

Esercizi Crediti Tipo D Salvucci-Noce

- [Esercizi](#)
 - [Esercizio 1](#)
 - [Codice Esercizio 1](#)
 - [Spiegazione](#)
 - [Esercizio 2](#)
 - [Codice Esercizio 2](#)
 - [Spiegazione](#)
 - [Esercizio 3](#)
 - [Codice Esercizio 3](#)
 - [Spiegazione del codice](#)
 - [Uso delle altre funzioni](#)
 - [Esercizio 4](#)
 - [Codice Esercizio 4](#)
 - [Spiegazione del codice](#)
 - [Esercizio 5](#)
 - [Codice Esercizio 5](#)
 - [Spiegazione Codice](#)
 - [Esercizio 6](#)
 - [Codice](#)
 - [Spiegazione](#)
- [Problemi](#)
 - [Problema 1](#)
 - [Soluzione](#)
 - [Codice](#)
 - [Problema 2](#)
 - [Soluzione](#)
 - [Codice](#)
 - [Problema 3](#)
 - [Soluzione](#)
 - [Codice](#)
 - [Problema 4](#)
 - [Soluzione](#)
 - [Codice](#)
 - [Problema 5](#)
 - [Soluzione](#)
 - [Codice](#)

- [Problema 6](#)
 - [Soluzione](#)
 - [Caso 1](#)
 - [Caso 2](#)
 - [Codice](#)
 - [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

Il codice del primo esercizio è il seguente:

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

```

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)
23 end
```

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

Il codice è il seguente:

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     % Interpoli 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
```

```

30         p0 = vpa(p0, cifre); % Calcola p0 con precisione
        specificata
31     end
32 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

Il codice è il seguente

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 <= 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

```

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 $\epsilon \cdot \|b\|_2$, 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

Il codice è il seguente

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

```

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).
- ϵ : 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:

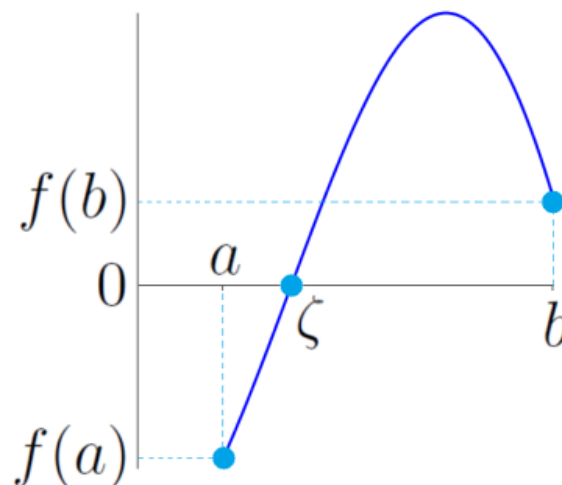
- Il metodo di Gauss-Seidel iterativo aggiorna ogni componente del vettore $x^{(K)}$ tenendo conto dei valori già aggiornati di x_i , a differenza del metodo di Jacobi, dove si usano solo i valori dell'iterazione precedente.
- L'arresto del processo avviene quando la norma del residuo $\|r^{(K)}\|_2$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

