

# Elaborato di Calcolo Numerico

Finoia Luca 6132811 Calabrese Filippo 5826217

July 20, 2019

CONTENTS

<b>1</b>	<b>ERRORI ED ARITMETICA FINITA</b>	<b>3</b>
1.1	Esercizio 1 . . . . .	3
1.2	Esercizio 2 . . . . .	5
1.3	Esercizio 3 . . . . .	6
1.4	Esercizio 4 . . . . .	8
<b>2</b>	<b>RADICI DI UN'EQUAZIONE</b>	<b>10</b>
2.1	Esercizio 5 . . . . .	10
2.2	Esercizio 6 . . . . .	15
2.3	Esercizio 7 . . . . .	16
<b>3</b>	<b>SISTEMI LINEARI E NON LINEARI</b>	<b>20</b>
3.1	Esercizio 8 . . . . .	20
3.2	Esercizio 9 . . . . .	22
3.3	Esercizio 10 . . . . .	23
3.4	Esercizio 11 . . . . .	26
3.5	Esercizio 12 . . . . .	27
3.6	Esercizio 13 . . . . .	28
<b>4</b>	<b>APPROSSIMAZIONE DI FUNZIONI</b>	<b>32</b>
4.1	Esercizio 14 . . . . .	32
4.2	Esercizio 15 . . . . .	34
4.3	Esercizio 16 . . . . .	36
4.4	Esercizio 18 . . . . .	38
<b>5</b>	<b>FORMULE DI QUADRATURA</b>	<b>39</b>
5.1	Esercizio 22 . . . . .	39
5.2	Esercizio 23 . . . . .	42

<b>6</b>	<b>CALCOLO DEL GOOGLE PAGERANK</b>	<b>45</b>
6.1	Esercizio 24 . . . . .	45
6.2	Esercizio 25 . . . . .	47
6.3	Esercizio 26 . . . . .	49
6.4	Esercizio 27 . . . . .	50
6.5	Esercizio 28 . . . . .	51

# CHAPTER 1

---

## ERRORI ED ARITMETICA FINITA

### 1.1 Esercizio 1

**Descrizione:** Verificare che, per  $h$  sufficientemente piccolo,  $\frac{3}{2}f(x) - 2f(x - h) + \frac{1}{2}f(x - 2h) = hf'(x) + O(h^3)$ .

*Soluzione:*

considerando il Polinomio di Taylor centrato in  $x_0$  al secondo ordine:

$$f(x_0 - h) = f(x_0) - f^1(x_0)h + \frac{1}{2}f^2(x_0)h^2 + O(h^3) \quad (1.1)$$

si effettua una sostituzione con  $f(x_0 - h)$  nella funzione e si ottiene:

$$\begin{aligned} \frac{2}{3}f(x) - 2 \left[ f(x) - f^1(x)h + \frac{1}{2}f^2(x)h^2 + O(h^3) \right] + \\ + \frac{1}{2} \left[ f(x) - f^1(x)2h + \frac{1}{2}f^2(x)4h^2 + O(h^3) \right] = \\ = hf^1(x) + O(h^3) \end{aligned} \quad (1.2)$$

da cui otteniamo:

$$\begin{aligned} \frac{3}{2}f(x) - 2f(x) + 2f^1(x)h - \\ - f^2(x)h^2 + \frac{1}{2}f(x) - \\ - f^1(x)h + f^2(x)h^2 + O(h^3) = \\ = hf^1(x) + O(h^3) \end{aligned} \quad (1.3)$$

quindi per  $h$  sufficientemente piccoli si ottiene:

$$hf'(x) + O(h^3) = hf'(x) + O(h^3) \quad (1.4)$$

## 1.2 Esercizio 2

**Descrizione:** Quanti sono i numeri di macchina normalizzati della doppia precisione IEEE? Argomentare la risposta.

*Soluzione:*

L'insieme dei numeri di macchina  $\mathcal{M}$  é un insieme finito di elementi, tramite i quali é possibile rappresentare un insieme denso  $\mathcal{I}$  e che rappresenta un sottoinsieme dei numeri reali  $\mathcal{I} \subset \mathbb{R}$ .

$$\mathcal{I} = [-real_{max}, -real_{min}] \cup \{0\} \cup [real_{min}, real_{max}] \quad (1.5)$$

con  $real_{max}$  e  $real_{min}$  che indicano rispettivamente il piú grande ed il piú piccolo in valore assoluto tra i numeri macchina diversi da 0.

Per trovare i numeri di macchina normalizzati che possono essere ottenuti della doppia precisione IEEE possiamo considerare la seguente formula

$$b^s \cdot b^m \cdot (b^e - b^s) \quad (1.6)$$

dove indichiamo con  $b$  la base utilizzata,  $s$  i bit riservati per il segno,  $m$  i bit riservati per la mantissa ed  $e$  i bit riservati per l'esponente.

Considerando che lo standard *ANSI/IEEE 754-1985* utilizza una base binaria, per la rappresentazione dei numeri reali in singola e doppia precisione, abbiamo  $b = 2$  indipendentemente dalla precisione utilizzata. Inoltre, per il formato a doppia precisione abbiamo 1 bit per il segno ( $s = 1$ ), 52 bit per la mantissa ( $m = 52$ ) e 11 bit per l'esponente ( $e = 11$ ),

andando a sostituire i valori appena ricavati nella formula (1.6) otteniamo i numeri di macchina normalizzati che possono essere ottenuti della doppia precisione IEEE, quindi:

$$\begin{aligned} &= 2^1 \cdot 2^{52} \cdot (2^{11} - 2^1) = \\ &= 2^{1+52} \cdot (2^{11} - 2^1) = \\ &= 2^{1+52+11} - 2^{1+52+1} = \\ &= 2^{64} - 2^{54} = 18,428,729,675,200,069,632 \end{aligned} \quad (1.7)$$

### 1.3 Esercizio 3

**Descrizione:** Eseguire il seguente script Matlab e spiegare i risultati ottenuti:

```
format long e
n=75;
u=1e-300; for i=1:n, u=u*2; end, for i=1:n, u=u/2; end, u
u=1e-300; for i=1:n, u=u/2; end, for i=1:n, u=u*2; end, u
```

**Svolgimento:**

```
>> format long e
>> n=75;
>> u=1e-300; for i=1:n, u=u*2; end, for i=1:n, u=u/2; end, u

u =

    1.0000000000000000e-300

>> u=1e-300; for i=1:n, u=u/2; end, for i=1:n, u=u*2; end, u

u =

    1.119916342203863e-300

>> |
```

in questo script Matlab viene moltiplicato per 2 un valore  $u=1e-300$  per 75 volte, successivamente questo valore  $u$  viene diviso per 2 per 75 volte e infine si visualizza il valore in  $u$ , poi, viene riportato  $u$  al valore di partenza, cioè  $u=1e-300$ , e viene prima diviso per 2 per 75 volte, poi moltiplicato per 2 per 75 volte e infine si visualizza il valore in  $u$ .

Se fossimo in matematica esatta, in entrambi i casi, avremmo  $u$  che sarebbe uguale al valore di partenza poiché è stato prima moltiplicato e diviso per la stessa cifra ( $2^{75}$ ) e poi diviso e moltiplicato per la stessa cifra ( $2^{75}$ ). Tuttavia in Matlab i calcoli vengono effettuati in matematica finita e pertanto nel primo caso, in cui la moltiplicazione precede la divisione otteniamo lo stesso risultato che otterremmo in matematica esatta ovvero  $1.0000000000000000e-300$ , nel secondo caso invece otteniamo un valore differente ovvero  $1.119916342203863e-300$ .

La causa per cui nel secondo caso otteniamo un valore diverso da quello iniziale è da attribuire ad un errore di round-off, per la precisione un errore di underflow, causato nel primo ciclo in cui viene  $u$  viene diviso per 2 per 75 volte, dove, durante una delle iterazioni viene a verificarsi che,  $0 < |u| < r_{min}$ . La

mantissa viene dunque denormalizzata e successivamente viene divisa per 2 diverse volte, portando alla generazione dell'errore.



## 1.4 Esercizio 4

**Descrizione:** Eseguire le seguenti istruzioni Matlab e spiegare i risultati ottenuti:

```
format long e  
a = 1.1111111111111111  
b = 1.1111111111111111  
a + b  
a - b
```

*Soluzione:*

```
>> format long e  
a = 1.1111111111111111;  
b = 1.1111111111111111;  
a + b  
a - b  
  
ans =  
  
2.2222222222222221e+00  
  
ans =  
  
8.881784197001252e-16  
  
fx >>
```

In questo script matlab vengono sommati due numeri  $a$  e  $b$  e poi viene fatta la differenza tra i due numeri  $a$  e  $b$ .

