Calcolo Scientifico e Metodi Numerici: Prova pratica

A.A. 2021/2022

Indice

1.1 – radice.m

Descrizione

Lo script prende in input dall'utente un valore compreso tra 0 e 50 tramite la funzione *input* e lo salva nella variabile *val*.

Tramite il costrutto *if* verifica che il valore *val* sia compreso tra 0 e 50. Se lo è: stampa a video la radice quadrata di *val*, se non lo è: stampa a video un messaggio di errore che notifica l'utente dell'errato inserimento.

Test

```
>> radice
Inserire un valore numerico compreso tra 0 e 50: 20
    4.4721

>> radice
Inserire un valore numerico compreso tra 0 e 50: 79
Il valore inserito non è compreso tra 0 e 50.
```

1.2 – menu.m

Descrizione

Lo script stampa a video un menu di 4 portate, ognuna associata ad un numero (da 1 a 4).

Viene chiesto all'utente di inserire il numero corrispondente alla portata da lui desiderata e l'input viene salvato nella variabile *val*.

Tramite il costrutto *switch-case-otherwise*, se il valore inserito è compreso tra 1 e 4 viene stampata a video la portata scelta, altrimenti viene stampato a video un messaggio di errore che notifica l'utente dell'errato inserimento.

Le stampe a video vengono effettuate tramite la funzione *disp*.

Test

```
>> menu
1 - Costata di maiale
2 - Pesce spada
3 - Pizza margherita
4 - Pasta al ragù

Inserire numero portata scelta: 3

Pizza margherita - 700 Kcal

>> menu
1 - Costata di maiale
2 - Pesce spada
3 - Pizza margherita
4 - Pasta al ragù

Inserire numero portata scelta: 10

Il numero della portata deve essere compreso tra 1 e 4.
```

2.1 - approx.m

Descrizione

Lo script prende tre valori in input (tramite la funzione *input*) e li salva nelle variabili a, b e c.

Vengono calcolate e salvate le seguenti quantità:

- $d1\_reale = (a + b) + c$
- $d2\_reale = a + (b + c)$

Successivamente, in un sistema a virgola mobile con $N = 3$ cifre significative, vengono calcolate e salvate le medesime quantità:

- $d1 = (a + b) + c$
- $d2 = a + (b + c)$

Per fare ciò viene utilizzata la funzione *round*.

Vengono calcolati gli errori relativi usando la seguente formula:

$$\frac{|d_{approssimato} - d_{reale}|}{|d_{reale}|}$$

Gli errori relativi vengono salvati rispettivamente nelle variabili: *p1* e *p2*.

Vengono infine stampate a video le quantità calcolate nel sistema a virgola mobile e gli errori relativi.

Test

```
>> approx
Inserire valore di a: 72.213
Inserire valore di b: 41.243
Inserire valore di c: -113.44

1) d1 = (a + b) + c = 0
1) Errore relativo = 1

2) d2 = a + (b + c) = 0.4
2) Errore relativo = 24
```

3.1 – matprod.m

Descrizione

Lo script prende in input dall'utente un valore n.

Vengono creati:

- Una matrice A_{nxn} con solo elementi uguali a 0 (tramite la funzione *zeros*);
- Una matrice B_{nxn} con solo elementi uguali a 1 (tramite la funzione *ones*);
- Un vettore colonna z_n con solo elementi uguali a 2.

Successivamente vengono calcolati e salvati i risultati dei seguenti prodotti:

- $b = A \cdot z$
- $c = z^T \cdot B$

Infine vengono stampati a video i dati relativi alle matrici e ai vettori.

Test

```
>> matprod
Inserire una dimensione: 3

A =
       0               0               0
       0               0               0
       0               0               0

B =
       1               1               1
       1               1               1
       1               1               1

z =
       2
       2
       2

b = A * z =
       0
       0
       0

c = trasp(z) * B =
       6               6               6
```

3.2 – vettnorm.m

Descrizione

Lo script prende in input dall'utente un valore n.

