOR(:,k) = x_current;
end

% Soluzione esatta
x_exact = A\b;

% Errori
norm_GS = vecnorm(X_GS - x_exact);
norm_SOR = vecnorm(X_SOR - x_exact);

% Tabelle
disp('Iterazioni Gauss-Seidel:');
disp(X_GS);

disp('Iterazioni SOR con omega_opt:');
disp(X_SOR);

% === Calcolo norma infinito ===
norminf_GS = zeros(1, 10);
norminf_SOR = zeros(1, 10);

for k = 1:10
    norminf_GS(k) = norm(X_GS(:,k) - x_exact, inf);
    norminf_SOR(k) = norm(X_SOR(:,k) - x_exact, inf);
end

% Grafico errori
```

```
figure;
semilogy(1:10, norminf_GS, '-o', 'LineWidth', 2);
hold on;
semilogy(1:10, norminf_SOR, '-x', 'LineWidth', 2);
xlabel('Numero di iterazioni');
ylabel('Norma errore ||x_k - x||_\infty');
legend('Gauss-Seidel', 'SOR \omega_{opt}');
title('Confronto convergenza tra Gauss-Seidel e SOR');
grid on;
hold off;
% Stampa norme infinito
disp('Norma infinito Gauss-Seidel:');
disp(norminf_GS);
disp('Norma infinito SOR con omega_opt:');
disp(norminf_SOR);
```

Problema 2.5

Problema 2.5. Consideriamo i seguenti due casi:

- $f(x) = x^3 + 3x - 1 - \mathrm{e}^{-x^2}$ e $[a, b] = [0, 1]$;
- $f(x) = \cos x - x$ e $[a, b] = [0, \pi]$.

Per ciascuno di questi due casi, risolvere i seguenti punti.

(a) Verificare che $f(a)f(b) < 0$.
(b) Tracciare il grafico di $f(x)$ su $[a, b]$ e verificare che $f(x)$ ha un unico zero ζ nell'intervallo (a, b).
(c) Dimostrare analiticamente che $f(x)$ ha un'unico zero ζ nell'intervallo (a, b).
(d) Costruire una tabella che riporti vicino ad ogni $\varepsilon \in \{10^{-1}, 10^{-2}, \dots, 10^{-10}\}$:
- un'approssimazione ξ_ε di ζ, calcolata con il metodo di bisezione, che soddisfa $|\xi_\varepsilon - \zeta| \leq \varepsilon$;
- il numero d'iterazioni K_ε effettuate dal metodo di bisezione per calcolare l'approssimazione ξ_ε;
- il valore $f(\xi_\varepsilon)$.

Caso 1

$$f(x) = x^3 + 3x - 1 - e^{-x^2}, [a, b] = [0, 1]$$

Punto (a): Verifica che $f(a)f(b) < 0$

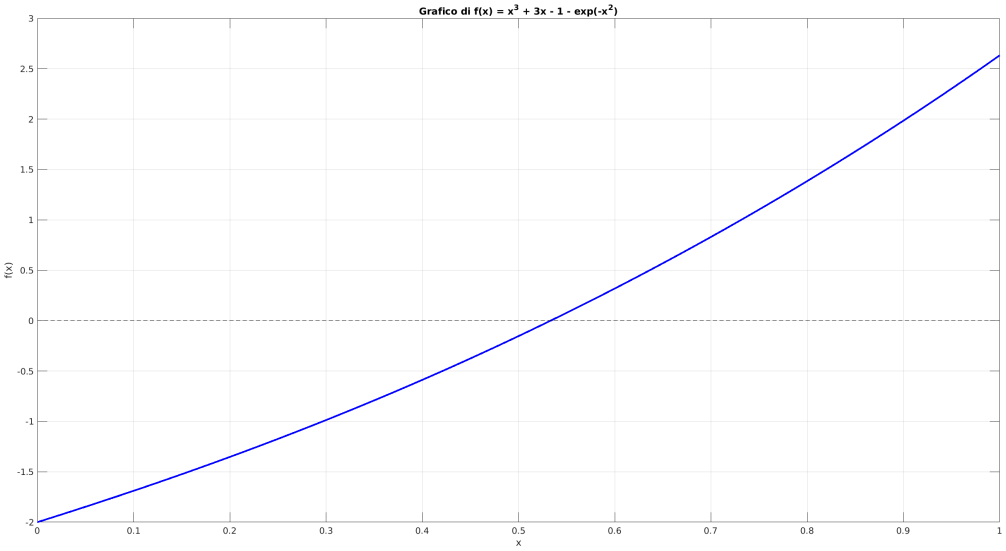
1. Calcoliamo $f(a)$ e $f(b)$:
   - $f(0) = 0^3 + 3(0) - 1 - e^{-0^2} = -1 - 1 = -2$,
   - $f(1) = 1^3 + 3(1) - 1 - e^{-1^2} = 1 + 3 - 1 - e^{-1} = 3 - e^{-1} \approx 2.63$.
2. Poiché $f(0) \cdot f(1) < 0$, (risulta $-2 \cdot 2,63 = -5,26$) possiamo procedere.

Punto (b): Grafico di $f(x)$ e verifica di uno zero unico

Tracciamo il grafico di $f(x)$ su $[0,1]$ con MATLAB per osservare che $f(x)$ ha un unico zero nell'intervallo $(0,1)$.

