

Report Crediti D Salvucci-Noce

1. [Esercizi](#)

1. [Esercizio 1](#)

1. [Codice Esercizio 1](#)
2. [Spiegazione](#)

2. [Esercizio 2](#)

1. [Codice Esercizio 2](#)
2. [Spiegazione](#)

3. [Esercizio 3](#)

1. [Codice Esercizio 3](#)
 1. [Spiegazione del codice](#)
 2. [Uso delle altre funzioni](#)

4. [Esercizio 4](#)

1. [Codice Esercizio 4](#)
2. [Spiegazione del codice](#)

5. [Esercizio 5](#)

1. [Codice Esercizio 5](#)
2. [Spiegazione Codice](#)

6. [Esercizio 6](#)

1. [Codice](#)
2. [Spiegazione](#)

2. [Problemi](#)

1. [Problema 1](#)

1. [Soluzione](#)
2. [Codice](#)

2. [Problema 2](#)

1. [Soluzione](#)
2. [Codice](#)
3. [Codice v2](#)

3. [Problema 3](#)

1. [Soluzione](#)
2. [Codice](#)

4. [Problema 4](#)

1. [Soluzione](#)
2. [Codice](#)

5. [Problema 5](#)

1. [Soluzione](#)

2. [Codice](#)6. [Problema 6](#)1. [Soluzione](#)1. [Caso 1](#)2. [Caso 2](#)2. [Codice](#)1. [Descrizione del Codice](#)

Esercizi

Esercizio 1

Il primo esercizio chiede di scrivere in MATLAB una function che calcoli l'algoritmo di **Ruffini-Horner** per la valutazione del polinomio d'interpolazione in un punto:

Esercizio d'implementazione dell'algoritmo di valutazione del polinomio d'interpolazione in più punti.

Codice Esercizio 1

Questo codice genera un vettore di coefficienti per le differenze divise:

Esercizio 1.1

```

1  function p_t = interpola_ruffini_horner(x, y, t)
2      % Input:
3      % x: vettore dei punti x0, x1, ..., xn (devono essere distinti)
4      % y: vettore dei valori corrispondenti y0, y1, ..., yn
5      % t: vettore dei punti t1, t2, ..., tm dove si vuole valutare il
        polinomio interpolante
6
7      % Output:
8      % p_t: vettore contenente le valutazioni del polinomio interpolante
        nei punti t
9
10     % Calcola i coefficienti del polinomio usando le differenze divise
11     coeff = differenze_divise(x, y);
12
13     % Valuta il polinomio nei punti t usando lo schema di Horner
14     p_t = horner_eval(coeff, x, t);
15 end
16
17 function coeff = differenze_divise(x, y)
18     % Calcola i coefficienti delle differenze divise
19     n = length(x);
20     coeff = y; % Copia il vettore y
21

```

```

22     % Costruisce la tabella delle differenze divise
23     for j = 2:n
24         for i = n:-1:j
25             coeff(i) = (coeff(i) - coeff(i-1)) / (x(i) - x(i-j+1));
26         end
27     end
28 end
29
30 function p_t = horner_eval(coeff, x, t)
31     % Valuta il polinomio usando lo schema di Horner
32     n = length(coeff);
33     m = length(t);
34     p_t = zeros(1, m);
35
36     for k = 1:m
37         % Inizializza il polinomio con il termine di grado più alto
38         p = coeff(n);
39
40         % Applica lo schema di Horner
41         for i = n-1:-1:1
42             p = p * (t(k) - x(i)) + coeff(i);
43         end
44
45         % Salva il risultato della valutazione nel punto t(k)
46         p_t(k) = p;
47     end
48 end

```

Questo codice genera la **matrice** delle differenze divise:

```

1  function p_t = interpola_ruffini_horner(x, y, t)
2      % Input:
3      % x: vettore dei punti x0, x1, ..., xn (devono essere distinti)
4      % y: vettore dei valori corrispondenti y0, y1, ..., yn
5      % t: vettore dei punti t1, t2, ..., tm dove si vuole valutare il
        polinomio interpolante
6
7      % Output:
8      % p_t: vettore contenente le valutazioni del polinomio interpolante
        nei punti t
9
10     % Calcola la matrice delle differenze divise
11     diff_matrix = differenze_divise(x, y);
12
13     % Estrai i coefficienti dalla diagonale principale della matrice
14     coeff = diag(diff_matrix);
15
16     % Valuta il polinomio nei punti t usando lo schema di Horner
17     p_t = horner_eval(coeff, x, t);

```

```

18 end
19
20 function diff_matrix = differenze_divise(x, y)
21     % Calcola la matrice delle differenze divise
22     n = length(x);
23     diff_matrix = zeros(n, n); % Inizializza la matrice delle
    differenze divise
24
25     % Copia il vettore y nella prima colonna
26     diff_matrix(:, 1) = y(:);
27
28     % Costruisce la tabella delle differenze divise
29     for j = 2:n
30         for i = j:n
31             diff_matrix(i, j) = (diff_matrix(i, j-1) - diff_matrix(i-1,
j-1)) / (x(i) - x(i-j+1));
32         end
33     end
34 end
35
36 function p_t = horner_eval(coeff, x, t)
37     % Valuta il polinomio usando lo schema di Horner
38     n = length(coeff);
39     m = length(t);
40     p_t = zeros(1, m);
41
42     for k = 1:m
43         % Inizializza il polinomio con il termine di grado più alto
44         p = coeff(n);
45
46         % Applica lo schema di Horner
47         for i = n-1:-1:1
48             p = p * (t(k) - x(i)) + coeff(i);
49         end
50
51         % Salva il risultato della valutazione nel punto t(k)
52         p_t(k) = p;
53     end
54 end
55

```

Spiegazione

1. Funzione principale (`interpola_ruffini_horner`):

- Prende in input i vettori `x` (ascisse), `y` (ordinate) e `t` (punti in cui valutare il polinomio).
- Prima usa la funzione `differenze_divise` per calcolare i coefficienti del polinomio interpolante nella forma di Newton.

- Poi usa la funzione `horner_eval` per valutare il polinomio nei punti desiderati applicando lo schema di Horner.

2. Calcolo delle differenze divise (`differenze_divise`):

- Costruisce la tabella delle differenze divise e restituisce i coefficienti del polinomio interpolante.
- L'algoritmo funziona partendo dai valori `y` e iterando per costruire le differenze successive.

3. Valutazione con lo schema di Horner (`horner_eval`):

- Prende i coefficienti del polinomio e valuta il polinomio in ciascun punto di `t` usando lo schema di Horner.
- Questo schema permette di valutare il polinomio in modo molto efficiente, riducendo il numero di operazioni necessarie.

Esercizio 2

Il secondo esercizio chiede di scrivere una function MATLAB per implementare la **formula dei trapezi** di una data funzione presa in input:

Esercizio d'implementazione della formula dei trapezi

Codice Esercizio 2

Esercizio 1.2

```

1  function In = formula_trapezi(f, a, b, n)
2      % Input:
3      % f: funzione da integrare (definita su [a,b])
4      % a: estremo sinistro dell'intervallo
5      % b: estremo destro dell'intervallo
6      % n: numero di sottointervalli (n >= 1)
7
8      % Output:
9      % In: approssimazione dell'integrale di f(x) su [a, b] usando la
      formula dei trapezi
10
11     % Larghezza di ciascun sottointervallo
12     h = (b - a) / n;
13
14     % Calcolo della somma dei valori intermedi
15     somma = 0;
16     for i = 1:n-1
17         xi = a + i*h;
18         somma = somma + f(xi);
19     end
20
21     % Formula dei trapezi
22     In = h*((f(a)+f(b))/2+somma)

```

Spiegazione

Funzione `formula_trapezi` :

- Prende in input la funzione da integrare `f` , gli estremi dell'intervallo `[a, b]` , e il numero di sottointervalli `n` .
- Calcola la larghezza di ciascun sottointervallo come $h = \frac{b-a}{n}$
- Usa la **formula dei trapezi** per calcolare un'approssimazione dell'integrale:

$$I_n = h \left[\frac{f(a) + f(b)}{2} + \sum_{j=1}^{n-1} f(x_j) \right]$$

- Restituisce l'approssimazione I_n della funzione $f(x)$ passata in input usando la formula dei trapezi.

Esercizio 3

Il terzo esercizio chiede di scrivere una function MATLAB per implementare il **metodo di estrapolazione** di una data funzione presa in input. Chiede inoltre di usare le function MATLAB usate per risolvere gli esercizi 1 e 2

Esercizio d'implementazione del metodo di estrapolazione

Codice Esercizio 3

Esercizio 1.3

```

1  function p0 = estrapol(f, a, b, n_vect)
2      % Input:
3      % f: funzione da integrare
4      % a: estremo sinistro dell'intervallo
5      % b: estremo destro dell'intervallo
6      % n_vect: vettore dei valori di n0, n1, ..., nm
7
8      % Output:
9      % p0: valore estrapolato p(0)
10
11     % Prealloca i vettori per h^2 e In
12     m = length(n_vect);
13     h_squared = zeros(1, m);
14     In_values = zeros(1, m);
15
16     % Calcola h^2 e In per ogni n in n_vect
17     for i = 1:m
18         n = n_vect(i);
19         h = (b - a) / n; % Passo di discretizzazione

```

```

20         h_squared(i) = h^2;
21         In_values(i) = formula_trapezi(f, a, b, n);
22     end
23
24     % Interpola i valori (h^2, In) usando le differenze divise
25     % La funzione interpola_ruffini_horner accetta vettori di x (qui
26     % h^2) e y (qui In_values)
27     % t=0 perché vogliamo estrapolare p(0)
28     p0 = interpola_ruffini_horner(h_squared, In_values, 0);
29
30     % Se viene specificato il numero di cifre, usa vpa per ottenere
31     % precisione
32     if nargin > 4
33         p0 = vpa(p0, cifre); % Calcola p0 con precisione
34         % specificata
35     end
36 end

```

Spiegazione del codice

1. Input:

- `f` : la funzione da integrare.
- `a` e `b` : gli estremi dell'intervallo su cui si calcola l'integrale.
- `n_vect` : un vettore di valori n_0, n_1, \dots, n_m usati per il calcolo degli integrali.

2. Output:

- `p0` : il valore estrapolato $p(0)$, dove $p(x)$ è il polinomio interpolante ottenuto dai valori di h^2 e I_n .

3. Calcolo di h^2 e I_n :

- Per ogni n_i nel vettore `n_vect`, il programma calcola il passo h e il corrispondente integrale approssimato I_n utilizzando la formula dei trapezi fornita nell'Esercizio 2.

4. Interpolazione:

- I valori h^2 e I_n vengono usati per ottenere il polinomio interpolante con la funzione di interpolazione `interpola_ruffini_horner`, che è la soluzione all'Esercizio 1.11.

5. Estrapolazione:

- Il programma valuta il polinomio interpolante nel punto $t = 0$ per ottenere $p(0)$.

Uso delle altre funzioni

- `interpola_ruffini_horner`, `differenze_divise` e `horner_eval` provengono dall'Esercizio 1.1.
- `formula_trapezi` viene dall'Esercizio 1.2 per approssimare gli integrali usando la formula dei trapezi.

- `vpa(p0, cifre)` viene usato per approssimare correttamente il risultato con il numero di cifre passate in input. Questa è una funzione del Toolbox **Symbolic Math Toolbox**

Esercizio 4

L'esercizio chiede di creare una function MATLAB per implementare il **metodo di Jacobi**.