Vengono creati:

- Una matrice R_{nxn} con valori pseudo-casuali compresi tra 0 e 1 (tramite la funzione *rand*);
- Un vettore colonna x con gli elementi della diagonale della matrice R (tramite la funzione *diag*);
- Una matrice D_{nxn} avente come diagonale il vettore x.

Vengono poi salvate: la parte triangolare superiore di D nella matrice U (tramite la funzione *triu*) e la parte triangolare inferiore di D nella matrice L (tramite la funzione *tril*).

Vengono effettuati dei controlli, utilizzando il costrutto *if-then-else*, per assicurarsi che:

- D sia diagonale (tramite la funzione *isdiag*);
- U sia triangolare superiore (tramite la funzione *istriu*);
- L sia triangolare inferiore (tramite la funzione *istril*).

Infine vengono stampati a video i dati relativi alle matrici e ai vettori.

Test

```
>> vettnorm
Inserire una dimensione: 3

R =
    1462/2557      1993/2273       477/1514
    2293/3683       371/1870       463/1135
     277/718        442/719       1715/5272

x =
    1462/2557
     371/1870
    1715/5272

D =
    1462/2557         0              0
       0           371/1870          0
       0              0          1715/5272

U =
    1462/2557         0              0
       0           371/1870          0
       0              0          1715/5272

L =
    1462/2557         0              0
       0           371/1870          0
       0              0          1715/5272

D è diagonale.
U è triangolare superiore.
L è triangolare inferiore.
```

3.3 – eigmat.m

Descrizione

Lo script prende in input dall'utente un valore n.

Viene creata una matrice S_{nxn} con elementi pseudo-casuali compresi tra 10 e 20 (tramite la funzione *rand*).

Viene fatto un controllo tramite la funzione *issymmetric* per verificare che la matrice sia simmetrica, se non lo è viene resa simmetrica tramite la seguente formula:

$$S = \frac{S + S^T}{2}$$

Viene stampata a video la matrice S seguita dal messaggio che comunica all'utente che la matrice è simmetrica.

Vengono salvati gli autovalori di S in un vettore d.

Infine vengono calcolate, salvate e stampate a video le norme: 1, 2 e ∞ del vettore d.

Test

```
>> eigmat
Inserire una dimensione: 4

S =
   15.3283   14.5044   16.2012   15.3535
   14.5044   16.2248   15.2898   12.1683
   16.2012   15.2898   12.3049   15.0751
   15.3535   12.1683   15.0751   12.2766

S è simmetrica.

Autovalori della matrice S =
   -3.5036
   -1.6078
    2.8262
   58.4197

Norma indice 1 = 66.3573
Norma indice 2 = 58.615
Norma indice inf = 58.4197
```

4.1 – test_gauss_lu.m

Descrizione

Lo script esegue dei test sulla risoluzione di 10 sistemi lineari tramite la fattorizzazione $A = LU$.

Vengono generate casualmente 10 matrici quadrate, aventi *nxn* entrate comprese tra 0 e 1 (n va da 100 a 1000 con passo 100).

Per ogni matrice:

1. Viene creato un vettore x contenente le soluzioni del sistema, avente tutte le entrate uguali a uno (tramite la funzione *ones*);
2. Viene creato un vettore b dei termini noti;
3. Vengono calcolate e salvate le matrici L e U di A, ottenute tramite la funzione *gauss_lu*;
4. Viene calcolata la soluzione x_1 tramite la fattorizzazione $A = LU$ e con essa viene calcolato l'errore relativo;
5. Vengono stampate a video tutte le informazioni riguardanti la matrice.

Test

```
>> test_gauss_lu
Matrice 100x100
    Errore relativo: 1.58E-12
    Indice di condizionamento: 2.90E+03
Matrice 200x200
    Errore relativo: 1.22E-11
    Indice di condizionamento: 1.02E+04
Matrice 300x300
    Errore relativo: 2.20E-10
    Indice di condizionamento: 2.03E+05
Matrice 400x400
    Errore relativo: 9.07E-12
    Indice di condizionamento: 6.47E+04
Matrice 500x500
    Errore relativo: 1.09E-09
    Indice di condizionamento: 3.73E+05
Matrice 600x600
    Errore relativo: 1.99E-11
    Indice di condizionamento: 2.50E+04
Matrice 700x700
    Errore relativo: 3.27E-10
    Indice di condizionamento: 4.64E+04
Matrice 800x800
    Errore relativo: 4.10E-09
    Indice di condizionamento: 4.13E+06
Matrice 900x900
    Errore relativo: 2.88E-10
    Indice di condizionamento: 8.73E+04
Matrice 1000x1000
    Errore relativo: 1.26E-10
    Indice di condizionamento: 8.07E+04
```