```
f = @(x) x.^3 + 3.*x - 1 - exp(-x.^2);
x = linspace(0, 1, 1000); % 1000 punti nell'intervallo [0, 1]
plot(x, f(x), 'b-', 'LineWidth', 2);
grid on;
xlabel('x');
ylabel('f(x)');
title('Grafico di f(x) = x^3 + 3x - 1 - e^{-x^2}');
```

Analisi: Osservando il grafico, si nota che $f(x)$ è continuo e cambia segno una sola volta tra 0 e 1.



Punto (c): Dimostrazione analitica che f(x) ha un unico zero

Usiamo il teorema di Bolzano e la monotonicità derivata dall'analisi di $f'(x)$:

$$f'(x) = 3x^2 + 3 + 2xe^{-x^2}.$$

1. $f'(x) > 0$ per ogni $x \in [0,1]$ (la funzione è strettamente crescente su $[0,1]$).
2. Poiché $f(x)$ è crescente e cambia segno in $[0,1]$, per il teorema di Bolzano esiste un unico zero $\zeta \in (0,1)$.

Punto (d): Tabella per $\varepsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$

Abbiamo usato il **metodo di bisezione** per calcolare:

- L'approssimazione ξ_ε,
- Il numero di iterazioni K_ε,

- Il valore $f(\xi_\varepsilon)$.

La tabella dei risultati è la seguente:

ϵ	x_i	K	$f(x_i)$
$1.0 \cdot 10^{-1}$	0.531250000000000	4	$-1.041995243049776 \cdot 10^{-2}$
$1.0 \cdot 10^{-2}$	0.535156250000000	7	$7.765312582933004 \cdot 10^{-3}$
$1.0 \cdot 10^{-3}$	0.533691406250000	10	$9.389559548024229 \cdot 10^{-4}$
$1.0 \cdot 10^{-4}$	0.533477783203125	14	$-5.586409047664276 \cdot 10^{-5}$
$1.0 \cdot 10^{-5}$	0.533489227294922	17	$-2.574612559369527 \cdot 10^{-6}$
$1.0 \cdot 10^{-6}$	0.533489704132080	20	$-3.542067064099541 \cdot 10^{-7}$
$1.0 \cdot 10^{-7}$	0.533489793539047	24	$6.211948844203619 \cdot 10^{-8}$
$1.0 \cdot 10^{-8}$	0.533489782363176	27	$1.007871253122516 \cdot 10^{-8}$
$1.0 \cdot 10^{-9}$	0.533489780034870	30	$-7.631157927789900 \cdot 10^{-10}$
$1.0 \cdot 10^{-10}$	0.533489780180389	34	$-8.550160579545718 \cdot 10^{-11}$

Codice MATLAB:

```
a = 0; b = 1;
f = @(x) x.^3 + 3.*x - 1 - exp(-x.^2);
epsilon_values = 10.^(-1:-1:-10); % Tolleranze
results = zeros(length(epsilon_values), 3); % Preallocazione: [xi_eps, K_eps,
f(xi_eps)]

for i = 1:length(epsilon_values)
    epsilon = epsilon_values(i);
    [xi, K, fx] = bisezione(a, b, f, epsilon);
    results(i, :) = [xi, K, fx];
end

% Mostra la tabella
disp('Tabella dei risultati:');
disp('epsilon        xi_eps         K_eps        f(xi_eps)');
disp(results);
```

Caso 2

$$f(x) = \cos x - x, [a, b] = [0, \pi]$$

Punto (a): Verifica che $f(a)f(b) < 0$

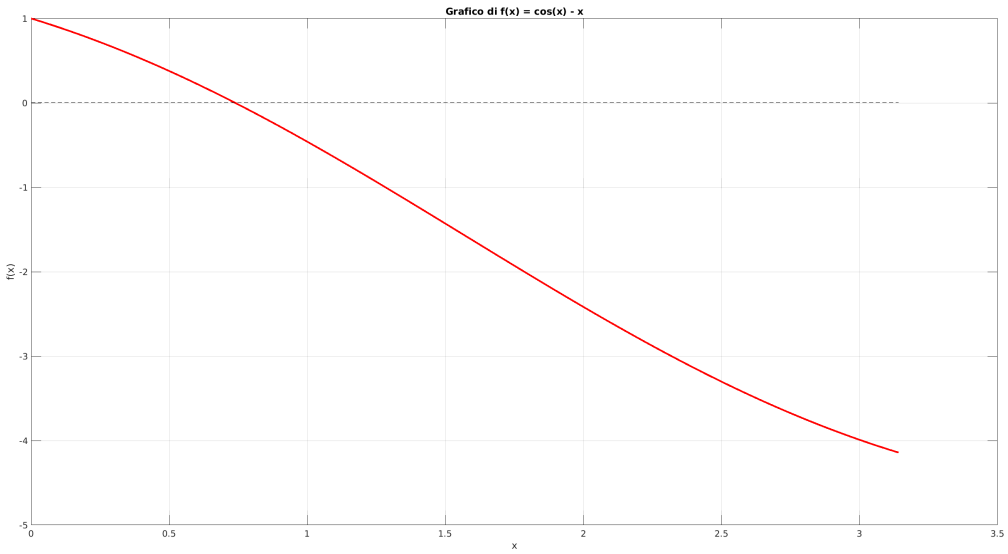
1. Calcoliamo $f(a)$ e $f(b)$:
   - $f(0) = \cos(0) - 0 = 1$,
   - $f(\pi) = \cos(\pi) - \pi = -1 - \pi < 0$.
2. Poiché $f(0) \cdot f(\pi) < 0$, possiamo procedere.

Punto (b): Grafico di $f(x)$ e verifica di uno zero unico

Codice MATLAB:

```
f = @(x) cos(x) - x;
x = linspace(0, pi, 1000); % 1000 punti nell'intervallo [0, pi]
plot(x, f(x), 'r-', 'LineWidth', 2);
grid on;
xlabel('x');
ylabel('f(x)');
title('Grafico di f(x) = cos(x) - x');
```

Analisi: Il grafico mostra che $f(x)$ è continuo e cambia segno una sola volta tra 0 e π.



Punto (c): Dimostrazione analitica che $f(x)$ ha un unico zero

Usiamo $f'(x) = -\sin(x) - 1$:

1. $f'(x) < 0$ per ogni $x \in [0, \pi]$ (la funzione è strettamente decrescente su $[0, \pi]$).
2. Poiché $f(x)$ è decrescente e cambia segno in $[0, \pi]$, per il teorema di Bolzano esiste un unico zero $\zeta \in (0, \pi)$.

Punto (d): Tabella per

$$\varepsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$$

Abbiamo usato il **metodo di bisezione** per calcolare:

- L'approssimazione ξ_ε,
- Il numero di iterazioni K_ε,
- Il valore $f(\xi_\varepsilon)$.

La tabella dei risultati è la seguente:

ϵ	x_i	K	$f(x_i)$
$1.0 \cdot 10^{-1}$	0.736310778185108	5	$4.640347169851511 \cdot 10^{-3}$
$1.0 \cdot 10^{-2}$	0.739378739760879	9	$-4.914153002637534 \cdot 10^{-4}$
$1.0 \cdot 10^{-3}$	0.738995244563908	12	$1.504357420498703 \cdot 10^{-4}$
$1.0 \cdot 10^{-4}$	0.739043181463529	15	$7.021030579146270 \cdot 10^{-5}$
$1.0 \cdot 10^{-5}$	0.739088122306924	19	$-5.002583233437718 \cdot 10^{-6}$
$1.0 \cdot 10^{-6}$	0.739085500757726	22	$-6.151237084139893 \cdot 10^{-7}$
$1.0 \cdot 10^{-7}$	0.739085173064076	25	$-6.669162500028136 \cdot 10^{-8}$
$1.0 \cdot 10^{-8}$	0.739085135028206	29	$-3.034334783436066 \cdot 10^{-9}$
$1.0 \cdot 10^{-9}$	0.739085133199558	32	$2.611200144997383 \cdot 10^{-11}$
$1.0 \cdot 10^{-10}$	0.739085133245275	35	$-5.039924033667376 \cdot 10^{-11}$

Codice MATLAB:

```
a = 0; b = pi;
f = @(x) cos(x) - x;
epsilon_values = 10.^(-1:-1:-10); % Tolleranze
results = zeros(length(epsilon_values), 3); % Preallocazione: [xi_eps, K_eps,
f(xi_eps)]

for i = 1:length(epsilon_values)
    epsilon = epsilon_values(i);
    [xi, K, fx] = bisezione(a, b, f, epsilon);
    results(i, :) = [xi, K, fx];
end

% Mostra la tabella
disp('Tabella dei risultati:');
```

```
disp('epsilon        xi_eps         K_eps        f(xi_eps)');
disp(results);
```

Codice Finale

```
% Funzioni e intervalli definiti dal problema
f1 = @(x) x.^3 + 3*x - 1 - exp(-x.^2); % Prima funzione
a1 = 0; b1 = 1; % Intervallo [a, b] per f1

f2 = @(x) cos(x) - x; % Seconda funzione
a2 = 0; b2 = pi; % Intervallo [a, b] per f2

% Lista di epsilon
epsilons = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10];

% Risoluzione per il primo caso
solve_case(f1, a1, b1, epsilons, 'f1(x) = x^3 + 3x - 1 - e^{-x^2}');

% Risoluzione per il secondo caso
solve_case(f2, a2, b2, epsilons, 'f2(x) = cos(x) - x');

% Funzione per risolvere ogni caso
function solve_case(f, a, b, epsilons, case_name)
    fprintf('\nSoluzione per %s:\n', case_name);

    % (a) Verifica che f(a)*f(b) < 0
    fa = f(a);
    fb = f(b);
    fprintf('(a) f(a)*f(b) = %.3f (segno opposto: %s)\n', fa * fb, ...
        string(fa * fb < 0));
    if fa * fb >= 0
        error('f(a) e f(b) devono avere segni opposti');
    end

    % (b) Tracciamento del grafico
    fprintf('(b) Tracciamento del grafico di f(x) su [%f, %f]\n', a, b);
    fplot(f, [a b]);
    hold on;
    grid on;
    plot(a, f(a), 'ro', 'DisplayName', 'f(a)');
    plot(b, f(b), 'bo', 'DisplayName', 'f(b)');
    xlabel('x'); ylabel('f(x)');
    title(['Grafico di f(x) - Caso ', case_name]);
    legend show;
```

```
    % (c) Dimostrazione analitica: fatta in modo separato (se necessario)