Esercizio d'implementazione del metodo di Jacobi

Codice Esercizio 4

Questo codice implementa il metodo di Jacobi componente per componente:

Esercizio 1.4

```

1  function [x, K, r_norm] = jacobi_method(A, b, x0, epsilon, N_max)
2      % Input:
3      % A: matrice del sistema lineare Ax = b
4      % b: vettore dei termini noti
5      % x0: vettore di innesco (stima iniziale di x)
6      % epsilon: soglia di precisione
7      % N_max: numero massimo di iterazioni consentite
8
9      % Output:
10     % x: vettore approssimato x^(K) dopo K iterazioni o x^(N_max)
11     % K: numero di iterazioni effettivamente eseguite
12     % r_norm: norma ||r^(K)||_2 del residuo alla fine del processo
13
14     % Numero di variabili (dimensione del sistema)
15     n = length(b);
16
17     % Inizializza la variabile per il vettore x^(K) (soluzione corrente)
18     x = x0;
19
20     % Itera il metodo di Jacobi
21     for K = 1:N_max
22         % Prealloca il vettore x^(K+1)
23         x_new = zeros(n, 1);
24
25         % Calcola ogni componente di x^(K+1)
26         for i = 1:n
27             % Somma degli elementi a sinistra di x^(K)
28             sum1 = A(i, 1:i-1) * x(1:i-1);
29             % Somma degli elementi a destra di x^(K)
30             sum2 = A(i, i+1:n) * x(i+1:n);
31             % Formula del metodo di Jacobi
32             x_new(i) = (b(i) - sum1 - sum2) / A(i, i);
33         end
34

```



```

35      % Calcola il residuo  $r^{(K)} = b - A * x^{(K)}$ 
36       $r = b - A * x_{new};$ 
37
38      % Calcola la norma del residuo  $||r^{(K)}||_2$ 
39       $r\_norm = norm(r, 2);$ 
40
41      % Condizione di arresto: se  $||r^{(K)}||_2 \leq \epsilon * ||b||_2$ 
42      if  $r\_norm \leq \epsilon * norm(b, 2)$ 
43           $x = x_{new};$ 
44          return; % Arresta l'algoritmo e restituisce il risultato
45      end
46
47      % Aggiorna la soluzione corrente  $x^{(K)}$  con  $x^{(K+1)}$ 
48       $x = x_{new};$ 
49  end
50
51      % Se si raggiunge N_max iterazioni senza soddisfare il criterio, si
restituisce
52      %  $x^{(N\_max)}$ , il relativo indice N_max e la norma del residuo
 $||r^{(N\_max)}||_2$ 
53  end

```

Questo codice implementa il metodo di Jacobi con il metodo iterativo:

```

1  function [x, K, r_norm] = jacobiIterativo(A, b, x0, epsilon, N_max)
2      % Metodo di Jacobi - Versione Iterativa
3      % Input:
4      % A: matrice del sistema lineare  $Ax = b$ 
5      % b: vettore dei termini noti
6      % x0: vettore di innesco (stima iniziale di x)
7      % epsilon: soglia di precisione
8      % N_max: numero massimo di iterazioni consentite
9
10     % Output:
11     % x: vettore approssimato  $x^{(K)}$  dopo K iterazioni o  $x^{(N\_max)}$ 
12     % K: numero di iterazioni effettivamente eseguite
13     % r_norm: norma  $||r^{(K)}||_2$  del residuo alla fine del processo
14
15     % Separazione di D, L e U dalla matrice A
16      $D = diag(diag(A));$  % Matrice diagonale
17      $L = tril(A, -1);$  % Parte triangolare inferiore
18      $U = triu(A, 1);$  % Parte triangolare superiore
19
20     % Pre-calcolo della matrice iterativa  $M = D^{(-1)} * (L + U)$ 
21      $D\_inv = inv(D);$  % Inversa della diagonale
22      $M = -D\_inv * (L + U);$  % Matrice di iterazione
23
24     % Pre-calcolo del termine costante  $c = D^{(-1)} * b$ 
25      $c = D\_inv * b;$ 

```

```

26
27     % Inizializza il vettore soluzione con la stima iniziale
28     x = x0;
29
30     % Itera il metodo di Jacobi
31     for K = 1:N_max
32         % Aggiornamento vettoriale:  $x^{(k+1)} = M * x^{(k)} + c$ 
33         x_new = M * x + c;
34
35         % Calcola il residuo  $r^{(K)} = b - A * x^{(K)}$ 
36         r = b - A * x_new;
37
38         % Calcola la norma del residuo  $\|r^{(K)}\|_2$ 
39         r_norm = norm(r, 2);
40
41         % Condizione di arresto:  $\|r^{(K)}\|_2 \leq \text{epsilon} * \|b\|_2$ 
42         if r_norm <= epsilon * norm(b, 2)
43             x = x_new;
44             return; % Arresta l'algoritmo e restituisce il risultato
45         end
46
47         % Aggiorna la soluzione corrente  $x^{(K)}$ 
48         x = x_new;
49     end
50
51     % Se si raggiunge N_max iterazioni senza soddisfare il criterio, si
    restituisce
52     %  $x^{(N\_max)}$ , il relativo indice N_max e la norma del residuo
     $\|r^{(N\_max)}\|_2$ 
53 end
54

```

Spiegazione del codice

1. Input:

- **A** : La matrice del sistema lineare.
- **b** : Il vettore dei termini noti.
- **x0** : Il vettore di innesco (cioè la stima iniziale di x).
- **epsilon** : La soglia di precisione per il residuo.
- **N_max** : Il numero massimo di iterazioni consentite.

2. Output:

- **x** : Il vettore soluzione $x^{(K)}$, dove K è il numero di iterazioni.
- **K** : Il numero di iterazioni effettivamente eseguite.
- **r_norm** : La norma $\|r^{(K)}\|_2$ del residuo $r^{(K)} = b - A \cdot x^{(K)}$.

3. Procedura:

- Il metodo di Jacobi viene applicato iterativamente fino a quando il residuo $\|r^{(K)}\|_2$ diventa minore o uguale a $\varepsilon \cdot \|b\|_2$, oppure si raggiunge il numero massimo di iterazioni N_{\max} .
- Se nessuna delle iterazioni soddisfa la condizione di arresto, il programma restituisce $x^{(N_{\max})}$.

Esercizio 5

L'esercizio chiede di creare una function MATLAB per implementare il **metodo di Gauss-Seidel**.

Esercizio d'implementazione del metodo di Gauss-Seidel

Codice Esercizio 5

Questo è il codice di Gauss-Seidel componente per componente

Esercizio 1.5

```

1  function [x, K, r_norm] = metodo_gauss_seidel(A, b, x0, epsilon, N_max)
2      % Input:
3      % A: matrice del sistema lineare Ax = b
4      % b: vettore dei termini noti
5      % x0: vettore di innesco (stima iniziale di x)
6      % epsilon: soglia di precisione
7      % N_max: numero massimo di iterazioni consentite
8
9      % Output:
10     % x: vettore approssimato x^(K) dopo K iterazioni o x^(N_max)
11     % K: numero di iterazioni effettivamente eseguite
12     % r_norm: norma ||r^(K)||_2 del residuo alla fine del processo
13
14     % Numero di variabili (dimensione del sistema)
15     n = length(b);
16
17     % Inizializza la soluzione corrente con il vettore di innesco x0
18     x = x0;
19
20     % Itera il metodo di Gauss-Seidel
21     for K = 1:N_max
22         % Memorizza la soluzione precedente x^(K-1)
23         x_old = x;
24
25         % Calcola ogni componente di x^(K)
26         for i = 1:n
27             % Somma degli elementi a sinistra di x^(K)
28             sum1 = A(i, 1:i-1) * x(1:i-1);
29             % Somma degli elementi a destra di x^(K-1)

```

```

30         sum2 = A(i, i+1:n) * x_old(i+1:n);
31         % Formula del metodo di Gauss-Seidel
32         x(i) = (b(i) - sum1 - sum2) / A(i, i);
33     end
34
35     % Calcola il residuo  $r^{(K)} = b - A * x^{(K)}$ 
36     r = b - A * x;
37
38     % Calcola la norma del residuo  $\|r^{(K)}\|_2$ 
39     r_norm = norm(r, 2);
40
41     % Condizione di arresto: se  $\|r^{(K)}\|_2 \leq \epsilon * \|b\|_2$ 
42     if r_norm <= epsilon * norm(b, 2)
43         return; % Arresta l'algoritmo e restituisce il risultato
44     end
45 end
46
47 % Se si raggiunge N_max iterazioni senza soddisfare il criterio, si
restituisce
48 %  $x^{(N\_max)}$ , il relativo indice N_max e la norma del residuo
 $\|r^{(N\_max)}\|_2$ 
49 end

```

Questo è il metodo di Gauss-Seidel iterativo

```

1  function [x, K, r_norm] = gauss_seidelIterativo(A, b, x0, epsilon,
N_max)
2      % Metodo di Gauss-Seidel - versione Iterativa
3      % Input:
4      % A: matrice del sistema lineare  $Ax = b$ 
5      % b: vettore dei termini noti
6      % x0: vettore di innesco (stima iniziale di x)
7      % epsilon: soglia di precisione
8      % N_max: numero massimo di iterazioni consentite
9
10     % Output:
11     % x: vettore approssimato  $x^{(K)}$  dopo K iterazioni o  $x^{(N\_max)}$ 
12     % K: numero di iterazioni effettivamente eseguite
13     % r_norm: norma  $\|r^{(K)}\|_2$  del residuo alla fine del processo
14
15     % Separazione della matrice A in E (triangolare inferiore) e U
(triangolare superiore)
16     E = tril(A); % Parte triangolare inferiore (inclusa
diagonale)
17     U = triu(A, 1); % Parte triangolare superiore (esclusa
diagonale)
18
19     % Pre-calcolo della matrice iterativa  $G = E^{(-1)} * U$ 
20     G = -E \ U; %  $G = -\text{inv}(E) * U$ 

```

```

21
22 % Pre-calcolo del termine costante  $c = E^{-1} * b$ 
23  $c = E \setminus b;$  %  $c = \text{inv}(E) * b$ 
24
25 % Inizializza la soluzione corrente con il vettore di innesco  $x_0$ 
26  $x = x_0;$ 
27
28 % Itera il metodo di Gauss-Seidel
29 for  $K = 1:N\_max$ 
30 % Aggiornamento vettoriale:  $x^{(k+1)} = G * x^{(k)} + c$ 
31  $x\_new = G * x + c;$ 
32
33 % Calcola il residuo  $r^{(K)} = b - A * x^{(K)}$ 
34  $r = b - A * x\_new;$ 
35
36 % Calcola la norma del residuo  $\|r^{(K)}\|_2$ 
37  $r\_norm = \text{norm}(r, 2);$ 
38
39 % Condizione di arresto:  $\|r^{(K)}\|_2 \leq \text{epsilon} * \|b\|_2$ 
40 if  $r\_norm \leq \text{epsilon} * \text{norm}(b, 2)$ 
41  $x = x\_new;$ 
42 return; % Arresta l'algoritmo e restituisce il risultato
43 end
44
45 % Aggiorna la soluzione corrente  $x^{(K)}$ 
46  $x = x\_new;$ 
47 end
48
49 % Se si raggiunge  $N\_max$  iterazioni senza soddisfare il criterio, si
restituisce
50 %  $x^{(N\_max)}$ , il relativo indice  $N\_max$  e la norma del residuo
 $\|r^{(N\_max)}\|_2$ 
51 end

```

Spiegazione Codice

1. Input:

- A : La matrice del sistema lineare.
- b : Il vettore dei termini noti.
- x_0 : Il vettore di innesco (cioè la stima iniziale di x).
- epsilon : La soglia di precisione per il residuo.
- N_max : Il numero massimo di iterazioni consentite.

2. Output:

- x : Il vettore soluzione $x^{(K)}$, dove K è il numero di iterazioni.
- K : Il numero di iterazioni effettivamente eseguite.
- r_norm : La norma $\|r^{(K)}\|_2$ del residuo $r^{(K)} = b - A \cdot x^{(K)}$.