4.2 – test_gauss_palu.m

Descrizione

Lo script esegue dei test sulla risoluzione di 10 sistemi lineari tramite la fattorizzazione $PA = LU$.

Vengono generate casualmente 10 matrici quadrate, aventi nxn entrate comprese tra 0 e 1 (n va da 100 a 1000 con passo 100).

Per ogni matrice:

1. Viene creato un vettore x contenente le soluzioni del sistema, avente tutte le entrate uguali a uno (tramite la funzione *ones*);
2. Viene creato un vettore b dei termini noti;
3. Vengono calcolate e salvate le matrici P, L e U di A, ottenute tramite la funzione *gauss_palu*;
4. Viene calcolata la soluzione x_1 tramite la fattorizzazione $PA = LU$ e con essa viene calcolato l'errore relativo;
5. Vengono stampate a video tutte le informazioni riguardanti la matrice.

Test

```
>> test_gauss_palu
Matrice 100x100
    Errore relativo: 5.92E-14
    Indice di condizionamento: 4.33E+03
Matrice 200x200
    Errore relativo: 1.49E-12
    Indice di condizionamento: 4.46E+04
Matrice 300x300
    Errore relativo: 9.00E-13
    Indice di condizionamento: 1.93E+04
Matrice 400x400
    Errore relativo: 2.17E-12
    Indice di condizionamento: 6.56E+04
Matrice 500x500
    Errore relativo: 3.98E-13
    Indice di condizionamento: 4.62E+04
Matrice 600x600
    Errore relativo: 9.77E-13
    Indice di condizionamento: 3.48E+04
Matrice 700x700
    Errore relativo: 1.61E-11
    Indice di condizionamento: 2.06E+06
Matrice 800x800
    Errore relativo: 3.07E-12
    Indice di condizionamento: 5.03E+04
Matrice 900x900
    Errore relativo: 4.50E-12
    Indice di condizionamento: 1.90E+05
Matrice 1000x1000
    Errore relativo: 2.84E-11
    Indice di condizionamento: 3.63E+05
```

4.3 – function_gs.m

Descrizione

Funzione che approssima la soluzione di una matrice tramite l'algebra iterativa di Gauss-Seidel.

Prende come input la matrice dei coefficienti A, il vettore dei termini noti b, la tolleranza *tol*, il numero massimo di iterazioni *kmax* e il vettore iniziale x^0.

Vengono inizializzate le matrici:

- D = matrice di zeri avente la diagonale di A;
- E = matrice triangolare superiore di A, con le entrate cambiate di segno e la diagonale pari a 0;
- F = matrice triangolare inferiore di A, con le entrate cambiate di segno e la diagonale pari a 0.

Vengono poi calcolati:

- la matrice di iterazione B_{gs} tramite la formula:
$$B_{gs} = (D - E)^{-1} \cdot F$$
- il vettore f tramite la formula:
$$f = (D - E)^{-1} \cdot b$$
- il raggio spettrale di B_{gs}: $max(abs(eig(B_{gs})))$

Tramite un ciclo *while* vengono calcolate le k-esime approssimazioni usando la formula: $x^{k+1} = B_{gs} \cdot x^k + f$.

Il ciclo termina quando non è vero che: $(|x^{k+1} - x^k|_2 > tol \cdot |x^k|_2) \ \wedge \ (k < k_{max})$

Viene stampato un messaggio di warning in caso si fosse raggiunto e/o superato il numero massimo di iterazioni.

Infine vengono restituiti:

- la soluzione x;
- il numero di iterazioni effettuate k.