    % (d) Tabella dei risultati per vari epsilon
    fprintf('(d) Calcolo del metodo di bisezione per diverse tolleranze
epsilon:\n');
    fprintf('epsilon       xi               K       f(xi)\n');
    fprintf('-----------------------------------------------\n');
    for epsilon = epsilons
        [xi, K, fx] = bisezione(a, b, f, epsilon);
        fprintf('%e   %.15f   %d   %.15e\n', epsilon, xi, K, fx);
    end
end
```

Spiegazione Codice Finale

1. **Caso 1 e Caso 2**:
   - Si calcolano $f(a)$ e $f(b)$ per verificare che il prodotto è negativo.
   - Si tracciano i grafici per osservare il comportamento di $f(x)$.
   - Si riportano i risultati delle tabelle usando il metodo di bisezione.
2. **Funzione `bisezione`**:
   - Implementa il metodo di bisezione per trovare l'approssimazione di uno zero di una funzione continua su un intervallo $[a, b]$.
   - Restituisce l'approssimazione ξ_ε, il numero di iterazioni K_ε, e il valore $f(\xi_\varepsilon)$.
3. **Tabelle dei Risultati**:
   - Si stampano le tabelle per ogni caso, con i valori di ϵ, ξ_ε,K_ε, e $f(\xi_\varepsilon)$.

Problema 2.6

Problema 2.6. Consideriamo le seguenti due funzioni e gli intervalli riportati a fianco di esse:

- $g(x) = \frac{1+e^{-x^2}}{x^2+3}$, $\quad [a, b] = [0, 1]$;
- $g(x) = \cos x$, $\quad [a, b] = [0, \pi/3]$.

Per ciascuno di questi due casi, risolvere i seguenti punti.

(a) Tracciare il grafico di $y = g(x)$ e il grafico di $y = x$ sull'intervallo $[a, b]$ e dedurre che l'equazione $x = g(x)$ ha un'unica soluzione $\alpha \in [a, b]$. Che cosa rappresenta α nel grafico tracciato?

(b) Dimostrare analiticamente che l'equazione $x = g(x)$ ha un'unica soluzione nell'intervallo $[a, b]$.

(c) Costruire una tabella che riporti vicino ad ogni $\varepsilon \in \{10^{-1}, 10^{-2}, \dots, 10^{-10}\}$:

- un'approssimazione α_ε di α—calcolata con il metodo d'iterazione funzionale (1.3) innescato partendo da un punto $x_0 \in [a, b]$—che soddisfa la condizione di arresto del metodo (1.3) proposta nell'Esercizio 1.6 usando come soglia di precisione ε;
- il punto d'innesco $x_0 \in [a, b]$ utilizzato per innescare il metodo (1.3) e ottenere α_ε;
- il numero d'iterazioni K_ε effettuate dal metodo (1.3) per calcolare l'approssimazione α_ε;
- il valore $|\alpha_\varepsilon - g(\alpha_\varepsilon)|$.

Punto (a) - Tracciare i grafici di $y = g(x)$ e $y = x$

```
% Dati del problema
g1 = @(x) (1 + exp(-x.^2)) ./ (x.^2 + 3);
g2 = @(x) cos(x);

% Intervalli
a1 = 0; b1 = 1;
a2 = 0; b2 = pi/3;

% Grafico g1(x) e y = x
figure;
fplot(g1, [a1 b1], 'LineWidth', 2); hold on;
fplot(@(x) x, [a1 b1], '--r', 'LineWidth', 2);
title('Grafico di g_1(x) e y=x');
legend('g_1(x)', 'y=x');
grid on;

