

Tesina CSMN

Daniele Lurani - 60/61/65932
daniele.lurani@tiscali.it

a.a. 2021/2022

Indice

1	Esercitazione 1	3
1.1	Script radice.m	3
1.2	Script menu.m	4
2	Esercitazione 2	6
2.1	Script approx.m	6
3	Esercitazione 3	7
3.1	Script matprod.m	7
3.2	Script vettnorm.m	8
3.3	Script eigmat.m	9
4	Esercitazione 4	11
4.1	Script test_gauss_lu.m	11
4.2	Script test_gauss_palu.m	11
4.3	Function gs.m	12
4.4	Script test_metodi_iter.m	13
5	Esercitazione 5	15
5.1	Cenni teorici	15
5.2	Function corde.m	16
5.3	Function secanti.m	16
5.4	Script test_nonlin.m	17
6	Esercitazione 6	20
6.1	Cenni teorici	20
6.2	Function canint.m	20
6.3	Function lagrint.m	21
6.4	Script test_interp.m	21

1 Esercitazione 1

1.1 Script radice.m

Viene creato il file `radice.m`, il quale contiene il codice che simula l'inserimento di un numero da parte dell'utente e la stampa a schermo della sua radice se il numero è compreso in un determinato range. Innanzitutto viene richiesto all'utente di digitare un numero tra 0 e 50, estremi compresi, il quale verrà salvato nella variabile `inpt`. Ciò avviene tramite il comando `"input"`. Ora, per verificare che l'utente abbia effettivamente inserito un numero valido, viene usato il costrutto `if`, che funziona come segue:

```
if condizione (espressione logica)
    blocco istruzione 1
end

oppure

if condizione (espressione logica)
    blocco istruzione 1
else (altrimenti)
    blocco istruzione 2
end
```

Se il numero scelto è effettivamente compreso tra 0 e 50, tramite il comando `"disp"` verrà stampata a schermo la sua radice, altrimenti verrà stampato un messaggio di errore.

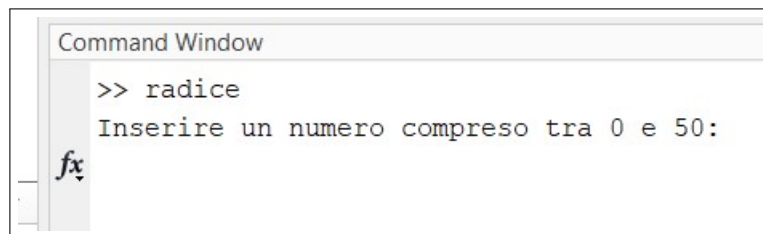


Figura 1: Visualizzazione all'avvio dello script

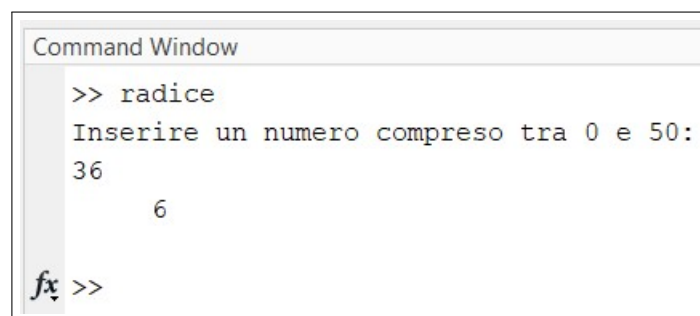


Figura 2: Visualizzazione in caso l'input sia corretto

```
>> radice
Inserire un numero compreso tra 0 e 50:
98
Errore. Il numero non fa parte del range specificato.
fx >>
```

Figura 3: Visualizzazione in caso l'input sia errato

1.2 Script menu.m

Il codice si trova all'interno del file menu.m, il quale simula la scelta di un piatto nel menu di un ristorante. Viene stampato a schermo un menu di 4 portate, con ogni portata assegnata ad un numero da 1 a 4. Viene chiesto all'utente di scegliere uno dei piatti e digitare il corrispettivo numero. Per verificare che l'utente scelga effettivamente un piatto nel menù, viene usato il costrutto switch, descritto in seguito:

```
switch variabile //cioè la variabile su cui stiamo effettuando il controllo
case 1 //il caso in cui la variabile valga 1
    blocco istruzione 1;
case 2
    blocco istruzione 2;
case 3
    blocco istruzione 3;
...

//in caso la variabile non assuma nessuno dei precedenti valori
otherwise:
    blocco istruzione;
end
```

In base alla scelta, verrà stampato a schermo il piatto scelto e le rispettive calorie. Altrimenti, se il numero scelto non corrisponde a nessuno dei piatti, verrà stampato un messaggio di errore.

```
Command Window

>> menu
Menù:
1 - Pasta alla carbonara
2 - Riso alla milanese
3 - Spaghetti alla bolognese
4 - Penne cacio e pepe
fx Scegliere una portata:
```

Figura 4: Visualizzazione all'avvio dello script

```
Command Window

>> menu
Menù:
1 - Pasta alla carbonara
2 - Riso alla milanese
3 - Spaghetti alla bolognese
4 - Penne cacio e pepe
Scegliere una portata: 4
Penne cacio e pepe, 480 kcal
fx >>
```

Figura 5: Visualizzazione in caso venga scelto un piatto appartenente al menù

```
Menù:
1 - Pasta alla carbonara
2 - Riso alla milanese
3 - Spaghetti alla bolognese
4 - Penne cacio e pepe
Scegliere una portata: 7
Numero non compreso nel menù.
fx >>
```

