Tesina Matlab

Esercitazione 1

Menu.m

Viene richiesto all'utente di inserire un numero n per selezionare da un menù il piatto corrispondente attraverso uno switch, l'inserimento di un numero non contenuto dal menù porta alla stampa di un messaggio d'errore altrimenti alla stampa di un piatto e delle sue calorie.

Risultati:

inserisci il numero di un ordine del menù [1-4]:

1

Spaghetti 158 cal

inserisci il numero di un ordine del menù [1-4]:

5

Non hai selezionato nessun piatto

Radice.m

Viene richiesti all'utente di inserire un numero, dopo di che avviene un controllo su tale dato, se è compreso fra 0 e 50 ne viene calcolata e stampata la radice altrimenti viene stampato un messaggio d'errore dove viene segnalata l'assenza del numero inserito nel range indicato.

inserire un numero fra 0 e 50:

6

radice del numero inserito: 2.4495

inserire un numero fra 0 e 50:

99

il numero inserito è al di fuori del range (0 - 50)

Esercitazione 2

Approx.m

Viene richiesto all'utente di inserire tre numeri per calcolare d1 = (a + b) + c d2 = a + (b + c) in un sistema in virgola mobile con 3 cifre significative.

```
a= round(a,3 ,'significant');
```

(viene svolto per tutti i dati inseriti dall'utente per renderli numeri in virgola mobile con 3 cifre significative).

d1 e d2 vengono calcolati con le seguenti operazioni:

```
% (a + b) + c con operazioni macchina
d1=round( round(a+b, 3, 'significant')+c, 3, 'significant');

% a + (b + c) con operazioni macchina
d2=round(round(b+c, 3, 'significant')+a, 3, 'significant');
```

Stampare a video gli errori relativi pari a valore assoluto della differenza fra espressione svolta usando operazioni macchina e espressione svolta usando espressioni algebriche diviso per valore il assoluto dell' espressione svolta usando operazioni macchina.

```
%errori relativi r1 e r2  differenza tra il valore misurato e il valore
%esatto divisi valore atteso

r1= abs(d1 - ( (a + b) + c)) / abs( (a + b) + c);
r2= abs(d2 - (  a + (b + c) )) / abs(  a + (b + c) );
```

Risultati:

inserisci il numero a:

72.213

inserisci il numero b:

41.243

inserisci il numero c:

-113.44

d1 = (a + b) + c = 0

d2 = a + (b + c) = 0.4

errori relativi calcolati su d1 e d2 rispettivamente r1 e r2

r1: 1 r2: 1.4155e-14

Esercitazione 3

matprod.m

Viene richiesto in input un numero che verrà usato per creare una matrice quadrata A di tale dimensione composta solo da 0 con il comando:

```
A= zeros(n);
```

e in seguito una matrice B di 1 di dimensione n(dimensione inserita dall'utente).

```
B= ones(n);
```

Per creare la matrice Z di dimensione n composta da elementi pari a 2 creiamo una matrice di 1 e la moltiplichiamo per due e estraiamo il vettore z, con le seguenti istruzioni:

```
%crea un vettore colonna z di lunghezza n contenente elementi uguali a 2;
Z=ones(n)*2; %creiamo prima la matrice contenente elementi pari a 2
z=Z(:,n);  %estraiamo il vettore
```

Dopo di che attuiamo il prodotto fra A e z (che conterrà sempre 0 dato che A contiene solo questo elemento) e poi calcoliamo il prodotto fra la trasposta del vettore z e B.

```
b=A*z;
c=transpose(z)*B;
```

Risultati:

inserisci dimensione n:

4

Matrice A:

0	*0*	*0*	*0*
0	*0*	*0*	*0*
0	*0*	*0*	*0*
0	*0*	*0*	*0*

Matrice B:

1	*1*	*1*	*1*
1	*1*	*1*	*1*
1	*1*	*1*	*1*
1	*1*	*1*	*1*

Vettore z:

2
2

2

2

Stampa vettore b pari al prodotto fra matrice A e vettore z:

0

0

0

0

Stampa vettore c pari al prodotto fra vettore trasposto z e matrice B:

8 8 8 8

Vettnorm.m

Viene richiesto all'utente di inserire una dimensione n per creare una matrice randomica contenente elementi fra 0 e 1, da cui viene estratta la diagonale principale poi inserita nella matrice D come diagonale principale usando il comando diag(). Vengono estratte la parte triangolare superiore e inferiore della matrice randomica R e inserite rispettivamente nelle variabili U e L.

```
%crea una matrice D che contiene il vettore x come diagonale
D=diag(x,0); %0 rappresenta la diagonale principale

%estrarre la parte triangolare superiore e inferiore  di R
U=tril(R);
L=triu(R);
```

Dobbiamo verificare che D sia diagonale e controllare che U e L siano triangolari superiori e inferiori, per fare ciò usiamo i tre comandi isdiag(D), istril(U), istriu(L), che restituiscono 1 se le condizioni che cerchiamo si verificano altrimenti restituisce 0.

Risultati:

inserisci dimensione matrice:

4

Matrice D:

0.8147	*0*	*0*	*0*
0	*0.0975*	*0*	*0*
0	*0*	*0.1576*	*0*
0	*0*	*0*	*0.1419*

D è diagonale

U è triangolare superiore

L è triangolare inferiore

Eigmat.m

Viene chiesto all'utente di inserire un numero dimensione n, e si crea una matrice quadrata S di dimensione n con elementi pseudo-casuali interi tra 10 e 20:

```
S=randi([10,20],n,n );
```

[10,20] indica l'intervallo fra cui verrano presi gli elementi casuali della matrice, n la dimensione.

Si verifica se S è simmetrica, evento che accadrà raramente, altrimenti renderla tale con questa formula (S + S ^T)/ 2 che produrrà una matrice con elementi decimali che verranno poi arrotondati con il seguente comando:

```
S=round(double(S));
```

Si calcolano gli autovalori della matrice, poi inseriti nel vettore d e si calcola la norma con indice 1, 2 e ∞ del vettore d, con il comando norm().

Risultati

inserisci n dimensione matrice:

4

Matrice S:

18	*16*	*20*	*20*
19	*11*	*20*	*15*
11	*13*	*11*	*18*
20	*16*	*20*	*11*

La matrice S di dimensione 4 non è simmetrica.

Stampa della matrice S dopo essere stata resa simmetrica:

18	*18*	*16*	*20*
18	*11*	*17*	*16*
16	*17*	*11*	*19*
20	*16*	*19*	*11*

Autovalori della matrice S

-9.1451

-4.5997

-1.2302

65.9750

Norme con indice 1, 2 e infinito del vettore d (il vettore d contiene gli autovalori della matrice):

80.9499

66.7757

65.9750

Esercitazione 4

test_gauss_lu.m

Effettua dei test sulla soluzione di un sistema lineare tramite fattorizzazione A=LU su 10 matrici generate randomicamente di dimensione fra 100 e 1000 con passo 100.

```
%crea 10 matrici di dimensione tra 100 e 1000 con passo 100
for n= 100:100:1000
    i=i+1;
    A=rand(n);
```

Viene calcolato b (b=A*x) imponendo una soluzione x contenente elementi pari a 1. Ignorando la soluzione imposta x dobbiamo trovare la soluzione x1 svolgendo la fattorizzazione A = LU con cui vengono trovate le matrici triangolari L e U. Le soluzioni x e x1 sono state usate per calcolare l'errore relativo da stampare poi con la dimensione della matrice e il suo condizionamento. Questa operazione è stata svolta su ogni matrice.