% Grafico g2(x) e y = x
figure;
fplot(g2, [a2 b2], 'LineWidth', 2); hold on;
fplot(@(x) x, [a2 b2], '--r', 'LineWidth', 2);
title('Grafico di g_2(x) e y=x');
legend('g_2(x)', 'y=x');
grid on;
```

Si noti che dai grafici generati vedremo che il parametro α rappresenta l'ascissa del punto di intersezione tra il grafico della funzione $g(x)$ e la retta $y = x$

Grafico $g_1(x)$

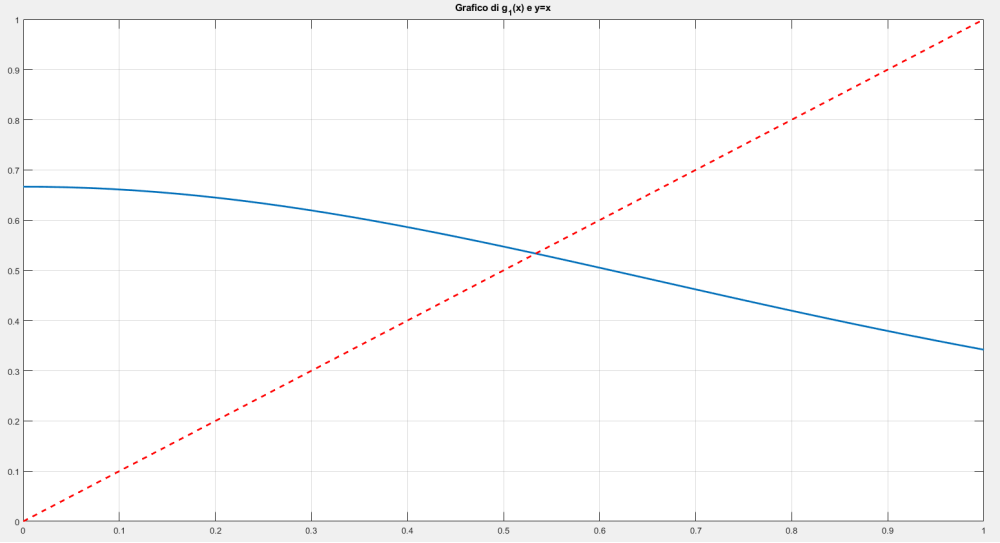
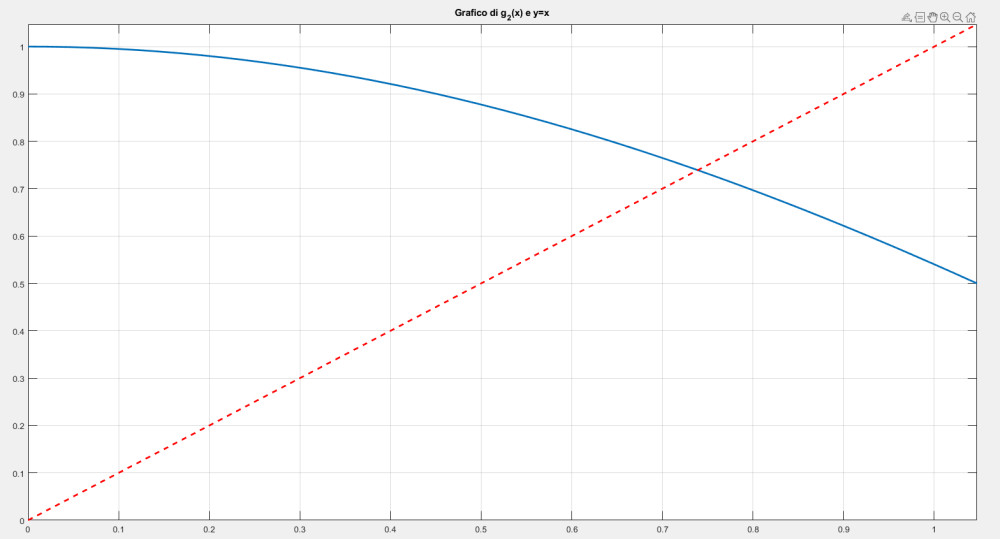


Grafico $g_2(x)$



Punto (b) - Dimostrazione analitica dell'unicitá

Dobbiamo verificare che:

- Le funzioni $g_1(x)$ e $g_2(x)$ sono di classe C^1 (cioè derivabili e con derivata continua);
- $|g_1'(x)| < 1$ e $|g_2'(x)| < 1$ nei rispettivi intervalli.

Se queste due condizioni sono soddisfatte, allora possiamo applicare il **Teorema del Punto Fisso**: l'equazione $x = g(x)$ ha **un'unica soluzione** α nell'intervallo dato.

Calcolo di $g_1'(x)$

La funzione è definita da:

$$g_1(x) = \frac{1 + e^{-x^2}}{x^2 + 3}$$

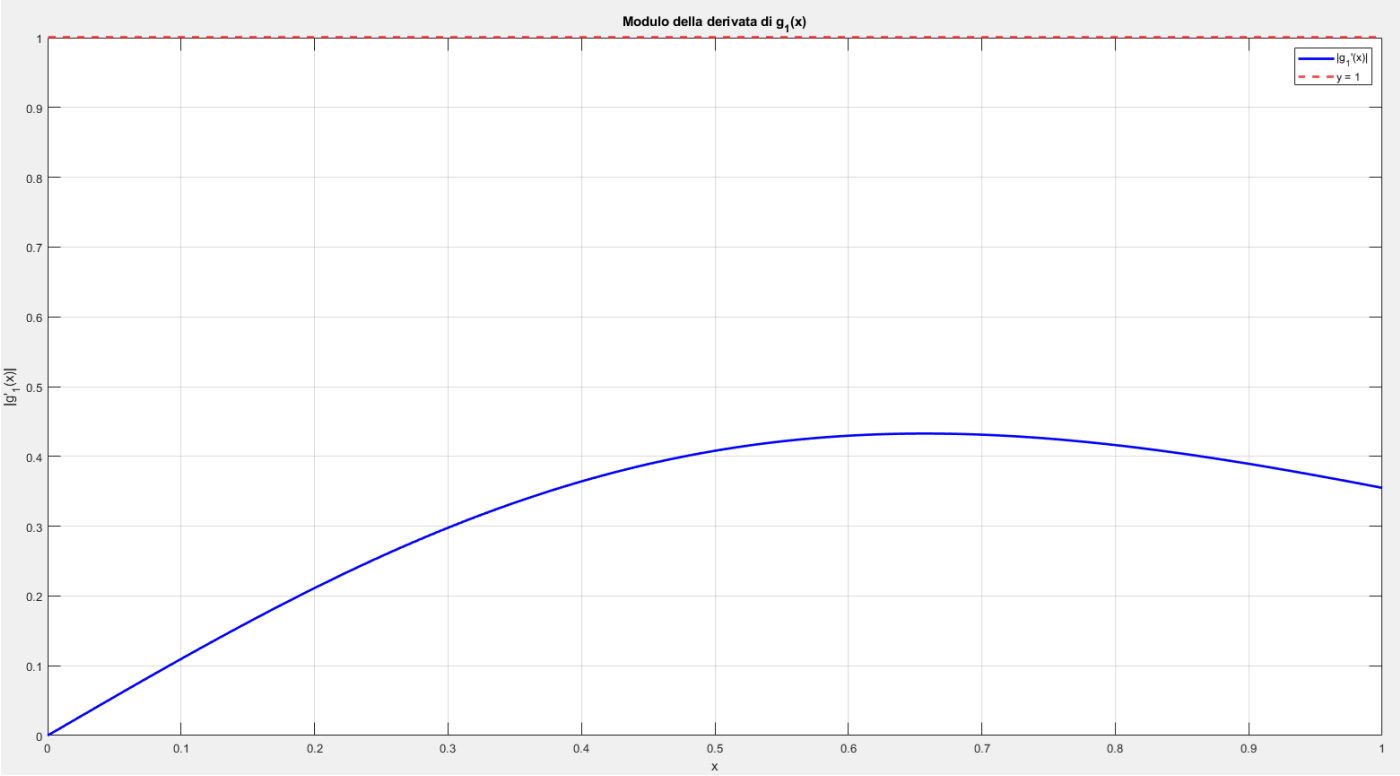
Applicando la regola del quoziente, otteniamo:

$$g_1'(x) = \frac{(x^2 + 3)(-2xe^{-x^2}) - (1 + e^{-x^2})(2x)}{(x^2 + 3)^2}$$

Questa derivata è continua su $[0, 1]$:

$$|g_1'(x)| < 1 \quad \text{per ogni} \quad x \in [0, 1]$$

Grafico



Codice