Figura 6: Visualizzazione in caso venga scelto un piatto non appartenente al menù

2 Esercitazione 2

2.1 Script approx.m

Questo script serve a dimostrare come la proprietà associativa della somma valga soltanto algebricamente e non su di un calcolatore, perchè nel calcolatore i dati vengono arrotondati e perdono la loro precisione, alterando così il risultato che si otterrebbe con le operazioni algebriche. Nella successiva immagine vengono illustrati i diversi risultati e i rispettivi errori dovuti al calcolatore, ottenuti semplicemente cambiando l'ordine delle operazioni.

```
Command Window
Inserire primo numero:
72.213
Inserire secondo numero:
41.243
Inserire terzo numero:
-113.44
Risultato della somma d1 = (a + b) + c = 0.000000.
Risultato della somma d2 = a + (b + c) = 0.400000.
Errore relativo p1 = 1.000000e+00.
Errore relativo p2 = 2.400000e+01.
```

Per fare ciò sono state usate la funzione `round(X, N, 'significant')` che arrotonda il numero X alla sua N cifra più significativa, contando dalla sua cifra più significativa a sinistra e la funzione `abs(X)` che ritorna il valore assoluto del valore X. Per ricavare l'errore relativo, viene usata la formula:

$$\rho = \frac{|a - a^*|}{|a|}$$

dove ρ è l'errore relativo, a è il valore normale, a^* è il valore approssimato.

3 Esercitazione 3

3.1 Script matprod.m

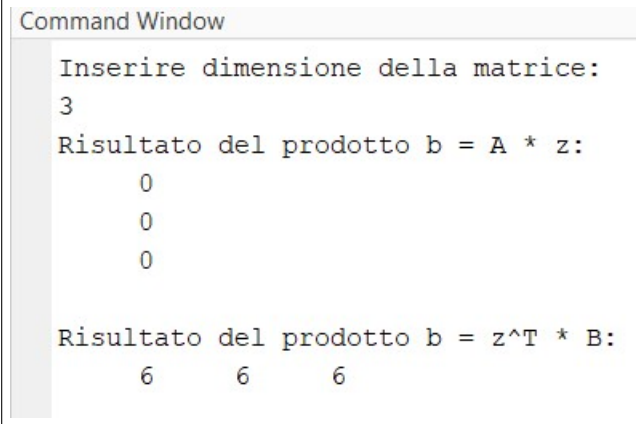
Lo script chiede all'utente di inserire una dimensione n la quale sarà la dimensione della matrice quadrata A contenente solo elementi 0, della matrice quadrata B contenente solo elementi 1 e del vettore colonna z che conterrà solo elementi 2. Successivamente viene calcolato il prodotto

$$b = A * z$$

e il prodotto

$$c = z^T * B$$

dove z^T è il vettore z trasposto.



```
Command Window
Inserire dimensione della matrice:
3
Risultato del prodotto b = A * z:
    0
    0
    0

Risultato del prodotto b = z^T * B:
    6    6    6
```

Figura 7: Una esecuzione dello script

Per creare la matrice A viene usata la funzione `zeros(n)` la quale crea una matrice quadrata con soli elementi 0 di dimensione n , mentre per la matrice B viene usata la funzione `ones(n)`, che si comporta allo stesso modo della precedente ma riempie la matrice di elementi 1.

Per creare il vettore colonna z di dimensione n e riempirlo di elementi 2, viene usato un ciclo `for`, il quale ad ogni iterazione aggiunge un elemento 2 in coda al vettore colonna, ciclando tante volte quante è n .

```

2  * Inizia il ciclo for, viene dichiarato un indice
3  * che ciclerà in base alla istruzione*
4
5  for indice : istruzione
6
7      *codice di ciò che deve avvenire all'interno del ciclo*
8
9  end *fine del ciclo*

```

Figura 8: Funzionamento di un ciclo for

Il vettore z^T viene creato tramite il comando $z^T = z'$, venendo così assegnato alla variabile z^T .

3.2 Script vettnorm.m

Lo script chiede all'utente di inserire una dimensione n la quale sarà la dimensione della matrice quadrata R contenente elementi pseudo casuali di grandezza compresa tra 0 e 1. Ciò avviene tramite la funzione $rand(n)$.

```

Inserire dimensione della matrice:
3
Creo la matrice R:
    0.9448    0.3377    0.1112
    0.4909    0.9001    0.7803
    0.4893    0.3692    0.3897

```

Successivamente viene estratta la diagonale della matrice R tramite la funzione $diag(R)$ ed inserita all'interno del vettore x ;

```

Estraggo la diagonale da R e la salvo nel vettore x:
    0.9448
    0.9001
    0.3897

```

Viene così creata, tramite la funzione $diag(x)$, la matrice diagonale D avente come elementi diagonali gli elementi del vettore x .

```

Creo la matrice D che ha nella diagonale gli elementi del vettore x:
    0.9448    0    0
    0    0.9001    0
    0    0    0.3897

```


Successivamente vengono estratte, rispettivamente tramite le funzioni *triu*(*R*) e *tril*(*R*), la parte triangolare superiore e triangolare inferiore della matrice *R*, ed inserite rispettivamente nelle matrici *U* ed *L*.

```
Estraggo la parte triangolare superiore della matrice R e la salvo nella matrice U:
0.9448    0.3377    0.1112
      0    0.9001    0.7803
      0      0    0.3897

Estraggo la parte triangolare inferiore della matrice R e la salvo nella matrice L:
0.9448      0      0
0.4909    0.9001      0
0.4893    0.3692    0.3897
```

Vengono infine eseguite le funzioni *isdiag*(*D*), *istriu*(*U*) e *istril*(*L*) per verificare se le matrici *D*, *U* ed *L* siano rispettivamente diagonale, triangolare superiore e triangolare inferiore.