Risultati

dimensione matrice: 100	errore relativo: 2.373460e-12	condizionamento: 6144.952737
dimensione matrice: 200	errore relativo: 4.933541e-12	condizionamento: 4941.864041
dimensione matrice: 300	errore relativo: 2.263276e-11	condizionamento: 11637.981901
dimensione matrice: 400	errore relativo: 2.773858e-11	condizionamento: 37768.997153
dimensione matrice: 500	errore relativo: 1.659598e-11	condizionamento: 8520.093900
dimensione matrice: 600	errore relativo: 7.570071e-11	condizionamento: 96486.827186
dimensione matrice: 700	errore relativo: 2.038550e-11	condizionamento: 24473.428712
dimensione matrice: 800	errore relativo: 8.633079e-11	condizionamento: 30610.846228
dimensione matrice: 900	errore relativo: 1.867438e-10	condizionamento: 23210.816238
dimensione matrice: 1000	errore relativo: 3.732460e-10	condizionamento: 81177.720924

test_gauss_palu.m

Vengono svolte le stesse operazioni dello script precedente ma al posto della fattorizzazione A=LU viene svolta la fattorizzazione PA=LU con cui vengono trovate le matrici triangolari L e U e la matrice di permutazione P.

Risultati

dimensione matrice: 100 errore relativo: 3.055092e-14 condizionamento: 1918.469723
dimensione matrice: 200 errore relativo: 7.753366e-14 condizionamento: 5275.840379
dimensione matrice: 300 errore relativo: 4.652544e-13 condizionamento: 40757.068413
dimensione matrice: 4
00 errore relativo: 7.152071e-12 condizionamento: 347722.736005
dimensione matrice: 500 errore relativo: 1.633885e-12 condizionamento: 56504.606686
dimensione matrice: 600 errore relativo: 3.766114e-13 condizionamento: 44896.066541
dimensione matrice: 700 errore relativo: 8.400232e-13 condizionamento: 27528.052881
dimensione matrice: 800 errore relativo: 7.733959e-13 condizionamento: 40389.305062
dimensione matrice: 900 errore relativo: 6.369643e-13 condizionamento: 53768.112963
dimensione matrice: 1000 errore relativo: 1.698996e-12 condizionamento: 47098.916482

Si nota che in entrambi gli script l'errore relativo tende ad aumentare di poco con l'aumentare della dimensione.

gs.m

L'algoritmo di Gauss-Seidel prende in input i termini noti b e la matrice A da cui estrae la matrice diagonale D e le matrici triangolari superiore e inferiore E e F, cambiate di segno. Calcola Bgs e f con le seguenti espressioni:

Bgs = (D-E)\F (da tenere a mente che \ sta ad indicare (D-E)^(-1) / F)
f = (D-E)\b

ad ogni iterazione viene moltiplicata la matrice Bgs con x_old (inizialmente x0 e in seguito la soluzione dell'iterazione precedente) e gli viene sommato f.

Il ciclo continua finchè non avviene la convergenza con la tolleranza o si raggiunge il numero massimo di iterazioni, entrambi parametri delle funzioni.

Il metodo di Jacobi è molto simile ma si differenzia per il calcolo della matrice di iterazione e del vettore f:

```
Bj = D\(E+F); %matrice di iterazione
f = D\b; %vettore f
```

test_metodi_iter.m

Lo script crea 10 matrici usando la funzione rand() di dimensione compresa fra 100 e 1000 con passo 100 e le rende strettamente diagonalmente dominanti. Viene imposta una soluzione x di elementi pari a 1 e viene calcolata b con b=A*x. Vengono in seguito chiamate la funzione di Jacobi e di Gauss-Seidel per calcolare le approssimazioni xj e xgs da cui sono stati calcolati gli errori relativi con x. Stampati poi con ogni dimensione di ogni matrice e il raggio spettrale della matrice di iterazione per entrambi i metodi.

Risultati:

Tabella che contiene raggio spettrale della matrice di iterazione per entrambi i metodi, dimensione della matrice e errore relativo calcolato col metodo Gauss-Seidel e Jacobi

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.031660
dimensione matrice: 100
errore relativo(Gauss-Seidel): 9.893e+00 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 9.900e+00 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.032067
dimensione matrice: 200
errore relativo(Gauss-Seidel): 1.407e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 1.407e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.031958
dimensione matrice: 300
errore relativo(Gauss-Seidel): 1.726e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 1.726e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.032090
dimensione matrice: 400

errore relativo(Gauss-Seidel): 1.995e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 1.995e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.031983
dimensione matrice: 500
errore relativo(Gauss-Seidel): 2.231e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 2.232e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.031972
dimensione matrice: 600
errore relativo(Gauss-Seidel): 2.445e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 2.445e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.032082
dimensione matrice: 700
errore relativo(Gauss-Seidel): 2.642e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 2.642e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.032048
dimensione matrice: 800
errore relativo(Gauss-Seidel): 2.825e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 2.825e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.032075
dimensione matrice: 900
errore relativo(Gauss-Seidel): 2.996e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 2.997e+01 passi(Jacobi):9