```
% Funzione g1(x)

g1 = @(x) (1 + exp(-x.^2)) ./ (x.^2 + 3);

% Derivata calcolata manualmente

g1_deriv = @(x) ((x.^2 + 3) .* (-2 .* x .* exp(-x.^2)) - (1 + exp(-x.^2)) .*
(2 .* x)) ./ (x.^2 + 3).^2;

% Valori di x su [0,1]

x_vals = linspace(0, 1, 1000);

% Valori del modulo della derivata

g1_deriv_vals = abs(g1_deriv(x_vals));

% Controllo se sempre < 1

if all(g1_deriv_vals < 1)

disp('La derivata è sempre minore di 1 su [0,1]');

else
```

```
disp('Attenzione: la derivata NON è sempre < 1 su [0,1]');
end
% Valore massimo
fprintf('Valore massimo di |g_1''(x)| su [0,1]: %.6f\n', max(g1_deriv_vals));
% Grafico
figure;
plot(x_vals, g1_deriv_vals, 'b', 'LineWidth', 2); hold on;
yline(1, '--r', 'LineWidth', 2);
xlabel('x'); ylabel('|g''_1(x)|');
title('Modulo della derivata di g_1(x)');
legend('|g_1''(x)|', 'y = 1');
grid on;
```

Calcolo di $g_2'(x)$

La funzione è:

$$g_2(x) = \cos(x)$$

e derivando otteniamo:

$$g_2'(x) = -\sin(x)$$

Poiché $|\sin(x)| \leq \sin\left(\frac{\pi}{3}\right) \approx 0.866$ su $[0, \pi/3]$, abbiamo:

$$|g_2'(x)| < 1 \quad \text{per ogni} \quad x \in [0, \pi/3]$$

Conclusione

In entrambi i casi:

- $g(x)$ è continua e derivabile,
- $|g'(x)| < 1$ nell'intervallo specificato,

quindi, per il **Teorema del Punto Fisso di Banach**, l'equazione $x = g(x)$ ha **esattamente una soluzione** α in ciascun intervallo.

$\boxed{\text{Soluzione unica garantita!}}$

In parole semplici:

Per dimostrare l'unicità, abbiamo controllato che la derivata di $g(x)$ è:

- Continua $(g \in C^1)$,
- Stretta in valore assoluto sotto 1 $(|g'(x)| < 1)$,

questo implica che g è una **contrazione**. Una funzione contrattiva ha sempre **un solo punto fisso**, che è la soluzione dell'equazione $x = g(x)$.

Punto (c) - Costruzione della tabella

Si crei una tabella per i valori di $\epsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$.
Preso un valore x_0 casuale all'interno dell'intervallo (Esempio: 0.5 per entrambi i casi)

```
% Funzioni
g1 = @(x) (1 + exp(-x.^2)) ./ (x.^2 + 3);
g2 = @(x) cos(x);

% Intervalli
a1 = 0; b1 = 1;
a2 = 0; b2 = pi/3;

% Punto di innesco
x0_1 = 0.5;
x0_2 = 0.5;

% Valori di epsilon
epsilons = 10.^(-(1:10));
Nmax = 1000; % per sicurezza

% Tabelle vuote
results1 = [];
results2 = [];

for i = 1:length(epsilons)
    eps = epsilons(i);

    % Primo problema (g1)
    [alpha1, K1, err1] = fixed_point_iteration(g1, eps, x0_1, Nmax);
```

```
    results1 = [results1; eps, x0_1, K1, abs(alpha1 - g1(alpha1))];

    % Secondo problema (g2)
    [alpha2, K2, err2] = fixed_point_iteration(g2, eps, x0_2, Nmax);
    results2 = [results2; eps, x0_2, K2, abs(alpha2 - g2(alpha2))];
end

% Visualizzazione tabella
disp('Tabella per g1(x)');
disp('epsilon      x0       K       |alpha - g(alpha)|');
disp(results1);

disp('Tabella per g2(x)');
disp('epsilon      x0       K       |alpha - g(alpha)|');
disp(results2);
```

Risultati per $g_1(x)$

ϵ	α_e	$NumIterazioni$	$ErroreModulo$
0.1	0.54732	1	$1.9635 \cdot 10^{-02}$
0.01	0.53591	3	$3.4297 \cdot 10^{-03}$
0.001	0.53331	6	$2.4999 \cdot 10^{-04}$
0.0001	0.5335	9	$1.8216 \cdot 10^{-05}$
$1e-05$	0.53349	11	$3.1778 \cdot 10^{-06}$
$1e-06$	0.53349	14	$2.3156 \cdot 10^{-07}$
$1e-07$	0.53349	16	$4.0397 \cdot 10^{-08}$
$1e-08$	0.53349	19	$2.9437 \cdot 10^{-09}$
$1e-09$	0.53349	22	$2.145 \cdot 10^{-10}$
$1e-10$	0.53349	24	$3.7421 \cdot 10^{-11}$

Risultati per $g_2(x)$

ϵ	α_e	$NumIterazioni$	$ErroreModulo$
0.1	0.69859	4	$6.7167 \cdot 10^{-02}$
0.01	0.73535	10	$6.2432 \cdot 10^{-03}$
0.001	0.73874	16	$5.8305 \cdot 10^{-04}$
0.0001	0.73905	22	$5.4469 \cdot 10^{-05}$

ϵ	α_e	$NumIterazioni$	$ErroreModulo$
$1e-05$	0.73908	28	$5.0887 \cdot 10^{-06}$
$1e-06$	0.73908	34	$4.7541 \cdot 10^{-07}$
$1e-07$	0.73909	39	$6.5935 \cdot 10^{-08}$
$1e-08$	0.73909	45	$6.1599 \cdot 10^{-09}$
$1e-09$	0.73909	51	$5.7549 \cdot 10^{-10}$
$1e-10$	0.73909	57	$5.3764 \cdot 10^{-11}$

Codice Finale

