Esercizi Matlab Crediti D

Domenico Sabatini (0324904)
Mihai Alexandru Sandu (0327308)

May 2, 2025

Contents

1 Esercizi

1.1 Esercizio 1.1

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

Esercizio 1.11. Scrivere un programma MATLAB che implementa l'algoritmo descritto. Il programma deve:

- prendere in input tre vettori (a componenti reali) $[x_0, x_1, \ldots, x_n]$, $[y_0, y_1, \ldots, y_n]$, $[t_1, \ldots, t_m]$, con $x_0, \ldots, x_n$ tutti distinti;
- restituire in output il vettore $[p(t_1), \ldots, p(t_m)]$ che contiene le valutazioni nei punti $t_1, \ldots, t_m$ del polinomio $p(x)$ interpolante i dati $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.

Verificare sperimentalmente la correttezza del programma, utilizzandolo in particolare per riottenere il risultato dell'Esempio 1.5 e per risolvere nuovamente l'Esercizio 1.10.

```
x = [0,1,2,3];
y = [0,3,1,1];
t = [2.3];

x1 = [0,1,4,9];
y1 = [0,1,8,27];
t1 = [0.5, 4];

function p_t = newton_interpolation(x, y, t)
    n = length(x);
    m = length(t);

    % Calcolo della tabella delle differenze divise
    div_diff = zeros(n, n);
    div_diff(:,1) = y(:); % Prima colonna    y

    for j = 2:n
        for i = 1:n-j+1
            div_diff(i,j) = (div_diff(i+1,j-1) - div_diff(i,j-1)) / (x(i+j-1) - x(i));
        end
    end

    coeff = div_diff(1, :);

    p_t = zeros(size(t));

    for k = 1:m
        tk = t(k);
        p_t(k) = coeff(1);
        term = 1;
        for j = 2:n
            term = term * (tk - x(j-1));
            p_t(k) = p_t(k) + coeff(j) * term;
        end
    end
end

% TEST esempio 1.5
p_t = newton_interpolation(x, y, t);

for i = 1:length(t)
    fprintf('p(%.2f) = %.6f\n', t(i), p_t(i));
end

% TEST esercizio 1.10
p_t = newton_interpolation(x1, y1, t1);

for i = 1:length(t1)
    fprintf('p(%.2f) = %.6f\n', t1(i), p_t(i));
end
```

1.2 Esercizio 1.2

Implementazione della formula dei trapezi.
Viene richiesto di scrivere un codice Matlab che implementi la formula dei trapezi e che se ne verifichi la correttezza calcolando l'approssimazione di $\int_0^2 xe^x\,dx$ ottenuta con I_n per $n = 20, 40, 80, 160, 320, 640, 1280, 2560, 5120$.

```
% Funzione da integrare
f = @(x) x .* exp(x);

% Estremi dell'intervallo
a = 0;
b = 1;

function I = formula_trapezi(f, a, b, n)
    x = linspace(a, b, n + 1);
    y = f(x);
    h = (b - a) / n;
    I = h * (sum(y) - 0.5 * (y(1) + y(end)));
end

% Valore esatto dell'integrale
I_esatto = integral(f,0,2);
fprintf("Valore esatto: %.15f\n", I_esatto);

% Vettore dei vari n
n_values = [20, 40, 80, 160, 320, 640, 1280, 2560, 5120];

% Calcolo e stampa degli errori
fprintf("  n\t\tI_n\t\tErrore\n");
for n = n_values
    I_n = formula_trapezi(f, 0, 2, n);
    errore = abs(I_n - I_esatto);
    fprintf("%5d\t%.15f\t%.10f\n", n, I_n, errore);
end
```

1.3 Esercizio 1.3

Implementazione del metodo di estrapolazione.

Esercizio 2.4. Usando i programmi creati per risolvere gli Esercizi 1.11 e 2.2, scrivere un programma MATLAB che implementa il metodo di estrapolazione. Il programma deve:

- prendere in input gli estremi a, b di un intervallo, una funzione $f(x)$ definita su $[a, b]$ e un vettore $[n_0, n_1, \ldots, n_m]$ di numeri $n_0, n_1, \ldots, n_m \geq 1$ tutti distinti;

- restituire in output il valore estrapolato $p(0)$, dove $p(x)$ è il polinomio d'interpolazione dei dati (h_0^2, I_{n_0}), (h_1^2, I_{n_1}), …, (h_m^2, I_{n_m}) e $h_0, h_1, \ldots, h_m$ sono i passi di discretizzazione delle formule dei trapezi I_{n_0}, I_{n_1}, …, I_{n_m} per approssimare $\int_a^b f(x)\mathrm{d}x$.