Test

```
>> [x,k] = function_gs(rand(3), [1 1 1]', 1e-20, 10, [0 0 0]')

x =
  -17557/239
   48444/79
  -52859/166

k =
      10
```

4.4 – test_metodi_iter.m

Descrizione

Lo script esegue dei test sulla risoluzione di 10 sistemi lineari tramite i metodi di Jacobi e Gauss-Seidel.

Vengono generate casualmente 10 matrici quadrate, aventi nxn entrate comprese tra 0 e 1 (n va da 100 a 1000 con passo 100).

Per ogni matrice:

1. Viene creata una matrice casuale A strettamente diagonalmente dominante;
2. Viene creato un vettore x contenente le soluzioni del sistema (in questo caso tutte uguali a 1) e un vettore b contenente i termini noti;
3. Vengono calcolate le matrici: D, E e F usate in seguito per il calcolo del raggio spettrale;
4. Vengono inizializzate le variabili relative alla tolleranza tol, al numero massimo di iterazioni k_{max} e il vettore iniziale x^0;
5. Vengono calcolate le matrici di iterazione B_j e B_{gs};
6. Vengono calcolate le soluzioni e il numero di iterazioni effettuate per ogni metodo;
7. Vengono calcolati gli errori relativi e il raggio spettrale per ciascuna matrice di iterazione (B_j e B_{gs});
8. Viene stampata una tabella con i risultati ottenuti.

Test

```
>> test_metodi_iter
           Metodo di Jacobi                                 Metodo di Gauss-Seidel
Dimensione N. Iterazioni   Errore relativo  Raggio spettrale N. Iterazioni   Errore relativo  Raggio spettrale
100        10              8.43e-04         4.93e-01         10              4.34e-10         1.05e-01
200        10              8.90e-04         4.95e-01         10              4.89e-10         1.07e-01
300        10              9.41e-04         4.98e-01         10              4.69e-10         1.06e-01
400        10              9.42e-04         4.98e-01         10              4.83e-10         1.07e-01
500        10              9.49e-04         4.99e-01         10              5.03e-10         1.07e-01
600        10              9.68e-04         5.00e-01         10              5.08e-10         1.07e-01
700        10              9.55e-04         4.99e-01         10              4.92e-10         1.07e-01
800        10              9.61e-04         4.99e-01         10              4.87e-10         1.07e-01
900        10              9.56e-04         4.99e-01         10              4.99e-10         1.07e-01
1000       10              9.63e-04         4.99e-01         10              4.97e-10         1.07e-01
```

5.1 – corde.m

Descrizione

Funzione per la risoluzione di un sistema non lineare tramite metodo delle corde:

$$x^{k+1} = x^k - \frac{f(x^k)}{m}$$

La funzione prende come parametri: la funzione *fun*, la pendenza m, il valore iniziale *x0*, la tolleranza *tol* e il numero massimo di iterazioni k_{max}.

Per prima cosa viene inizializzata la variabile k usata per calcolare le iterazioni e controlla che $fun(x0)$ ed m non siano troppo piccoli (o non siano uguali a 0).

Vengono poi effettuate le iterazioni all'interno di un *while*, andando ad incrementare il valore di k, salvando il vecchio valore di x in *x0* e il nuovo valore in *x_new*.

Il ciclo termina quando non è vero che: $(|x^{k+1} - x^k| > tol \cdot max\{1, |x^{k+1}|\}) \ \wedge \ (k < k_{max})$.

Stampa un messaggio di warning in caso si fosse raggiunto e/o superato il numero massimo di iterazioni.

Infine vengono restituiti:

- l'approssimazione x della radice;
- il numero di iterazioni effettuate k.

Test

```
>> fun = @(x) cos(x.*4) - x.*2 - 1/4;
>> fund = @(x) -4 * sin(4*x) - 2;
>> [x,k] = corde(fun,fund(0),0,1e-10,2)

x =
   -0.0896

k =
     2
```

Nota: *x0* assume il valore di x^k mentre *x_new* assume il valore di x^{k+1} all'interno del ciclo *while*.