3.3 Script eigmat.m

Lo script chiede all'utente di inserire una dimensione *n* la quale sarà la dimensione della matrice quadrata *S* contenente elementi pseudo casuali di grandezza compresa tra 10 e 20. Ciò avviene tramite la funzione *randi*([10, 20], *n*).

```
Inserire dimensione della matrice:
3
Creo una matrice S di dimensione n composta da elementi
pseudo-casuali interi compresi tra 10 e 20:
15    17    17
14    18    17
17    13    11
```

Viene successivamente controllato che la matrice creata *S* sia simmetrica, cioè sia una matrice che ha la proprietà di essere la trasposta di se stessa, ovvero $S^T = S$. Nel caso in cui la matrice non fosse simmetrica, viene resa simmetrica sostituendo *S* con

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

```
S non è simmetrica, la rendo simmetrica.
15.0000    15.5000    17.0000
15.5000    18.0000    15.0000
17.0000    15.0000    11.0000
```

Vengono poi calcolati gli autovalori della nuova matrice S e salvati nel vettore d tramite la funzione $\text{eig}(S)$. Infine vengono calcolate le norme con indice 1, 2 ed ∞ del vettore d .

```
Calcolo gli autovalori della matrice S e li salvo nel vettore d:  
-4.2124  
 1.7619  
46.4505  
  
Norma con indice 1:  52.4248  
  
Norma con indice 2:  46.6744  
  
Norma con indice infinito:  46.4505
```

4 Esercitazione 4

4.1 Script test_gauss_lu.m

Questo script effettua dei test per la risoluzione di sistemi lineari tramite la fattorizzazione $A = LU$. La fattorizzazione $A = LU$, utilizzando l'algoritmo di Gauss senza pivoting, scompone la matrice A in due matrici L ed U , le quali sono rispettivamente una matrice triangolare inferiore composta dai moltiplicatori trovati e la diagonale composta da elementi uguali ad 1 ed una matrice triangolare superiore equivalente a quella di partenza. Una volta nota la fattorizzazione, la soluzione del sistema lineare $Ax = b$ può essere ricondotta alla risoluzione a cascata di due sistemi triangolari, ovvero

$$\begin{cases} Ly = b \\ Ux = y \end{cases}$$

Lo script crea 10 matrici di dimensione crescente da 100 a 1000 e passo 100 e impostata una soluzione di elementi uguali ad 1, crea il vettore b dei termini noti. Successivamente viene invocata la funzione $\text{gauss_lu}(A)$, la quale prende in input la matrice A e, dopo aver effettuato la fattorizzazione, restituisce le due matrici L ed U . Vengono infine calcolati e stampati a schermo errore relativo e numero di condizionamento per ognuna delle matrici create.

```
Errore relativo alla dimensione 100 = 1.927465e-11
Numero di condizionamento alla dimensione 100 = 2.579499e+04

Errore relativo alla dimensione 200 = 2.827498e-12
Numero di condizionamento alla dimensione 200 = 2.674335e+03
```

Figura 9: Un esempio di esecuzione dello script

4.2 Script test_gauss_palu.m

Questo script effettua dei test per la risoluzione di sistemi lineari tramite la fattorizzazione $PA = LU$. La fattorizzazione $PA = LU$, utilizzando l'algoritmo di Gauss con pivoting, scompone la matrice A in due matrici L ed U , le quali sono rispettivamente una matrice triangolare inferiore composta dai moltiplicatori trovati e scambiati di riga in base ai cambi di riga effettuati durante l'algoritmo di Gauss con pivoting e la diagonale composta da elementi uguali ad 1 ed una matrice triangolare superiore equivalente a quella di partenza. A differenza della fattorizzazione $A = LU$ restituisce anche la matrice P , cioè la matrice Identità I con le righe scambiate seguendo gli scambi effettuati durante l'algoritmo di Gauss. Una volta nota la fattorizzazione, la soluzione del sistema lineare $Ax = b$ può

essere ricondotta alla risoluzione a cascata di due sistemi triangolari, ovvero

$$\begin{cases} Ly = Pb \\ Ux = y \end{cases}$$

Lo script crea 10 matrici di dimensione crescente da 100 a 1000 e passo 100 e impostata una soluzione di elementi uguali ad 1, crea il vettore b dei termini noti. Successivamente viene invocata la funzione *gauss_palu*(A), la quale prende in input la matrice A e, dopo aver effettuato la fattorizzazione, restituisce le tre matrici L , P ed U . Vengono infine calcolati e stampati a schermo errore relativo e numero di condizionamento per ognuna delle matrici create.

```
Errore relativo alla dimensione 100 = 2.632212e-14
Numero di condizionamento alla dimensione 100 = 1.687630e+03

Errore relativo alla dimensione 200 = 2.264022e-13
Numero di condizionamento alla dimensione 200 = 1.229273e+04
```