Verificare la correttezza del programma usandolo in particolare per riottenere il risultato dell'Esempio 2.4(c).

```
f = @(x) x .* exp(x);
n_vect = [12,24,30]
a = 0
b = 2

function I = formula_trapezi(f, a, b, n)
        x = linspace(a, b, n + 1);
        y = f(x);
        h = (b - a) / n;
        I = h * (sum(y) - 0.5 * (y(1) + y(end)));
    end

function p_t = newton_interpolation(x, y, t)
        n = length(x);
        m = length(t);

    % Calcolo della tabella delle differenze divise
        div_diff = zeros(n, n);
        div_diff(:,1) = y(:); % Prima colonna    y

        for j = 2:n
            for i = 1:n-j+1
                div_diff(i,j) = (div_diff(i+1,j-1) - div_diff(i,j-1)) / (x(i+j-1) - x(i)
                    );
            end
        end

        coeff = div_diff(1, :);

        p_t = zeros(size(t));

        for k = 1:m
            tk = t(k);
            p_t(k) = coeff(1);
            term = 1;
            for j = 2:n
                term = term * (tk - x(j-1));
                p_t(k) = p_t(k) + coeff(j) * term;
            end
        end
    end

function p0 = estrapolazione(f, a, b, n_vect)

    m = length(n_vect);

    for i = 1:m
       I(i) = formula_trapezi(f,a,b,n_vect(i));
       H(i) = ((b - a) / n_vect(i))^2;
    end

```

```
51      p0 = newton_interpolation(H,I,0);
52
53  end
54
55  poli = estrapolazione(f,a,b,n_vect);
56
57  fprintf('\n%.15f',poli);
```

1.4 Esercizio 1.4

Implementazione del metodo di Jacobi.

Esercizio 4.2. Scrivere un programma MATLAB che implementa il metodo di Jacobi. Il programma deve:

- prendere in input la matrice A e il termine noto $\mathbf{b}$ del sistema lineare da risolvere $A\mathbf{x} = \mathbf{b}$, una soglia di precisione ε, un vettore d'innesco $\mathbf{x}^{(0)}$ e un numero massimo d'iterazioni consentite $N_{\max}$;
- restituire in output il primo vettore $\mathbf{x}^{(K)}$ calcolato dal metodo di Jacobi (con $0 \leq K \leq N_{\max}$) che soddisfa la condizione di arresto del residuo $\|\mathbf{r}^{(K)}\|_2 \leq \varepsilon \|\mathbf{b}\|_2$, il relativo indice K che conta il numero d'iterazioni effettuate, e la norma $\|\mathbf{r}^{(K)}\|_2$ del residuo a cui ci si arresta. Se nessuno dei vettori $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(N_{\max})}$ soddisfa la condizione di arresto del residuo allora il programma deve restituire in output $\mathbf{x}^{(N_{\max})}$, il relativo indice $N_{\max}$, e la norma $\|\mathbf{r}^{(N_{\max})}\|_2$ dell'ultimo residuo.

```
function [x, K, res_norm] = jacobi(A, b, eps, x0, Nmax)
    % Metodo di Jacobi per risolvere Ax = b
    n = length(b);
    x_old = x0;
    D = diag(diag(A));
    P = inv(D)*(D - A);
    q = inv(D)*b;

    for k = 1:Nmax
        x = P * x_old + q;
        r = b - A * x;
        res_norm = norm(r, 2);

        if res_norm <= eps * norm(b, 2)
            K = k;
            return
        end
        x_old = x;
    end

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

2 Problemi

2.1 Problema 2.1

Interpolazione della funzione $\sqrt{x}$.

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

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

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

Calcolare il vettore (colonna)

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

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

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

(a) Calcolo del vettore degli errori

Sono stati scelti come nodi di interpolazione i punti $x_i = \left\{0, \frac{1}{64}, \frac{4}{64}, \ldots, 1\right\}$ e sono stati calcolati i corrispondenti valori $y_i = \sqrt{x_i}$.

Il polinomio di interpolazione $p(x)$ è stato ottenuto utilizzando il codice già ricavato per svolgere l'Esercizio 1.1. Successivamente è stato valutato in 21 punti $\zeta_i = \frac{i-1}{20}$ per $i = 1, \ldots, 21$, confrontando i risultati con $\sqrt{\zeta_i}$.

Codice Matlab:

```
x_nodes = [0, 1/64, 4/64, 9/64, 16/64, 25/64, 36/64, 49/64, 1];
y_nodes = sqrt(x_nodes);

zeta = (0:20) / 20;
sqrt_zeta = sqrt(zeta);

function p_t = newton_interpolation(x, y, t)
    n = length(x);
    m = length(t);

    % Calcolo della tabella delle differenze divise
    div_diff = zeros(n, n);
    div_diff(:,1) = y(:); % Prima colonna è y

    for j = 2:n
        for i = 1:n-j+1
            div_diff(i,j) = (div_diff(i+1,j-1) - div_diff(i,j-1)) / (x(i+j-1) - x(i));
        end
    end

    coeff = div_diff(1, :);

    p_t = zeros(size(t));

    for k = 1:m
        tk = t(k);
        p_t(k) = coeff(1);
        term = 1;
        for j = 2:n
```

```
            term = term * (tk - x(j-1));
            p_t(k) = p_t(k) + coeff(j) * term;
        end
    end
end


p = newton_interpolation(x_nodes, y_nodes, zeta);

diff_vector = p - sqrt_zeta;


% stampo il vettore
disp('Vettore differenze:');
for i = 1:length(diff_vector)
    fprintf('%d: %1.5f\n', i, diff_vector(i));
end
```

Il vettore delle differenze mostra quanto il polinomio d'interpolazione si discosta dalla funzione esatta $\sqrt{x}$ in ciascun punto ζ_i.

Output del vettore

```
Vettore differenze (per verifica vedi script Problema21):
1: -0.00000
2:  0.00937
3: -0.01662
4:  0.00627
5:  0.02606
6:  0.00000
7: -0.04680
8: -0.05284
9:  0.01904
10: 0.13666
11: 0.19597
12: 0.07022
13: -0.29867
14: -0.79383
15: -1.04786
16: -0.46169
17: 1.60012
18: 5.33760
19: 9.64872
20: 10.73148
21: -0.00000
```

(b) Grafico di $\sqrt{x}$ e del polinomio $p(x)$

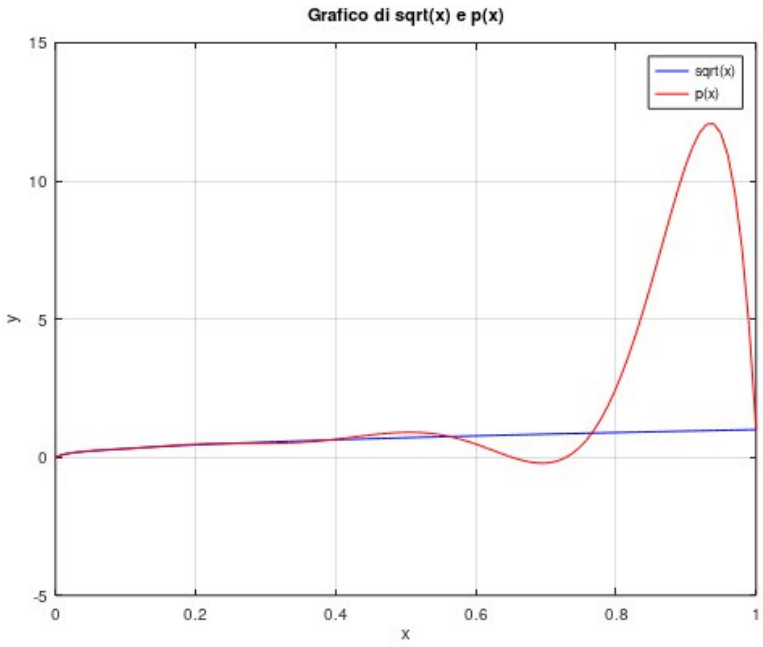


Figure 1: Confronto tra $\sqrt{x}$ (blu) e polinomio d'interpolazione $p(x)$ (rosso).

Il grafico per mostrare il confronto tra la funzione $\sqrt{x}$ e il polinomio $p(x)$ è stato creato usando il seguente codice MatLab:

```
% Creazione di punti per il grafico
x_plot = linspace(0, 1, 100);
% Valutazione del polinomio nei punti x_plot
p_plot = polyval(p, x_plot);
% Valutazione della funzione sqrt(x) nei punti x_plot
sqrt_plot = sqrt(x_plot);

figure;
plot(x_plot, sqrt_plot, 'b-', 'DisplayName', 'sqrt(x)');
hold on;
plot(x_plot, p_plot, 'r-', 'DisplayName', 'p(x)');
xlabel('x');
ylabel('y');
title('Grafico di sqrt(x) e p(x)');
legend;
```

2.2 Problema 2.2

Approssimazione dell'integrale $\int_0^1 e^x\,dx$ *con la formula dei trapezi.*

Problema 2.2. 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

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

(a) Per ogni fissato $\varepsilon > 0$ determinare un $n = n(\varepsilon)$ tale che $|I - I_n| \leq \varepsilon$.

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

(c) Calcolare le approssimazioni di I ottenute con le formule dei trapezi I_2, I_4, I_8, I_{16} e confrontarle con il valore esatto I.

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

(a)

Si applica il Teorema dell'errore o del resto della formula dei trapezi, secondo il quale:

$$|I - I_n| = |-\frac{(b-a)f''(\mu)}{12}h^2|$$

Dunque:

$$|I - I_n| = |-\frac{e}{12n^2}| \leq \frac{e}{12n^2} \leq \varepsilon \quad \rightarrow \quad n^2 \geq \frac{e}{12\varepsilon} \quad \rightarrow n \geq \sqrt{\frac{e}{12\varepsilon}}$$

$$n(\varepsilon) = \sqrt{\frac{e}{12\varepsilon}}$$

(b) Determinazione di $n(\varepsilon)$ per vari ε e costruzione della tabella:

Il valore esatto dell'integrale è:

$$I = \int_0^1 e^x\,dx = e - 1 \approx 1.718281828459045$$

Abbiamo implementato un algoritmo che, per ogni $\varepsilon \in \{10^{-1}, \ldots, 10^{-10}\}$, determina il numero minimo n tale che l'errore $|I - I_n| \leq \varepsilon$, dove I_n è l'approssimazione dell'integrale con la formula dei trapezi di ordine n.

Codice Matlab dell'algoritmo:

```
function I = formula_trapezi(f, a, b, n)
    x = linspace(a, b, n + 1);
    y = f(x);
    h = (b - a) / n;
    I = h * (sum(y) - 0.5 * (y(1) + y(end)));
end
```

Epsilon	n	I_n	Error
1.0e-01	2	1.753931092464826	3.56493e-02
1.0e-02	5	1.724005619782788	5.72379e-03
1.0e-03	16	1.718841128579994	5.59300e-04
1.0e-04	48	1.718343976513113	6.21481e-05
1.0e-05	151	1.718288108448857	6.27999e-06
1.0e-06	476	1.718282460433049	6.31974e-07
1.0e-07	1506	1.718281891593030	6.31340e-08
1.0e-08	4760	1.718281834778786	6.31974e-09
1.0e-09	15051	1.718281829091139	6.32093e-10
1.0e-10	47595	1.718281828522253	6.32083e-11

Figure 2: Tabella di confronto

(c) Approssimazioni per $n = 2, 4, 8, 16$

Val	I	Error: |I_n - I|
I2	1.753931092464826	3.56493e-02
I4	1.727221904557517	8.94008e-03
I8	1.720518592164302	2.23676e-03
I16	1.718841128579994	5.59300e-04

(d) Interpolazione e calcolo di $p(0)$

Abbiamo considerato il polinomio d'interpolazione $p(x)$ dei valori I_2, I_4, I_8, I_{16} in funzione dei nodi $h_2^2, ..., h_{16}^2$ (dove $h_n = (1-0)/n$), e valutato il polinomio nel punto 0 ($p(0)$).

- $p(0) = 1.718281828460388$
- Errore: $|p(0) - I| = 1.343 \cdot 10^{-12}$

Questo risultato mostra che valutare il polinomio d'interpolazione $p(x)$ come definito sopra nel punto 0 è un'approssimazione molto più accurata di $\int_0^1 e^x\,dx$ rispetto all'utilizzo delle singole formule dei trapezi I_2, I_4, I_8, I_{16}.

Appendice