Raggio spettrale della matrice di iterazione Bj: 0.166667
Raggio spettrale della matrice di iterazione Bgs: 0.032054
dimensione matrice: 1000
errore relativo(Gauss-Seidel): 3.159e+01 passi(Gauss_Seidel): 3
errore relativo(Jacobi): 3.159e+01 passi(Jacobi):9

Come possiamo notare il numero di passi effettuati da Gauss-Seidel è minore al numero di passi effettuati con il metodo di Jacobi per ogni matrice di ogni dimensione n, ne deduciamo che Gauss-Seidel è più veloce. L'errore relativo calcolato coi due metodi su una data matrice è molto vicino, ciò sta ad indicare che i risultati calcolati coi due metodi sono quasi identici e quindi la loro accuratezza è quasi la stessa. Il raggio della matrice di iterazione per il metodo di Jacobi rimane uguale indipendentemente dalla dimensione della matrice, mentre il raggio della matrice di iterazione del metodo Gauss varia di poco.

Esercitazione 5

test_nonlin.m

Confrontiamo vari metodi per la risoluzione di un sistema lineare usando i dati presenti nelle tabelle 7.1, 7.2, 7.3 e 7.4 del libro con delle funzioni che possiedono come parametro la funzione da risolvere col metodo, un numero massimo di iterazioni e una tolleranza(10^(-8) che determinano quando terminare il calcolo e poi una serie di parametri tipici per ogni metodo :

- **Metodo di Bisezione:**

Prima funzione radice: 1.414214 passi: 29 errore: 1.873798e-09
Seconda funzione radice: 0.693147 passi: 29 errore: 1.820636e-09
Terza funzione radice: 0.333333 passi: 29 errore: 1.241763e-09
Quarta funzione radice: 3.000326 passi: 11 errore: 3.255208e-04

Il **metodo di Bisezione** prende come parametri degli intervalli e genera una serie di intervalli di larghezza decrescente assumendo ad ogni passo il punto medio come approssimazione della radice e continua finché l'ultimo valore calcolato non è sufficientemente piccolo da assicurarci un buona approssimazione.
Si basa sul teorema di esistenza degli zeri per una funzione continua, il quale garantisce l'esistenza di almeno una radice della funzione su un intervallo se le due funzioni calcolate per i suoi punti estremi hanno segno opposto (si dice in tal caso che il metodo converge).

Osserviamo che con l'aumentare del numero di iterazioni l'errore tende a 0, l'errore delle prime tre funzioni risolte in 29 passi è infatti più piccolo dell'errore calcolato nella quarta funzione, con numero di passi 11, inferiore a 29.
Il metodo ha ordine 1 e il criterio d'arresto è l'annullarsi della funzione nel punto medio, il rimpicciolirsi dell'intervallo al di sotto di una tolleranza(10^(-8) tolleranza valida per questo e tutti i test seguenti) e il superamento di un numero massimo di iterazioni in questo caso 100.

La velocità di convergenza dipende dall'ampiezza dell'intervallo iniziale, dall'equazione e dalla molteplicità della radice. Gli intervalli su cui vengono svolte le funzioni sono [0,2] per le prime tre e [4/3,10/3] per la quarta che necessita di meno passi delle altre.

- Metodo di Newton:
Prima funzione radice: 1.414214 passi: 11 errore: 7.460699e-13

Warning: N. massimo di iterazioni raggiunto
> In newton (line 46)
In test_nonlin (line 64)
Seconda funzione radice: Inf passi: 100 errore: Inf

Terza funzione radice: 0.333333 passi: 7 errore: 5.551115e-17
Quarta funzione radice: 2.999657 passi: 14 errore: 3.425487e-04

Metodo newton
La funzione usata per testare il metodo prende in input un punto iniziale e la derivata della funzione da risolvere.
Sappiamo che f di x è continua in un intervallo ed è ivi dotata di un solo 0. Fissiamo un punto x0 presso la radice e consideriamo la retta tangente alla funzione data nel punto x0.