Nel caso della somma tra i due numeri il risultato ottenuto in Matlab corrisponde con il risultato ottenuto in matematica esatta in quanto una somma di numeri concorde é sempre ben condizionata poiché abbiamo

$$|\varepsilon_y| \leq \frac{|a| + |b|}{|a + b|} \varepsilon_x \equiv k \cdot \varepsilon_x \quad (1.8)$$

in cui  $k$  é il numero di condizionamento

$$k = \frac{|a| + |b|}{|a + b|} \quad (1.9)$$

e per una somma di numeri concorde  $k=1$  e quindi é sempre ben condizionata.

Nel caso della differenza tra i due numeri  $a$  e  $b$  invece il risultato ottenuto in Matlab differisce dal risultato ottenuto in matematica esatta perché esso é 0.000000000000001 . In questo caso si verifica quindi il fenomeno della cancellazione numerica in cui si ha una perdita di cifre significative nel momento in cui viene effettuata una somma tra due numeri prossimi in valore assoluto ma opposti in segno. Infatti in questo caso, in cui  $a \approx b$ , si ottiene un numero di condizionamento arbitrariamente grande e quindi l'operazione di somma algebrica risulta mal condizionata.

## CHAPTER 2

---

## RADICI DI UN'EQUAZIONE

### 2.1 Esercizio 5

**Descrizione:** Scrivere function Matlab distinte che implementino efficientemente i seguenti metodi per la ricerca degli zeri di una funzione:

- metodo di bisezione
- metodo di Newton
- metodo delle secanti
- metodo delle corde

Detta  $x_i$  l'approssimazione al passo  $i$ -esimo, utilizzando come criterio di arresto  $|\Delta x_i| \leq \text{tol} \cdot (1 + |x_i|)$  essendo  $\text{tol}$  una opportuna tolleranza specificata in ingresso.

*Soluzione:*

## metodo di bisezione

```
1 function [x,i] = bisezione(fun,a,b,tolx)
2 %   x=bisezione(fun,a,b,tolx)
3 %   parametri in ingresso
4 %   fun -> funzione in cui si effettua la ricerca degli zeri
5 %   a,b -> estremi dell'intervallo di ricerca
6 %   tolx -> parametro che indica la tolleranza su x
7 %   valori di ritorno
8 %   x -> radice della funzione
9 %   i -> numero di iterazioni
10 %   il metodo di bisezione restituisce la radice di una funzione fun in
11 %   un intervallo [a,b] ed il numero di iterazioni effettuate
12 if a>b, error('intervallo non valido'),end
13 fa=feval(fun,a);
14 fb=feval(fun,b);
15 if fa*fb>0,error('nessuna radice in [a,b]'),end
16 if fa==0,x=a;return; elseif fb==0,x=b;return, end
17 nmax = ceil(log2((b-a)/tolx));
18 for i=1:nmax
19     df=abs((fb-fa)/(b-a));
20     x=(a+b)/2; fx=feval(fun,x);
21     if abs(fx)<=df*(1+abs(x))*tolx,return;
22     elseif fa*fx<0
23         b=x;
24         fb=fx;
25     else
26         a=x;
27         fa=fx;
28     end
29 end
30 return
```

## metodo di Newton

```
1 function [x,i]= Newton(f,f1,x0,tol,itmax)
2 %   x=newton(f,f1,x0,tol,itmax)
3 %   parametri in ingresso
4 %   f -> funzione in cui si effettua la ricerca degli zeri
5 %   f1 -> derivata della funzione f
6 %   x0 -> punto di partenza
7 %   tol -> parametro che indica la tolleranza
8 %   itmax -> numero massimo di iterazioni
9 %   valori di ritorno
10 %   x -> radice della funzione
11 %   i -> numero di iterazioni
12 %   il metodo di newton restituisce la radice di una funzione f
13 %   ed il numero di iterazioni effettuate
14 fx = feval(f,x0);
15 f1x = feval(f1,x0);
16 x = x0-fx/f1x;
17 i = 0;
18 while (i<itmax) && (abs(x-x0)>=(1+abs(x))*tol)
19     i = i+1;
20     x0 = x;
21     fx = feval(f,x0);
22     f1x = feval(f1,x0);
23     x = x0-fx/f1x;
24 end
25 if abs(x-x0) >= (1+abs(x))*tol
26     error('il metodo non converge')
27 end
28 return
```

## metodo delle secanti

```
1 function [x,i]= secanti(f,f1,x0,tol,itmax)
2 %   x=secanti(f,f1,x0,tol,itmax)
3 %   parametri in ingresso
4 %   f -> funzione in cui si effettua la ricerca degli zeri
5 %   f1 -> derivata della funzione f
6 %   x0 -> punto di partenza
7 %   tol -> parametro che indica la tolleranza
8 %   itmax -> numero massimo di iterazioni
9 %   valori di ritorno
10 %   x -> radice della funzione
11 %   i -> numero di iterazioni
12 %   il metodo delle secanti restituisce la radice di una funzione f
13 %   ed il numero di iterazioni effettuate
14 fx = feval(f,x0);
15 f1x = feval(f1,x0);
16 x = x0-fx/f1x;
17 i = 0;
18 while (i<itmax) && (abs(x-x0)>=(1+abs(x))*tol)
19     i=i+1;
20     fx0 = fx;
21     fx = feval(f,x);
22     x1 = (fx*x0-fx0*x)/(fx-fx0);
23     x0 = x;
24     x = x1;
25 end
26 if abs(x-x0) >= (1+abs(x))*tol
27     error('il metodo non converge')
28 end
29 return
```

## metodo delle corde

```
1 function [x,i]= corde(f,f1,x0,tol,itmax)
2 %   x=corde(f,f1,x0,tol,itmax)
3 %   parametri in ingresso
4 %   f -> funzione in cui si effettua la ricerca degli zeri
5 %   f1 -> derivata della funzione f
6 %   x0 -> punto di partenza
7 %   tol -> parametro che indica la tolleranza
8 %   itmax -> numero massimo di iterazioni
9 %   valori di ritorno
10 %   x -> radice della funzione
11 %   i -> numero di iterazioni
12 %   il metodo delle corde restituisce la radice di una funzione f
13 %   ed il numero di iterazioni effettuate
14 fx = feval(f,x0);
15 f1x = feval(f1,x0);
16 x = x0-fx/f1x;
17 i = 0;
18 while (i<itmax) && (abs(x-x0)>=(1+abs(x))*tol)
19     x0 = x;
20     fx = feval(f,x0);
21     x = x0-fx/f1x ;
22     i = i+1;
23 end
24 if abs(x-x0) >= (1+abs(x))*tol
25     error('il metodo non converge')
26 end
27 return
```

## 2.2 Esercizio 6

**Descrizione:** Utilizzare le function del precedente esercizio per determinare una approssimazione della radice della funzione  $f(x) = x - e^{-x}\cos(x/100)$ , per  $tol = 10^{-i}$ ,  $i = 1, 2, \dots, 12$ , partendo da  $x_0 = -1$ . Per il metodo di bisezione, utilizzare  $[-1, 1]$ , come intervallo di confidenza iniziale. Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare il relativo costo computazionale.

*Soluzione:*

tolleranza	bisezione		Newton		secanti		corde	
	radice	iterazioni	radice	iterazioni	radice	iterazioni	radice	iterazioni
$10^{-1}$	5.000000000000000e-01	2	5.663058026182579e-01	2	5.662928457961099e-01	3	4.021808606806709e-01	2
$10^{-2}$	5.625000000000000e-01	5	5.671373451065942e-01	3	5.671339926160822e-01	4	5.495185718942192e-01	6
$10^{-3}$	5.664062500000000e-01	9	5.671373451065942e-01	3	5.671339926160822e-01	4	5.651741531554760e-01	10
$10^{-4}$	5.671386718750000e-01	13	5.671374702931882e-01	4	5.671374697616252e-01	5	5.670103627779298e-01	15
$10^{-5}$	5.671386718750000e-01	13	5.671374702931882e-01	4	5.671374697616252e-01	5	5.671232371727831e-01	19
$10^{-6}$	5.671386718750000e-01	13	5.671374702931882e-01	4	5.671374702931907e-01	6	5.671358764608520e-01	23
$10^{-7}$	5.671374797821045e-01	23	5.671374702931882e-01	4	5.671374702931907e-01	6	5.671372918143774e-01	27
$10^{-8}$	5.671374797821045e-01	23	5.671374702931911e-01	5	5.671374702931907e-01	6	5.671374503069613e-01	31
$10^{-9}$	5.671374704688787e-01	30	5.671374702931911e-01	5	5.671374702931907e-01	6	5.671374689985149e-01	36
$10^{-10}$	5.671374702360481e-01	33	5.671374702931911e-01	5	5.671374702931911e-01	7	5.671374701482120e-01	40
$10^{-11}$	5.671374702942558e-01	35	5.671374702931911e-01	5	5.671374702931911e-01	7	5.671374702769563e-01	44
$10^{-12}$	5.671374702942558e-01	35	5.671374702931911e-01	5	5.671374702931911e-01	7	5.671374702913732e-01	48

Nella tabella sono riportate le radici ricavate con ogni metodo e il relativo numero di iterazioni necessarie per ottenere tale risultato per ogni  $tol = 10^{-i}$ ,  $i = 1, 2, \dots, 12$ .