```
function matlab()
    % Risoluzione del Problema 2.6 completo

    % Definizione delle funzioni g1 e g2
    g1 = @(x) (1 + exp(-x.^2)) ./ (x.^2 + 3);
    g2 = @(x) cos(x);

    % Intervalli
    intervallo1 = [0, 1];
    intervallo2 = [0, pi/3];

    % Valori di epsilon da considerare
    epsilons = 10.^(-1:-1:-10);

    % Punto di innesco iniziale
    x0_1 = 0.5; % Puoi scegliere un valore qualsiasi in [0,1]
    x0_2 = pi/6; % Un valore qualsiasi in [0, pi/3]

    % Numero massimo di iterazioni
    Nmax = 1000;

    % Grafici
    figure;
    fplot(g1, intervallo1, 'r', 'LineWidth', 1.5);
    hold on;
    fplot(@(x) x, intervallo1, 'b--', 'LineWidth', 1.5);
    title('Grafico g1(x) e y=x');
    legend('g1(x)', 'y=x');
    grid on;

    figure;
    fplot(g2, intervallo2, 'r', 'LineWidth', 1.5);
```

```
    hold on;
    fplot(@(x) x, intervallo2, 'b--', 'LineWidth', 1.5);
    title('Grafico g2(x) e y=x');
    legend('g2(x)', 'y=x');
    grid on;

    % Tabelle risultati
    disp('--- Risultati per g1(x) ---');
    tabella1 = calcolaTabella(g1, x0_1, epsilons, Nmax);
    disp(tabella1);

    disp('--- Risultati per g2(x) ---');
    tabella2 = calcolaTabella(g2, x0_2, epsilons, Nmax);
    disp(tabella2);
end

function risultati = calcolaTabella(g, x0, epsilons, Nmax)
    % Calcolo approssimazioni alpha_e per diverse soglie epsilon

    n = length(epsilons);
    risultati = table('Size',[n 4],'VariableTypes',
{'double','double','double','double'}, ...
                      'VariableNames',
{'Epsilon','Alpha_e','NumIterazioni','ErroreModulo'});

    for i = 1:n
        epsilon = epsilons(i);
        [alpha_e, K, errore] = punto_fisso(g, x0, epsilon, Nmax);
        risultati.Epsilon(i) = epsilon;
        risultati.Alpha_e(i) = alpha_e;
        risultati.NumIterazioni(i) = K;
        risultati.ErroreModulo(i) = errore;
    end
end

function [xk, k, errore] = punto_fisso(g, x0, epsilon, Nmax)
    % Metodo di iterazione funzionale (punto fisso)

    xk = x0;
    for k = 1:Nmax
        xk1 = g(xk);
        if abs(xk1 - xk) <= epsilon
            xk = xk1;
            errore = abs(xk - g(xk));
            return
        end
```

```
        xk = xk1;
    end
    % Se non si soddisfa la condizione entro Nmax iterazioni
    errore = abs(xk - g(xk));
end
```

Spiegazione Codice Finale

- **problema()** é il programma principale: definisce le funzioni, traccia i grafici, chiama il calcolo tabellare.
- **calcolaTabella()** cicla sui valori di ϵ e salva tutti i risultati in una tabella MATLAB.
- **puntoFisso()** implementa esattamente il metodo di iterazione funzionale.
- Se il metodo converge in meno di **Nmax** iterazioni, restituisce il valore trovato; se no, dà comunque l'ultima iterazione.

Esercizi

Esercizio 1.1

```
function [V] = ValPol(X, Y, T)
    % input:
    % X = vettore a componenti reali tutti distinti
    % Y = vettore a componenti reali della stessa lunghezza di X
    % T = vettore contenente i punti in cui verrà calcolato p(x)
    %
    % output:
    % V = vettore che contiene le valutazioni nei punti del vettore T del
    %     polinomio p(x) interpolante i valori Y sui nodi X.

    Z =zeros(length(Y));
    %Z=tabelle delle differenze divise
    for i=1:length(Y)
        Z(i,1)=Y(i);
    end

    C =CNewton(X,Y);
    %C=vettore dei coefficienti di Newton

    for k=1:length(T)
        V(k)=RH(T(k),C,X);
    end
```

```
    %V=vettore dei polinomi calcolati con t

    function [f]=CNewton(X,Y)
        Z =zeros(length(Y));
         %Z=tabelle delle differenze divise
        for i=1:length(Y)
            Z(i,1)=Y(i);
        end
       % f(1)=Z(1,1);
        for j=2:length(Y)
            for i=j:length(Y)
                Z(i,j)=(Z(i,j-1)-Z(j-1,j-1))/(X(i)-X(j-1));
             %   if i==j
              %      f(j)=Z(i,j);
               % end
            end
        end
        f = diag(Z);
    end


    function [g]=RH(t,C,X)
        g = 0;
        n = length(C);
        for i = n:-1:1
            g = g * (t - X(i)) + C(i);
        end
    end
end
```

Esercizio 1.2

```
function [app] = Trapezi(a,b,n,f)
    % input:
    % a = estremo sinistro dell'intervallo
    % b = estremo destro dell'intervallo
    % n = numero naturale >=1
    % f = funzione integrabile su [a,b]
    %
    % output:
    % app = approssimazione dell'integrale su [a,b] della
    %    funzione f ottenuta mediante la formula dei trapezi
    %    di ordine n
    r=0;
    h=(b-a)/n;
```