```
% Funzione da integrare
f = @(x) exp(x);

% Calcolo dell'integrale
a = 0;
b = 1;
I_exact = integral(f,a,b);
fprintf('Valore esatto dell integrale: %.15f\n\n', I_exact);

%% Parte (a) e (b): Determinazione di n(eps) per vari valori di eps
for i = 1:10
    epsilon_values(i) = 10^(-i);
end

n_values = zeros(size(epsilon_values));
```

```
I_n_values = zeros(size(epsilon_values));
errors = zeros(size(epsilon_values));

%% Funzione per la formula dei trapezi
function I = formula_trapezi(f, a, b, n)
    x = linspace(a, b, n + 1); % Punti equidistanti
    y = f(x); % Valutazione della funzione
    h = (b - a) / n; % Passo di discretizzazione
    I = h * (sum(y) - 0.5 * (y(1) + y(end))); % Formula dei trapezi
end

for i = 1:length(epsilon_values)
    epsilon = epsilon_values(i);
    n = ceil(sqrt(exp(1)/(12*epsilon)));
    I_n = formula_trapezi(f, a, b, n);
    n_values(i) = n;
    I_n_values(i) = I_n;
    errors(i) = abs(I_exact - I_n);
end

% tabella
fprintf('    Epsilon        n     I_n                Error\n');
fprintf('-------------------------------------------------------\n');
for i = 1:length(epsilon_values)
    fprintf('%10.1e    %4d    %.15f    %.5e\n', epsilon_values(i), n_values(i), I_n_values(
        i), errors(i));
end
fprintf('\n');

%% Parte (c): Approssimazioni per n = 2, 4, 8, 16
n_test = [2, 4, 8, 16];
I_test = arrayfun(@(n) formula_trapezi(f, a, b, n), n_test);
errors_test = abs(I_test - I_exact);

% Stampa tabella con I2, I4, I8, I16
fprintf('Val          I                  Error: |I_n - I|\n');
fprintf('-------------------------------------------------\n');
for i = 1:length(n_test)
    fprintf('I%d        %.15f      %.5e\n', n_test(i), I_test(i), errors_test(i));
end
fprintf('\n');

h_values = 1 ./ n_test; % Passi di discretizzazione h = 1/n

function p_t = newton_interpolation(x, y, t)
        n = length(x);
        m = length(t);

    % Calcolo della tabella delle differenze divise
        div_diff = zeros(n, n);
        div_diff(:,1) = y(:); % Prima colonna    y

        for j = 2:n
            for i = 1:n-j+1
                div_diff(i,j) = (div_diff(i+1,j-1) - div_diff(i,j-1)) / (x(i+j-1) - x(i)
                    );
            end
        end

        coeff = div_diff(1, :);

        p_t = zeros(size(t));

        for k = 1:m
            tk = t(k);
            p_t(k) = coeff(1);
            term = 1;
            for j = 2:n
                term = term * (tk - x(j-1));
                p_t(k) = p_t(k) + coeff(j) * term;
            end
        end
    end
```