Il metodo consiste nel considerare come approssimazione successiva il punto x1 intersezione della retta con l'asse delle ascisse e nell'iterare il procedimento.

X k+1-esimo è uguale a:

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

a patto che la derivata non si annulli mai nei punti della successione generata dal metodo.
Il metodo converge quando l'approssimazione iniziale x0 `e "sufficientemente" vicina alla radice x.
Il punto iniziale x0 è posto a 200 per le prime due funzioni, per la terza 0.1 e la quarta 2.9.

Metodo veloce (più veloce del metodo delle Bisecanti) per il calcolo di radici semplici ma potrebbe non convergere se il punto iniziale non è abbastanza vicino alla radice, è il caso della seconda funzione dove il numero di iterazioni supera 100 numero impostato come massimo valore di iterazioni raggiungibili, 200, punto iniziale x0, è infatti troppo distante dalla radice (0.693147 calcolata con gli altri metodi).

Metodi Quasi-Newton (metodo delle corde e delle secanti)

Nei metodi **quasi-Newton** sostituisco al denominatore della funzione del metodo Newton il coefficiente angolare m k-esimo. Nel metodo delle corde m è uguale alla derivata sul punto iniziale e viene mantenuto per tutte le iterazioni, non ha prestazioni soddisfacenti rispetto agli altri metodi, ma vediamolo più nel dettaglio.

Il metodo delle corde consiste nel costruire una successione di punti con il seguente criterio: assegnati due punti iniziali x0 e x1, per ogni n maggiore o uguale a 1, il punto x n+1 esimo sia lo zero della retta passante per il punto:

$$(x_n, f(x_n))$$

E con coefficiente angolare :

$$m = \frac{f(a) - f(x_n)}{a - x_n},$$

La funzione usata nell'esercitazione per questo metodo prende in input un punto iniziale x0 e la derivata della funzione.

- Metodo delle Corde:
Prima funzione radice: 1.414216 passi: 1991 errore: 1.972009e-06

Warning: N. massimo di iterazioni raggiunto
> In corde (line 62)
In test_nonlin (line 90)
Seconda funzione radice: Inf passi: 2000 errore: Inf

Terza funzione radice: 0.333333 passi: 159 errore: 3.181485e-08

Warning: N. massimo di iterazioni raggiunto
> In corde (line 62)
In test_nonlin (line 98)

Quarta funzione radice: Inf passi: 2000 errore: Inf

Il numero massimo di iterazioni è 2000, x0 è impostato rispettivamente a 200, 200, 0.1, 2.9. Notiamo che il metodo delle corde è il peggiore poiché il numero di passi per ogni funzione è maggiore rispetto al numero di passi necessario per ogni altro metodo studiato ed è anche il meno accurato poiché l'errore tende ad essere maggiore rispetto agli altri metodi, non converge nella seconda e nella quarta funzione che necessitano di un numero maggiore di passi rispetto al numero massimo impostato.

- Metodo delle Secanti:
Prima funzione radice: 1.414214 passi: 6 errore: 0.000000e+00
Seconda funzione radice: 0.693147 passi: 8 errore: 4.340694e-11
Terza funzione radice: NaN passi: 10 errore: NaN
Quarta funzione radice: 2.999584 passi: 28 errore: 4.160803e-04

La funzione usata per testare questo metodo prende in input due punti iniziali x0 e x1.

Uno dei metodi più semplici per il calcolo approssimato di un'equazione, consiste nell'approssimare il coefficiente angolare con quello della retta secante la curva nei punti di ascissa x k-esimo e x k-1 esimo, inisialmente x1 e x0.

Coefficiente angolare per il metodo delle secanti è invece:

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

$$x_{n+1} = x_n - \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})} f(x_n)$$

Inoltre la convergenza è locale, cioè dipende dalla scelta dei punti iniziali .