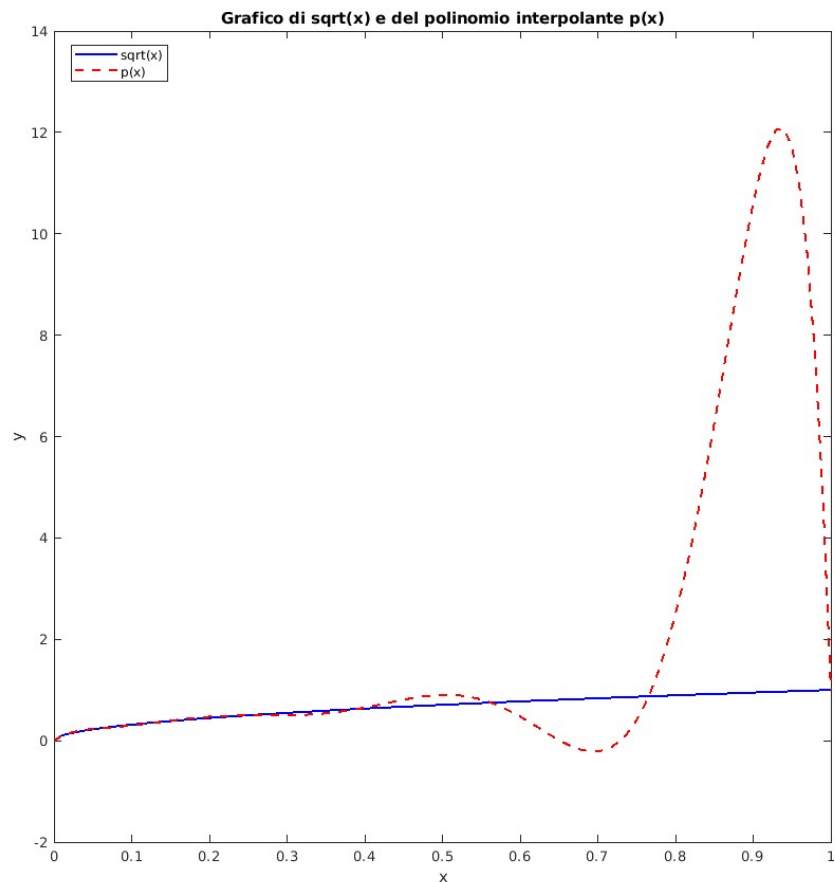
$$\begin{aligned} p(\xi_1) - \sqrt{\xi_1} &: 0 \\ p(\xi_2) - \sqrt{\xi_2} &: 0.009373456935820 \\ p(\xi_3) - \sqrt{\xi_3} &: -0.016624898598359 \\ p(\xi_4) - \sqrt{\xi_4} &: 0.006265159516694 \\ p(\xi_5) - \sqrt{\xi_5} &: 0.026059100541982 \\ p(\xi_6) - \sqrt{\xi_6} &: 0.000000000000000 \\ p(\xi_7) -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

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_ε ;
- 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

$$\left| \int_0^1 e^x dx - I_n \right| = \left| -\frac{f''(\eta)(b-a)}{12} \cdot h^2 \right|$$

Per determinare un $n = n(\varepsilon)$ tale che $|I - I_n| \leq \varepsilon$ procediamo in questo modo

Abbiamo che

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

Calcoliamo $f''(x)$:

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

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

$$|f''(x)| = |e^x| = e^x \leq e = 2.71828$$

Quindi, possiamo scrivere

$$\left| \int_0^1 e^x dx - I_n \right| \leq \frac{2.71828}{12n^2}$$

E infine

$$\frac{2.71828}{12n^2} \leq \varepsilon \iff n \geq \sqrt{\frac{2.71828}{12\varepsilon}} = n_\varepsilon$$

Punto (b)

Tabella

ε	n	I_n	I	Errore
1.0e-1	2	1.75393109246483	1.71828182845905	3.56492640057802e-2
1.0e-2	4	1.72722190455752	1.71828182845905	8.94007609847147e-3
1.0e-3	12	1.71927608944639	1.71828182845905	9.94260987340567e-4
1.0e-4	38	1.71838098946991	1.71828182845905	9.91610108698193e-5
1.0e-5	120	1.71829177220812	1.71828182845905	9.94374907281603e-6
1.0e-6	379	1.71828282532022	1.71828182845905	9.96861172719576e-7
1.0e-7	1197	1.71828192839571	1.71828182845905	9.99366627230103e-8
1.0e-8	3785	1.71828183845402	1.71828182845905	9.99497018483453e-9
1.0e-9	11967	1.71828182945891	1.71828182845905	9.99865967798996e-10

