Appello 28-06-22

Esercizio 1

1a) *Descrivere il problema dell'interpolazione polinomiale:*
Consiste nel trovare un polinomio che passi per un insieme dato di punti.
Dati $n+1$ punti distinti $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, si cerca un polinomio $p(x)$ di grado al più n tale che:

$$p(x_i) = y_i$$

per ogni $i = 0, \ldots, n$.
I punti distinti sono detti nodi e il polinomio è detto interpolatore.

1b) *impostare "analiticamente" il calcolo del polinomio interpolante i punti di coordinate* (x, y) *seguenti:*

x	-1	0	1	2
y	-0.1	2.03	3.5	6

1c) *In Matlab, in una stessa finestra fare il grafico del polinomio interpolante, dei punti di interpolazione e della retta di approssimazione ai minimi quadrati.*

```
clear
clc
close all

%%%%% Esercizio 1

x = [-1 0 1 2];
y = [-0.1 2.03 3.5 6];

x_val = linspace(x(1), x(end));

p = polyfit(x, y, length(x)-1);
p_val = polyval(p, x_val);

p_ls = polyfit(x, y, 1);
y_ls = polyval(p_ls, x_val);

plot(x_val, p_val)
hold on
grid on
plot(x, y, '*')
hold on
plot(x_val, y_ls, '-r')

diff = abs(p_val - y_ls);

figure
plot(x_val, diff)
```

Creazione della griglia di valutazione:

```
x_val = linspace(x(1), x(end));
```

- crea un vettore di punti equidistanti tra `x(1)` e `x(end)`. Serve per tracciare i polinomi in modo continuo e liscio.

Interpolazione polinomiale

```
p = polyfit(x, y, length(x)-1);
```

- `polyfit(x, y, n)` restituisce i coefficienti del polinomio interpolante di grado n.

```
p_val = polyval(p, x_val);
```

- `polyval(p, x_val)` valuta il polinomio interpolante sui punti di `x_val`.

Approssimazione ai minimi quadrati (retta)

```
p_ls = polyfit(x, y, 1);
```

- troviamo la retta (grado 1) che meglio approssima i dati secondo il criterio dei minimi quadrati (cioè non passa per tutti i punti, ma minimizza l'errore quadratico).

```
y_ls = polyval(p_ls, x_val);
```

- valuta la retta sui punti `x_val`.

Differenza tra i due polinomi

```
diff = abs(p_val - y_ls);
```

- calcola la differenza punto per punto tra il polinomio interpolante e la retta dei minimi quadrati.

Esercizio 2

2a) *Dato il sistema lineare $Ax = b$ con*:

$$A = \begin{bmatrix} 1 & 1 & \beta \\ -1 & 2 & 0.2 \\ 1 & -0.1 & 2 \end{bmatrix}$$

scegliere un valore β per cui il sistema abbia un'unica soluzione e il metodo di Jacobi converga alla soluzione del sistema. Motivare la scelta.

Per far si che il sistema abbia una sola soluzione è necessario che il $det(A) \neq 0$

$$det(A) = 1 \cdot \begin{bmatrix} 2 & 0.2 \\ 0.1 & 2 \end{bmatrix} - (-1) \cdot \begin{bmatrix} 1 & \beta \\ -0.1 & 2 \end{bmatrix} + 1 \cdot \begin{bmatrix} 1 & \beta \\ 2 & 0.2 \end{bmatrix} = -1.9\beta + 6.18$$

dunque

$$\beta \neq 3.2526$$

Per far si che il metodo converga, A deve essere **diagonalmente dominante** per righe, ovvero:

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

- riga 1: $1 > |1| + |\beta| \rightarrow$ Mai soddisfatta.
  Altro criterio sufficiente per far si che il metodo converga è che il raggio spettrale della matrice d'iterazione sia maggiore di 1.
  La matrice d'iterazione è: $M = -D^{-1} \cdot C$ e la convergenza è garantita solo se $\rho(M) = max_i |\lambda_i| < 1$
  Script Matlab per trovare un valore adatto di β

```
for beta=[-5:0.01:5]
    A = [1 1 beta; -1 2 0.2; 1 -0.1 2];
    D = diag(diag(A));
    C = A - D;

    M = -(inv(D))*C;
    ragg_spettr = max(abs(eig(M)));

    if ragg_spettr < 1
        disp('la matrice converge per beta uguale')
        disp(beta)
        disp('con raggio spettrale')
        disp(ragg_spettr)
        break
    end
end
```

2b) *Implementare il metodo di Jacobi e approssimare la soluzione, fissata una tolleranza* 10^{-3}

```
for beta=[-5:0.01:5]
    A = [1 1 beta; -1 2 0.2; 1 -0.1 2];
    D = diag(diag(A));
    C = A - D;

    M = -(inv(D))*C;
    ragg_spettr = max(abs(eig(M)));

    if ragg_spettr < 1
        disp('la matrice converge per beta uguale')
        disp(beta)
        disp('con raggio spettrale')
        disp(ragg_spettr)
        break
    end
end

b = [1; 2; 3];

x_0 = zeros(3,1);
x_new = M*x_0 + inv(D)*b;

while (norm(A*x_new - b) > 10^-3)
    x_0 = x_new;
```

```
    x_new = M*x_0 + inv(D)*b;
end

disp('soluzione approssimata:')
disp(x_new)
```