Il metodo di bisezione ed il metodo delle corde hanno una valutazione di funzione e convergenza lineare mentre il metodo di Newton ha due valutazioni di funzioni, una per la funzione e una per la sua derivata, e convergenza quadratica. Da quanto detto, consegue che col metodo di Newton si ha una maggiore velocità nella ricerca della radice ma si ha anche un costo computazionale maggiore per singola iterazione. Il metodo delle secanti ha una valutazione di funzione, quindi un costo computazionale minore del metodo di Newton, e una convergenza  $\approx 1.618$ , quindi ha una velocità maggiore dei metodi di bisezione e delle secanti.



## 2.3 Esercizio 7

**Descrizione:** Calcolare la molteplicità della radice nulla della funzione  $f(x) = x^2 \cdot \sin(x^2)$ . Confrontare, quindi, i metodi di Newton, Newton modificato, e di Aitken, per approssimarla per gli stessi valori di  $\text{tol}$  del precedente esercizio (ed utilizzando il medesimo criterio di arresto), partendo da  $x_0 = 1$ . Tabulare e commentare i risultati ottenuti.

*Soluzione:*

La molteplicità  $m$  della radice nulla della funzione  $f(x) = x^2 \cdot \sin(x^2)$  è  $m = 4$ . Per calcolarla, è stato utilizzato il seguente script matlab:

### Calcolo della Molteplicità

```
1 function y = molteplicita (fun , x0)
2 %
3 % y = molteplicita (fun , x0)
4 % Funzione che calcola e rende la molteplicita '
5 % dello zero di funzione (x0) specificato .
6 %
7 if feval (fun , x0 )~= 0
8     error ( 'Il valore in x0 deve essere uno zero per la funzione .' );
9 end
10 syms x;
11 f1 = fun ;
12 y = 0;
13 while ( feval (f1 , x0) == 0)
14     f1 = eval ([ '@(x)' char ( diff (f1(x )))]);
15     y = y +1;
16 end
17 return
```

## metodo di Newton modificato

```
1 function [x, i] = NewtonMod(f, df, m, x0, tol, itmax)
2 %   x=NewtonMod(f,df,m,x0,tol,itmax)
3 %   parametri in ingresso
4 %   f -> funzione in cui si effettua la ricerca degli zeri
5 %   df -> derivata della funzione f
6 %   m -> molteplicità della radice
7 %   x0 -> punto di partenza
8 %   tol -> parametro che indica la tolleranza
9 %   itmax -> numero massimo di iterazioni
10 %   valori di ritorno
11 %   x -> radice della funzione
12 %   i -> numero di iterazioni
13 %   il metodo di newton restituisce la radice di una funzione f
14 %   ed il numero di iterazioni effettuate
15 i=1;
16 fx=feval(f,x0);
17 dfx=feval(df,x0);
18 x=x0-m*(fx/dfx);
19 while(i<itmax && abs(x - x0) >= tol*(1+abs(x)))
20     i=i+1;
21     x0=x;
22     fx=feval(f,x0);
23     dfx=feval(df,x0);
24     x=x0-m*(fx/dfx);
25 end
26 if (abs(x - x0) >= tol*(1+abs(x)))
27     error('il metodo non converge');
28 end
29 return
```

## metodo di Aitken

```
1 function [x,i] = aitken(f, f1, x0, tol, itmax)
2 %   x=aitken(f,f1,x0,tol,itmax)
3 %   parametri in ingresso
4 %   f -> funzione in cui si effettua la ricerca degli zeri
5 %   f1 -> derivata della funzione f
6 %   x0 -> punto di partenza
7 %   tol -> parametro che indica la tolleranza
8 %   itmax -> numero massimo di iterazioni
9 %   valori di ritorno
10 %   x -> radice della funzione
11 %   i -> numero di iterazioni
12 %   il metodo di aitken restituisce la radice di una funzione f
13 %   ed il numero di iterazioni effettuate
14 i=0;
15 x=x0;
16 vai=1;
17 while(i<itmax) && vai
18     i=i+1;
19     x0=x;
20     fx=feval(f,x0);
21     f1x=feval(f1,x0);
22     x1=x0-fx/f1x;
23     fx=feval(f,x1);
24     f1x=feval(f1,x1);
25     x=x1-fx/f1x;
26     x=(x*x0-x1^2)/(x-2*x1+x0);
27     vai = abs(x-x0)>tol*(1+abs(x));
28 end
29 if vai, error('il metodo non converge'),end
30 return
```

tolleranza	Newton		NewtonMod		Aitken	
	radice	iterazioni	radice	iterazioni	radice	iterazioni
10 <sup>-1</sup>	3.84317880607062e-01	2	0	3	6.4929e-19	3
10 <sup>-2</sup>	2.8805139309385e-02	11	0	3	6.4929e-19	3
10 <sup>-3</sup>	2.883766303035e-03	19	0	3	6.4929e-19	3
10 <sup>-4</sup>	2.887022508866152e-04	27	0	3	0	4
10 <sup>-5</sup>	2.890282391459780e-05	35	NaN	4	0	4
10 <sup>-6</sup>	2.893545954951113e-06	43	NaN	4	0	4
10 <sup>-7</sup>	2.896813203496438e-07	51	NaN	4	0	4
10 <sup>-8</sup>	2.900084141256734e-08	59	NaN	4	0	4
10 <sup>-9</sup>	2.903358772397679e-09	67	NaN	4	0	4
10 <sup>-10</sup>	2.906637101089656e-10	75	NaN	4	0	4
10 <sup>-11</sup>	2.909919131507756e-11	83	NaN	4	0	4
10 <sup>-12</sup>	2.913204867831785e-12	91	NaN	4	0	4

dai risultati in tabella si evince che il metodo di Newton richiede un numero elevato di iterazioni nel caso di radici multiple, diventando inefficiente in confronto al metodo di Newton modificato e il metodo di Aitken che richiedono un numero di iterazioni molto minore e risultando così più efficienti nel caso di radici multiple ripristinando la convergenza quadratica.

## CHAPTER 3

---

## SISTEMI LINEARI E NON LINEARI

### 3.1 Esercizio 8

**Descrizione:** Scrivere una function Matlab che, data in ingresso una matrice  $A$ , restituisca una matrice,  $LU$ , che contenga l'informazione sui suoi fattori  $L$  ed  $U$ , ed un vettore  $p$  contenente la relativa permutazione, della fattorizzazione  $LU$  con pivoting parziale di  $A$ : **function**  $[LU, p] = palu(A)$ . Curare particolarmente la scrittura e l'efficienza della function.

*Soluzione:*

#### fattorizzazione LU con pivoting parziale

```
1 function [LU, p] = palu(A)
2 % function [LU, p] = palu(A)
3 % parametri
4 % A -> matrice quadrata non singolare
5 %
6 % funzione che, data una matrice n*n, non singolare, la fattorizza in A=LU
7 % tramite pivoting parziale. restituisce le informazioni dei fattori
8 % memorizzati in A e il vettore che memorizza le permutazioni effettuate
9 LU=A;
10 [m,n]=size(LU);
11 if m~=n, error('matrice non quadrata');end
12 p=1:n;
13 for i=1:n-1
```

```

14     [mi, ki]=max(abs(A(i:n, i)));
15     if mi==0, error('matrice singolare');end
16     ki=ki+i-1;
17     if ki>i
18         LU([i ki],:)=LU([ki i],:);
19         p([i ki])=p([ki i]);
20     end
21     LU(i+1:n, i)=LU(i+1:n, i)/LU(i, i);
22     LU(i+1:n, i+1:n)= LU(i+1:n, i+1:n)-LU(i+1:n, i)*LU(i, i+1:n);
23 end
24 return

```

## 3.2 Esercizio 9

**Descrizione:** Scrivere una function Matlab che, data in ingresso la matrice  $LU$  ed il vettore  $p$  creati dalla function del precedente esercizio, ed il termine noto del sistema lineare  $Ax = b$ , ne calcoli la soluzione: **function**  $x = \text{lusolve}(LU, p, b)$ . Curare particolarmente la scrittura e l'efficienza della function.