Figura 10: Un esempio di esecuzione dello script

4.3 Function *gs.m*

Questa funzione calcola l'approssimazione della soluzione di un sistema lineare tramite l'algoritmo iterativo di Gauss-Seidel. Un metodo iterativo è un tipo di metodo numerico nel quale le successive approssimazioni della soluzione al problema sono ottenute a partire dalle precedenti e comportano quindi la presenza di un valore stimato iniziale da cui partire. Inoltre un metodo iterativo comporta che le approssimazioni convergano alla soluzione, cioè sia possibile non giungere alla soluzione esatta in un numero finito di passi. Tramite la strategia dello splitting additivo, che consiste nello scrivere la matrice del sistema nella forma

$$A = P - N$$

e sostituendo nella formula di risoluzione di un sistema lineare si avrà

$$Px = Nx + b$$

che porta alla definizione del metodo iterativo

$$Px^{(k+1)} = Nx^{(k)} + b$$

Nel metodo di Gauss-Seidel $P = D - E$ mentre $N = F$, il che ci porta a

$$x^{(k+1)} = (D - E)^{-1} * (Fx^{(k)} + b)$$

Dalla formula precedente ricaviamo che la matrice di iterazione B_{GS} e il vettore f sono

$$B_{GS} = (D - E)^{-1} * F, f = (D - E)^{-1} * b$$

La condizione per l'applicabilità del metodo è che il raggio spettrale di B_{GS} sia minore di 1. Se inoltre la matrice A del sistema è diagonalmente strettamente dominante o definita positiva, allora il metodo di Gauss-Seidel converge.

Nell' applicazione di un metodo iterativo è cruciale utilizzare un criterio di arresto per le iterazioni in quanto esso dipende dalla qualità dell' approssimazione della soluzione del sistema. In questo caso viene utilizzato il Criterio di Cauchy, ovvero si esamina lo scarto tra due iterazioni successive. Fissata una tolleranza $\tau > 0$ e scelta una norma vettoriale, la condizione di arresto è data da:

$$||x^{(k)} - x^{(k-1)}|| < \tau$$

La funzione quindi implementa l'algoritmo di Gauss-Seidel, chiedendo in ingresso una matrice dei coefficienti A , il vettore dei termini noti b , una tolleranza tol , il numero massimo di iterazioni $kmax$ e il vettore iniziale $x0$. Esegue quindi l'algoritmo creando le matrici B_{GS} e il vettore f , esegue le varie iterazioni fino a quando non è soddisfatto il criterio di arresto o si è raggiunto un numero massimo di iterazioni e rende infine in uscita l'approssimazione x e il numero di iterazioni effettuate k .

4.4 Script test_metodi_iter.m

Questo script effettua dei test sulla risoluzione di un sistema lineare tramite gli algoritmi di Jacobi e Gauss-Seidel. Il metodo di Jacobi è anch'esso un metodo iterativo e si comporta in modo simile al metodo di Gauss-Seidel descritto in precedenza utilizzando la medesima strategia. A differenza di Gauss-Seidel nel metodo di Jacobi $P = D$ mentre $N = E + F$, il che ci porta a

$$x^{(k+1)} = D^{-1} * ((E + F)x^{(k)} + b)$$

Dalla formula precedente ricaviamo che la matrice di iterazione B_J e il vettore f sono

$$B_J = D^{-1} * (E + F), f = D^{-1} * b$$

La condizione per l'applicabilità del metodo è che il raggio spettrale di B_J sia minore di 1. Se inoltre la matrice A del sistema è diagonalmente strettamente dominante allora il metodo di Jacobi converge.

Lo script utilizza le funzioni *gs_rho.m* e *jacobi_rho.m*, che sono delle versioni modificate delle funzioni *gs.m* e *jacobi.m* che semplicemente restituiscono un valore in più, ossia il raggio spettrale della matrice di iterazione del rispettivo metodo (*rho*). Lo script genera tramite un ciclo for 10 matrici di dimensione da 100 a 1000 con passo 100 e dopo averle rese strettamente diagonalmente dominanti, in modo da rispettare la condizione sufficiente per la convergenza dei metodi iterativi, chiama le funzioni per l'esecuzione dei metodi iterativi, le quali restituiscono la soluzione del sistema lineare, il numero di iterazioni e il

raggio spettrale della matrice di iterazione del rispettivo metodo. Successivamente viene calcolato l'errore relativo per entrambi i metodi e infine viene stampata una tabella che riporta i dati relativi all'errore relativo e al raggio spettrale di ogni matrice.

Tabella errore relativo e raggio spettrale				
Dimensione matrice	Errore relativo J	Errore relativo GS	Raggio spettrale J	Raggio spettrale GS
100	9.999994e-11	1.224098e-10	1.000000e-02	1.280852e-03
200	9.999995e-11	1.269021e-10	1.000000e-02	1.303461e-03
300	1.000000e-10	1.281861e-10	1.000000e-02	1.308360e-03
400	1.000000e-10	1.304072e-10	1.000000e-02	1.321161e-03
500	1.000000e-10	1.288832e-10	1.000000e-02	1.313092e-03
600	1.000000e-10	1.309041e-10	1.000000e-02	1.319890e-03
700	9.999996e-11	1.301730e-10	1.000000e-02	1.318848e-03
800	1.000000e-10	1.303780e-10	1.000000e-02	1.317819e-03
900	9.999997e-11	1.298739e-10	1.000000e-02	1.319309e-03
1000	1.000001e-10	1.295472e-10	1.000000e-02	1.318064e-03

Figura 11: Un esempio di esecuzione dello script

5 Esercitazione 5

5.1 Cenni teorici

La risoluzione di equazioni e di sistemi di equazioni non lineari è un problema frequente, alcune possono avere una forma complessa e per via della loro forma è difficile la verifica dell'esistenza o unicità di una soluzione. In generale, non è possibile risolvere esattamente un'equazione non lineare, ma mediante gli algoritmi iterativi possiamo arrivare ad una approssimazione numerica delle sue radici. Qualsiasi equazione non lineare può essere scritta nella forma $f(x) = 0$ e risolvere questo problema significa trovare le radici dell'equazione o gli zeri della funzione $f(x)$. Si utilizzano quindi degli algoritmi iterativi, cioè dei metodi che partendo da un punto iniziale x_0 , forniscono una regola per costruire una successione che converge ad una radice α tale che $f(\alpha) = 0$. Nei successivi test verranno usati alcuni di questi metodi, come il metodo di bisezione, di Newton, delle corde e delle secanti.