5.2 - secanti.m

Descrizione

Funzione per la risoluzione di un sistema non lineare tramite il metodo delle secanti:

$$x^{k+1} = \frac{x^{k-1} \cdot f(x^k) - x^k \cdot f(x^{k-1})}{f(x^k) - f(x^{k-1})}$$

La funzione prende come parametri: la funzione *fun*, i valori iniziali *x0* e *x1*, la tolleranza *tol* e il numero massimo di iterazioni k_{max}.

Per prima cosa viene inizializzata la variabile *k* per le iterazioni, viene controllato se la funzione calcolata in *x0* o *x1* restituisca un valore molto piccolo (in quel caso interrompe l'esecuzione della funzione) e verifica che il denominatore non sia troppo piccolo (o uguale a 0).

Vengono poi effettuate le iterazioni all'interno di un *while*, andando ad incrementare il valore di *k*, salvando il vecchio valore di *x1* in *x0* e il nuovo valore di *x_new* in *x1*.

Il ciclo termina quando non è vero che: $(|x^{k+1} - x^k| > tol \cdot max\{1, |x^{k+1}|\}) \ \wedge \ (k < k_{max})$.

Stampa un messaggio di warning in caso si fosse raggiunto e/o superato il numero massimo di iterazioni.

Infine vengono restituiti:

- l'approssimazione *x* della radice;
- il numero di iterazioni effettuate *k*.

Test

```
>> fun = @(x) cos(x.*4) - x.*2 - 1/4;
>> [x,k] = secanti(fun,0,1/4,1e-10,3)

x =
    0.2089

k =
     3
```

5.3 - test_nonlin.m

Descrizione

Lo script esegue dei test sui metodi di risoluzione di un sistema non lineare utilizzando i dati forniti dalle tabelle 7.1, 7.2, 7.3 e 7.4 del libro di testo.

Lo script è strutturato in questo modo:

- 7.1 risoluzione sistemi tramite metodo di bisezione (*bisec.m*)
- 7.2 risoluzione sistemi tramite metodo di Newton (*newton.m*)
- 7.3 risoluzione sistemi tramite metodo delle corde (*corde.m*)
- 7.4 risoluzione sistemi tramite metodo delle secanti (*secanti.m*)

Per ogni metodo sono dati 4 sistemi da risolvere e per ogni sistema sono dati 2 input distinti.

Vengono stampati i risultati a video.

Test

```
>> test_nonlin

7.1) Metodo di bisezione

1) f(x) = x^2 - 2, alpha = sqrt(2)
    [0, 2],   |x - alpha| = 1.87e-09, n. iter. = 28
    [0, 200], |x - alpha| = 2.90e-09, n. iter. = 35
2) f(x) = e^x - 2, alpha = log(2)
    [0, 2],   |x - alpha| = 1.82e-09, n. iter. = 28
    [0, 200], |x - alpha| = 2.17e-09, n. iter. = 35
3) f(x) = 1/x - 3, alpha = 1/3
    [0, 2],   |x - alpha| = 1.24e-09, n. iter. = 28
    [0, 200], |x - alpha| = 6.60e-10, n. iter. = 35
4) f(x) = (x - 3)^3, alpha = 3
    [1.33, 3.33],   |x - alpha| = 1.24e-09, n. iter. = 28
    [1.33, 201.33], |x - alpha| = 2.52e-09, n. iter. = 35

7.2) Metodo di Newton

1) f(x) = x^2 - 2, alpha = sqrt(2)
    2,   |x - alpha| = 0.00e+00, n. iter. = 5
    200, |x - alpha| = 0.00e+00, n. iter. = 12
2) f(x) = e^x - 2, alpha = log(2)
    2,   |x - alpha| = 0.00e+00, n. iter. = 6
    200, |x - alpha| = 9.93e+01, n. iter. = 100
3) f(x) = 1/x - 3, alpha = 1/3
    2,    |x - alpha| = Inf,      n. iter. = 9
    0.10, |x - alpha| = 5.55e-17, n. iter. = 7
4) f(x) = (x - 3)^3, alpha = 3
    2,    |x - alpha| = 4.02e-08, n. iter. = 42
    2.90, |x - alpha| = 4.58e-08, n. iter. = 36
```

```
7.3) Metodo delle corde

1) f(x) = x^2 - 2, alpha = sqrt(2)
    2,   |x - alpha| = 2.64e-09, n. iter. = 15
    200, |x - alpha| = 1.97e-06, n. iter. = 1991
2) f(x) = e^x - 2, alpha = log(2)
    2,   |x - alpha| = 2.13e-08, n. iter. = 54
    200, |x - alpha| = 1.92e+02, n. iter. = 2000
3) f(x) = 1/x - 3, alpha = 1/3
    2,    |x - alpha| = 2.40e+04, n. iter. = 2000
    0.10, |x - alpha| = 9.87e-08, n. iter. = 147
4) f(x) = (x - 3)^3, alpha = 3
    2,    |x - alpha| = 2.73e-02, n. iter. = 2000
    2.90, |x - alpha| = 2.73e-03, n. iter. = 2000

7.4) Metodo delle secanti

1) f(x) = x^2 - 2, alpha = sqrt(2)
    [1, 2],     |x - alpha| = 0.00e+00, n. iter. = 7
    [199, 200], |x - alpha| = 2.22e-16, n. iter. = 17
2) f(x) = e^3, alpha = log(2)
    [2, 3],     |x - alpha| = 1.11e-16, n. iter. = 10
    [199, 200], |x - alpha| = 1.30e+02, n. iter. = 100
3) f(x) = 1/x - 3, alpha = 1/3
    [2, 3],       |x - alpha| = Inf,      n. iter. = 9
    [0.10, 0.11], |x - alpha| = 5.00e-16, n. iter. = 10
4) f(x) = (x - 3)^3, alpha = 3
    [1, 2],       |x - alpha| = 9.03e-08, n. iter. = 59
    [2.50, 2.90], |x - alpha| = 7.00e-08, n. iter. = 52
```