**Svolgimento:**

### Risoluzione matrice fattorizzata LU

```
1 function x = lusolve(LU,p,b)
2 % x=lusolve(LU,p,b)
3 % parametri
4 % LU-> matrice nxn fattorizzata LU
5 % p->vettore delle permutazioni
6 % b->vettore dei termini noti
7 % Data una matrice nxn fattorizzata LU, p vettore delle permutazioni e
8 % b di lunghezza n, risolve Ax=b, cioè: LUx=b Ux=y e Ly=b
9 [m,n]= size (LU);
10 lb = length (b);
11 if m~=n, error ( 'La matrice LU non e quadrata .' ); end
12 if length (p )~= n
13     error ( 'Vettore di permutazione non consistente con la matrice ' ); end
14 if length (b )~= n
15     error ( 'Vettore di permutazione non consistente con la matrice ' ); end
16 b=b(p);
17 x=b(:);
18 for j=2:n
19     x(j:n)=x(j:n)-LU(j:n,j-1)*x(j-1);
20 end
21 for j=n:-1:1
22     x(j)=x(j)/LU(j,j);
23     x(1:j-1)=x(1:j-1)-LU(1:j-1,j)*x(j);
24 end
25 return
```

### 3.3 Esercizio 10

**Descrizione:** Scaricare la function *cremat* che crea sistemi lineari  $n \times n$  la cui soluzione è il vettore  $v = (1 \dots n)^T$ . Eseguire, quindi, lo script Matlab (vedere esercizio) per testare le function dei precedenti esercizi. Confrontare i risultati ottenuti con quelli attesi, e dare una spiegazione esauriente degli stessi.

*Soluzione:*

Una volta scaricata la funzione *cremat*

#### **cremat**

```
1 function [A,b] = cremat(n,k,simme)
2 %
3 % [A,b] = cremat(n,k,simme) Crea una matrice A nxn ed un termine noto b,
4 % in modo che la soluzione del sistema lineare
5 % A*x=b sia x = [1,2,...,n]^T.
6 % k non ve lo dico a cosa serve.
7 % simme, se specificato, crea una matrice
8 % simmetrica e definita positiva.
9 %
10 if nargin==1
11     sigma = 1/n;
12 else
13     sigma = 10^(-k);
14 end
15 rng(0);
16 [q1,r1] = qr(rand(n));
17 if nargin==3
18     q2 = q1';
19 else
20     [q2,r1] = qr(rand(n));
21 end
22 A = q1*diag([sigma 2/n:1/n:1])*q2;
23 x = [1:n]';
24 b = A*x;
25 return
```

ed eseguito il seguente script



```

n = 10;
x = zeros(n,15);
for i = 1:15
    [A,b] = cremat(n,i);
    [LU,p] = palu(A);
    x(:,i) = lusolve(LU,p,b);
end

```

otteniamo i seguenti risultati

```

disp(x(1:10,1:5))
1.0000    1.0000    1.0000    1.0000    1.0000
2.0000    2.0000    2.0000    2.0000    2.0000
3.0000    3.0000    3.0000    3.0000    3.0000
4.0000    4.0000    4.0000    4.0000    4.0000
5.0000    5.0000    5.0000    5.0000    5.0000
6.0000    6.0000    6.0000    6.0000    6.0000
7.0000    7.0000    7.0000    7.0000    7.0000
8.0000    8.0000    8.0000    8.0000    8.0000
9.0000    9.0000    9.0000    9.0000    9.0000
10.0000   10.0000   10.0000   10.0000   10.0000

```

```

disp(x(1:10,6:10))
1.0000    1.0000    1.0000    1.0000    1.0000
2.0000    2.0000    2.0000    2.0000    2.0000
3.0000    3.0000    3.0000    3.0000    3.0000
4.0000    4.0000    4.0000    4.0000    4.0000
5.0000    5.0000    5.0000    5.0000    5.0000
6.0000    6.0000    6.0000    6.0000    6.0000
7.0000    7.0000    7.0000    7.0000    7.0000
8.0000    8.0000    8.0000    8.0000    8.0000
9.0000    9.0000    9.0000    9.0000    9.0000
10.0000   10.0000   10.0000   10.0000   10.0000

```

```

disp(x(1:10,11:15))
1.0000    0.9999    1.0022    1.0060    0.8926
2.0000    2.0002    1.9935    1.9825    2.3131
3.0000    2.9999    3.0021    3.0056    2.8997
4.0000    4.0002    3.9931    3.9815    4.3314
5.0000    4.9995    5.0174    5.0466    4.1649
6.0000    6.0003    5.9914    5.9769    6.4133
7.0000    7.0002    6.9934    6.9823    7.3175
8.0000    8.0000    7.9994    7.9983    8.0300
9.0000    9.0002    8.9940    8.9839    9.2881
10.0000   10.0001    9.9976    9.9936   10.1138

```

notiamo che i risultati ottenuti coincidono con i risultati attesi per  $k \leq 12$ , quindi abbiamo dei risultati perturbati.

Questo vuol dire che per il condizionamento del problema per le matrici

$$\frac{\|\Delta x\|}{\|x\|} \leq k(A) \cdot \left( \frac{\|\Delta b\|}{\|b\|} + \frac{\|\Delta A\|}{\|A\|} \right) \quad (3.1)$$

abbiamo un numero di condizionamento di A molto piccolo e quindi una matrice ben condizionata per  $1 \leq k < 12$  ed un numero di condizionamento di A  $\gg 1$  e quindi una matrice mal condizionata per  $k \geq 12$

### 3.4 Esercizio 11

**Descrizione:** Scrivere una function Matlab che, data in ingresso una matrice  $A \in \mathbb{R}^{m \times n}$ , con  $m \geq n = \text{rank}(A)$ , restituisca una matrice,  $QR$ , che contenga l'informazione sui fattori  $Q$  ed  $R$  della fattorizzazione  $QR$  di  $A$ : **function**  $QR = \text{myqr}(A)$ . Curare particolarmente la scrittura e l'efficienza della function.  
*Soluzione:*

#### fattorizzazione QR

```
1 function QR=myqr(A)
2 % QR=myqr(A)
3 % parametri
4 % A -> matrice da fattorizzare QR
5 % funzione che fattorizza QR una matrice A
6 QR=A;
7 [m,n]=size(QR);
8 if m<n,error('il numero di righe è minore del numero di colonne'),end
9 for i=1:n
10     alfa=norm(QR(i:m,i));
11     if alfa==0,error('la matrice A non ha rango massimo'),end
12     if QR(i,i)>=0, alfa=-alfa;end
13     v1 = QR(i,i)-alfa;
14     QR(i,i)=alfa;
15     QR(i+1:m,i)=QR(i+1:m,i)/v1;
16     beta = -v1/alfa;
17     QR(i:m,i+1:n)=QR(i:m,i+1:n)-(beta*[1;QR(i+1:m,i)])*...
18         ([1;QR(i+1:m,i)]'*QR(i:m,i+1:n));
19 end
20 return
```

### 3.5 Esercizio 12

**Descrizione:** scrivere una function Matlab che, data in ingresso la matrice  $QR$  creata dalla function del precedente esercizio, ed il termine noto del sistema lineare  $Ax = b$ , ne calcoli la soluzione nel senso dei minimi quadrati: function  $x = qrsolve(QR,b)$ . Curare particolarmente la scrittura e l'efficienza della function.

**Svolgimento:**

#### risoluzione matrice fattorizzata QR

```
1 function x= qrsolve(A,b)
2 % x=qrsolve(A,b)
3 % parametri
4 % A -> Matrice fattorizzata QR
5 % b -> vettore dei termini noti
6 % la funzione risolve il sistema lineare Ax=b dove A è
7 % una matrice fattorizzata QR
8 [m,n]=size(A);
9 x=b;
10 if length(b)~=m,error('dati inconsistenti');end
11 for i=1:n
12     v=[1; A(i+1:m,i)];
13     beta = 2/(v'*v);
14     x(i:m)=x(i:m)-(beta*(v'*x(i:m)))*v;
15 end
16 x=x(1:n);
17 for i=n:-1:1
18     x(i)=x(i)/A(i,i);
19     if i>1
20         x(1:i-1)=x(1:i-1)-x(i)*A(1:i-1,i);
21     end
22 end
23 return
```

### 3.6 Esercizio 13

**Descrizione:** Scaricare la function *cremat1* che crea sistemi lineari  $m \times n$ , con  $m \geq n$ , la cui soluzione (nel senso dei minimi quadrati) é il vettore  $x = (1 \cdots n)^T$ . Eseguire, quindi, lo script Matlab (vedere esercizio) per testare le function dei precedenti esercizi.