Il metodo di bisezione è il metodo numerico più semplice per trovare le radici di una funzione. La sua efficienza è scarsa ma ha il pregio di essere stabile in ogni occasione. Supponiamo che una funzione continua $f(x)$ abbia un solo zero interno all'intervallo $[a, b]$ e consideriamo il punto medio $c = \frac{a+b}{2}$ dell'intervallo. La radice α o coincide con c e si ha quindi $f(c) = 0$, oppure cade in uno dei due semi intervalli $[a, c]$ e $[c, b]$. Per capire in quale è sufficiente verificare in quale intervallo la funzione cambia di segno, ossia analizzare il segno del prodotto $f(a) \cdot f(c)$: se è negativo la radice si trova nel primo intervallo altrimenti nel secondo. La successiva iterazione del metodo consiste nel ripetere questo procedimento dopo aver sostituito l'intervallo $[a, b]$ con $[a, c]$ o $[c, b]$ a seconda del segno di $f(a) \cdot f(c)$. L'algoritmo genera quindi una successione di intervalli tali che

$$b_k - a_k = \frac{b_{k-1} - a_{k-1}}{2}$$

assumendo ad ogni passo il punto medio

$$x_k = \frac{a_k + b_k}{2}$$

come approssimazione della radice α . Col metodo di bisezione la convergenza cresce linearmente a prescindere che la radice sia multipla o semplice.

Il metodo di Newton si applica dopo avere determinato un intervallo $[a, b]$ che contiene una sola radice e consiste nel sostituire alla curva $y = f(x)$ la tangente alla curva stessa, partendo da un qualsiasi punto. Per semplicità si può iniziare da uno dei due punti che hanno come ascissa gli estremi dell'intervallo $[a, b]$ e assumere, come valore approssimato della radice, l'ascissa x_t del punto in cui la tangente interseca l'asse delle x internamente all'intervallo $[a, b]$. Procedendo in modo iterativo si dimostra che la relazione di ricorrenza del metodo è

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

La convergenza è uguale a 2 se la radice è semplice, mentre è uguale a 1 se la radice è multipla (≥ 2).

Il metodo di Newton ha un svantaggio di tipo computazionale, infatti oltre a dover valutare la funzione bisogna valutare anche la sua derivata e questa potrebbe non essere disponibile. I metodi quasi-Newton considerano l'iterazione

$$x_{k+1} = x_k - \frac{f(x_k)}{m_k}$$

dove il coefficiente angolare m_k ci fornisce una approssimazione del valore della derivata. Due di questi metodi sono il metodo delle corde e il metodo delle secanti, che verranno descritti in seguito negli appositi paragrafi.

5.2 Function corde.m

Questa function implementa il metodo delle corde per la risoluzione di un'equazione non lineare. Prende in input la funzione *fun*, il valore *m* della pendenza della retta, il valore iniziale *x0*, la tolleranza *tol* e il numero massimo di iterazioni *kmax*, mentre ritornerà in uscita l'approssimazione *x* della radice e il numero di iterazioni *k* effettuate.

Nel metodo delle corde si considera un coefficiente angolare costante $m_k = m$, che può essere scelto in base all'osservazione del grafico di $f(x)$ oppure valutando la derivata f' sul punto iniziale x_0 . Il metodo ha un convergenza lineare ($p = 1$) e non ha delle prestazioni soddisfacenti ma permette di ridurre la complessità computazionale.

La funzione controlla innanzitutto che il valore della funzione nel punto iniziale non sia tendente allo 0, altrimenti si può considerare già come buona approssimazione della radice, e controlla che il coefficiente angolare non sia nullo. Superati questi controlli si inizia con le iterazioni del metodo delle corde, tramite un ciclo *while*, il quale esegue una determinata porzione di codice finché non vengono soddisfatte determinate condizioni. In questo caso il ciclo continua finché l'intervallo di riferimento è ancora troppo grande rispetto alla *tol* definita o finché non ho raggiunto il numero massimo di iterazioni o finché la approssimazione della radice trovata non è praticamente nulla.

5.3 Function secanti.m

Questa function implementa il metodo delle secanti per la risoluzione di un'equazione non lineare. Prende in input la funzione *fun*, i valori iniziali *x0* e *x1*, la tolleranza *tol* e il numero massimo di iterazioni *kmax*, mentre ritornerà in uscita l'approssimazione *x* della radice e il numero di iterazioni *k* effettuate.