```
    for j=1:(n-1)
        r=r+f(a+j*h);
    end
    app=((f(a)+f(b))/2 + r)*h;
end
```

Esercizio 1.3

```
function [Sn] = CavaSimp(a,b,f,n)
    % input:
    % a = estremo sinistro dell'intervallo
    % b = estremo destro dell'intervallo
    % f = funzione integrabile su [a,b]
    % n = numero naturale >=1
    %
    % output:
    % Sn = approssimazione dell'integrale su [a,b] della
    %      funzione f ottenuta mediante la formula di
    %      Cavalieri-Simpson di ordine n

    h=(b-a)/n;
    %s1 rappresenta la prima sommatoria della formula
    s1=0;
    for j=1:(n-1)
        xj=a+(j*h);
        s1=s1+f(xj);
    end

    %s2 rappresenta la seconda sommatoria della formula
    s2=0;
    for j=0:(n-1)
        x0=a+(j*h);
        x1=a+((j+1)*h);
        x2=(x0+x1)/2;
        s2=s2+f(x2);
    end

    %Sn rappresenta la formula di Cavalieri-Simpson di ordine n
    Sn=(h/6)*(f(a)+f(b)+2*s1+4*s2);
end
```

Esercizio 1.4

```
function [x, K, res_norm] = metodo_SOR(A, b, omega, epsilon, x0, Nmax)
    % Metodo SOR per risolvere il sistema lineare Ax = b
    %
    % INPUT:
    % - A: matrice del sistema (n x n)
    % - b: vettore termine noto (n x 1)
    % - omega: parametro di rilassamento (omega ≠ 0)
    % - epsilon: soglia di precisione per il residuo
    % - x0: vettore di innesco iniziale (n x 1)
    % - Nmax: numero massimo di iterazioni
    %
    % OUTPUT:
    % - x: approssimazione della soluzione
    % - K: numero di iterazioni effettuate
    % - res_norm: norma 2 del residuo all'ultima iterazione

    n = length(b);
    x = x0;

    % Decomposizione della matrice
    D = diag(diag(A));       % Parte diagonale
    E = -tril(A, -1);         % Parte inferiore (senza diagonale)
    F = -triu(A, 1);          % Parte superiore (senza diagonale)

    % Matrice del metodo
    M = (1/omega)*D - E;
    N = ((1-omega)/omega)*D + F;

    % Iterazione
    for K = 1:Nmax
        x_new = M \\ (N*x + b);    % \\ = risoluzione sistema lineare

        % Residuo
        r = b - A*x_new;
        res_norm = norm(r, 2);

        % Controllo condizione di arresto
        if res_norm <= epsilon
            x = x_new;
            return
        end

        % Aggiorna iterato
        x = x_new;
    end
```

```
    % Se non converge entro Nmax iterazioni
    r = b - A*x;
    res_norm = norm(r, 2);
end
```

Esercizio 1.5

```
function [xi, K, fx] = bisezione(a, b, f, epsilon)
%BISEZIONE Trova un'approssimazione di una radice della funzione f
nell'intervallo [a, b]
%[xi, K, fx] = BISEZIONE(a, b, f, epsilon) applica il metodo di bisezione per
trovare
%un'approssimazione xi della radice della funzione f nell'intervallo [a, b],
con precisione epsilon. La funzione restituisce:
%- xi: approssimazione della radice
%- K: numero di iterazioni eseguite
%- fx: valore della funzione calcolato in xi
%Richiede che f(a) e f(b) abbiano segni opposti (ovvero che la radice sia
garantita nell'intervallo).

% Verifica che f(a) e f(b) abbiano segno opposto
    % Verifica che f(a) e f(b) abbiano segno opposto
    if f(a) * f(b) > 0
        error('f(a) e f(b) devono avere segni opposti');
    end
    % Inizializzazione degli estremi dell'intervallo e contatore delle
iterazioni
    alpha_k = a;
    beta_k = b;
    K = 0;
    % Ripeti finché la lunghezza dell'intervallo è maggiore della precisione
richiesta
    while (beta_k - alpha_k) / 2 > epsilon
        % Calcola il punto medio dell'intervallo
        xi = (alpha_k + beta_k) / 2;
        % Aggiorna gli estremi dell'intervallo in base al segno di f(xi)
        if f(alpha_k) * f(xi) <= 0
            beta_k = xi;
        else
            alpha_k = xi;
        end

        % Incrementa il contatore delle iterazioni
        K = K+1
    end
```

```
  % Calcola l'approssimazione finale di xi come punto medio dell'ultimo
intervallo
    xi = (alpha_k + beta_k) / 2;
    fx = f(xi); % Calcola il valore di f in xi
end
```

Esercizio 1.6

```
function [xK, K, err] = fixed_point_iteration(g, epsilon, x0, Nmax)
    % fixed_point_iteration risolve l'equazione x = g(x) usando il metodo di
iterazione funzionale
    % Input:
    %   g       - funzione handle per g(x)
    %   epsilon - soglia di precisione per il criterio di arresto
    %   x0      - punto di innesco
    %   Nmax    - numero massimo di iterazioni
    % Output:
    %   xK      - ultima approssimazione calcolata
    %   K       - numero di iterazioni effettuate
    %   err     - errore finale |xK - g(xK)|

    x_prev = x0; % primo valore iniziale

    for K = 1:Nmax
        xK = g(x_prev); % calcola il nuovo valore
        if abs(xK - x_prev) <= epsilon
            err = abs(xK - g(xK));
            return
        end
        x_prev = xK; % aggiorna per l'iterazione successiva
    end

    % Se raggiungo Nmax iterazioni senza soddisfare la condizione
    err = abs(xK - g(xK));

end
```