3. Procedura:

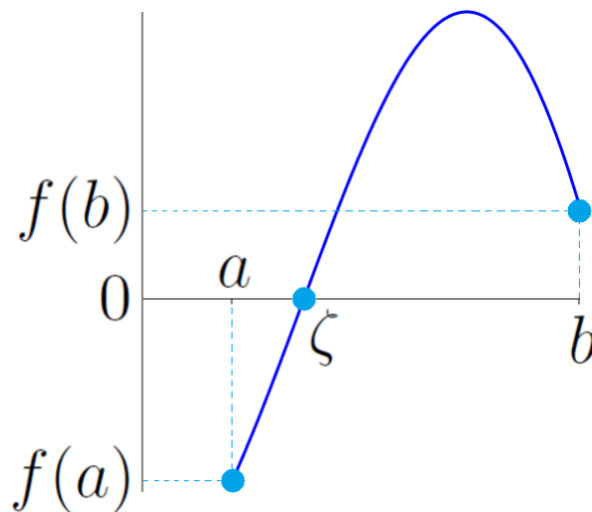
- Il metodo di Gauss-Seidel iterativo aggiorna ogni componente del vettore $x^{(K)}$ tenendo conto dei valori già aggiornati di x_i , a differenza del metodo di Jacobi, dove si usano solo i valori dell'iterazione precedente.
- L'arresto del processo avviene quando la norma del residuo $\|r^{(K)}\|_2$ è inferiore o uguale a $\varepsilon \cdot \|b\|_2$ oppure si raggiunge il numero massimo di iterazioni N_{\max} .

Esercizio 6

L'esercizio 6 chiede di creare una function MATLAB che implementi il **metodo della bisezione**, ovvero il metodo che permette di trovare il punto ξ di una funzione $f(x)$ definita su intervallo $[a, b]$ tale che $f(\xi) = 0$

Sia $f : [a, b] \rightarrow \mathbb{R}$ una funzione continua su $[a, b]$ tale che $f(a)$ e $f(b)$ hanno segno opposto : $f(a)f(b) < 0$. Un teorema dell'analisi matematica (teorema degli zeri) garantisce che la funzione $f(x)$ ha almeno uno zero nell'intervallo (a, b) , cioè esiste un punto $\zeta \in (a, b)$ tale che $f(\zeta) = 0$;

Figura 1.1



Una funzione continua $f : [a, b] \rightarrow \mathbb{R}$ tale che $f(a)f(b) < 0$ possiede almeno uno zero $\zeta \in (a, b)$.

Supponiamo che $f(x)$ abbia un unico zero ζ in (a, b) . Un metodo per determinare un'approssimazione ξ di ζ è il metodo di bisezione: fissata una soglia di precisione $\varepsilon > 0$, il metodo costruisce la successione di intervalli

$$[\alpha_k, \beta_k], \quad k = 0, 1, 2, \dots$$

in cui $[\alpha_0, \beta_0] = [a, b]$ e, per $k \leq 1$,

$$[\alpha_k, \beta_k] = \begin{cases} [\alpha_{k-1}, \frac{\alpha_{k-1} + \beta_{k-1}}{2}], & \text{se } \zeta \in [\alpha_{k-1}, \frac{\alpha_{k-1} + \beta_{k-1}}{2}] \text{ cioè } f(\alpha_{k-1})f(\frac{\alpha_{k-1} + \beta_{k-1}}{2}) \leq 0, \\ [\frac{\alpha_{k-1} + \beta_{k-1}}{2}, \beta_{k-1}], & \text{altrimenti.} \end{cases}$$

La successione di intervalli così costruita gode delle seguenti proprietà:

- $\zeta \in [\alpha_k, \beta_k]$ per tutti i $k \geq 0$;
- ogni intervallo è metà del precedente e dunque la lunghezza di $[\alpha_k, \beta_k]$ è $\beta_k - \alpha_k = \frac{b-a}{2^k}$ per ogni $k \geq 0$.

Il metodo si arresta al primo indice K tale che $\beta_K - \alpha_K \leq \varepsilon$ e restituisce come risultato il punto medio ξ dell'intervallo $[\alpha_K, \beta_K]$ dato da $\xi = \frac{\alpha_K + \beta_K}{2}$. In questo modo, siccome $\zeta \in [\alpha_K, \beta_K]$, si ha $|\xi - \zeta| \leq \frac{\varepsilon}{2}$.

Osserviamo che l'indice di arresto K è il più piccolo intero ≥ 0 tale che

$$\beta_k - \alpha_k \leq \varepsilon \iff \frac{b-a}{2^K} \leq \varepsilon \iff 2^K \geq \frac{b-a}{\varepsilon} \iff K \geq \log_2\left(\frac{b-a}{\varepsilon}\right),$$

cioè $K = \lceil \log_2(\frac{b-a}{\varepsilon}) \rceil$.

Scrivere un programma Matlab che implementa il metodo di bisezione. Il programma deve:

- prendere in input gli estremi a, b di un intervallo, una funzione continua $f : [a, b] \rightarrow \mathbb{R}$, con $f(a)f(b) < 0$ e con un unico zero $\zeta \in (a, b)$, e un $\varepsilon > 0$;
- restituire in output l'approssimazione ξ di ζ ottenuta con il metodo di bisezione sopra descritto, l'indice di arresto K del metodo, e il valore $f(\xi)$ (che sarà all'incirca pari a $0 = f(\zeta)$).

Codice

Esercizio 1.6

```

1  function [xi, K, fx] = bisezione(a, b, f, epsilon)
2      % Verifica che f(a) e f(b) abbiano segno opposto
3      if f(a) * f(b) > 0
4          error('f(a) e f(b) devono avere segni opposti');
5      end
6
7      % Inizializzazione degli estremi dell'intervallo e contatore delle
      iterazioni
8      alpha_k = a;
9      beta_k = b;
10     K = 0;
11
12     % Ripeti finché la lunghezza dell'intervallo è maggiore della
      precisione richiesta
13     while (beta_k - alpha_k) / 2 > epsilon
14         % Calcola il punto medio dell'intervallo
15         xi = (alpha_k + beta_k) / 2;
16
17         % Aggiorna gli estremi dell'intervallo in base al segno di f(xi)
18         if f(alpha_k) * f(xi) <= 0
19             beta_k = xi;
20         else
21             alpha_k = xi;

```

```

22         end
23
24         % Incrementa il contatore delle iterazioni
25         K = K + 1;
26     end
27
28     % Calcola l'approssimazione finale di xi come punto medio
    dell'ultimo intervallo
29     xi = (alpha_k + beta_k) / 2;
30     fx = f(xi); % Calcola il valore di f in xi
31 end

```

Spiegazione

- **Verifica dei segni:** la funzione controlla che $f(a)$ e $f(b)$ abbiano segno opposto, come richiesto dal teorema degli zeri.
- **Inizializzazione:** definisce $\alpha_k = a$ e $\beta_k = b$ e imposta il contatore $K = 0$
- **Iterazione del metodo di bisezione:** continua a suddividere l'intervallo finché la metà della sua lunghezza è maggiore di ε . Ad ogni iterazione:
 - Calcola il punto medio ξ .
 - Aggiorna gli estremi in base al segno di $f(\xi)$ rispetto a $f(\alpha_k)$.
 - Incrementa K .
- **Output finale:** restituisce l'approssimazione ξ , l'indice K , e $f(\xi)$.

Problemi

Problema 1

Si consideri la funzione \sqrt{x} .

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

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

Calcolare il vettore (colonna)

$$[p(\zeta_1) - \sqrt{\zeta_1} \quad p(\zeta_2) - \sqrt{\zeta_2} \quad \dots \quad p(\zeta_{21}) - \sqrt{\zeta_{21}}]^T$$

dove $\zeta_i = \frac{i-1}{20}$ per $i = 1, \dots, 21$, 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)

Con $\xi_i = \frac{i-1}{20}$, il vettore colonna $p(\xi_1) - \sqrt{\xi_1}, \dots, p(\xi_{21}) - \sqrt{\xi_{21}}$ è

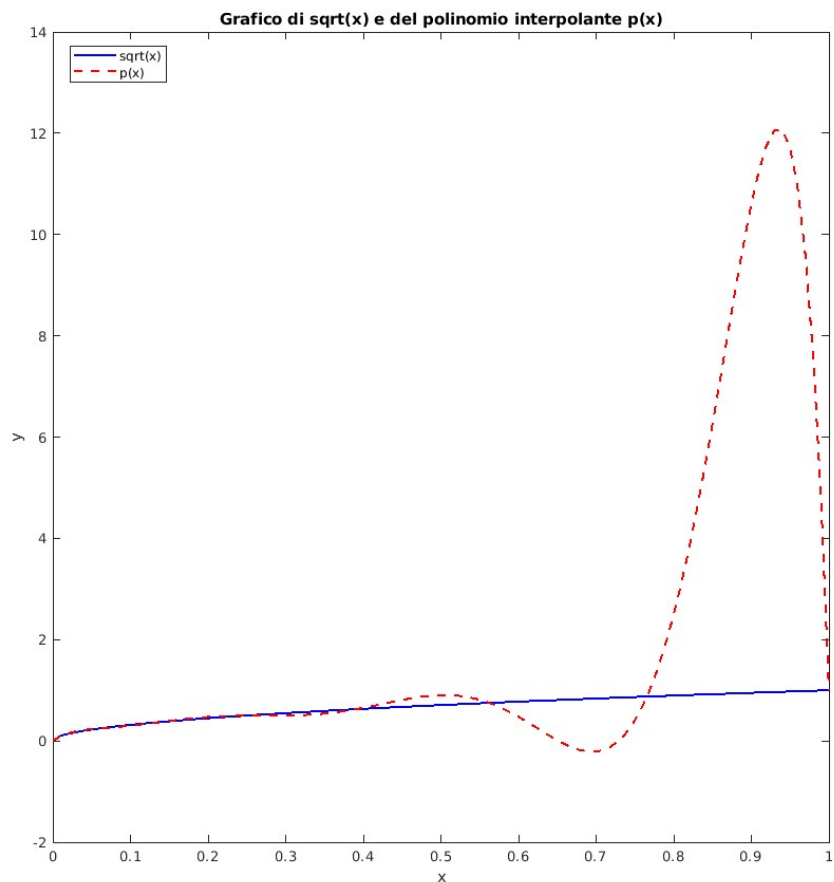
$$\begin{aligned}
 p(\xi_1) - \sqrt{\xi_1} &: 0 \\
 p(\xi_2) - \sqrt{\xi_2} &: 0.009373456935820 \\
 p(\xi_3) - \sqrt{\xi_3} &: -0.016624898598359 \\
 p(\xi_4) - \sqrt{\xi_4} &: 0.006265159516694 \\
 p(\xi_5) - \sqrt{\xi_5} &: 0.026059100541982 \\
 p(\xi_6) - \sqrt{\xi_6} &: 0.000000000000000 \\
 p(\xi_7) - \sqrt{\xi_7} &: -0.046798842893448 \\
 p(\xi_8) - \sqrt{\xi_8} &: -0.052843679514480 \\
 p(\xi_9) - \sqrt{\xi_9} &: 0.019043791981465 \\
 p(\xi_{10}) - \sqrt{\xi_{10}} &: 0.136657922266046 \\
 p(\xi_{11}) - \sqrt{\xi_{11}} &: 0.195969221000572 \\
 p(\xi_{12}) - \sqrt{\xi_{12}} &: 0.070222900207986 \\
 p(\xi_{13}) - \sqrt{\xi_{13}} &: -0.298665479678417 \\
 p(\xi_{14}) - \sqrt{\xi_{14}} &: -0.793827451939188 \\
 p(\xi_{15}) - \sqrt{\xi_{15}} &: -1.047857448417138 \\
 p(\xi_{16}) - \sqrt{\xi_{16}} &: -0.461689802877381 \\
 p(\xi_{17}) - \sqrt{\xi_{17}} &: 1.600121563949965 \\
 p(\xi_{18}) - \sqrt{\xi_{18}} &: 5.337600132745608 \\
 p(\xi_{19}) - \sqrt{\xi_{19}} &: 9.648720381277402 \\
 p(\xi_{20}) - \sqrt{\xi_{20}} &: 10.731478361986454 \\
 p(\xi_{21}) - \sqrt{\xi_{21}} &: -0.000000000000004
 \end{aligned}$$

Osservando i valori numerici, si può notare che:

- **L'errore non è costante:** La differenza $p(\xi_i) - \sqrt{\xi_i}$ assume sia valori positivi che negativi, indicando che il polinomio a volte sovrastima e a volte sottostima la funzione radice quadrata.
- **L'errore varia in modo significativo a seconda del punto:** In alcuni punti l'errore è molto piccolo (quasi nullo), mentre in altri è molto grande.