Con questo metodo si va ad approssimare il coefficiente angolare con quello della retta secante la curva nei due punti di ascissa x_k e x_{k-1} ottenendo

$$m_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

e sostituendo m_k nella equazione si ha

$$x_{k+1} = \frac{x_{k-1}f(x_k) - x_kf(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

Questa formula dipende dalle valutazioni di f sui due punti precedenti, quindi necessità di essere inizializzata su x_0 e x_1 . L'ordine di convergenza di questo metodo è

$$p = \frac{1 - \sqrt{5}}{2} \simeq 1.618$$

La funzione si comporta in modo molto simile alla precedente, differenziandosi nel fatto che i controlli vengono eseguiti sui valori della funzione in entrambi i punti iniziali che verranno poi sostituiti in modo sequenziale.

5.4 Script test_nonlin.m

Questo script effettua dei test sulla risoluzione di un'equazione non lineare. Il test usa i dati presenti nelle tabelle 7.1, 7.2, 7.3 e 7.4 del libro di testo e li testa con i metodi di bisezione, di Newton, delle corde e delle secanti. Di seguito vengono riportati alcuni risultati dei test per ogni metodo utilizzato.

----- Test con metodo di bisezione -----	
Equazione 1 intervallo 1	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	22
Iterazioni attese:	28
Errore assoluto ottenuto:	9.500606e-08
Errore assoluto atteso:	1.900000e-09
Equazione 1 intervallo 2	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	29
Iterazioni attese:	35
Errore assoluto ottenuto:	1.397095e-07
Errore assoluto atteso:	2.900000e-09

Figura 12: Un esempio di esecuzione del test con il metodo di bisezione

Test con metodo di Newton	
Equazione 1 punto 1	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	5
Iterazioni attese:	4
Errore assoluto ottenuto:	0.000000e+00
Errore assoluto atteso:	0.000000e+00
Equazione 1 punto 2	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	12
Iterazioni attese:	11
Errore assoluto ottenuto:	0.000000e+00
Errore assoluto atteso:	0.000000e+00

Figura 13: Un esempio di esecuzione del test con il metodo di Newton

Test con metodo delle corde	
Equazione 1 punto 1	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	11
Iterazioni attese:	14
Errore assoluto ottenuto:	3.591387e-07
Errore assoluto atteso:	2.600000e-09
Equazione 1 punto 2	
Risultato ottenuto:	1.414411e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	1342
Iterazioni attese:	1990
Errore assoluto ottenuto:	1.972649e-04
Errore assoluto atteso:	1.900000e-06

Figura 14: Un esempio di esecuzione del test con il metodo delle corde

Test con metodo delle secanti	
Equazione 1 intervallo 1	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	6
Iterazioni attese:	6
Errore assoluto ottenuto:	0.000000e+00
Errore assoluto atteso:	0.000000e+00
Equazione 1 intervallo 2	
Risultato ottenuto:	1.414214e+00
Risultato atteso:	1.414214e+00
Iterazioni ottenute:	16
Iterazioni attese:	16
Errore assoluto ottenuto:	2.220446e-16
Errore assoluto atteso:	2.200000e-16

Figura 15: Un esempio di esecuzione del test con il metodo delle secanti

6 Esercitazione 6

6.1 Cenni teorici

Per interpolazione si intende un metodo per individuare nuovi punti del piano cartesiano a partire da un insieme finito di punti dati, nell'ipotesi che tutti i punti si possano riferire ad una funzione $f(x)$ di una data famiglia di funzioni di una variabile reale. Nelle attività scientifiche e tecnologiche, e in genere negli studi quantitativi di qualsiasi fenomeno, accade molto spesso di disporre di un certo numero di punti del piano ottenuti con un campionamento o con apparecchiature di misura e di ritenere opportuno individuare una funzione che passi per tutti i punti dati o almeno nelle loro vicinanze. Assegnate $n + 1$ coppie di numeri reali (x_i, y_i) con $i = 0, \dots, n$, una funzione $\phi(x)$ interpola i punti dati se

$$\Phi(x) = y_i, \forall i = 0, \dots, n$$

Scelte $n + 1$ funzioni $\phi_j(x)$ con $j = 0, \dots, n$ si vuole approssimare $f(x)$ mediante una loro combinazione lineare

$$\Phi(x) = \sum_{j=0}^n a_j \phi_j(x)$$

che sia interpolante. Per ottenere i coefficienti della combinazione lineare basta imporre che la funzione $\phi(x)$ passi per tutte le y ottenendo il sistema di equazioni lineari

$$\sum_{j=0}^n \phi_j(x_i) \cdot a_j = y_i, i = 0, \dots, n$$

la cui soluzione $a = (a_0, \dots, a_n)^T$ fornisce i coefficienti cercati. Un sistema ha un'unica soluzione (condizione di unisolvenza) ed è risolvibile se e solo se tutte le x dei punti sono diverse tra loro e si ha il determinante diverso da 0. Nell'interpolazione polinomiale si cerca come interpolante un polinomio di un grado opportuno. Verranno descritte in seguito due rappresentazioni di interpolazioni polinomiale.

6.2 Function canint.m

Questa funzione costruisce il polinomio interpolante nella forma canonica. Prende in input i vettori di ascisse e ordinate di interpolazione x e y e le ascisse per la costruzione del grafico xx . Ritorna in output il valore del polinomio nelle ascisse date. Il polinomio interpolante in forma canonica $\{1, x, x^2, \dots, x^n\}$ si esprime come

$$p_n(x) = \sum_{j=0}^n a_j x^j$$

La condizione di unisolvenza è che il determinante sia diverso da zero. Imponendo le condizioni di interpolazione si ottiene il sistema lineare

$$\sum_{j=0}^n a_j x^j = y_i \quad i = 0, \dots, n$$

la cui matrice dei coefficienti è la matrice di Vandermonde e con essa possiamo risolvere il sistema e ottenere così il polinomio interpolante.