**Soluzione:**

una volta scaricata la function *cremat1*

#### **cremat1**

```
1 function [A,b] = cremat1(m,n)
2 %
3 % [A,b] = cremat1(m,n)      Crea una matrice A nxn ed un termine noto b,
4 %                          in modo che la soluzione del sistema lineare ,
5 %                          A*x=b, nel senso dei minimi quadrati, sia
6 %                          x = [1,2,...,n]^T.
7 %
8 rng(0);
9 A = rand(m,n);
10 [q,r] = qr(A);
11 b = r*[1:n]';
12 b(n+1:m) = rand(m-n,1);
13 b = q*b;
14 return
```

ed aver eseguito lo script

```
for n = 5:10
    xx = [1:n]';
    for m = n:n+10
        [A,b] = cremat1(m,n);
        QR = myqr(A);
        x = qrsolve(QR,b);
        disp([m n norm(x-xx)])
    end
end
```

otteniamo i seguenti risultati

n	m	norm(x-xx)
5	5	0.1556e <sup>-12</sup>
6	5	0.0215e <sup>-12</sup>
7	5	0.0130e <sup>-12</sup>
8	5	0.0249e <sup>-12</sup>
9	5	0.0053e <sup>-12</sup>
10	5	0.0117e <sup>-12</sup>
11	5	0.0097e <sup>-12</sup>
12	5	0.0040e <sup>-12</sup>
13	5	0.0056e <sup>-12</sup>
14	5	0.0054e <sup>-12</sup>
15	5	0.0047e <sup>-12</sup>
6	6	0.0539e <sup>-12</sup>
7	6	0.0129e <sup>-12</sup>
8	6	0.0261e <sup>-12</sup>
9	6	0.0170e <sup>-12</sup>
10	6	0.0137e <sup>-12</sup>
11	6	0.0210e <sup>-12</sup>
12	6	0.0128e <sup>-12</sup>
13	6	0.0106e <sup>-12</sup>
14	6	0.0056e <sup>-12</sup>
15	6	0.0084e <sup>-12</sup>
16	6	0.0066e <sup>-12</sup>
7	7	0.0275e <sup>-12</sup>
8	7	0.0536e <sup>-12</sup>
9	7	0.0093e <sup>-12</sup>
10	7	0.0215e <sup>-12</sup>
11	7	0.0265e <sup>-12</sup>
12	7	0.0188e <sup>-12</sup>
13	7	0.0104e <sup>-12</sup>
14	7	0.0236e <sup>-12</sup>

n	m	norm(x-xx)
15	7	0.0165e <sup>-12</sup>
16	7	0.0077e <sup>-12</sup>
17	7	0.0158e <sup>-12</sup>
8	8	0.0941e <sup>-12</sup>
9	8	0.0227e <sup>-12</sup>
10	8	0.0414e <sup>-12</sup>
11	8	0.0324e <sup>-12</sup>
12	8	0.0290e <sup>-12</sup>
13	8	0.0140e <sup>-12</sup>
14	8	0.0121e <sup>-12</sup>
15	8	0.0378e <sup>-12</sup>
16	8	0.0151e <sup>-12</sup>
17	8	0.0115e <sup>-12</sup>
18	8	0.0136e <sup>-12</sup>
9	9	0.0703e <sup>-12</sup>
10	9	0.0712e <sup>-12</sup>
11	9	0.0590e <sup>-12</sup>
12	9	0.0692e <sup>-12</sup>
13	9	0.0222e <sup>-12</sup>
14	9	0.0149e <sup>-12</sup>
15	9	0.0435e <sup>-12</sup>
16	9	0.0285e <sup>-12</sup>
17	9	0.0165e <sup>-12</sup>
18	9	0.0218e <sup>-12</sup>
19	9	0.0263e <sup>-12</sup>
10	10	0.0613e <sup>-12</sup>
11	10	0.1235e <sup>-12</sup>
12	10	0.0683e <sup>-12</sup>
13	10	0.0270e <sup>-12</sup>
14	10	0.0481e <sup>-12</sup>
15	10	0.0395e <sup>-12</sup>
16	10	0.0555e <sup>-12</sup>
17	10	0.0236e <sup>-12</sup>
18	10	0.0193e <sup>-12</sup>
19	10	0.0219e <sup>-12</sup>
20	10	0.0188e <sup>-12</sup>



## CHAPTER 4

---

## APPROSSIMAZIONE DI FUNZIONI

### 4.1 Esercizio 14

**Descrizione:** Scrivere un programma che implementi efficientemente il calcolo del polinomio interpolante su un insieme di ascisse distinte.

*Soluzione:*

#### metodo interpolante di Newton

```
1 function y = NewtonInterp(xi, fi, x)
2 %
3 %   y = newton(xi, fi, x) Calcolo il polinomio
4 %                           interpolante le coppie (xi, fi)
5 %                           nelle ascisse x
6 n=length(xi);
7 if length(fi)~= n, error('dati inconsistenti'); end
8 for i=1:n-1
9     for j=i+1:n
10         if xi(i)==xi(j), error('ascisse non distinte');
11         end
12     end
13 end
14 f=divdif(xi, fi);
15 y=f(n);
16 for i=n-1:-1:1
17     y=y.*(x-xi(i))+f(i);
18 end
19 return
```

```

1 function f=dividif(xi,fi)
2 %
3 % f=dividif(xi,fi)  funzione che calcola
4 %                  le differenze divise
5 %
6 n=length(xi);
7 f=fi;
8 for i=1:n-1
9     for j=n:-1:i+1
10        f(j)=(f(j)-f(j-1))/(xi(j)-xi(j-i));
11    end
12 end
13 return

```

## 4.2 Esercizio 15

**Descrizione:** Scrivere un programma che implementi efficientemente il calcolo del polinomio interpolante di Hermite su un insieme di ascisse distinte.

*Soluzione:*

### Hermite

```
1 function y = Hermite(xi, fi, dfi, x)
2 %
3 %   y = Hermite(xi, fi, dfi, x)
4 %                               Calcolo il polinomio
5 %                               interpolante le coppie (xi, fi)
6 %                               nelle ascisse x
7 n=length(xi);
8 if length(fi)~=n, error('dati inconsistenti'); end
9 for i=1:n-1
10     for j=i+1:n
11         if xi(i)==xi(j), error('ascisse non distinte');
12         end
13     end
14 end
15 fh=zeros(2*n);
16 xh=zeros(2*n);
17 for i=1:n
18     fh(2*i-1)=fi(i);
19     fh(2*i)=dfi(i);
20     xh(2*i-1)=xi(i);
21     xh(2*i)=xi(i);
22 end
23 fh=divdifH(fh, xh);
24 y=fh(2*n);
25 for i=2*n-1:-1:1
26     y=y.*(x-xh(i))+fh(i);
27 end
28 return
```

```

1 function f = dividifH(f,x)
2 %   f=dividifH(f,x)
3 %   calcola le differenze divise secondo il
4 %   polinomio interpolante di Hermite
5 n=length(x)-1;
6 for i=n:-2:3
7     f(i)=(f(i)-f(i-2))/(x(i)-x(i-1));
8 end
9 for j=2:n
10     for i=n+1:-1:j+1
11         f(i)=(f(i)-f(i-1))/(x(i)-x(i-j));
12     end
13 end
14 return

```

## 4.3 Esercizio 16

**Descrizione:** Scrivere un programma che implementi efficientemente il calcolo di una spline cubica naturale interpolante su una partizione assegnata.