Con numero di iterazioni pari a 100 e i punti iniziali x0 e x1 usati 1 e 2 per la prima e quarta funzione e 2 e 3 per la seconda e terza.
Il metodo delle secanti termina le proprie iterazioni quando si supera un certo valore di tolleranza o un numero massimo di iterazioni, in questo caso 100.
Gli errori calcolati col metodo delle secanti sono i più bassi fra quelli calcolati coi metodi visti fin ora, notiamo infatti che quello relativo alla prima funzione è 0.
Nella terza funzione non avviene convergenza.

L’ordine di convergenza è il rapporto aureo, il costo computazionale produce una riduzione dell’ errore maggiore rispetto al metodo Newton e non richiede la conoscenza della derivata della funzione. Fino ad ora è il metodo più conveniente.

Esercitazione 6

test_interp.m

Attua dei test sul polinomio interpolante (forma canonica e forma di Lagrange), con due metodi diversi per scegliere le ascisse di interpolazione, punti equispaziati calcolati con il comando:

```
x=linspace(-pi,pi,n)'; %campionamento del segmento da -p1 a pi prendendo n punti,
utilizzo del metodo dei punti equidistanti
```

o con gli zeri del polinomio di Chebychev:

```
x=cos(((2*k+1)*pi)./(2*n+2)); %zeri del polinomio di Chebychev
```

su due diverse funzioni da interpolare: f(x) = 1/(1+25x^2) e f(x) = sin (2πx).

La forma canonica è calcolata nello script canint.m mentre la forma di Lagrage da lagrint.m,

entrambi gli script stampano un grafico contenente i punti ottenuti.

Nell’interpolazione polinomiale si cerca di interpolante un polinomio di un grado opportuno nel caso dell’interpolazione lineare pari a 1. Quando si hanno n punti esiste esattamente un polinomio di grado n−1 che passa attraverso tutti tali punti. L’errore di interpolazione è proporzionale alla distanza fra i punti dati elevata alla potenza n-esima.

I grafici dei risultati ottenuti si trovano nella cartella grafici e sono stati calcolati ponendo n=30, numero di punti per campionare il segmento con l’istruzione x=linspace(-pi,pi,n)';

Forma canonica e Lagrange a confronto

Forma Canonica ha un costo computazionale elevato rispetto ad altri algoritmi e in quanto a stabilità della base che amplifica sul risultato piccoli errori sui coefficienti.

La principale differenza fra la forma Canonica e di Lagrange da ricordare è che la forma Canonica necessità di una complessità computazionale maggiore O(n^3) mentre la forma di Lagrange ha come complessità computazionale O(n^2).

Complessità computazionale: numero di operazioni necessarie a risolvere un determinato problema in funzione del numero di dati da trattare.

Quindi possiamo affermare che l'interpolazione di Lagrange è più veloce e quindi più conveniente.

Prima funzione

f(x) = 1/(1+25x^2)

Nei grafici i punti rossi sono i dati da interpolare, la linea blu il polinomio interpolante.

Forma Canonica

Nella prima immagine sono stati usati i nodi equispaziati nella seconda i nodi di Chebychev. Le oscillazioni sono più accentuare nel caso dei nodi equispaziati, agli estremi del grafico, invece che nei nodi di Chebychev con i quali l'interpolazione segue maggiormente l'andamento dei dati.

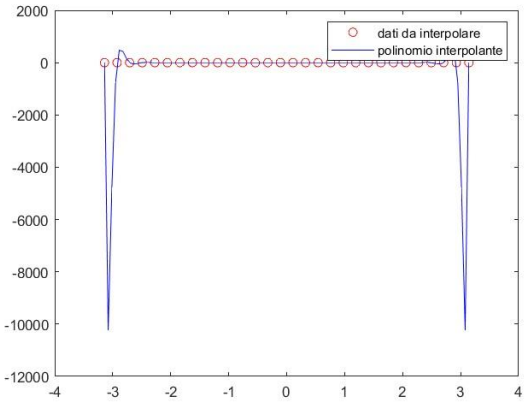
Uno dei parametri di fondamentale importanza nell'interpolazione polinomiale è la distribuzione dei nodi; infatti questa influisce pesantemente sulla precisione