6.3 Function lagrint.m

Questa funzione costruisce il polinomio interpolante con la rappresentazione di Lagrange. Prende in input i vettori di ascisse e ordinate di interpolazione x e y e le ascisse per la costruzione del grafico xx . Ritorna in output il valore del polinomio nelle ascisse date. La rappresentazione di Lagrange del polinomio interpolante adotta la base più semplice possibile e tutto lo sforzo computazionale è destinato al calcolo dei coefficienti. Dati (x_i, y_i) con $i = 0, \dots, n$ i polinomi caratteristici di Lagrange $L_j(x)$ con $j = 0, \dots, n$ sono definiti da

$$L_j(x) = \prod_{k=0, k \neq j}^n \frac{x - x_k}{x_j - x_k} = \frac{(x - x_0) \cdot (x - x_{j-1}) \cdot (x - x_{j+1}) \cdot \dots \cdot (x - x_n)}{(x_j - x_0) \cdot (x_j - x_{j-1}) \cdot (x_j - x_{j+1}) \cdot \dots \cdot (x_j - x_n)}$$

Il polinomio interpolante di Lagrange è:

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

che interpola i dati assegnati in quanto

$$p_n(x_i) = \sum_{j=0}^n y_j \delta_{ij} = y_i \quad i = 0, \dots, n$$

6.4 Script test_interp.m

Questo script esegue dei test con le due forme diverse del polinomio interpolante (forma canonica e forma di Lagrange), con due modi diversi di scegliere le ascisse di interpolazione (punti equispaziati e zeri del polinomio di Chebychev) e due diverse funzioni da interpolare

$$f1(x) = \frac{1}{1 + 25x^2} \quad e \quad f2(x) = \sin(2\pi x)$$

Per genere le ascisse si utilizzano due metodi: il primo, quello dei punti equispaziati, prevede l'utilizzo della funzione *linspace* che consente di generare un numero n di punti equispaziati all'interno di un intervallo specificato. Il secondo, il metodo degli zeri del polinomio di Chebychev, si definisce come segue

$$T_{n+1}(x) = \cos((n+1)\theta), \quad \text{per } x = \cos(\theta) \text{ e } \theta \in [0, \pi]$$

Si possono così trovare gli zeri del polinomio imponendo

$$x_k = \cos\left(\frac{2k+1}{2n+2}\pi\right), \quad k = 0, 1, \dots, n$$

Dopo la generazione delle ascisse con i due diversi metodi vengono chiamate le funzioni *canint* e *lagrint* per creare l'interpolazione in forma canonica e nella forma di Lagrange. Successivamente viene creato e mostrato a schermo il grafico di ognuno dei test effettuati tramite la funzione *plot*.