*Soluzione:*

### Spline naturale

```
1 function s = naturalSpline(x,y,xp)
2 %   s = naturalSpline(x,y,xp)
3 %   funzione per valutare le ascisse tramite spline naturale
4 %   parametri
5 %   x-> ascisse di interpolazione
6 %   y-> valori della funzione in x
7 %   xp-> punti di interpolazione
8 n=length(x);
9 hi=x(2:n) - x(1:n -1);
10 divdif=(y(2:n)-y(1:n-1))./hi;
11 divdif=6*((divdif(2: end) - divdif(1: end -1))./(x(3: end )-x(1: end -2)));
12 sub=(hi(2: end -1))./(hi(1: end -2)+hi(2: end -1));
13 sup=(hi(2: end -1))./(hi(2: end -1)+hi(3: end ));
14 m=tridiag(sub,sup,divdif);
15 m = [0,m,0];
16 ri = zeros (1, n);
17 qi = zeros (1, n);
18 for i = 2:n
19     ri(i -1) = y(i -1) - (( hi(i -1) ^2) /6) * m(i -1);
20     qi(i -1) =(y(i)-y(i -1))/ hi(i -1) - (hi(i -1) /6) * (m(i) - m(i -1));
21 end
22 s=evalSpline(ri ,qi ,xp,x,m,hi);
23 return
```

```

1 function m = tridiag(sub,sup,divdif)
2 % m = tridiag(sub,sup,divdif)
3 % funzione che restituisce una matrice tridiagonale
4 n=length(divdif);
5 l=zeros(1,n);
6 v=zeros(n);
7 v(1)=2;
8 for i = 2:n
9     v(i)=2-l(i)*sup(i-1);
10    l(i)=sub(i-1)/v(i-1);
11 end
12 y=zeros(n);
13 y(1)=divdif(1);
14 for j= 2:n
15     y(j)=divdif(j)-l(j)*y(j-1);
16 end
17 m(n)=y(n)/v(n);
18 for k = n-1:-1:1
19     m(k)=(y(k)-sup(k)*m(k+1))/v(k);
20 end
21 return

1 function s=evalSpline(ri,qi,xp,xi,m,hi)
2 % s=evalSpline(ri,qi,xp,xi,m,hi)
3 n = length(xp);
4 s = zeros(n,1)';
5 for j = 1 : n
6     i = range(xi, xp(j));
7     s(j) = ( ((xp(j)-xi(i-1))^3*m(i) + ((xi(i)-xp(j))^3)*m(i-1))/ ...
8     (6*hi(i-1))) + qi(i-1)*(xp(j)-xi(i-1)) + ri(i-1);
9 end
10 return

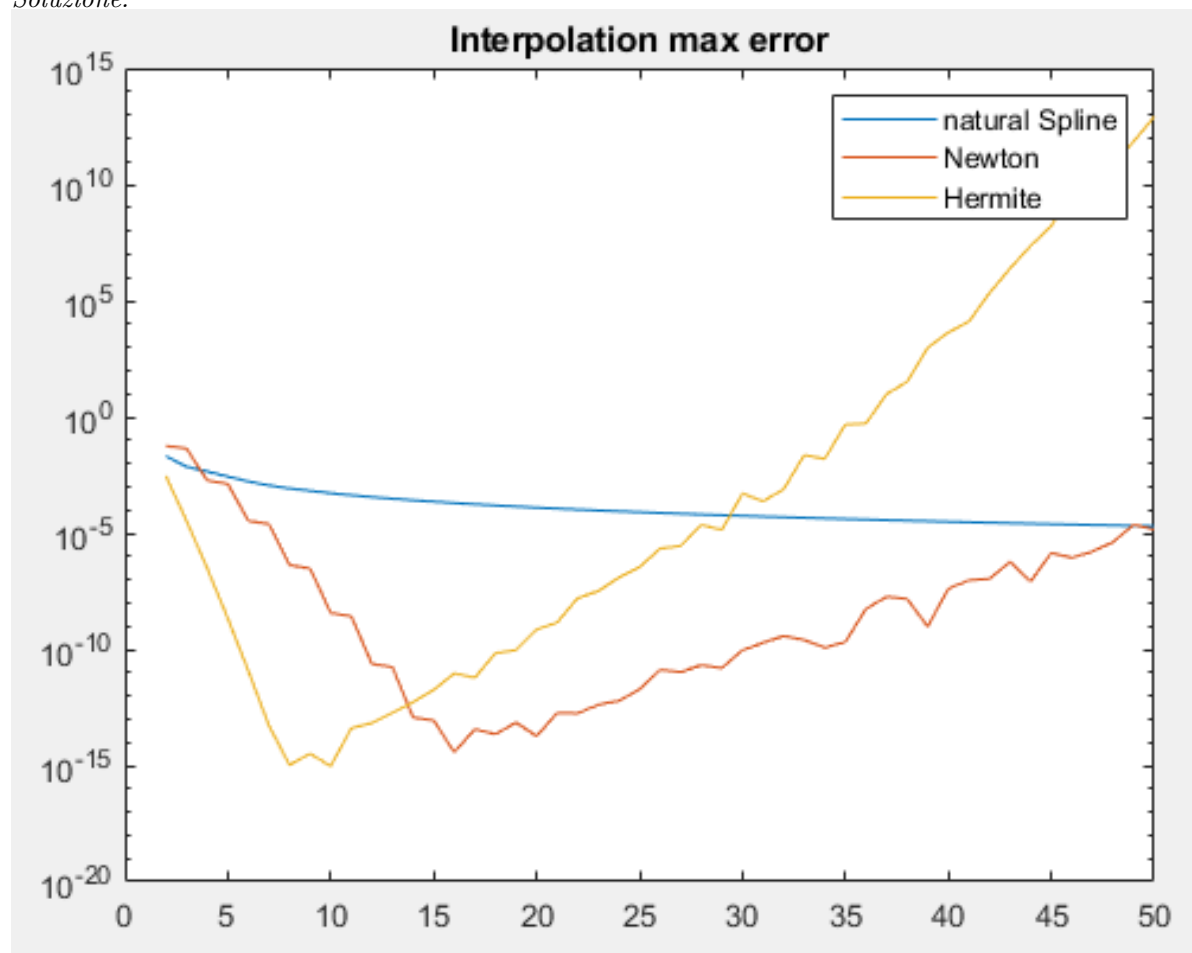
1 function i = range(xi, xp)
2 % i=range(xi, xp)
3 for i=2:length(xi)
4     if xi(i)>=xp
5         return;
6     end
7 end
8 return

```

## 4.4 Esercizio 18

**Descrizione:** Confrontare i codici degli esercizi 14–17 per approssimare la funzione  $f(x) = \sin(x)$  sulle ascisse  $x_i = i\pi/n, i = 0, 1, \dots, n$ , per  $n = 1, 2, \dots, 10$ . Graficare l'errore massimo di approssimazione verso  $n$  (in semilog), calcolato su una griglia uniforme di 10001 punti nell'intervallo  $[0, \pi]$ .

Soluzione:



# CHAPTER 5

## FORMULE DI QUADRATURA

### 5.1 Esercizio 22

**Descrizione:** Scrivere due functions che implementino efficientemente le formule adattative dei trapezi e di Simpson.

*Soluzione:*

#### formula dei trapezi adattiva

```
1 function I2 = trapad(fun,a,b,tol,fa,fb)
2 % I2 = trapad(fun,a,b,tol,fa,fb)
3 % parametri
4 % fun->funzione
5 % a,b->intervallo
6 % tol->tolleranza
7 % calcola un' approssimazione dell' integrale definito di fun(x)
8 % da a a b con tolleranza tol ( default 1e -5).
9 x1=(a+b)/2;
10 if nargin<=4
11     fa=feval(fun,a);
12     fb=feval(fun,b);
13     if nargin==3
14         tol=1e-5;
15     end
16 end
17 f1=feval(fun,x1);
18 h=(b-a)/2;
```



```

19 I1=h*( fa+fb );
20 I2=I1/2+h*f1 ;
21 err=abs(I2-I1)/3;
22 if err>tol
23     I2=trapad( fun ,a ,x1 , tol/2 ,fa , f1 )+...
24         +trapad( fun ,x1 ,b , tol/2 ,f1 ,fb );
25 end
26 return

```

## formula di Simpson adattiva

```
1 function I2 = simpad(fun,a,b,tol,fa,fb,f1)
2 % I2 = simpad(fun,a,b,tol,fa,fb)
3 % parametri
4 % fun->funzione
5 % a,b->intervallo
6 % tol->tolleranza
7 % calcola un' approssimazione dell' integrale definito di fun(x)
8 % da a a b con tolleranza tol ( default 1e -5).
9 x1=(a+b)/2;
10 if nargin<=4
11     fa=feval(fun,a);
12     fb=feval(fun,b);
13     f1=feval(fun,x1);
14     if nargin==3
15         tol=1e-5;
16     end
17 end
18 h=(b-a)/6;
19 I1=h*(fa+4*f1+fb);
20 x2=(a+x1)/2;
21 x3=(x1+b)/2;
22 f2=feval(fun,x2);
23 f3=feval(fun,x3);
24 I2 = 0.5*h*(fa+4*f2+2*f1+4*f3+fb);
25 err=abs(I2-I1)/15;
26 if err>tol
27     I2=simpad(fun,a,x1,tol/2,fa,f1,f2)+...
28         +simpad(fun,x1,b,tol/2,f1,fb,f3);
29 end
30 return
```

## 5.2 Esercizio 23

*Descrizione:* Sapendo che

$$I(x) = \int_0^{\arctan(30)} (1 + \tan^2(x)) dx = 30 \quad (5.1)$$

tabulare il numero dei punti richiesti dalle formule composite dei trapezi e di Simpson per approssimare  $I(f)$  con tolleranze  $tol = 10^{-i}$ , con  $i = 2, \dots, 8$

*Soluzione:*

per prima cosa é stato necessario effettuare una modifica al metodo dei trapezi adattivo e al metodo di Simpson adattivo per fare in modo che ritornassero il numero di punti

```
1 function [I2,punti] = trapadcount(fun,a,b,tol,fa,fb)
2 % I2 = trapadcount(fun,a,b,tol,fa,fb)
3 % parametri
4 % fun->funzione
5 % a,b->intervallo
6 % tol->tolleranza
7 % calcola un' approssimazione dell' integrale definito di fun(x)
8 % da a a b con tolleranza tol ( default 1e -5).
9 persistent count;
10 x1=(a+b)/2;
11 if nargin<=4
12     fa=feval(fun,a);
13     fb=feval(fun,b);
14     count=0;
15     if nargin==3
16         tol=1e-5;
17     end
18 end
19 count=count+1;
20 f1=feval(fun,x1);
21 h=(b-a)/2;
22 I1=h*(fa+fb);
23 I2=I1/2+h*f1;
24 err=abs(I2-I1)/3;
25 if err>tol
26     I2=trapadcount(fun,a,x1,tol/2,fa,f1)+...
27         +trapadcount(fun,x1,b,tol/2,f1,fb);
28 end
29 punti=count+2;
30 return
```

```

1 function [I2,punti] = simpadcount(fun,a,b,tol,fa,fb,f1)
2 % I2 = simpadcount(fun,a,b,tol,fa,fb)
3 % parametri
4 % fun->funzione
5 % a,b->intervallo
6 % tol->tolleranza
7 % calcola un' approssimazione dell' integrale definito di fun(x)
8 % da a a b con tolleranza tol ( default 1e -5).
9 persistent count;
10 x1=(a+b)/2;
11 if nargin<=4
12     fa=feval(fun,a);
13     fb=feval(fun,b);
14     f1=feval(fun,x1);
15     count=0;
16     if nargin==3
17         tol=1e-5;
18     end
19 end
20 count=count+3;
21 h=(b-a)/6;
22 I1=h*(fa+4*f1+fb);
23 x2=(a+x1)/2;
24 x3=(x1+b)/2;
25 f2=feval(fun,x2);
26 f3=feval(fun,x3);
27 I2 = 0.5*h*(fa+4*f2+2*f1+4*f3+fb);
28 err=abs(I2-I1)/15;
29 if err>tol
30     I2=simpadcount(fun,a,x1,tol/2,fa,f1,f2)+...
31         +simpadcount(fun,x1,b,tol/2,f1,fb,f3);
32 end
33 punti=count+2;
34 return