Osservazione: nei test 7.2, 7.3 e 7.4 il numero di iterazioni effettuate risulta maggiore di 1 rispetto al valore riportato sul libro. Testando le stesse funzioni sui sistemi visti a lezione questo non accade.

6.1 - canint.m

Descrizione

La funzione interpola dei dati punti tramite il polinomio in base canonica.

Prende come input:

- le ascisse di interpolazione x;
- le ordinate di interpolazione y;
- le ascisse per disegnare il grafico della funzione interpolante xx.

La funzione trasforma i vettori x, y e xx in vettori colonna (se già non lo sono) e crea la matrice dei coefficienti X:

$$X = \begin{bmatrix} x_0^0 & x_0^1 & \dots & x_0^n \\ x_1^0 & x_1^1 & \dots & x_1^n \\ \vdots & \vdots & \ddots & \vdots \\ x_n^0 & x_n^1 & \dots & x_n^n \end{bmatrix}$$

Salva la dimensione del vettore x in n e la dimensione del vettore xx in m.

Calcola la soluzione del sistema $a = X^{-1} \cdot y$ e inizializza il vettore yy che conterrà le ordinate del polinomio interpolante in corrispondenza dei punti xx dati.

Costruisce il polinomio interpolante e ne calcola i valori assunti nei punti xx:

$$p_n(x) = \sum_{k=0}^{n} a_k x^k \implies yy_j = \sum_{i=1}^{n} a_i \cdot xx_j^{i-1} \; con \; j = 0, ..., m$$

Infine restituisce il vettore delle ordinate yy e stampa a video il grafico del polinomio interpolante.

Test

```
>> x = [0 1 2 3];
>> y = [3 1 0 2];
>> xx = -1:0.1:4;
>> yy = canint(x,y,xx);
```