Punto (b)

Il grafico delle funzioni \sqrt{x} e $p(x)$ è il seguente



Codice

Problema2.1

```

1  % Definisci i nodi di interpolazione e i valori corrispondenti di
   sqrt(x)
2  x_nodes = [0, 1/64, 4/64, 9/64, 16/64, 25/64, 36/64, 49/64, 1];
3  y_nodes = sqrt(x_nodes);
4
5  % Definisci i punti zeta_i dove valutare il polinomio interpolante
6  i = 1:21;
7  zeta = (i-1) / 20;
8
9  % Calcola il polinomio interpolante nei punti zeta usando
   Interpola_Ruffini_Horner
10 p_zeta = interpola_ruffini_horner(x_nodes, y_nodes, zeta);
11
12 % Calcola la funzione sqrt nei punti zeta
13 sqrt_zeta = sqrt(zeta);
14
15 % Calcola il vettore delle differenze p(zeta) - sqrt(zeta)
16 diff_vector = p_zeta - sqrt_zeta;
17
18 % Visualizza il vettore delle differenze
19 disp('Vettore delle differenze p(zeta_i) - sqrt(zeta_i):');

```

```

20  disp(diff_vector. ');
21
22  % Traccia il grafico di sqrt(x) e p(x) sull'intervallo [0, 1]
23  x_plot = linspace(0, 1, 100); % Punti per il grafico
24  p_x_plot = interpola_ruffini_horner(x_nodes, y_nodes, x_plot);
25  sqrt_x_plot = sqrt(x_plot);
26
27  figure;
28  plot(x_plot, sqrt_x_plot, 'b-', 'LineWidth', 1.5); hold on;
29  plot(x_plot, p_x_plot, 'r--', 'LineWidth', 1.5);
30  legend('sqrt(x)', 'p(x)', 'Location', 'best');
31  xlabel('x');
32  ylabel('y');
33  title('Grafico di sqrt(x) e del polinomio interpolante p(x)');
34  hold off;

```

Problema 2

Si consideri la funzione

$$f(x) = e^x.$$

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

$$I = \int_0^1 f(x) dx = 1.7182818284590 \dots$$

(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 $\varepsilon \in \{10^{-1}, 10^{-2}, \dots, 10^{-10}\}$:

- il numero $n(\varepsilon)$;
- il valore I_n per $n = n(\varepsilon)$;
- il valore esatto I (per 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 di 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)

Per il teorema sull'errore o resto della formula dei trapezi, abbiamo che

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

Per determinare un $n = n(\varepsilon)$ tale che $|I - I_n| \leq \varepsilon$, calcoliamo $f''(x)$:

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

per ogni $x \in [0, 1]$ si ha che:

$$|f''(x)| = |e^x| = e^x \leq e$$

Quindi, possiamo scrivere

$$\left| \int_0^1 e^x dx - I_n \right| \leq \frac{e}{12n^2}$$

E infine

$$\frac{e}{12n^2} \leq \varepsilon \iff n \geq \sqrt{\frac{e}{12\varepsilon}}$$

Dunque prenderemo

$$n = n(\varepsilon) = \left\lceil \sqrt{\frac{e}{12\varepsilon}} \right\rceil$$

Punto (b)

Tabella per ogni $\varepsilon \in \{10^{-1}, 10^{-2}, \dots, 10^{-10}\}$

ϵ	n	I_n	Error
1.0×10^{-1}	2	1.753931092464825	$3.564926400578017 \times 10^{-2}$
1.0×10^{-2}	5	1.724005619782788	$5.723791323742899 \times 10^{-3}$
1.0×10^{-3}	16	1.718841128579994	$5.593001209494020 \times 10^{-4}$
1.0×10^{-4}	48	1.718343976513114	$6.214805406878909 \times 10^{-5}$
1.0×10^{-5}	151	1.718288108448857	$6.279989812174591 \times 10^{-6}$
1.0×10^{-6}	476	1.718282460433048	$6.319740029070431 \times 10^{-7}$
1.0×10^{-7}	1506	1.718281891593031	$6.313398559498751 \times 10^{-8}$
1.0×10^{-8}	4760	1.718281834778786	$6.319740952775987 \times 10^{-9}$
1.0×10^{-9}	15051	1.718281829091138	$6.320926004832472 \times 10^{-10}$
1.0×10^{-10}	47595	1.718281828522237	$6.319145207100973 \times 10^{-11}$

Punto (c)

Le approssimazioni di I ottenute con la formula dei trapezi sono le seguenti :

$$I_2 = 1.75393109246482525876 \text{ (Errore} = 3.5649264006 \cdot 10^{-2}\text{)}$$

$$I_4 = 1.72722190455751656302 \text{ (Errore} = 8.9400760985 \cdot 10^{-3}\text{)}$$

$$I_8 = 1.72051859216430180766 \text{ (Errore} = 2.2367637053 \cdot 10^{-3}\text{)}$$

$$I_{16} = 1.71884112857999449275 \text{ (Errore} = 5.5930012095 \cdot 10^{-4}\text{)}$$

Valore esatto di I è : 1.718281828459045

Punto (d)

Il valore di $p(0) = 1.718281828460389$

Confronto con il valore esatto di $I = 1.718281828459045$

Si nota che il valore $p(0)$ si avvicina di molto al valore esatto di I , infatti l'errore

$$|p(0) - I| = 1.343813949006289 \cdot 10^{-12} \text{ (ovvero } 1.3438 \cdot 10^{-12}\text{)}.$$

Codice

Questo è il codice che **NON** utilizza il metodo dell'extrapolazione, ma utilizza al suo posto Ruffini-Horner e formula dei trapezi separatamente; inoltre, per il **punto (b)** viene calcolato per ogni ε il **miglior** n .

Usando `tic; toc` di MatLab, vediamo che il codice impiega tempo 18.120515 sec.

Problema 2.2

```

1  % Definizione della funzione
2  f = @(x) exp(x);
3
4  % Valore esatto dell'integrale
5  I_exact = 1.718281828459045;
6
7  % --- Punto (b) ---
8  % Tolleranze epsilon da verificare
9  epsilons = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-
10 10];
11
12 % Inizializzazione tabella
13 results = [];
14
15 for epsilon = epsilons
16     n = 1;
17     In = formula_trapezi(f, 0, 1, n);
18     error = abs(I_exact - In);
19
20     % Incrementa n fino a soddisfare la condizione di errore
21     while error > epsilon
22         n = n + 1;
23         In = formula_trapezi(f, 0, 1, n);
24         error = abs(I_exact - In);
25     end
26
27     % Aggiungi i risultati per questo epsilon

```

```

27     results = [results; epsilon, n, In, I_exact, error];
28 end
29
30 % --- Formattazione e visualizzazione dei risultati ---
31 % Cambia formato per Epsilon in esponenziale
32 format("shortE");
33 epsilon_col = results(:,1);
34
35 % Cambia formato per I_n e I_exact in formato long
36 format("long");
37 In_col = results(:,3);
38 I_exact_col = results(:,4);
39
40 % Cambia formato per il resto dei valori in compatto
41 format("compact");
42 n_col = results(:,2);
43 error_col = results(:,5);
44
45 % Mostra la tabella formattata
46 disp('Tabella dei risultati per il punto (b):');
47 disp(table(epsilon_col, n_col, In_col, I_exact_col, error_col, ...
48     'VariableNames', {'Epsilon', 'n', 'In', 'I_exact', 'Error'}));
49
50 % --- Punto (c) ---
51 n_values = [2, 4, 8, 16];
52 I_values = zeros(size(n_values));
53
54 for i = 1:length(n_values)
55     I_values(i) = formula_trapezi(f, 0, 1, n_values(i));
56 end
57
58 % Visualizza i risultati per il punto (c) con formato long per I_values
59 disp('Risultati per il punto (c):');
60 format("long");
61 for i = 1:length(n_values)
62     fprintf('I_%d = %.20f (Errore = %.10e)\n', n_values(i), I_values(i),
63         abs(I_exact - I_values(i)));
64 end
65 disp('Valore esatto I:');
66 disp(I_exact);
67
68 % --- Punto (d) ---
69 % Passi di discretizzazione
70 h_values = [1/2, 1/4, 1/8, 1/16];
71 h_squared = h_values.^2;
72
73 % Calcola il polinomio interpolante usando le funzioni fornite
74 p0 = interpola_ruffini_horner(h_squared, I_values, 0);

```

```

75 % Visualizza il risultato dell'interpolazione per il punto (d) in
    formato long
76 disp('Risultato per il punto (d):');
77 disp(['p(0) = ', num2str(p0, '%.15f')]);
78 disp(['Confronto con il valore esatto I: ', num2str(I_exact, '%.15f')]);
79 disp(abs(p0-I_exact));
80 % Reset del formato al default per successive esecuzioni
81 format("default");

```

Codice v2

Questo codice risolve il punto **(b)** andando a sostituire il valore di ε in modo corretto, ovvero non calcolando ogni volta il **miglior** n

Usando `tic;toc` di MatLab notiamo una differenza significativa nel tempo di esecuzione del codice, che in questo caso è di soli 0.012795 sec.

Problema 2 versione 2

```

1  % Definizione degli epsilon
2  epsilon_values = 10.^(-1:-1:-10); % {10^-1, 10^-2, ..., 10^-10}
3
4  % Funzione da integrare
5  f = @(x) exp(x);
6
7  % Intervallo di integrazione
8  a = 0;
9  b = 1;
10
11 % Valore esatto dell'integrale
12 I_exact = exp(1) - 1;
13
14 % Preallocazione per risultati
15 n_values = zeros(size(epsilon_values));
16 I_n_values = zeros(size(epsilon_values));
17 errors = zeros(size(epsilon_values));
18
19 % Calcolo di n e I_n
20 for i = 1:length(epsilon_values)
21     epsilon = epsilon_values(i);
22
23     % Calcolo di n (formula di stima dell'errore)
24     n = ceil(sqrt(exp(1) / (12 * epsilon)));
25     n_values(i) = n;
26
27     % Calcolo di I_n usando la formula dei trapezi
28     I_n = formulaTrapeziEs2(f, a, b, n);
29     error = abs(I_exact - I_n);
30     I_n_values(i) = I_n;
31     errors(i) = error;

```

```

32 end
33
34 % Visualizzazione dei risultati
35 disp('Epsilon      n      I_n      Error');
36 disp([epsilon_values(:), n_values(:), I_n_values(:), errors(:)]);
37

```

Questo codice risolve il punto **(d)** utilizzando il metodo dell'estrapolazione.

Punto (d)

```

1  % Vettore di n
2  n_vect = [2, 4, 8, 16];
3
4  % Estrapolazione polinomiale
5  p0 = estrapolazioneEs3(f, a, b, n_vect);
6
7  % Calcolo degli I_n e confronto con p(0)
8  I_n_values = zeros(size(n_vect));
9  errors = zeros(size(n_vect));
10
11 for i = 1:length(n_vect)
12     n = n_vect(i);
13     I_n_values(i) = formulaTrapeziEs2(f, a, b, n);
14     errors(i) = abs(I_exact - I_n_values(i));
15 end
16
17 % Confronto finale
18 disp('n      I_n      Error');
19 disp([n_vect(:), I_n_values(:), errors(:)]);
20
21 disp(['Valore estrapolato p(0): ', num2str(p0)]);
22 disp(['Errore tra p(0) e I esatto: ', num2str(abs(I_exact - p0))]);
23

```

Problema 3

Consideriamo la funzione $f(x) = x^2 e^{-x}$ e indichiamo con I_n la formula dei trapezi di ordine n per approssimare $I = \int_0^1 f(x) dx$.

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

(b) Calcolare I_5 , I_{10} , I_{20} , I_{40} .