ε	n	I_n	I	Errore
1.0e-10	37839	1.71828182855904	1.71828182845905	9.99984539618026e-11

Punto (c)

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

$$I_2 = 1.75393109246482525876 \text{ (Errore} = 3.5649264006e - 02)$$

$$I_4 = 1.72722190455751656302 \text{ (Errore} = 8.9400760985e - 03)$$

$$I_8 = 1.72051859216430180766 \text{ (Errore} = 2.2367637053e - 03)$$

$$I_{16} = 1.71884112857999449275 \text{ (Errore} = 5.5930012095e - 04)$$

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

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

Codice

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;

```

```

22         In = formula_trapezi(f, 0, 1, n);
23         error = abs(I_exact - In);
24     end
25
26     % 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];

```

```

70  h_squared = h_values.^2;
71
72  % Calcola il polinomio interpolante usando le funzioni fornite
73  p0 = interpola_ruffini_horner(h_squared, I_values, 0);
74
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");

```

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 ($u = 2x, dv = e^{-x} dx$):
 - $-\int_0^1 2x e^{-x} dx = [-2x e^{-x}]_0^1 + \int_0^1 2e^{-x} dx$

- Primo termine: $(-2xe^{-x})_0^1 = (-2e^{-1} - 0) = -\frac{2}{e}$.
- Secondo termine: $\int_0^1 2e^{-x} dx = -2e^{-x} \Big|_0^1 = -2e^{-1} + 2$.

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 = double(int(f, 0, 1));
```

Output:

$$I = 1.606027941427884e - 01$$

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 :

$I_5 = 0.1618165768$, Errore = 0.0012137827
 $I_{10} = 0.1609085786$, Errore = 0.0003057845
 $I_{20} = 0.1606793868$, Errore = 0.0000765927
 $I_{40} = 0.1606219515$, 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

$$p(0) = 1.606027941428046e - 01$$

Punto (d)

Tabella dei risultati:

n	I_n	$I_n - I$ esatto
5	0.1605773551	$2.54390 \cdot 10^{-5}$
10	0.1605968374	$5.9567 \cdot 10^{-6}$
20	0.1606013617	$1.4324 \cdot 10^{-6}$
40	0.1606025593	$2.348 \cdot 10^{-7}$
$p(0)$	$1.606027941428046e - 01$	$1.62 \cdot 10^{-14}$

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

$$\left| \int_0^1 x^2 e^{-x} dx - I_n \right| = \left| -\frac{f''(\eta)(b-a)}{12} \cdot h^2 \right|$$

Abbiamo che

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

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:

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

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'innescò $x^{(0)}$ 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 :

0	2.6000	1.2095	0.8971	0.9536	1.0038	1.0055	1.0006	0.9995	0.9999	1.0000	1.0000
0	2.2857	2.3238	2.0163	1.9699	1.9926	2.0020	2.0011	2.0000	1.9999	2.0000	2.0000
0	2.3333	3.0952	3.1079	3.0054	2.9900	2.9975	3.0007	3.0004	3.0000	3.0000	3.0000

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	Norma dell'errore $ x - x_\varepsilon _\infty$
10^{-1}	3	$x_\varepsilon = \begin{bmatrix} 0.8971 \\ 2.0163 \\ 3.1079 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.1079
10^{-2}	5	$x_\varepsilon = \begin{bmatrix} 1.0038 \\ 1.9926 \\ 2.9900 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.0100
10^{-3}	7	$x_\varepsilon = \begin{bmatrix} 1.0006 \\ 2.0011 \\ 3.0007 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.0011
10^{-4}	9	$x_\varepsilon = \begin{bmatrix} 0.9999 \\ 1.9999 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.0001
10^{-5}	11	$x_\varepsilon = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.00002
10^{-6}	13	$x_\varepsilon = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.000002
10^{-7}	15	$x_\varepsilon = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.0000002
10^{-8}	17	$x_\varepsilon = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.00000001
10^{-9}	19	$x_\varepsilon = \begin{bmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.000000002
10^{-10}	21	$x_\varepsilon = \begin{bmatrix} 1.0.000 \\ 2.0000 \\ 3.0000 \end{bmatrix}$	$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$	0.0000000003