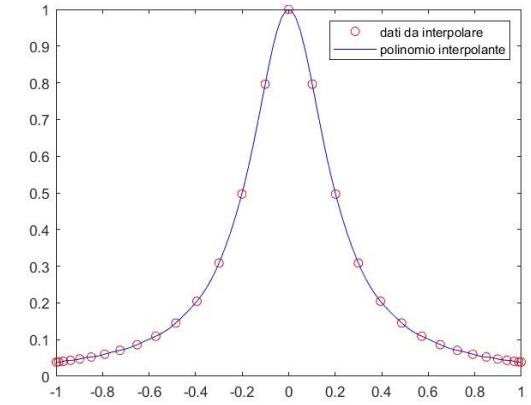
dell'interpolazione.

La distribuzione dei nodi di Chebychev risulta ottimale, nel senso che fa convergere a zero l'errore di interpolazione per n tendente all'infinito, n è il numero dei dati da interpolare, notiamo che l'approssimazione della funzione, rappresentata dalla linea blu segue perfettamente l'andamento dei dati da interpolare per questa quantità di punti, ovvero 30

(ipunti rossi sono i dati da interpolare).

Al contrario con l'utilizzo dei nodi equispaziati all'aumentare di n si ha una minor precisione dell'interpolazione, specialmente agli estremi del grafico.

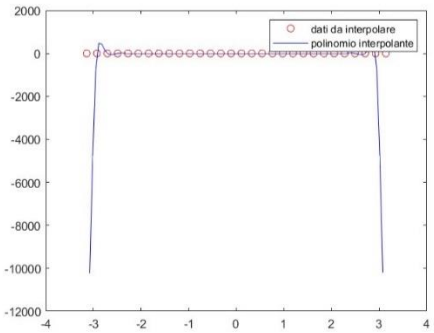


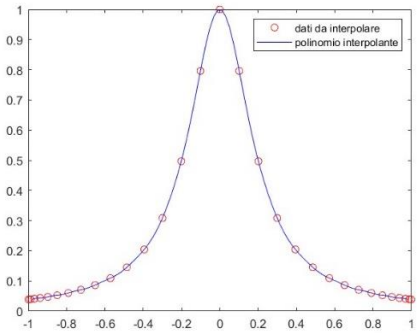


Forma di Lagrange

Il primo grafico è stato ricavato usando i nodi Equispaziati il secondo usando i nodi di Chebychev.

Otteniamo gli stessi grafici ottenuti con la forma Canonica, l'utilizzo dei nodi di Chebychev si dimostra più preciso dell'utilizzo dei nodi equispaziati. Possiamo trarre conclusioni molto simili a quelle osservate per la forma Canonica notiamo inoltre che l'interpolazione di Lagrange è meno precisa con l'utilizzo dei punti equispaziati infatti un dato per ogni estremo del grafico non viene interpolato. Riguardo alla scelta dei punti usando i nodi di Chebychev otteniamo due grafici uguali per i due metodi, in questo caso potremmo preferire Lagrange perché ha una complessità computazionale inferiore alla forma Canonica.





Seconda Funzione

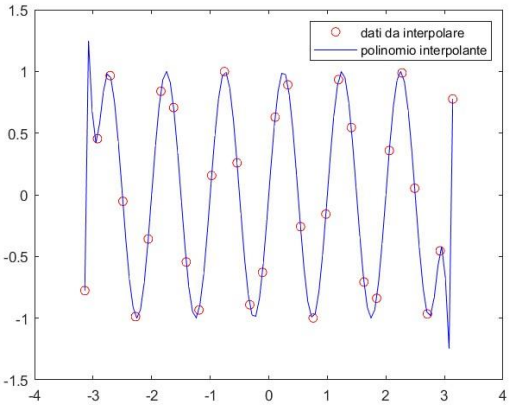
$f(x) = \sin(2\pi x)$

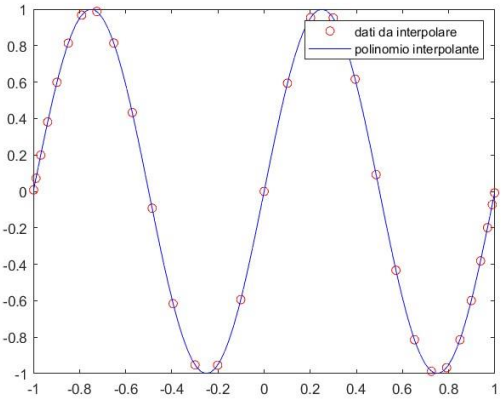
Possiamo osservare le stesse conclusioni osservate con la funzione analizzata precedentemente.