```

poi si é eseguito il seguente script per visualizzare i risultati

```
f = @(x) 1 + (tan(x)).^2;
a = 0;
b = atan(30);
for i=2:8
    [I2, counter] = simpadcount(f, a, b, 10^-i);
    disp(counter)
    err = abs(30 - I2);
    disp(err)
    [I2, counter] = trapadcount(f, a, b, 10^-i);
    disp(counter)
    err = abs(30 - I2);
    disp(err)
end
```

ottenendo così i risultati della seguente tabella

tolleranza	Simpson		trapezi	
	#punti	errore	#punti	errore
10 <sup>2</sup>	71	2.3725e-03	375	4.7645e-03
10 <sup>3</sup>	113	6.1653e-04	1181	5.7372e-04
10 <sup>4</sup>	203	5.0167e-05	3687	5.2639e-05
10 <sup>5</sup>	371	2.2207e-06	11883	5.4546e-06
10 <sup>6</sup>	635	3.9818e-07	37273	5.6355e-07
10 <sup>7</sup>	1145	3.5653e-08	116747	5.2552e-08
10 <sup>8</sup>	2045	3.5877e-09	375793	5.5077e-09

## CHAPTER 6

---

## CALCOLO DEL GOOGLE PAGERANK

### 6.1 Esercizio 24

*Descrizione:* Scrivere una function che implementi efficientemente il metodo delle potenze.

*Soluzione:*

#### metodo delle potenze

```
1 function [l1,x1] = potenze(A,tol,itmax)
2 % [f1,x1] = potenze(A,tol,itmax)
3 % parametri
4 % A-> matrice
5 % tol->tolleranza
6 % itmax-> numero massimo di iterazioni
7 % funzione che implementa il metodo delle potenze per calcolare
8 % l'autovalore dominante e il corrispettivo autovettore
9 [m,n]=size(A);
10 if m~=n,error('dati inconsistenti'),end
11 if nargin<=2
12     if nargin<=1
13         tol=1e-6;
14     else
15         if tol>= 0.1 || tol<=0, error('tolleranza non valida');end
16     end
17     itmax = ceil (- log10 ( tol))*n;
18 end
```

```

19 x = rand(n, 1);
20 l1=0;
21 for k = 1: itmax
22     x1=x/norm(x);
23     x=A*x1;
24     l0=l1;
25     l1=x'*x1;
26     err=abs(l1-l0);
27     if err<=tol*(1+abs(l1)), break, end
28 end
29 if err>tol*(1+abs(l1)), error('convergenza non ottenuta');end
30 return

```

## 6.2 Esercizio 25

**Descrizione:** Sia data la matrice di Toeplitz simmetrica  $A_N$  in cui le extra-diagonali più esterne sono le none. Partendo dal vettore  $\underline{u}_0 = (1, \dots, 1)^T \in \mathbb{R}^N$ , applicare il metodo delle potenze con tolleranza  $\text{tol} = 10^{-10}$  per  $N = 10 : 10 : 500$ , utilizzando la function dell'esercizio 24. Graficare il valore dell'autovalore dominante, e del numero di iterazioni necessarie per soddisfare il criterio di arresto, rispetto ad  $N$ . Utilizzare la funzione `spdiags` di Matlab per creare la matrice e memorizzarla come matrice sparsa.

*Soluzione:*

### metodo delle potenze

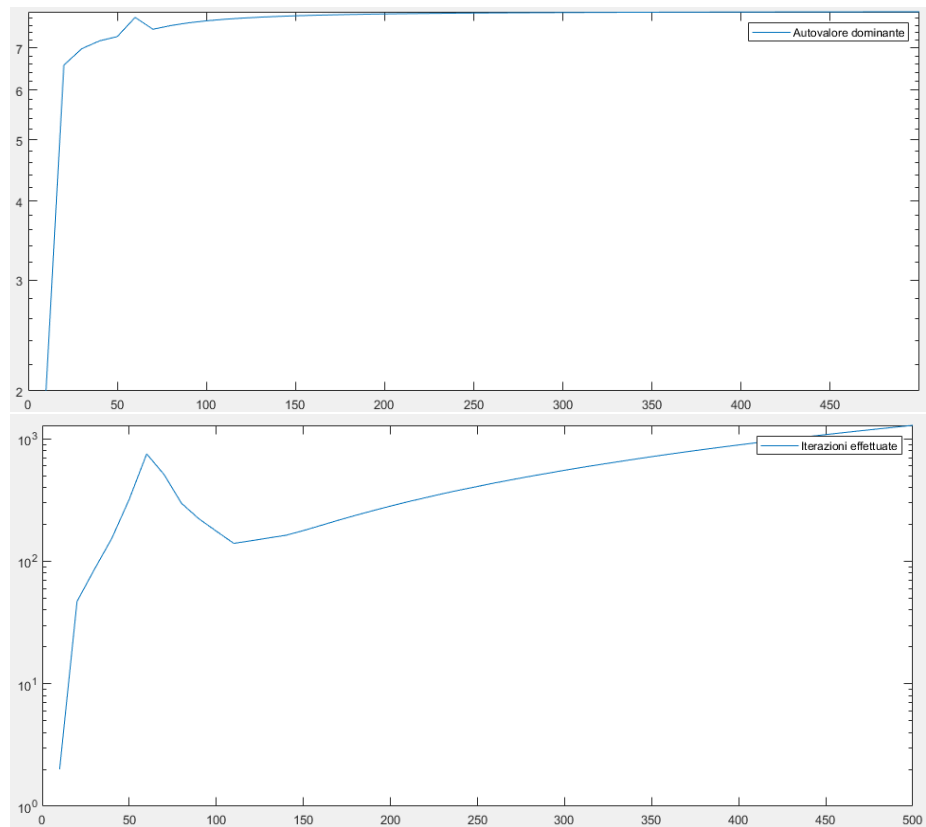
```
1 function [l1,q,k] = potenze(A,tol,itmax)
2 % [f1,q,k] = potenze(A,tol,itmax,u)
3 % parametri
4 % A-> matrice
5 % tol->tolleranza
6 % itmax-> numero massimo di iterazioni
7 % funzione che implementa il metodo delle potenze per calcolare
8 % l'autovalore dominante e il corrispettivo autovettore avendo
9 % come vettore di partenza u=(1,...,1)T,
10 [m,n]=size(A);
11 if m~=n,error('dati inconsistenti'),end
12 if nargin<=2
13     if nargin<=1
14         tol=1e-6;
15     else
16         if tol>= 0.1 || tol<=0, error('tolleranza non valida');end
17     end
18     itmax = ceil (- log10 ( tol))*n*n;
19 end
20 u = ones(n, 1);
21 l1=0;
22 for k = 1: itmax
23     q=u/norm(u);
24     u=A*q;
25     l0=(q'*u)/(q'*q);
26     err=abs(l0-l1);
27     l1=l0;
28     if err<=tol*(abs(l1)),break,end
29 end
30 if err>tol*(abs(l1)),error('convergenza non ottenuta');end
31 return
```



```

1  tol=10e-10;
2  j=1;
3  res = zeros (1,50) ;
4  iter = zeros (1,50) ;
5  for N = 10:10:500
6      B = -1* ones (N ,5);
7      B (: ,3) = B (: ,3) *( -4);
8      A = spdiags (B , [-9 -1 0 1 9] , N, N);
9      [l1 , x1 , k] = potenzecount (A, tol);
10     res (1,j) = l1 ;
11     iter (1,j) = k;
12     j = j+1;
13 end
14 figure
15 semilogy (10:10:500 , res (:));
16 legend('Autovalore dominante');
17 figure
18 semilogy (10:10:500 , iter (:));
19 legend('Iterazioni effettuate');

```



## 6.3 Esercizio 26

**Descrizione:** Scrivere una function che implementi efficientemente un metodo iterativo, per risolvere un sistema lineare, definito da un generico splitting della matrice dei coefficienti.