(c) Calcolare $p(0)$, dove $p(x)$ è il polinomio d'interpolazione dei dati

(h_0^2, I_5) , (h_1^2, I_{10}) , (h_2^2, I_{20}) , (h_3^2, I_{40}) e h_0, h_1, h_2, h_3 sono i passi di discretizzazione delle formule dei trapezi I_5 , I_{10} , I_{20} , I_{40} .

(d) Riportare in una tabella:

- i valori I_5 , I_{10} , I_{20} , I_{40} , $p(0)$;

- gli errori $|I_5 - I|$, $|I_{10} - I|$, $|I_{20} - I|$, $|I_{40} - I|$, $|p(0) - I|$.

(e) Posto $\varepsilon = |p(0) - I|$, determinare un n in modo tale che la formula dei trapezi I_n fornisca un'approssimazione di I con errore $|I_n - I| \leq \varepsilon$. Calcolare successivamente I_n e verificare che effettivamente $|I_n - I| \leq \varepsilon$.

Soluzione

Punto (a)

Calcolo manuale (Integrazione per parti):

$$I = \int_0^1 x^2 e^{-x} dx$$

- Prima integrazione per parti ($u = x^2$, $dv = e^{-x} dx$):
 - $I = [-x^2 e^{-x}]_0^1 + \int_0^1 2x e^{-x} dx$
 - Primo termine: $(-x^2 e^{-x})_0^1 = (-1^2 e^{-1} - 0) = -\frac{1}{e}$.
 - Secondo termine: $\int_0^1 2x e^{-x} dx$.
- Seconda integrazione per parti ($u = 2x$, $dv = e^{-x} dx$):
 - $\int_0^1 2x e^{-x} dx = [-2x e^{-x}]_0^1 + \int_0^1 2 e^{-x} dx$
 - Primo termine: $(-2x e^{-x})_0^1 = (-2e^{-1} - 0) = -\frac{2}{e}$.
 - Secondo termine: $\int_0^1 2 e^{-x} dx = -2e^{-x} \Big|_0^1 = -2e^{-1} + 2$.

Riassumendo:

$$I = -\frac{1}{e} + \left(-\frac{2}{e} + \left(-\frac{2}{e} + 2 \right) \right) = 2 - \frac{5}{e}.$$

Il valore esatto è:

$$I = 2 - \frac{5}{e} \approx 0.1606027941$$

Calcolo simbolico

```
1  syms x
2  f = x^2 * exp(-x);
3  I_exact = int(f, 0, 1);
```

Output:

$$I = 1.606027941427884 \cdot 10^{-1}$$

Punto (b)

Per calcolare I_n , usiamo la formula dei trapezi:

$$I_n = h \left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(a + ih) \right),$$

dove $h = \frac{b-a}{n} = \frac{1}{n}$.

```

1  % Funzione e intervallo
2  f = @(x) x.^2 .* exp(-x); % Definizione della funzione
3  a = 0;
4  b = 1;
5
6  % Calcolo delle approssimazioni con la formula dei trapezi
7  I_5 = formula_trapezi(f, a, b, 5);
8  I_10 = formula_trapezi(f, a, b, 10);
9  I_20 = formula_trapezi(f, a, b, 20);
10 I_40 = formula_trapezi(f, a, b, 40);
11
12 % Calcolo del valore esatto
13 I_exact = 2 - 5 / exp(1); % Valore calcolato analiticamente
14
15 % Calcolo degli errori
16 error_5 = abs(I_5 - I_exact);
17 error_10 = abs(I_10 - I_exact);
18 error_20 = abs(I_20 - I_exact);
19 error_40 = abs(I_40 - I_exact);
20
21 % Stampa dei risultati a schermo
22 fprintf('Risultati:\n');
23 fprintf('I_5    = %.10f, Errore = %.10f\n', I_5, error_5);
24 fprintf('I_10   = %.10f, Errore = %.10f\n', I_10, error_10);
25 fprintf('I_20   = %.10f, Errore = %.10f\n', I_20, error_20);
26 fprintf('I_40   = %.10f, Errore = %.10f\n', I_40, error_40);
27

```

Risultati :

$I_5 = 0.1618165768, \text{Errore} = 0.0012137827$

$I_{10} = 0.1609085786, \text{Errore} = 0.0003057845$

$I_{20} = 0.1606793868, \text{Errore} = 0.0000765927$

$I_{40} = 0.1606219515, \text{Errore} = 0.0000191573$

Punto (c)

Dati i nodi (h^2, I_n) , con:

$$h_0^2 = \left(\frac{1}{5}\right)^2, \quad h_1^2 = \left(\frac{1}{10}\right)^2, \quad h_2^2 = \left(\frac{1}{20}\right)^2, \quad h_3^2 = \left(\frac{1}{40}\right)^2$$

$$x = [0.04, 0.01, 0.0025, 0.000625], \quad y = [I_5, I_{10}, I_{20}, I_{40}]$$

Usiamo il metodo di Ruffini-Horner per interpolare $p(x)$ e valutiamo $p(0)$.

```

1  % Interpolazione dei nodi (h^2, I_n)
2  x = [0.04, 0.01, 0.0025, 0.000625]; % h^2 valori (passi quadratici)
3  y = [I_5, I_10, I_20, I_40]; % Valori approssimati
4
5  % Calcolo del valore interpolato p(0)
6  p_0 = interpolaRuffiniHornerEs1(x, y, 0);
7
8  % Calcolo errore di interpolazione
9  error_p0 = abs(p_0 - I_exact);
10
11 % Stampa dei risultati dell'interpolazione
12 fprintf('\nInterpolazione:\n');
13 fprintf('p(0) = %.10f, Errore = %.10f\n', p_0, error_p0);

```

Il valore di $p(0)$ è quindi

$$p(0) = 1.606027941428046 \cdot 10^{-1}$$

Punto (d)

Tabella dei risultati:

n	I_n	$I_n - I_{\text{exact}}$
5	$1.618165768206828 \cdot 10^{-1}$	$1.213782677894459 \cdot 10^{-3}$
10	$1.609085786320963 \cdot 10^{-1}$	$3.057844893079031 \cdot 10^{-4}$
20	$1.606793868113391 \cdot 10^{-1}$	$7.659266855072899 \cdot 10^{-5}$
40	$1.606219514748572 \cdot 10^{-1}$	$1.915733206886427 \cdot 10^{-5}$
$p(0)$	$1.606027941428046 \cdot 10^{-1}$	$1.618150058391166 \cdot 10^{-14}$

Punto (e)

Preso $\varepsilon = |p(0) - I|$, per trovare un $n = n_\varepsilon$ tale che $|I - I_n| \leq \varepsilon$ bisogna fare così

Per il teorema sull'errore o resto della formula dei trapezi, abbiamo che

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

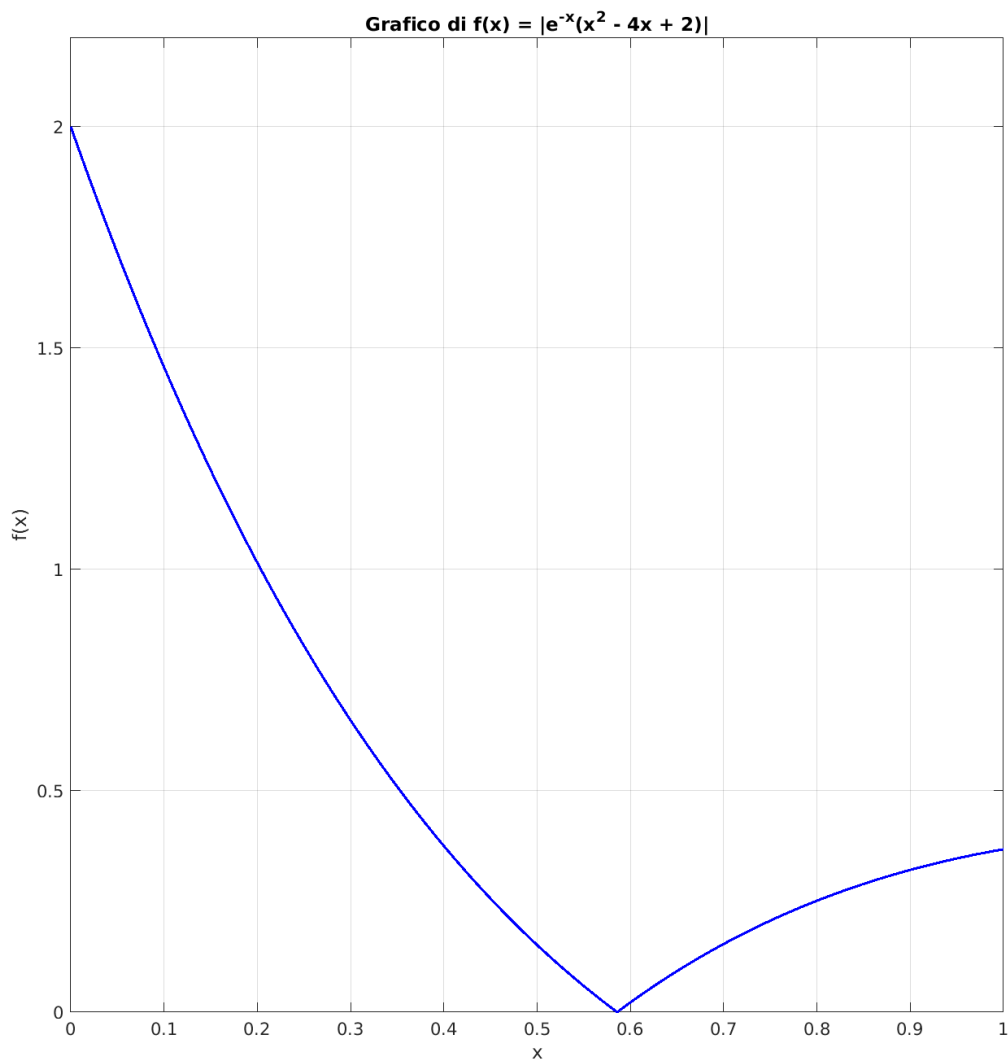
Calcoliamo $f''(x)$:

$$\begin{aligned} f'(x) &= 2xe^{-x} - x^2e^{-x} \\ f''(x) &= e^{-x}(x^2 - 4x + 2) \end{aligned}$$

per ogni $x \in [0, 1]$ si ha che:

$$|f''(x)| = |e^{-x}(x^2 - 4x + 2)| \leq 2$$

Questo lo possiamo verificare guardando il grafico di $|f''(x)|$, che è il seguente



Quindi, possiamo scrivere

$$\left| \int_0^1 x^2 e^{-x} dx - I_n \right| \leq \frac{2}{12n^2}$$

E infine

$$\frac{2}{12n^2} \leq \varepsilon \iff n \geq \sqrt{\frac{2}{12\varepsilon}} = n(\varepsilon)$$

Quindi, dato che $\varepsilon = 1.62 \cdot 10^{-14}$, $n = n(\varepsilon) \geq 3.2075 \cdot 10^6$

Codice

```

1
2 % Punto (a): Calcolo dell'integrale esatto
3 syms x;
4 f_sym = x^2 * exp(-x); % Funzione simbolica
5 I_exact = double(int(f_sym, 0, 1)); % Calcolo simbolico del valore
    esatto
6 fprintf('Punto (a):\n');
7 fprintf('Valore esatto dell\'integrale I = %.10f\n\n', I_exact);
8
9 % Definizione della funzione come funzione anonima
10 f = @(x) x.^2 .* exp(-x);
11
12 % Punto (b): Calcolo di I_5, I_10, I_20, I_40
13
14 I_5 = formula_trapezi(f, 0, 1, 5);
15 I_10 = formula_trapezi(f, 0, 1, 10);
16 I_20 = formula_trapezi(f, 0, 1, 20);
17 I_40 = formula_trapezi(f, 0, 1, 40);
18
19 fprintf('Punto (b):\n');
20 fprintf('I_5 = %.10f\n', I_5);
21 fprintf('I_10 = %.10f\n', I_10);
22 fprintf('I_20 = %.10f\n', I_20);
23 fprintf('I_40 = %.10f\n\n', I_40);
24
25
26 % Punto (c): Interpolazione di p(0)
27 % Passi h e h^2
28 h = [1/5, 1/10, 1/20, 1/40]; % Passi di discretizzazione
29 h2 = h.^2; % h^2 per interpolazione
30 I_values = [I_5, I_10, I_20, I_40]; % Valori I_5, I_10, I_20, I_40
31
32 % Calcolo del polinomio interpolante tramite interpolaRuffiniHornerEs1
33 p_coeff = interpolaRuffiniHornerEs1(h2, I_values); % Coefficienti del
    polinomio
34 p_0 = p_coeff(end); % Valore di p(0), cioè il termine noto
35 fprintf('Punto (c):\n');
36 fprintf('Valore interpolato p(0) = %.10f\n\n', p_0);
37
38 % Punto (d): Tabella dei risultati
39 % Errori calcolati
40 error_5 = abs(I_5 - I_exact);