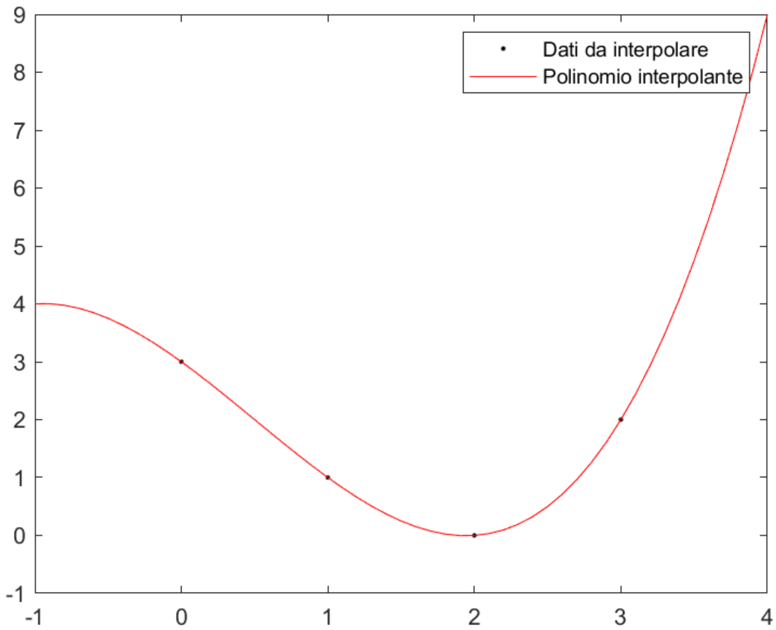

Figura 1: Polinomio interpolante in forma canonica.

6.2 - lagrint.m

Descrizione

La funzione interpola dei dati punti tramite il polinomio polinomio interpolante di Lagrange.

Prende come input:

- le ascisse di interpolazione x;
- le ordinate di interpolazione y;
- le ascisse per disegnare il grafico della funzione interpolante xx.

La funzione trasforma i vettori x, y e xx in vettori colonna.

Tramite ciclo *for* vengono calcolati i polinomi caratteristici di Lagrange:

$$L_j(x) = \prod_{k=0,k\neq j}^{n} \frac{x - x_k}{x_j - x_k} \; con \; j = 0, ..., n$$

Viene poi costruito e calcolato, nei punti dati e all'interno dello stesso ciclo *for*, il polinomio interpolante di Lagrange:

$$P_n(x) = \sum_{j=0}^{n} y_j L_j(x)$$

Infine restituisce il vettore delle ordinate yy e stampa a video il grafico del polinomio interpolante.

Test

```
>> x = [0 1 2];
>> y = [-1 1 -1];
>> xx = -1:0.1:3;
>> yy = lagrint(x,y,xx);
```

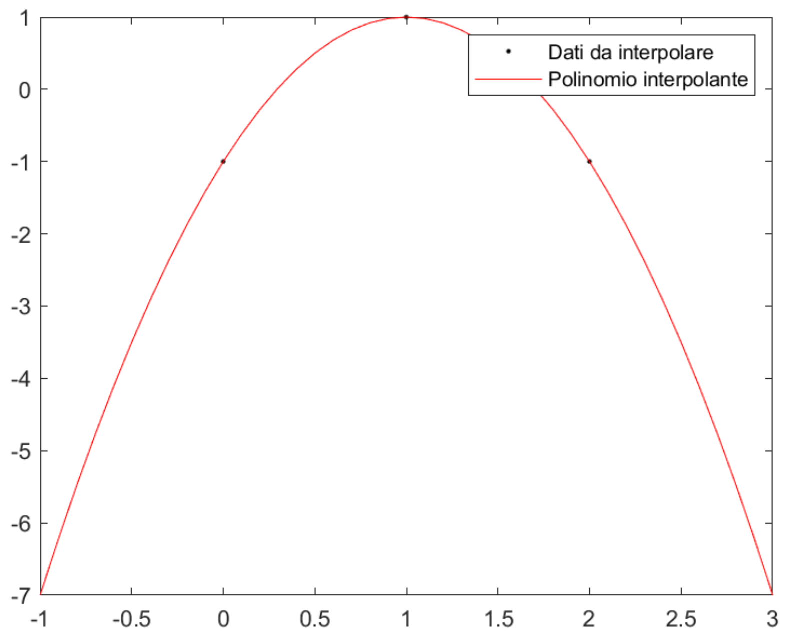

Figura 2: Polinomio interpolante di Lagrange.

6.3 - test_interp.m

Descrizione

Lo script effettua dei test con le funzioni *canint.m* e *lagrint.m*.