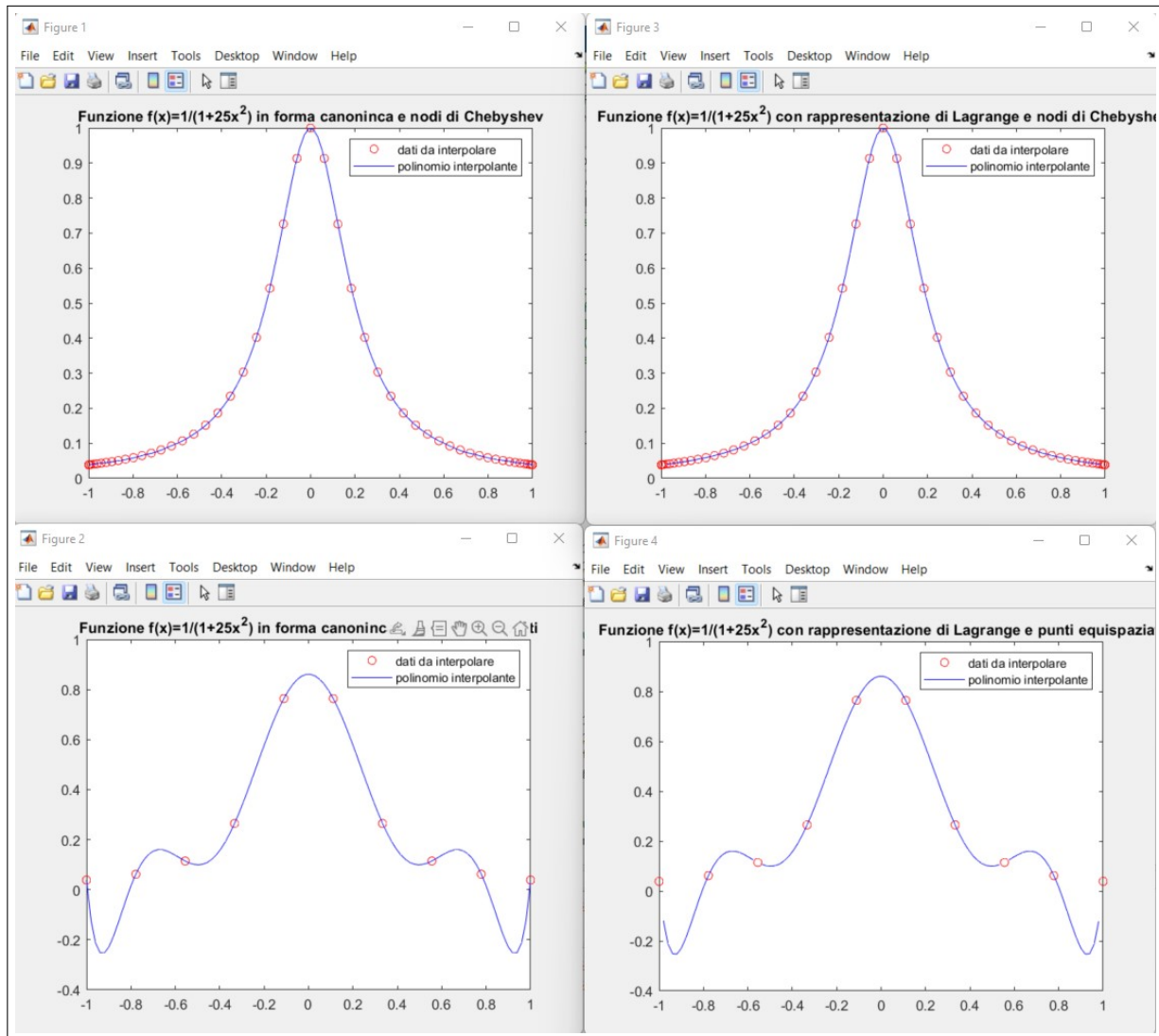


Figura 16: Esecuzione del test con la prima funzione

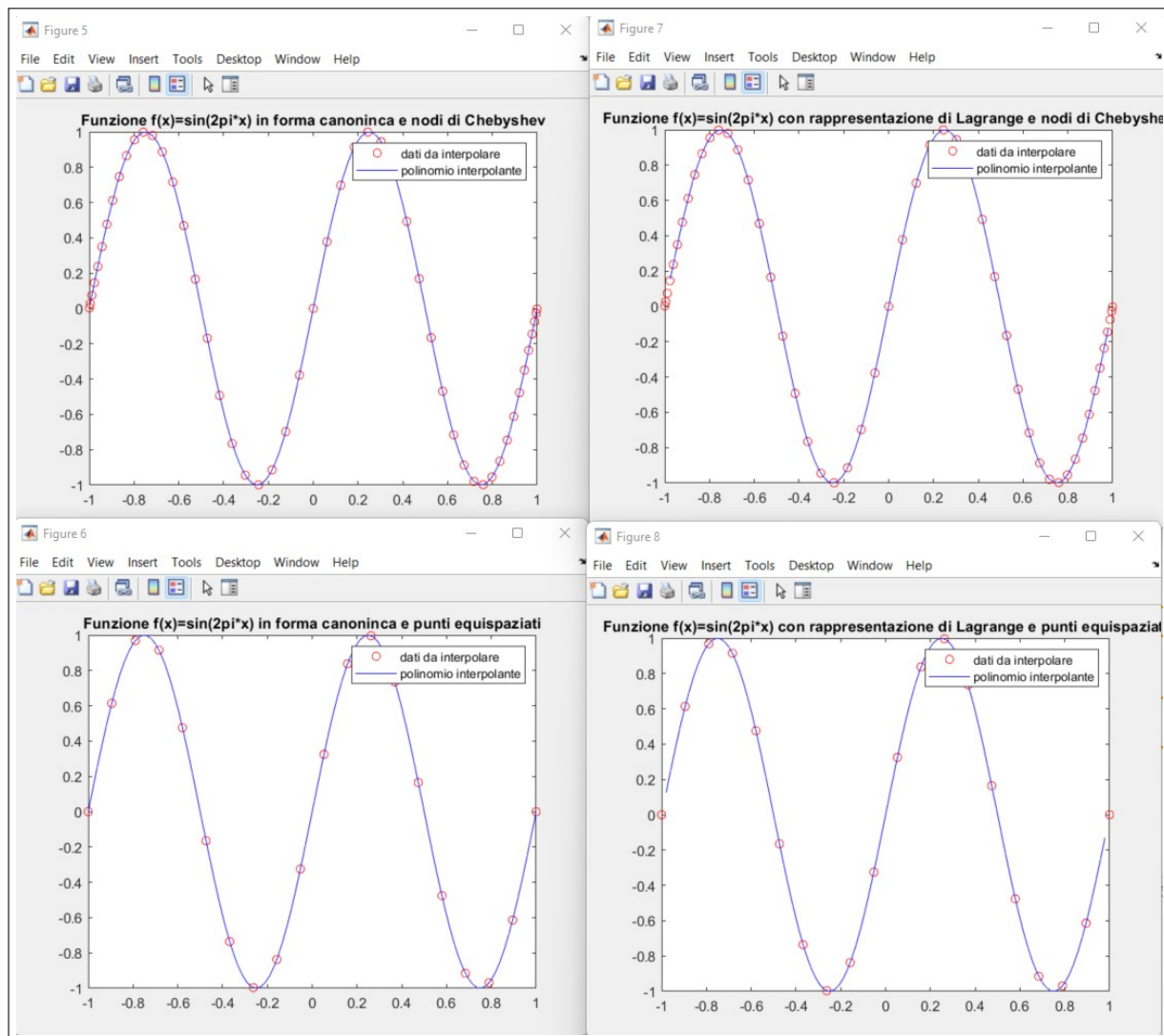


Figura 17: Esecuzione del test con la seconda funzione

Si può confermare osservando i grafici che utilizzando i nodi equispaziati come metodo di generazione delle ascisse si incorre in un errore, che tende infatti all'infinito all'aumentare del grado del polinomio. Utilizzando invece il metodo dei nodi di Chebychev l'errore converge a zero.