```

```

41 error_10 = abs(I_10 - I_exact);
42 error_20 = abs(I_20 - I_exact);
43 error_40 = abs(I_40 - I_exact);
44 error_p0 = abs(p_0 - I_exact);
45
46 fprintf('Punto (d): Tabella dei risultati\n');
47 fprintf('n          I_n          |I_n - I_exact|\n');
48 fprintf('%-9d %.10f %.10f\n', 5, I_5, error_5);
49 fprintf('%-9d %.10f %.10f\n', 10, I_10, error_10);
50 fprintf('%-9d %.10f %.10f\n', 20, I_20, error_20);
51 fprintf('%-9d %.10f %.10f\n', 40, I_40, error_40);
52 fprintf('p(0)          %.10f %.10f\n\n', p_0, error_p0);

```

Problema 4

Si consideri il sistema lineare $Ax = b$, dove:

$$A = \begin{bmatrix} 5 & 1 & 2 \\ -1 & 7 & 1 \\ 0 & 1 & -3 \end{bmatrix}, b = \begin{bmatrix} 13 \\ 16 \\ -7 \end{bmatrix}.$$

(a) Si calcoli la soluzione x del sistema dato con **MATLAB**.

(b) La matrice A è a diagonale dominante in senso stretto per cui il metodo di Jacobi è convergente ossia partendo da un qualsiasi vettore d'innescio $x^{(0)}$ la successione prodotta dal metodo di Jacobi converge (componente per componente) alla soluzione x del sistema dato. Calcolare le prime 10 iterazioni $x^{(1)}, \dots, x^{(10)}$ del metodo di Jacobi partendo dal vettore nullo $x^{(0)} = [0, 0, 0]^T$ e confrontarle con la soluzione esatta x ponendo iterazioni e soluzione esatta in un'unica matrice S di dimensioni 3×12 le cui colonne sono nell'ordine $x^{(0)}, x^{(1)}, \dots, x^{(10)}, x$.

(c) Consideriamo il metodo di Jacobi per risolvere il sistema dato. Conveniamo d'innescare il metodo di Jacobi con il vettore nullo $x^{(0)} = [0, 0, 0]^T$. Costruire una tabella che riporti vicino ad ogni $\varepsilon \in \{10^{-1}, 10^{-2}, \dots, 10^{-10}\}$:

- il numero d'iterazioni K_ε necessarie al metodo di Jacobi per convergere entro la precisione ε ;
- la soluzione approssimata x_ε calcolata dal metodo di Jacobi;
- la soluzione esatta x (in modo da confrontarla con la soluzione approssimata x_ε);
- la norma ∞ dell'errore $\|x - x_\varepsilon\|_\infty$.

Soluzione

Punto (a)

La soluzione al sistema lineare $Ax = b$, trovata con MATLAB è la seguente :

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Il codice MATLAB per fare ciò è il seguente :

```
1  A = [5, 1, 2; -1, 7, 1; 0, 1, -3];
2  b = [13; 16; -7];
3
4  x_exact = A \ b; % Soluzione esatta
```

Punto (b)

La matrice S di dimensione 3×12 contenente le prime 10 iterazioni del metodo di Jacobi è la seguente :

Abbiamo diviso la matrice S in due matrici, ognuna contenente 6 colonne per maggior chiarezza.

$$S_1 = \begin{bmatrix} 0.0000000 & 2.6000000 & 1.2095238 & 0.8971429 & 0.9535601 & 1.0038458 \\ 0.0000000 & 2.2857143 & 2.3238095 & 2.0163265 & 1.9698866 & 1.9925883 \\ 0.0000000 & 2.3333333 & 3.0952381 & 3.1079365 & 3.0054422 & 2.9899622 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 1.0054975 & 1.0005916 & 0.9995079 & 0.9998502 & 1.0000262 & 1.0000000 \\ 2.0019834 & 2.0011383 & 1.9999901 & 1.9998755 & 1.9999791 & 2.0000000 \\ 2.9975294 & 3.0006611 & 3.0003794 & 2.9999967 & 2.9999585 & 3.0000000 \end{bmatrix}$$

Punto (c)

Tabella riportante le soluzioni fornite dal metodo di Jacobi, per ogni ε richiesto

ε	K_ε	Soluzione approssimata x_ε	Soluzione esatta x	$\ x - x_\varepsilon\ _\infty$
10^{-1}	3	$x_\varepsilon = \begin{bmatrix} 0.8971429 \\ 2.0163265 \\ 3.1079365 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.079365 \cdot 10^{-1}$
10^{-2}	5	$x_\varepsilon = \begin{bmatrix} 1.0038458 \\ 1.9925883 \\ 2.9899622 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.003779 \cdot 10^{-2}$
10^{-3}	7	$x_\varepsilon = \begin{bmatrix} 1.0005916 \\ 2.0011383 \\ 3.0006611 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.138291 \cdot 10^{-3}$
10^{-4}	9	$x_\varepsilon = \begin{bmatrix} 0.9998502 \\ 1.9998755 \\ 2.9999967 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.497845 \cdot 10^{-4}$
10^{-5}	11	$x_\varepsilon = \begin{bmatrix} 1.0000208 \\ 2.0000097 \\ 2.9999930 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$2.078563 \cdot 10^{-5}$
10^{-6}	13	$x_\varepsilon = \begin{bmatrix} 0.9999979 \\ 1.9999997 \\ 3.0000013 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$2.083214 \cdot 10^{-6}$
10^{-7}	15	$x_\varepsilon = \begin{bmatrix} 1.0000001 \\ 2.0000000 \\ 2.9999998 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.621496 \cdot 10^{-7}$
10^{-8}	17	$x_\varepsilon = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.450418 \cdot 10^{-8}$
10^{-9}	19	$x_\varepsilon = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$1.823506 \cdot 10^{-9}$
10^{-10}	21	$x_\varepsilon = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$x = \begin{bmatrix} 1.0000000 \\ 2.0000000 \\ 3.0000000 \end{bmatrix}$	$2.567879 \cdot 10^{-10}$

Codice

Problema 2.4

```

1  % Dati del problema
2  A = [5, 1, 2; -1, 7, 1; 0, 1, -3];
3  b = [13; 16; -7];
4
5  % Punto (a): Soluzione esatta del sistema
6  x_exact = A \ b; % Soluzione esatta
7  disp('Soluzione esatta:');
8  disp(x_exact);
9
10 % Punto (b): Metodo di Jacobi per le prime 10 iterazioni
11 x0 = [0; 0; 0]; % Vettore iniziale

```



```

12 N_iter = 10; % Numero di iterazioni
13 n = length(b);
14 X_iterations = zeros(n, N_iter+2); % Matrice per conservare le
    iterazioni
15 X_iterations(:, 1) = x0; % Inizializzazione con x^(0)
16
17 for k = 1:N_iter
18     x_new = zeros(n, 1);
19     for i = 1:n
20         sum1 = A(i, 1:i-1) * X_iterations(1:i-1, k);
21         sum2 = A(i, i+1:n) * X_iterations(i+1:n, k);
22         x_new(i) = (b(i) - sum1 - sum2) / A(i, i);
23     end
24     X_iterations(:, k+1) = x_new;
25 end
26 X_iterations(:, end) = x_exact; % Aggiunge la soluzione esatta come
    ultima colonna
27
28 disp('Iterazioni del metodo di Jacobi (prime 10):');
29 disp(X_iterations);
30
31 % Punto (c): Metodo di Jacobi con variazione della precisione
32 epsilons = 10.^(-1:-1:-10); % Precisioni {10^-1, ..., 10^-10}
33 N_max = 1000; % Numero massimo di iterazioni
34 results = []; % Per conservare i risultati
35
36 for epsilon = epsilons
37     [x_approx, K, r_norm] = jacobi_method(A, b, x0, epsilon, N_max);
38     error_norm = norm(x_exact - x_approx, inf); % Norma dell'errore
    infinito
39     results = [results; struct('epsilon', epsilon, 'K', K, 'x_approx',
    x_approx, ...
40                               'error_norm', error_norm)];
41 end
42
43 % Stampa dei risultati in formato tabella
44 disp('Tabella dei risultati per le varie precisioni:');
45 disp('Epsilon | Iterazioni K | x_epsilon | Norma
    errore ||x - x_approx||_inf');
46 for i = 1:length(results)
47     r = results(i);
48     fprintf('%.1e|%.3d|[%7.4f, %7.4f, %7.4f]|%e\n', ...
49             r.epsilon, r.K, r.x_approx(1), r.x_approx(2), r.x_approx(3),
    r.error_norm);
50 end

```

Problema 5

Si consideri il sistema lineare $A_n x = b_n$, dove $b_n = [1, 1, \dots, 1]^T$ e A_n è la matrice $n \times n$ definita nel modo seguente:

$$(A_n)_{ij} = \begin{cases} 3, & \text{se } i = j, \\ -\left(\frac{1}{2}\right)^{\max(i,j)-1}, & \text{se } i \neq j. \end{cases}$$

(a) Scrivere esplicitamente A_n per $n = 5$.

(b) Dimostrare che, qualunque sia n , A_n è una matrice a diagonale dominante in senso stretto per righe e per colonne. Dedurre che i metodi di Jacobi e Gauss-Seidel per risolvere un sistema lineare di matrice A_n sono convergenti.

(c) Risolvere con il comando `\` il sistema lineare $A_n x = b_n$ per $n = 5, 10, 20$.

(d) Risolvere il sistema lineare $A_n x = b_n$ per $n = 5, 10, 20$ con i metodi di Jacobi e Gauss-Seidel entro una soglia di precisione $\varepsilon = 10^{-7}$, partendo dal vettore d'innescio $x^{(0)} = 0$.

(e) Costruire una tabella che, vicino ad ogni $n = 5, 10, 20$, riporti:

- la soluzione esatta x del sistema $A_n x = b_n$ ottenuta al punto (c);
- le soluzioni approssimate x_J e x_G ottenute con i metodi di Jacobi e Gauss-Seidel al punto (d);
- gli errori $\|x_J - x\|_\infty$ e $\|x_G - x\|_\infty$;
- i numeri K_J e K_G , che contano le iterazioni effettuate da Jacobi e Gauss-Seidel per calcolare x_J e x_G , rispettivamente.

Soluzione

Punto (a)

La matrice A_n è definita come:

$$(A_n)_{ij} = \begin{cases} 3, & i = j \\ -\left(\frac{1}{2}\right)^{\max(i,j)-1}, & i \neq j \end{cases}$$

Per $n = 5$ la matrice A_5 è:

$$A_5 = \begin{bmatrix} 3 & -\frac{1}{2} & -\frac{1}{4} & -\frac{1}{8} & -\frac{1}{16} \\ -\frac{1}{2} & 3 & -\frac{1}{4} & -\frac{1}{8} & -\frac{1}{16} \\ -\frac{1}{4} & -\frac{1}{4} & 3 & -\frac{1}{8} & -\frac{1}{16} \\ -\frac{1}{8} & -\frac{1}{8} & -\frac{1}{8} & 3 & -\frac{1}{16} \\ -\frac{1}{16} & -\frac{1}{16} & -\frac{1}{16} & -\frac{1}{16} & 3 \end{bmatrix}.$$

Punto (b)

Una matrice $A \in \mathbb{C}^{n \times n}$ è definita:

- A diagonale dominante in senso stretto (per righe) se $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ per ogni $i = 1, \dots, n$

- A diagonale dominante in senso stretto (per colonne) se $|a_{jj}| > \sum_{i \neq j} |a_{ij}|$ per ogni $i = 1, \dots, n$

Data la matrice A_5 , si nota che essa è a diagonale dominante in senso stretto sia per righe che per colonne.