```
87
88  function p0 = estrapolazione(f, a, b, n_vect)
89
90      m = length(n_vect);
91
92      for i = 1:m
93          I(i) = formula_trapezi(f,a,b,n_vect(i));
94          H(i) = ((b - a) / n_vect(i))^2;
95      end
96
97      p0 = newton_interpolation(H,I,0);
98
99  end
100
101 p0 = estrapolazione(f,a,b,n_test);
102
103 fprintf('p(0): %.15f\n', p0);
104 fprintf('Errore interpolazione |p(0) - I|: %.15e\n', abs(p0 - I_exact));
```

2.3 Problema 2.3

Consideriamo la funzione $f(x) = x^2e^{-x}$ *e vogliamo approssimare l'integrale* $\int_0^1 f(x)\,dx$ *con la formula dei trapezi.*

Problema 2.3. Consideriamo la funzione $f(x) = x^2\mathrm{e}^{-x}$ e indichiamo con I_n la formula dei trapezi di ordine n per approssimare $I = \int_0^1 f(x)\mathrm{d}x$.

(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 $|I_n - I| \leq \varepsilon$.

(a) Calcolo Integrale

$$I = \int_0^1 x^2 e^{-x}\,dx = -x^2 e^{-x} + \int 2x e^{-x}\,dx$$
$$= -x^2 e^{-x} - 2x e^{-x} + \int 2e^{-x}\,dx$$
$$= \left[-x^2 e^{-x} - 2x e^{-x} - 2e^{-x}\right]_0^1 = -\frac{1}{e} - \frac{2}{e} - \frac{2}{e} - (-2)$$
$$= 2 - \frac{5}{e} \approx 0.160602794$$

(INTEGRAZIONE PER PARTI $\int f(x)\,g'(x)\,dx = f(x)g(x) - \int f'(x)g(x)\,dx$)

Figure 3: Calcolo dell'integrale manualmente

Abbiamo calcolato il valore esatto dell'integrale numericamente tramite la funzione `integral` di MATLAB:

$$I \approx 0.160602794142788$$

(b) Calcolo delle approssimazioni $I_5, I_{10}, I_{20}, I_{40}$

Ancora una volta utilizziamo l'algoritmo ricavato nel Problema 2.2 per calcolare la formula dei trapezi:

```
f = @(x) x.^2 .* exp(-x);

function I = formula_trapezi(f, a, b, n)
    x = linspace(a, b, n + 1);
    y = f(x);
    h = (b - a) / n;
    I = h * (sum(y) - 0.5 * (y(1) + y(end)));
end
```

I valori ottenuti per $n = 5, 10, 20, 40$ sono riportati nella tabella del punto (d).

(c) Calcolo di $p(0)$ tramite interpolazione su h^2

Abbiamo considerato il polinomio d'interpolazione $p(x)$ dei valori $I_5, I_{10}, I_{20}, I_{40}$ in funzione dei nodi $h_5^2, ..., h_{40}^2$ (dove $h_n = (1-0)/n$), e valutato il polinomio nel punto 0 ($p(0)$).

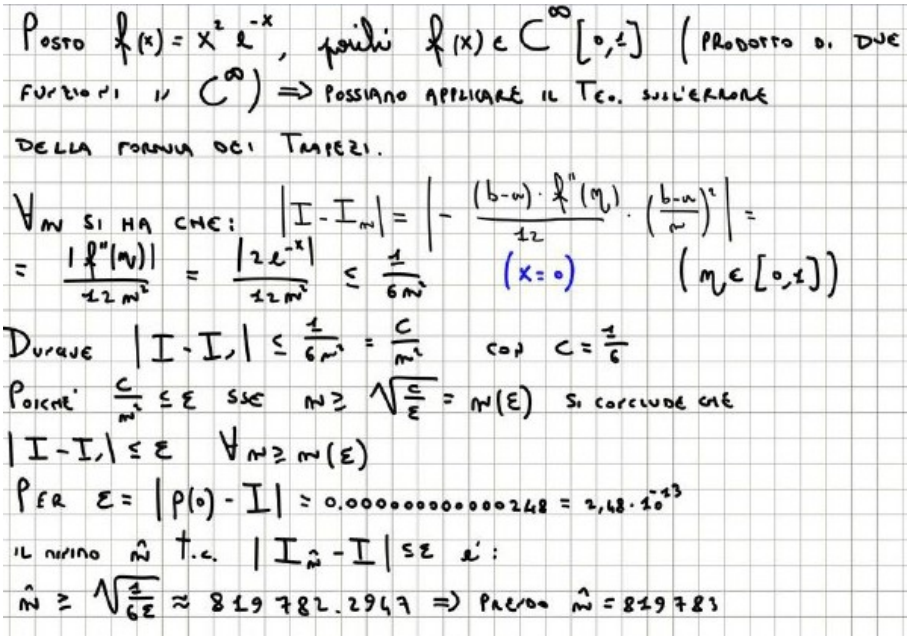
$$p(0) = 0.160602794143036$$

(d) Tabella dei risultati

n	I_n	Errore |I_n - I|
5	0.161816576820683	1.21378e-03
10	0.160908578632096	3.05784e-04
20	0.160679386811339	7.65927e-05
40	0.160621951474857	1.91573e-05
p(0)	0.160602794142805	1.61815e-14

Figure 4: Approssimazioni dell'integrale con la formula dei trapezi e interpolazione.

(e) Determinazione di n tale che $|I_n - I| \leq |p(0) - I|$



Appendice