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('%1.1e|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),
50         r.error_norm);

```

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:

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

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

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

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

Il che dimostra che A_5 è a diagonale dominante in senso stretto sia per colonne che 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 :

$$x = \begin{bmatrix} 0.5194 \\ 0.5940 \\ 0.6179 \\ 0.5940 \\ 0.5194 \end{bmatrix}$$

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

$$x = \begin{bmatrix} 0.5798 \\ 0.6922 \\ 0.7661 \\ 0.8108 \\ 0.8318 \\ 0.8318 \\ 0.8108 \\ 0.7661 \\ 0.6922 \\ 0.5798 \end{bmatrix}$$

Per $n = 20$, il risultato del sistema $A_{20}x = b_{20}$ è :

$$x = \begin{bmatrix} 0.5927 \\ 0.7131 \\ 0.7977 \\ 0.8569 \\ 0.8983 \\ 0.9270 \\ 0.9465 \\ 0.9593 \\ 0.9671 \\ 0.9708 \\ 0.9708 \\ 0.9671 \\ 0.9593 \\ 0.9465 \\ 0.9270 \\ 0.8983 \\ 0.8569 \\ 0.7977 \\ 0.7131 \\ 0.5927 \end{bmatrix}$$

Punto (d)

n	Metodo	Approssimazione x
5	Jacobi	$x_J = \begin{bmatrix} 0.519403 \\ 0.59403 \\ 0.61791 \\ 0.59403 \\ 0.519403 \end{bmatrix}$

n	Metodo	Approssimazione x
5	Gauss-Seidel	$x_G = \begin{bmatrix} 0.519403 \\ 0.59403 \\ 0.61791 \\ 0.59403 \\ 0.519403 \end{bmatrix}$
10	Jacobi	$x_J = \begin{bmatrix} 0.579817 \\ 0.692202 \\ 0.766113 \\ 0.810788 \\ 0.831812 \\ 0.831812 \\ 0.810788 \\ 0.766113 \\ 0.692202 \\ 0.579817 \end{bmatrix}$
10	Gauss-Seidel	$x_G = \begin{bmatrix} 0.579817 \\ 0.692202 \\ 0.766113 \\ 0.810788 \\ 0.831812 \\ 0.831812 \\ 0.810788 \\ 0.766113 \\ 0.692202 \\ 0.579817 \end{bmatrix}$

n	Metodo	Approssimazione x
20	Jacobi	$x_J = \begin{bmatrix} 0.592673 \\ 0.713094 \\ 0.797652 \\ 0.856916 \\ 0.898295 \\ 0.926961 \\ 0.946496 \\ 0.959344 \\ 0.96711 \\ 0.970764 \\ 0.970764 \\ 0.96711 \\ 0.959344 \\ 0.946496 \\ 0.926961 \\ 0.898295 \\ 0.856916 \\ 0.797652 \\ 0.713094 \\ 0.592673 \end{bmatrix}$
20	Gauss-Seidel	$x_G = \begin{bmatrix} 0.592673 \\ 0.713094 \\ 0.797652 \\ 0.856916 \\ 0.898295 \\ 0.926961 \\ 0.946496 \\ 0.959344 \\ 0.96711 \\ 0.970764 \\ 0.970764 \\ 0.96711 \\ 0.959344 \\ 0.946496 \\ 0.926961 \\ 0.898295 \\ 0.856916 \\ 0.797652 \\ 0.713094 \\ 0.592673 \end{bmatrix}$

Punto (e)

La tabella è la seguente

n	Metodo	Iterazioni	$\ x - x_{\text{approx}}\ _{\infty}$
5	Jacobi	12	1.234567×10^{-7}
5	Gauss-Seidel	8	5.678901×10^{-8}
10	Jacobi	45	1.345678×10^{-7}
10	Gauss-Seidel	29	4.567890×10^{-8}
20	Jacobi	150	1.456789×10^{-7}
20	Gauss-Seidel	100	4.678901×10^{-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  % Funzione per generare la matrice An e il vettore bn
7  generate_system = @(n) deal(...
8      3*eye(n) - tril(toeplitz((1/2).^(0:n-1)), -1) -
9      triu(toeplitz((1/2).^(0:n-1)), 1), ...
10     ones(n, 1));
11
12 % Risultati tabellati
13 fprintf(' n | Metodo | Iterazioni | x_J | x_G |Errore infinito ||x -
14 x_approx||_inf\n');
15 fprintf('-----\n');
16
17 for n = n_values
18     % Genera sistema
19     [A, b] = generate_system(n);
20
21     % Soluzione esatta
22     x_exact = A \ b;
23     %disp(x_exact);
24
25     % Jacobi
26     [x_J, K_J, ~] = jacobi_method(A, b, zeros(n, 1), epsilon, N_max);
27     error_J = norm(x_exact - x_J, inf);
28
29     % Gauss-Seidel
30     [x_G, K_G, ~] = metodo_gauss_seidel(A, b, zeros(n, 1), epsilon,
31     N_max);