Infatti preso $|a_{ii}| = |a_{jj}| = |3|, \forall i, j$, abbiamo che

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \text{ con } |a_{ij}| = \left(\frac{1}{2}\right)^{\max(i,j)-1}$$

$$|a_{jj}| > \sum_{i \neq j} |a_{ij}|, \text{ con } |a_{ij}| = \left(\frac{1}{2}\right)^{\max(i,j)-1}$$

Dimostriamo che la matrice A_5 è a diagonale dominante:

La condizione di dominanza diagonale per righe richiede che:

$$|A_{ii}| > \sum_{j \neq i} |A_{ij}|.$$

Nel nostro caso:

- $|A_{ii}| = 3$.
- La somma $\sum_{j \neq i} |A_{ij}|$ si divide in due parti:
 - **Prima della diagonale** ($j < i$): tutti i termini sono uguali a $\left(\frac{1}{2}\right)^{i-1}$.
 - **Dopo la diagonale** ($j > i$): i termini sono della forma $\left(\frac{1}{2}\right)^i, \left(\frac{1}{2}\right)^{i+1}, \dots$

Pertanto, possiamo scrivere:

$$\sum_{j \neq i} |A_{ij}| = \underbrace{(i-1) \cdot \left(\frac{1}{2}\right)^{i-1}}_{\text{prima della diagonale}} + \underbrace{\sum_{k=0}^{n-i-1} \left(\frac{1}{2}\right)^{i+k}}_{\text{dopo la diagonale}}.$$

Analisi prima parte:

La somma degli elementi prima della diagonale è:

$$(i-1) \cdot \left(\frac{1}{2}\right)^{i-1}.$$

Analisi seconda parte:

Gli elementi dopo la diagonale formano una serie geometrica:

$$\sum_{k=0}^{n-i-1} \left(\frac{1}{2}\right)^{i+k}.$$

Usando la formula per la somma di una serie geometrica:

$$\sum_{k=0}^m r^k = \frac{1 - r^{m+1}}{1 - r},$$

qui $r = \frac{1}{2}$, $m = n - i - 1$, e il primo termine della serie è $\left(\frac{1}{2}\right)^i$.

Quindi otteniamo che:

$$\sum_{k=0}^{n-i-1} \left(\frac{1}{2}\right)^{i+k} = \left(\frac{1}{2}\right)^i \cdot \frac{1 - \left(\frac{1}{2}\right)^{n-i}}{1 - \frac{1}{2}} = 2 \cdot \left(\frac{1}{2}\right)^i \cdot \left(1 - \left(\frac{1}{2}\right)^{n-i}\right).$$

Combinando le due parti, otteniamo:

$$\sum_{j \neq i} |A_{ij}| = (i-1) \cdot \left(\frac{1}{2}\right)^{i-1} + 2 \cdot \left(\frac{1}{2}\right)^i \cdot \left(1 - \left(\frac{1}{2}\right)^{n-i}\right).$$

Di conseguenza, la condizione di dominanza diagonale per righe $|A_{ii}| > \sum_{j \neq i} |A_{ij}|$ diventa:

$$3 > (i-1) \cdot \left(\frac{1}{2}\right)^{i-1} + 2 \cdot \left(\frac{1}{2}\right)^i \cdot \left(1 - \left(\frac{1}{2}\right)^{n-i}\right).$$

Verifica

Per $i = 1$:

$$3 > 0 + 2 \cdot \left(\frac{1}{2}\right)^1 \cdot \left(1 - \left(\frac{1}{2}\right)^{n-1}\right)$$

La disuguaglianza è soddisfatta poiché il lato destro è minore di 1.

Per $i = n$:

$$3 > (n-1) \cdot \left(\frac{1}{2}\right)^{n-1}.$$

Anche qui la disuguaglianza è verificata perché $\left(\frac{1}{2}\right)^{n-1}$ decresce rapidamente.

In generale, la disuguaglianza è verificata per ogni i , dimostrando che A_n è diagonale dominante per righe.

Usando i **teoremi di convergenza**, sappiamo che i metodi di Jacobi e Gauss-Seidel convergono se la matrice $A \in \mathbb{C}^{n \times n}$ soddisfa almeno una delle seguenti condizioni :

- A è a diagonale dominante e irriducibile
- A è a diagonale dominante in senso stretto per righe
- A è a diagonale dominante per colonne e irriducibile
- A è a diagonale dominante in senso stretto per colonne

Abbiamo dimostrato che A_5 rispetta sia la seconda che quarta condizione, quindi i metodi di Jacobi e Gauss-Seidel applicati alla matrice A_5 convergono.

Punto (c)

Per $n = 5$, il risultato del sistema $A_5 x = b_5$ è :

$$x = \begin{bmatrix} 4.728395611573806 \cdot 10^{-1} \\ 4.728395611573807 \cdot 10^{-1} \\ 4.364672872221975 \cdot 10^{-1} \\ 3.986401223296070 \cdot 10^{-1} \\ 3.704330527472200 \cdot 10^{-1} \end{bmatrix}$$

Per $n = 10$, il risultato del sistema $A_{10} x = b_{10}$ è :

$$x = \begin{bmatrix} 4.829209469162112 \cdot 10^{-1} \\ 4.829209469162111 \cdot 10^{-1} \\ 4.457731817688103 \cdot 10^{-1} \\ 4.071395060155133 \cdot 10^{-1} \\ 3.783310350848758 \cdot 10^{-1} \\ 3.595809878517137 \cdot 10^{-1} \\ 3.481971245527415 \cdot 10^{-1} \\ 3.415564320733823 \cdot 10^{-1} \\ 3.377789759887189 \cdot 10^{-1} \\ 3.356667963132605 \cdot 10^{-1} \end{bmatrix}$$

del sistema $A_{20} x = b_{20}$ è :

$$x = \begin{bmatrix} 4.832359353604220 \cdot 10^{-1} \\ 4.832359353604221 \cdot 10^{-1} \\ 4.460639403326973 \cdot 10^{-1} \\ 4.074050655038636 \cdot 10^{-1} \\ 3.785778040537910 \cdot 10^{-1} \\ 3.598155269758799 \cdot 10^{-1} \\ 3.484242384753038 \cdot 10^{-1} \\ 3.417792145594933 \cdot 10^{-1} \\ 3.379992946036253 \cdot 10^{-1} \\ 3.358857372447301 \cdot 10^{-1} \\ 3.347186246657436 \cdot 10^{-1} \\ 3.340803134057811 \cdot 10^{-1} \\ 3.337338945501267 \cdot 10^{-1} \\ 3.335470848650332 \cdot 10^{-1} \\ 3.334468884268689 \cdot 10^{-1} \\ 3.333933967162749 \cdot 10^{-1} \\ 3.333649547349902 \cdot 10^{-1} \\ 3.333498858470857 \cdot 10^{-1} \\ 3.333419274986317 \cdot 10^{-1} \\ 3.333377363838597 \cdot 10^{-1} \end{bmatrix}$$

Punto (d)

n	Metodo	Soluzione x_J/x_G
5	Jacobi	$x_J = \begin{bmatrix} 4.7284 \cdot 10^{-1} \\ 4.7284 \cdot 10^{-1} \\ 4.3647 \cdot 10^{-1} \\ 3.9864 \cdot 10^{-1} \\ 3.7043 \cdot 10^{-1} \end{bmatrix}$
5	Gauss-Seidel	$x_G = \begin{bmatrix} 4.7284 \cdot 10^{-1} \\ 4.7284 \cdot 10^{-1} \\ 4.3647 \cdot 10^{-1} \\ 3.9864 \cdot 10^{-1} \\ 3.7043 \cdot 10^{-1} \end{bmatrix}$

n	Metodo	Soluzione x_J/x_G
10	Jacobi	$x_J = \begin{bmatrix} 4.8292 \cdot 10^{-1} \\ 4.8292 \cdot 10^{-1} \\ 4.4577 \cdot 10^{-1} \\ 4.0714 \cdot 10^{-1} \\ 3.7833 \cdot 10^{-1} \\ 3.5958 \cdot 10^{-1} \\ 3.4820 \cdot 10^{-1} \\ 3.4156 \cdot 10^{-1} \\ 3.3778 \cdot 10^{-1} \\ 3.3567 \cdot 10^{-1} \end{bmatrix}$
10	Gauss-Seidel	$x_G = \begin{bmatrix} 4.8292 \cdot 10^{-1} \\ 4.8292 \cdot 10^{-1} \\ 4.4577 \cdot 10^{-1} \\ 4.0714 \cdot 10^{-1} \\ 3.7833 \cdot 10^{-1} \\ 3.5958 \cdot 10^{-1} \\ 3.4820 \cdot 10^{-1} \\ 3.4156 \cdot 10^{-1} \\ 3.3778 \cdot 10^{-1} \\ 3.3567 \cdot 10^{-1} \end{bmatrix}$
20	Jacobi	$x_J = \begin{bmatrix} 4.8324 \cdot 10^{-1} \\ 4.8324 \cdot 10^{-1} \\ 4.4606 \cdot 10^{-1} \\ 4.0741 \cdot 10^{-1} \\ 3.7858 \cdot 10^{-1} \\ 3.5982 \cdot 10^{-1} \\ 3.4842 \cdot 10^{-1} \\ 3.4178 \cdot 10^{-1} \\ 3.3800 \cdot 10^{-1} \\ 3.3589 \cdot 10^{-1} \\ 3.3472 \cdot 10^{-1} \\ 3.3408 \cdot 10^{-1} \\ 3.3373 \cdot 10^{-1} \\ 3.3355 \cdot 10^{-1} \\ 3.3345 \cdot 10^{-1} \\ 3.3339 \cdot 10^{-1} \\ 3.3336 \cdot 10^{-1} \\ 3.3335 \cdot 10^{-1} \\ 3.3334 \cdot 10^{-1} \\ 3.3334 \cdot 10^{-1} \end{bmatrix}$

n	Metodo	Soluzione x_J/x_G
20	Gauss-Seidel	$x_G = \begin{bmatrix} 4.8324 \cdot 10^{-1} \\ 4.8324 \cdot 10^{-1} \\ 4.4606 \cdot 10^{-1} \\ 4.0741 \cdot 10^{-1} \\ 3.7858 \cdot 10^{-1} \\ 3.5982 \cdot 10^{-1} \\ 3.4842 \cdot 10^{-1} \\ 3.4178 \cdot 10^{-1} \\ 3.3800 \cdot 10^{-1} \\ 3.3589 \cdot 10^{-1} \\ 3.3472 \cdot 10^{-1} \\ 3.3408 \cdot 10^{-1} \\ 3.3373 \cdot 10^{-1} \\ 3.3355 \cdot 10^{-1} \\ 3.3345 \cdot 10^{-1} \\ 3.3339 \cdot 10^{-1} \\ 3.3336 \cdot 10^{-1} \\ 3.3335 \cdot 10^{-1} \\ 3.3334 \cdot 10^{-1} \\ 3.3334 \cdot 10^{-1} \end{bmatrix}$

Punto (e)

La tabella è la seguente

n	Metodo	Iterazioni	Norma errore $\ x - x_J/x_G\ _\infty$
5	Jacobi	12	4.051786×10^{-8}
5	Gauss-Seidel	7	6.545649×10^{-8}
10	Jacobi	12	4.884032×10^{-8}
10	Gauss-Seidel	7	9.323449×10^{-8}
20	Jacobi	12	4.897032×10^{-8}
20	Gauss-Seidel	7	9.398408×10^{-8}