```
f = @(x) x.^2 .* exp(-x);

%% (a) Calcolo dell'integrale
a = 0;
b = 1;
I_exact = integral(f, a, b);
fprintf('(a) Valore esatto dell integrale: %.15f\n\n', I_exact);

%% (b) Calcolo di I5, I10, I20, I40
n_vals = [5, 10, 20, 40];
```

```
I_n = zeros(size(n_vals));

function I = formula_trapezi(f, a, b, n)
        x = linspace(a, b, n + 1);
        y = f(x);
        h = (b - a) / n;
        I = h * (sum(y) - 0.5 * (y(1) + y(end)));
    end

for i = 1:length(n_vals)
    I_n(i) = formula_trapezi(f, a, b, n_vals(i));
end

%% (c) Interpolazione su h^2 e calcolo di p(0)
function p_t = newton_interpolation(x, y, t)
        n = length(x);
        m = length(t);

    % Calcolo della tabella delle differenze divise
        div_diff = zeros(n, n);
        div_diff(:,1) = y(:); % Prima colonna    y

        for j = 2:n
            for i = 1:n-j+1
                div_diff(i,j) = (div_diff(i+1,j-1) - div_diff(i,j-1)) / (x(i+j-1) - x(i)
                    );
            end
        end

        coeff = div_diff(1, :);

        p_t = zeros(size(t));

        for k = 1:m
            tk = t(k);
            p_t(k) = coeff(1);
            term = 1;
            for j = 2:n
                term = term * (tk - x(j-1));
                p_t(k) = p_t(k) + coeff(j) * term;
            end
        end
    end

function p0 = estrapolazione(f, a, b, n_vect)

    m = length(n_vect);

    for i = 1:m
       I(i) = formula_trapezi(f,a,b,n_vect(i));
       H(i) = ((b - a) / n_vect(i))^2;
    end

    p0 = newton_interpolation(H,I,0);

end

p0 = estrapolazione(f,a,b,n_vals);

fprintf('(c) p(0) = %.15f\n', p0);

%% (d) Tabella dei risultati
errors = abs(I_n - I_exact);
error_p0 = abs(p0 - I_exact);

fprintf('\n(d) Tabella riassuntiva:\n');
fprintf(' n       I_n                    Errore |I_n - I|\n');
fprintf('-----------------------------------------------\n');
for i = 1:length(n_vals)
    fprintf('%2d      %.15f      %.5e\n', n_vals(i), I_n(i), errors(i));
end
fprintf('p(0)     %.15f      %.5e\n', p0, error_p0);
```

2.4 Problema 2.4

Problema 2.4. Si consideri il sistema lineare $A\mathbf{x} = \mathbf{b}$, dove

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

(a) Si calcoli la soluzione $\mathbf{x}$ del sistema dato con MATLAB.

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

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

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

(a) Soluzione esatta

La soluzione esatta del sistema lineare è stata calcolata in Matlab tramite il seguente codice:

```
x_exact = A \ b;
fprintf('(a) Soluzione esatta:\n');
disp(x_exact);
```

La soluzione esatta restituita è il vettore colonna $x = [1.0000, 2.0000, 3.0000]^T$

(b) Metodo di Jacobi – prime 10 iterazioni

Si è applicato il metodo di Jacobi con vettore iniziale $x^{(0)} = [0, 0, 0]^T$. Le prime 10 iterazioni sono state salvate in una matrice $S \in \mathbb{R}^{3\times 12}$, dove: - le colonne da 1 a 11 sono $x^{(0)}, x^{(1)}, \ldots, x^{(10)}$, - l'ultima colonna è la soluzione esatta x.

Il codice Matlab implementa la formula iterativa:

$$x^{(k+1)} = Px^{(k)} + q, \quad \text{con } P = MatriceIterazione = D^{-1}(D - A),\ q = D^{-1}b$$