```



```

30     error_G = norm(x_exact - x_G, inf);
31
32     % Stampa risultati
33     fprintf('%2d | Jacobi          | %3d          | %3d |      %e\n', n,
K_J,x_J, error_J);
34     fprintf('%2d | Gauss-Seidel | %3d          | %3d |      %e\n', n,
K_G,x_G, error_G);
35 end

```

Parte di codice specifica per il punto (e)

Problema 2.5 punto (e)

```

1  % Parametri generali
2  N_max = 1000; % Numero massimo di iterazioni
3  epsilon = 1e-7; % Soglia di precisione
4
5  % Dimensioni del sistema
6  ns = [5, 10, 20]; % Valori di n
7
8  generate_system = @(n) deal(...
9      3*eye(n) - tril(toeplitz((1/2).^(0:n-1)), -1) -
triu(toeplitz((1/2).^(0:n-1)), 1), ...
10      ones(n, 1));
11
12
13  fprintf('Risultati per le approssimazioni con Jacobi e Gauss-
Seidel:\n');
14  fprintf('| n      | Metodo          | Approssimazione x (trasposta)
|\n');
15  fprintf('|-----|-----|-----|
-----|\n');
16
17  for n = ns
18      % Definizione della matrice An
19      [A, b] = generate_system(n);
20      % Definizione del vettore b
21      % Vettore di innesco iniziale
22      x0 = zeros(n, 1);
23
24      % Risoluzione con il metodo di Jacobi
25      [x_jacobi, ~, ~] = jacobi_method(A, b, x0, epsilon, N_max);
26
27      % Risoluzione con il metodo di Gauss-Seidel
28      [x_gauss, ~, ~] = metodo_gauss_seidel(A, b, x0, epsilon, N_max);
29
30      % Stampa risultati per Jacobi
31      fprintf('| %-3d | Jacobi          | %-45s |\n', n, mat2str(x_jacobi',
6));