Forma Canonica

Nodi equidistanti e nodi di Chebychev

Notiamo che usando i nodi di Chebychev il polinomio interpolante approssima meglio la funzione, l'utilizzo di nodi equidistanti comporta picchi improvvisi agli estremi. I due grafici sono rispettivamente nodi equidistanti e di Chebychev



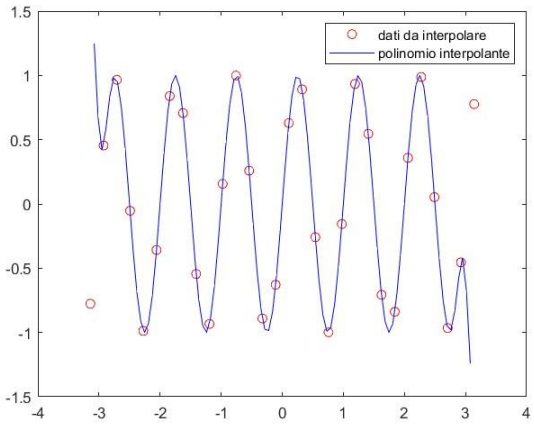


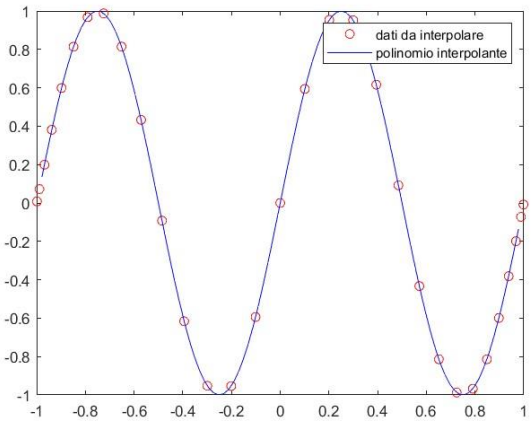
Forma di Lagrange

Nodi equidistanti e nodi di Chebychev

Coi nodi equidistanti alcuni dati agli estremi del grafico non vengono interpolati dal polinomio interpolante quindi l'approssimazione della funzione è meno accurata che con l'utilizzo dei nodi di Chebychev.

I due grafici sottostanti utilizzano rispettivamente nodi equidistanti e nodi di Chebychev.





Conclusioni

Se la funzione passa per tutti i punti di interpolazione, l'approssimazione della funzione è buona. Per valori molto grandi di n (n=30 o n>30) il calcolo di punti equispaziati presenta delle oscillazioni agli estremi e l'approssimazione è meno accurata che con i nodi di Chebychev la cui approssimazione è invece migliore con l'aumentare del numero dei punti e quindi con l'aumentare di n, per i nodi equispaziati è il contrario, sia nella prima che nella seconda funzione. La forma Canonica risulta più precisa per entrambi i metodi di scelta dei punti mentre usando Lagrange e punti equidistanti assieme alcuni dati non vengono interpolati.

Penso che il metodo di scelta migliore dei punti sia il metodo dei nodi di Chebychev perché il polinomio interpolante segue meglio l'andamento dei dati da interpolare, il metodo di interpolazione migliore penso sia l'interpolazione di Lagrange perché ha una complessità computazionale inferiore alla forma Canonica, ma è da usare solo con i nodi di Chebychev.

Mentre se usiamo la forma di Lagrange con il metodo dei nodi equispaziati notiamo che alcuni dati non vengono interpolati dal polinomio interpolante e questo ci fa preferire la forma Canonica se utilizziamo questo metodo di scelta dei punti.

Federica Meloni