```
(b) Matrice S contenente x^(0), ..., x^(10), x esatta:
         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
```

Figure 5: Matrice 3x12

(c) Convergenza al variare di ε

Si è analizzata la convergenza del metodo di Jacobi per vari epsilon $\varepsilon \in \{10^{-1}, 10^{-2}, \ldots, 10^{-10}\}$. Per ciascuna soglia sono stati registrati:

- il numero minimo di iterazioni K_ε affinché $\|x^{(k)} - x^{(k-1)}\|_\infty \leq \varepsilon$,
- la soluzione approssimata x_ε,
- l'errore rispetto alla soluzione esatta: $\|x - x_\varepsilon\|_\infty$.

Di seguito la tabella risultante in output:

```
 epsilon = 1.0e-01, K = 5, soluzione approssimata:
    1.0038
    1.9926
    2.9900

errore norma infinito = 1.00e-02

 epsilon = 1.0e-02, K = 6, soluzione approssimata:
    1.0055
    2.0020
    2.9975

errore norma infinito = 5.50e-03

 epsilon = 1.0e-03, K = 9, soluzione approssimata:
    0.9999
    1.9999
    3.0000

errore norma infinito = 1.50e-04

 epsilon = 1.0e-04, K = 11, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 2.08e-05


 epsilon = 1.0e-05, K = 13, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 2.08e-06

 epsilon = 1.0e-06, K = 15, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 1.62e-07

 epsilon = 1.0e-07, K = 17, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 1.45e-08

 epsilon = 1.0e-08, K = 19, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 1.82e-09
```

```
 epsilon = 1.0e-09, K = 21, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 2.57e-10

 epsilon = 1.0e-10, K = 23, soluzione approssimata:
    1.0000
    2.0000
    3.0000

errore norma infinito = 3.25e-11
```

Appendice

```
%% Problema 2.4 - Metodo di Jacobi per risolvere Ax = b

clear; clc;

% Dati del sistema
A = [5, 1, 2;
    -1, 7, 1;
     0, 1, -3];

b = [13; 16; -7];

%% (a) Soluzione esatta del sistema
x_exact = A \ b;
fprintf('(a) Soluzione esatta:\n');
disp(x_exact);

%% (b) Metodo di Jacobi - prime 10 iterazioni
x0 = [0; 0; 0];  % Vettore di partenza
n = length(b);
S = zeros(n, 12);  % 10 iterazioni + x(0) + soluzione esatta

S(:,1) = x0;
x_k = x0;

function [x, K] = jacobi(A, b, eps, x0, Nmax)
    n = length(b);
    x_old = x0;
    D = diag(diag(A));
    P = inv(D)*(D - A);
    q = inv(D)*b;

    for k = 1:Nmax
        x = P * x_old + q;
        r = norm(x - x_old, inf);  % errore relativo tra iterazioni
        if r < eps
            K = k;
            return;
        end
        x_old = x;
    end
    K = Nmax;
end

for k = 1:10
     [vettor, del1] = jacobi(A,b,0,x0,k);
     S(:,k+1) = vettor;
end
S(:,12) = x_exact;

fprintf('(b) Matrice S contenente x^(0), ..., x^(10), x esatta:\n');
disp(S);

```

```
%% (c) Iterazioni per vari epsilon
for i = 1:10
    epsilons(i) = 10^(-i);
end
results = zeros(length(epsilons), 4);  % [K, errore, , soluzione approssimata]

for i = 1:length(epsilons)
    eps = epsilons(i);
    Nmax = 1000;  % un limite massimo per sicurezza
    [x_eps, K] = jacobi(A, b, eps, x0, Nmax);

    err = norm(x_exact - x_eps, inf);
    results(i,:) = [K, eps, err, NaN];

    fprintf('\n epsilon = %.1e, K = %d, soluzione approssimata:\n', eps, K);
    disp(x_eps);
    fprintf('errore norma infinito = %.2e\n', err);
end
```