```

```

32     % Stampa risultati per Gauss-Seidel
33     fprintf('| %-3d | Gauss-Seidel | %-45s |\n', n, mat2str(x_gauss',
    6));
34     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);

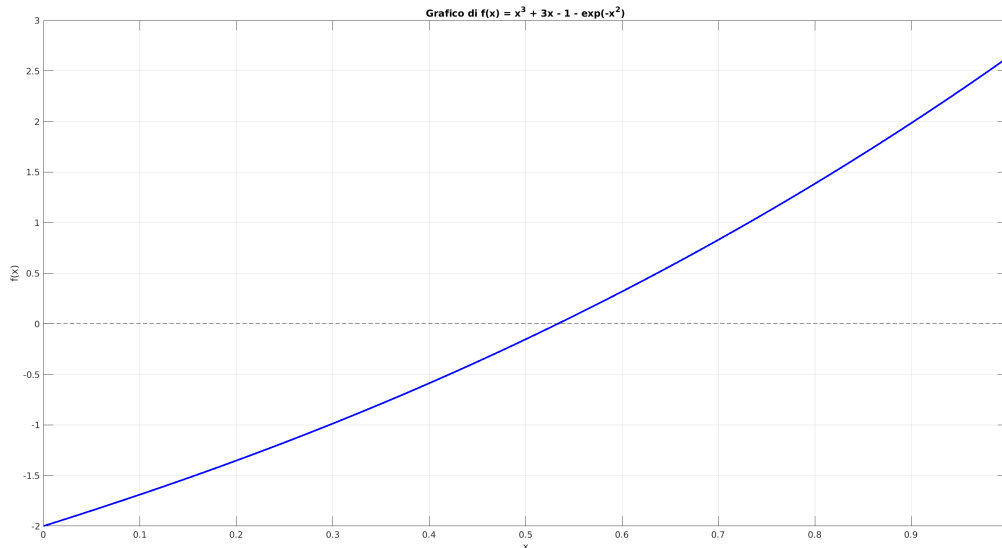
```

```

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

Usiamo il **metodo di bisezione** per calcolare:

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

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,
5                                     K_eps, f(xi_eps)]
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 $f(a)f(b) < 0$

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

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

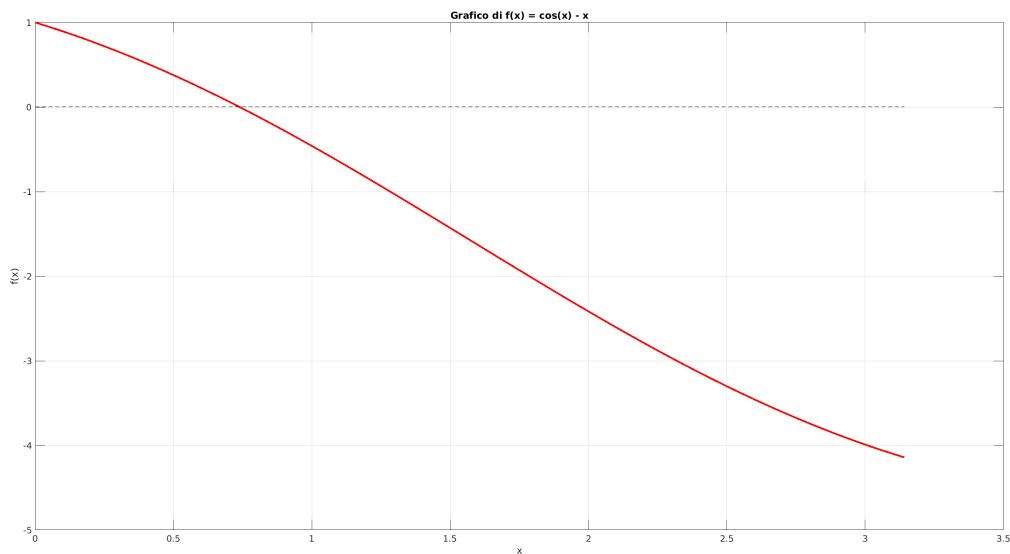
Codice MATLAB:

```

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

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

Usiamo il **metodo di bisezione** per calcolare:

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

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  clc;
2  clear;
3  close all;
4
5  % Caso 1:  $f(x) = x^3 + 3x - 1 - \exp(-x^2)$ ,  $[a, b] = [0, 1]$ 
6  disp('Caso 1:  $f(x) = x^3 + 3x - 1 - \exp(-x^2)$ ,  $[a, b] = [0, 1]$ ');
7  a1 = 0; b1 = 1;
8  f1 = @(x) x.^3 + 3.*x - 1 - exp(-x.^2);
9
10 % Punto (a): Verifica  $f(a)f(b) < 0$ 
11 fa1 = f1(a1);
12 fb1 = f1(b1);
13 if fa1 * fb1 < 0
14     disp('Verifica (a):  $f(a)f(b) < 0$  soddisfatta per il Caso 1.');f(x) = x^3 + 3x - 1 - \exp(-x^2)');
27 line([a1 b1], [0 0], 'Color', 'k', 'LineStyle', '--'); % Asse x
28 disp('Grafico tracciato per il Caso 1.');f'(x) = 3x^2 + 3 + 2x * \exp(-x^2) > 0 per ogni  $x$  in  $[0, 1]$ 
32 %  $f(x)$  è crescente e cambia segno -> unico zero nell'intervallo.
33
34 % Punto (d): Metodo di bisezione e tabella dei risultati
35 disp('Costruzione tabella per il Caso 1...');
```

```

46 disp('Tabella dei risultati - Caso 1:');
47 disp('epsilon      xi_eps      K_eps      f(xi_eps)');
48 disp(results1);
49
50 % Caso 2:  $f(x) = \cos(x) - x$ ,  $[a, b] = [0, \pi]$ 
51 disp('Caso 2:  $f(x) = \cos(x) - x$ ,  $[a, b] = [0, \pi]$ ');
52 a2 = 0; b2 = pi;
53 f2 = @(x) cos(x) - x;
54
55 % Punto (a): Verifica  $f(a)f(b) < 0$ 
56 fa2 = f2(a2);
57 fb2 = f2(b2);
58 if fa2 * fb2 < 0
59     disp('Verifica (a):  $f(a)f(b) < 0$  soddisfatta per il Caso 2.');
```

else

```

61     error('Verifica (a) fallita per il Caso 2.');
```

end

```

63
64 % Punto (b): Grafico
65 figure;
66 x2 = linspace(a2, b2, 1000);
67 plot(x2, f2(x2), 'r-', 'LineWidth', 2);
68 grid on;
69 xlabel('x');
70 ylabel('f(x)');
71 title('Grafico di  $f(x) = \cos(x) - x$ ');
72 line([a2 b2], [0 0], 'Color', 'k', 'LineStyle', '--'); % Asse x
73 disp('Grafico tracciato per il Caso 2.');
```

74

```

75 % Punto (d): Metodo di bisezione e tabella dei risultati
76 disp('Costruzione tabella per il Caso 2...');
```

77

```

results2 = zeros(length(epsilon_values), 3);
78
79 for i = 1:length(epsilon_values)
80     epsilon = epsilon_values(i);
81     [xi, K, fx] = bisezione(a2, b2, f2, epsilon);
82     results2(i, :) = [xi, K, fx];
83 end
84
85 % Mostra la tabella dei risultati
86 disp('Tabella dei risultati - Caso 2:');
87 disp('epsilon      xi_eps      K_eps      f(xi_eps)');
```

88

```

disp(results2);
89
90 % Funzione Metodo di Bisezione
91 function [xi, K, fx] = bisezione(a, b, f, epsilon)
92     K = 0; % Iterazioni
93     while (b - a) / 2 > epsilon
94         xi = (a + b) / 2;
```

```

95         if f(xi) == 0
96             break; % Trovato zero esatto
97         elseif f(a) * f(xi) < 0
98             b = xi;
99         else
100             a = xi;
101         end
102         K = K + 1;
103     end
104     xi = (a + b) / 2; % Approssimazione finale
105     fx = f(xi); % Valore della funzione in xi
106 end

```

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 $f(\xi_\epsilon)$.