I test vengono effettuati su due diverse funzioni:

1. $f(x) = \frac{1}{1+25x^2}$
2. $f(x) = sin(2\pi x)$

Come ascisse vengono utilizzati: dei punti equispaziati x_1 e gli zeri del polinomio di Chebyshev x_2, questi ultimi ottenuti tramite la formula:

$$x_k = cos(\frac{2k+1}{2n+2}\pi) \; con \; k = 0, ..., n$$

Vengono stampati a video i grafici dei polinomi interpolanti.

Test

```
>> test_interp
```

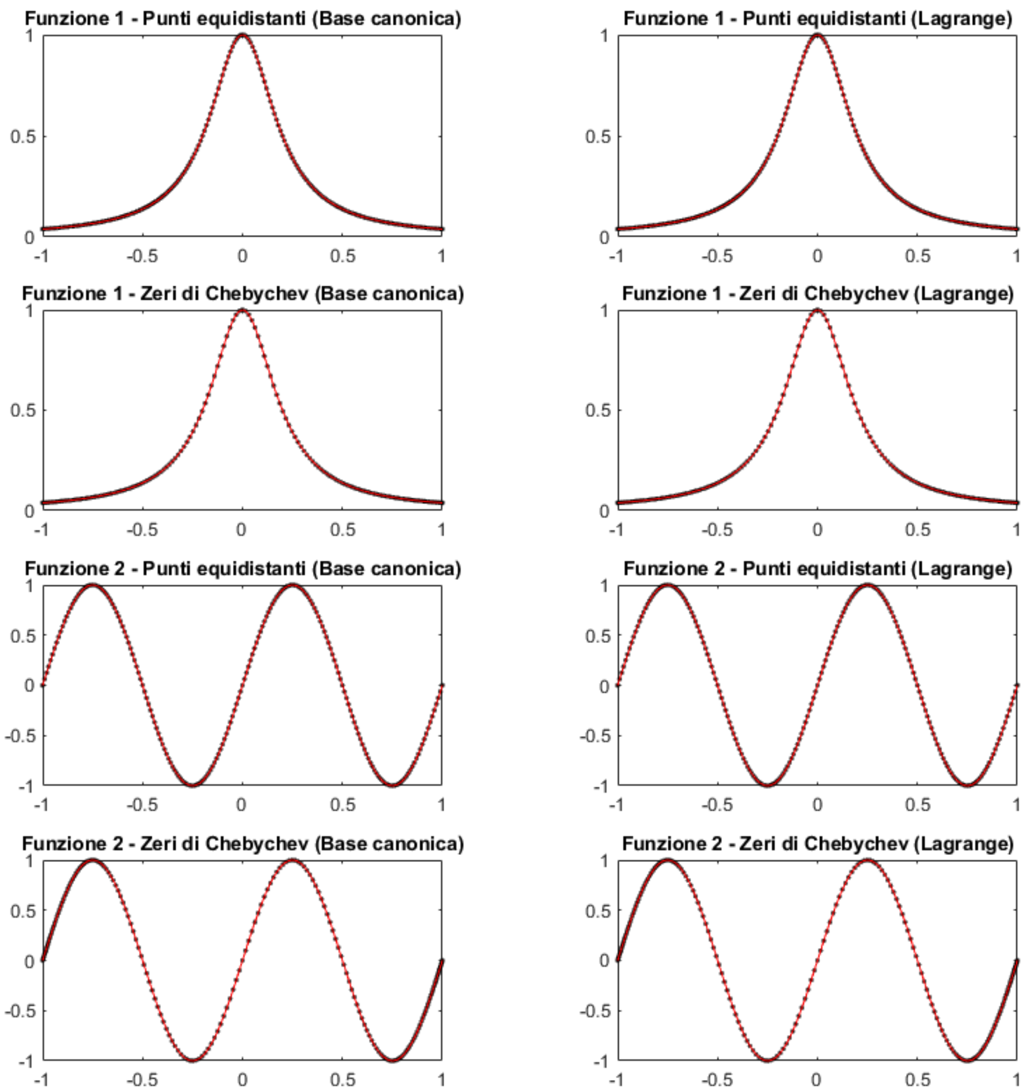

Figura 3: Grafici dei test effettuati.