Codice

Problema 2.5

```
1 % Parametri del problema
2 n_values = [5, 10, 20];
3 epsilon = 1e-7;
```



```

4  N_max = 500;
5
6  % Inizializza output per le tabelle
7  tabella1 = "";
8  tabella2 = "";
9
10 % Genera i risultati per entrambe le tabelle
11 for n = n_values
12     % Genera sistema
13     [A, b] = generate_system(n);
14
15     % Soluzione esatta
16     x_exact = A \ b;
17
18     % Jacobi
19     [x_J, K_J, ~] = JacobiIterativo(A, b, zeros(n, 1), epsilon, N_max);
20     error_J = norm(x_exact - x_J, inf);
21
22     % Gauss-Seidel
23     [x_G, K_G, ~] = GaussSeidelIt(A, b, zeros(n, 1), epsilon, N_max);
24     error_G = norm(x_exact - x_G, inf);
25
26     % Aggiorna Tabella 1
27     tabella1 = tabella1 + sprintf('%2d | Jacobi          | [%s]\n', n,
num2str(x_J, '%.4e '));
28     tabella1 = tabella1 + sprintf('%2d | Gauss-Seidel   | [%s]\n', n,
num2str(x_G, '%.4e '));
29
30     % Aggiorna Tabella 2
31     tabella2 = tabella2 + sprintf('%2d | Jacobi          | %3d          |
%e\n', n, K_J, error_J);
32     tabella2 = tabella2 + sprintf('%2d | Gauss-Seidel   | %3d          |
%e\n', n, K_G, error_G);
33 end
34
35 % Stampa Tabella 1
36 fprintf('Tabella 1: Soluzioni approssimate (x_J e x_G)\n');
37 fprintf(' n | Metodo          | Soluzione x_J/x_G\n');
38 fprintf('-----\n');
39 fprintf('%s', tabella1);
40
41 % Stampa Tabella 2
42 fprintf('\nTabella 2: Iterazioni e norma dell errore\n');
43 fprintf(' n | Metodo          | Iterazioni | Norma errore ||x -
x_J/x_G||_inf\n');
44 fprintf('-----\n');
45 fprintf('%s', tabella2);
46
47 function [A, b] = generate_system(n)

```

```

48     A = zeros(n);
49     b = ones(n, 1);
50
51     for i = 1:n
52         for j = 1:n
53             if i == j
54                 A(i,j) = 3;
55             else
56                 A(i,j) = -0.5^(max(i,j)-1);
57             end
58         end
59     end
60 end

```

Problema 6

Consideriamo i seguenti due casi:

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

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 $f(\xi_\varepsilon)$.

Soluzione

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 $f(b)$:

- $f(0) = 0^3 + 3(0) - 1 - e^{-0^2} = -1 - 1 = -2$,
- $f(1) = 1^3 + 3(1) - 1 - e^{-1^2} = 1 + 3 - 1 - e^{-1} = 3 - e^{-1} \approx 2.63$.

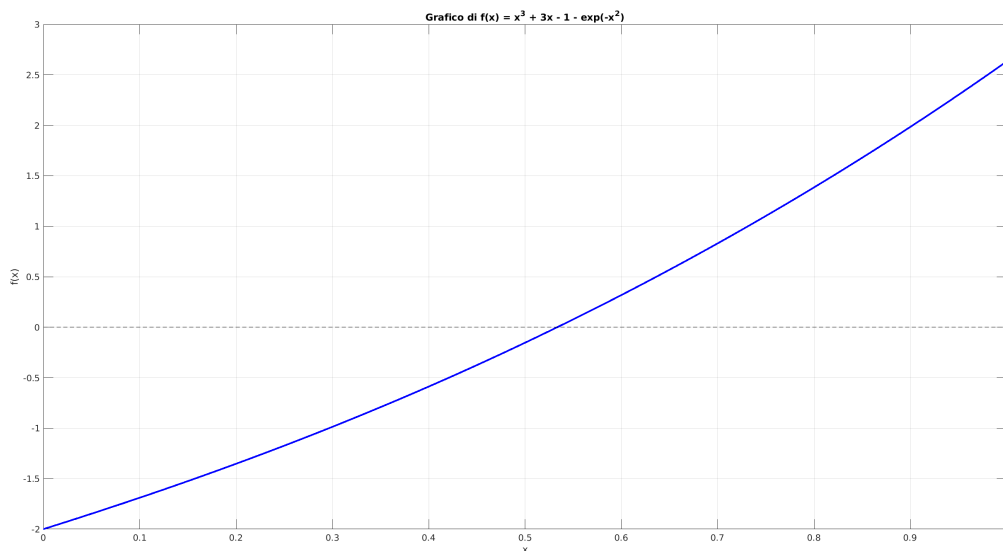
2. Poiché $f(0) \cdot f(1) < 0$, (risulta $-2 \cdot 2,63 = -5,26$) possiamo procedere.

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 $(0, 1)$. Il codice MATLAB è il seguente:

```
1  f = @(x) x.^3 + 3.*x - 1 - exp(-x.^2);
2  x = linspace(0, 1, 1000); % 1000 punti nell'intervallo [0, 1]
3  plot(x, f(x), 'b-', 'LineWidth', 2);
4  grid on;
5  xlabel('x');
6  ylabel('f(x)');
7  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 $\varepsilon \in \{10^{-1}, 10^{-2}, \dots, 10^{-10}\}$

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	$f(x_i)$
$1.0 \cdot 10^{-1}$	0.5312500000000000	4	$-1.041995243049776 \cdot 10^{-2}$
$1.0 \cdot 10^{-2}$	0.5351562500000000	7	$7.765312582933004 \cdot 10^{-3}$
$1.0 \cdot 10^{-3}$	0.5336914062500000	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:

```

1  a = 0; b = 1;
2  f = @(x) x.^3 + 3.*x - 1 - exp(-x.^2);
3  epsilon_values = 10.^(-1:-1:-10); % Tolleranze
4  results = zeros(length(epsilon_values), 3); % Preallocazione: [xi_eps,
   K_eps, f(xi_eps)]
5
6  for i = 1:length(epsilon_values)
7      epsilon = epsilon_values(i);
8      [xi, K, fx] = bisezione(a, b, f, epsilon);
9      results(i, :) = [xi, K, fx];
10 end
11
12 % Mostra la tabella
13 disp('Tabella dei risultati:');
14 disp('epsilon      xi_eps      K_eps      f(xi_eps)');
15 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 $f(b)$:

- $f(0) = \cos(0) - 0 = 1$,
- $f(\pi) = \cos(\pi) - \pi = -1 - \pi < 0$.

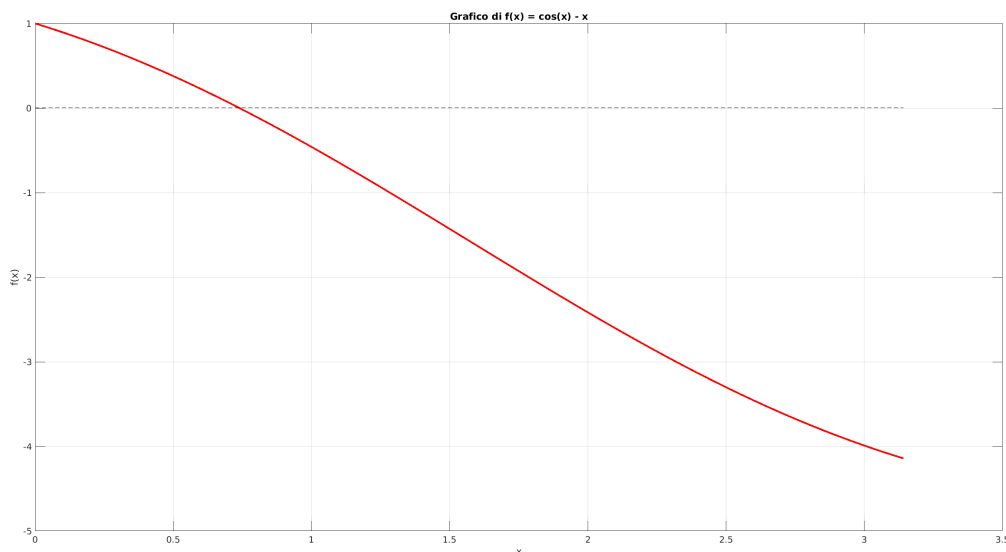
2. Poiché $f(0) \cdot f(\pi) < 0$, possiamo procedere.

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

Codice MATLAB:

```
1 f = @(x) cos(x) - x;
2 x = linspace(0, pi, 1000); % 1000 punti nell'intervallo [0, pi]
3 plot(x, f(x), 'r-', 'LineWidth', 2);
4 grid on;
5 xlabel('x');
6 ylabel('f(x)');
7 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

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

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:

```

1  a = 0; b = pi;
2  f = @(x) cos(x) - x;
3  epsilon_values = 10.^(-1:-1:-10); % Tolleranze
4  results = zeros(length(epsilon_values), 3); % Preallocazione: [xi_eps,
   K_eps, f(xi_eps)]
5
6  for i = 1:length(epsilon_values)
7      epsilon = epsilon_values(i);
8      [xi, K, fx] = bisezione(a, b, f, epsilon);
9      results(i, :) = [xi, K, fx];
10 end
11
12 % Mostra la tabella
13 disp('Tabella dei risultati:');
14 disp('epsilon      xi_eps      K_eps      f(xi_eps)');
15 disp(results);

```

Codice

```

1  % Funzioni e intervalli definiti dal problema
2  f1 = @(x) x.^3 + 3*x - 1 - exp(-x.^2); % Prima funzione
3  a1 = 0; b1 = 1; % Intervallo [a, b] per f1
4
5  f2 = @(x) cos(x) - x; % Seconda funzione
6  a2 = 0; b2 = pi; % Intervallo [a, b] per f2
7
8  % Lista di epsilon
9  epsilons = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-
   10];

```

```

10
11 % Risoluzione per il primo caso
12 solve_case(f1, a1, b1, epsilons, 'f1(x) = x^3 + 3x - 1 - e^{-x^2}');
13
14 % Risoluzione per il secondo caso
15 solve_case(f2, a2, b2, epsilons, 'f2(x) = cos(x) - x');
16
17 % Funzione per risolvere ogni caso
18 function solve_case(f, a, b, epsilons, case_name)
19     fprintf('\nSoluzione per %s:\n', case_name);
20
21     % (a) Verifica che f(a)*f(b) < 0
22     fa = f(a);
23     fb = f(b);
24     fprintf('(a) f(a)*f(b) = %.3f (segno opposto: %s)\n', fa * fb, ...
25         string(fa * fb < 0));
26     if fa * fb >= 0
27         error('f(a) e f(b) devono avere segni opposti');
28     end
29
30     % (b) Tracciamento del grafico
31     fprintf('(b) Tracciamento del grafico di f(x) su [%f, %f]\n', a, b);
32     fplot(f, [a b]);
33     hold on;
34     grid on;
35     plot(a, f(a), 'ro', 'DisplayName', 'f(a)');
36     plot(b, f(b), 'bo', 'DisplayName', 'f(b)');
37     xlabel('x'); ylabel('f(x)');
38     title(['Grafico di f(x) - Caso ', case_name]);
39     legend show;
40
41     % (c) Dimostrazione analitica: fatta in modo separato (se
42     necessario)
43
44     % (d) Tabella dei risultati per vari epsilon
45     fprintf('(d) Calcolo del metodo di bisezione per diverse tolleranze
46     epsilon:\n');
47     fprintf('epsilon      xi      K      f(xi)\n');
48     fprintf('-----\n');
49     for epsilon = epsilons
50         [xi, K, fx] = bisezione(a, b, f, epsilon);
51         fprintf('%e  %.15f  %d  %.15e\n', epsilon, xi, K, fx);
52     end
53 end
54

```

Descrizione del Codice

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 $f(\xi_\epsilon)$.

3. Tabelle dei Risultati:

- Si stampano le tabelle per ogni caso, con i valori di ϵ , ξ_ϵ , K_ϵ , e $f(\xi_\epsilon)$.