*Soluzione:*

### Splitting generico

```
1 function x = splitting(b,A,Msolve,tol)
2 % x= splitting(b,A,Msolve,tol)
3 % parametri
4 % b -> vettore colonna dei termini noti
5 % A -> Matrice passata
6 % Msolve -> funzione di Jacobi o di GaussSaider
7 % tol -> tolleranza
8 % la funzione calcola il sistema lineare Ax=b utilizzando un generico
9 % splitting che viene passato tramite il parametro Msolve
10
11 [n,m]=size(A);
12 if(n~=m || n~=length(b)),error('dati inconsistenti'),end
13 itmax=ceil(-log10(tol))*m;
14 x=zeros(n,1);
15 tolb=tol*norm(b,inf);
16 for i=1:itmax
17     r=A*x-b;
18     nr=norm(r,inf);
19     if nr<=tolb,break,end
20     u=Msolve(r,A);
21     x=x-u;
22 end
23 if nr>tolb,error('tolleranza non raggiunta'),end
24 return
```

## 6.4 Esercizio 27

**Descrizione:** Scrivere le function ausiliarie, per la function del precedente esercizio, che implementano i metodi iterativi di Jacobi e Gauss-Seidel.

Soluzione:

### il metodo di Jacobi

```
1 function y = Jacobi(x,A)
2 % y = Jacobi(x,A)
3 % parametri
4 % x -> colonna vettore dei termini noti
5 % A -> matrice nxn
6 % funzione per la risoluzione tramite Jacobi del sistema
7 D=diag(A);
8 y=x./D;
9 return
```

### il metodo di GaussSeidel

```
1 function y = GaussSeidel(x,A)
2 % y = GaussSeidel(x,A)
3 % parametri
4 % x -> colonna vettore dei termini noti
5 % A -> matrice nxn
6 % funzione per la risoluzione tramite GaussSeidel del sistema
7 n=length(x);
8 y=x;
9 for i=1:n
10     y(i)=y(i)/A(i,i);
11     y(i+1:n)=y(i+1:n)-y(i)*A(i+1:n,i);
12 end
13 return
```

## 6.5 Esercizio 28

**Descrizione:** Con riferimento alla matrice  $A_N$  definita in (1), risolvere il sistema lineare  $A_N x = (1, \dots, 1)^T \in \mathbb{R}^N$  con i metodi di Jacobi e Gauss-Seidel, per  $N = 10 : 10 : 500$ , partendo dalla approssimazione nulla della soluzione, ed imponendo la norma del residuo sia minore di  $10^{-8}$ . Utilizzare, a tale fine, la function dell'esercizio 26, scrivendo function ausiliare adhoc (vedi esercizio 27) che sfruttino convenientemente la struttura di sparsità (nota) della matrice  $A_N$ . Graficare il numero delle iterazioni richieste dai due metodi iterativi, rispetto ad  $N$ , per soddisfare il criterio di arresto prefissato. Soluzione:

```

1 function [x, i]=splitting(b, matvec, msolve, tol)
2 % [m,i]=splitting(b, matvec, msolve, tol)
3 % parametri in ingresso
4 % b -> vettore dei termini noti
5 % matvec -> funzione per creare la matrice ad-hoc per esercizio 25
6 % msolve -> funzione per risolvere il sistema lineare
7 % tol -> tolleranza desiderata
8 % Funzione per la risoluzione ad-hoc di un sistema lineare con splitting di
9 % Jacobi o Gauss-Seidel
10 n = length(b);
11 itmax = ceil(-log10(tol))*(n*n);
12 x=zeros(n,1);
13 tolb=tol*norm(b,inf);
14 for i=1:itmax
15     r=matvec(x)-b;
16     nr=norm(r,inf);
17     if nr <= tolb
18         break;
19     end
20     u=msolve(r);
21     x=x-u;
22 end
23 if nr>tolb
24     fprintf('Convergenza non raggiunta'); end
25 return

1 function y = matvec (x)
2 % y = matvec (x)
3 % parametri in ingresso
4 % x -> vettore da moltiplicare con la matrice A dell' es 25
5 % Metodo che ritorna il prodotto tra la
6 % matrice A dell' es 25 ed il vettore x.
7 y = x *4;
8 y(9 : end) = y(9 : end) - x(1 : end - 8);
9 y(2 : end) = y(2 : end) - x(1 : end - 1);

```

```

10 y(1 : end - 1) = y(1 : end - 1) - x(2 : end);
11 y(1 : end - 8) = y(1 : end - 8) - x(9 : end);
12 return

1 function y=jacobi(x)
2 % y=jacobi(x)
3 % parametri in ingresso
4 % x -> vettore dei tettrmini noti
5 % Risolve un sistema diagonale per la matrice dell'esercizio 25
6 y=x./4;
7 return

1 function y=gaussSeidel(x)
2 % y=gaussSeidel(x)
3 % parametri in ingresso
4 % x -> vettore dei termini noti
5 % Risolve un sistema triangolare inferiore
6 % per la matrice dell'esercizio 25
7 n=length(x);
8 y=x;
9 y(1)=y(1)/4;
10 for i=2:n
11     y(i)=(y(i)+y(i-1))/4;
12 end
13 for i=9:n
14     y(i)=(y(i)+y(i-1))/4;
15 end
16 return

1 % -----
2 % Generated by MATLAB on 20-Jul-2019 19:00:48
3 % MATLAB version: 9.5.0.944444 (R2018b)
4 % -----
5
6
7 b = [1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
8      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
9      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
10     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
11     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
12     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
13     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
14     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
15     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
16     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
17     1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...

```

```

18      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
19      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
20      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
21      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
22      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
23      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
24      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
25      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
26      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
27      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
28      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
29      1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; 1; ...
30
31      i = 67217;
32
33      iter = ...
34      [10 36 83;
35       20 88 178;
36       30 159 341;
37       40 254 555;
38       50 373 828;
39       60 517 1153;
40       70 684 1531;
41       80 873 1958;
42       90 1087 2440;
43       100 1323 2974;
44       110 1583 3559;
45       120 1866 4197;
46       130 2172 4887;
47       140 2502 5630;
48       150 2855 6425;
49       160 3231 7272;
50       170 3631 8171;
51       180 4054 9123;
52       190 4500 10127;
53       200 4969 11183;
54       210 5462 12292;
55       220 5977 13453;
56       230 6517 14666;
57       240 7079 15932;
58       250 7665 17250;
59       260 8274 18620;
60       270 8906 20043;
61       280 9562 21518;
62       290 10240 23046;
63       300 10942 24626;

```

```

64     310 11668 26258;
65     320 12416 27942;
66     330 13188 29679;
67     340 13983 31468;
68     350 14802 33310;
69     360 15644 35204;
70     370 16508 37150;
71     380 17397 39149;
72     390 18308 41200;
73     400 19243 43303;
74     410 20201 45459;
75     420 21182 47667;
76     430 22187 49928;
77     440 23215 52240;
78     450 24266 54606;
79     460 25341 57023;
80     470 26438 59493;
81     480 27559 62015;
82     490 28703 64590;
83     500 29871 67217];
84
85     n = 500;
86
87     tol = 1E-8;
88
89     x = [17.646381149776253; 24.283338324109526; 27.375685724311293; 29.206722101672
90           30.607881967658329; 32.016943926239023; 33.90861273057974; 37.3080513110213
91           45.302186275120739; 51.111286422492071; 55.01268247161574; 57.8433207148790
92           60.207861842889734; 62.551281006893696; 65.30945568524578; 69.0214062385938
93           74.143026218167748; 78.8469386200157; 82.7204370257197; 85.94601644431647;
94           88.828963683143; 91.670862574257043; 94.756522766025768; 98.325091741124965
95           102.40157374022733; 106.41300481504193; 110.07611056834662; 113.39134435498
96           116.49111387262231; 119.54668284252995; 122.72068106509737; 126.12086422134
97           129.72517218834153; 133.32739633341336; 136.77965607953095; 140.05213653662
98           143.19746461183198; 146.30407386020369; 149.45865443261013; 152.71251189299
99           156.05085446063271; 159.39175225305425; 162.66298088211522; 165.84008110257
100          168.94253418036809; 172.01349355639059; 175.09735091475355; 178.21967446005
101          181.37398151087302; 184.52577733884868; 187.64043409614911; 190.70267281412
102          193.71909745363936; 196.71001527326405; 199.6975812130444; 202.694853524744
103          205.69961978716151; 208.6969414985837; 211.6703053528351; 214.6110786075009
104          217.52116755024363; 220.40988887347939; 223.28810514297129; 226.16253864232
105          229.0327026181173; 231.89206351921982; 234.73276721289469; 237.550168716648
106          240.34460527025692; 243.1202675313975; 245.88241184704773; 248.634493287557
107          251.37658852788809; 254.10584275146979; 256.81853126711809; 259.51222378024
108          262.18681728709277; 264.84416413921616; 267.48678143073079; 270.11643413748
109          272.7333154589848; 275.33618769628538; 277.92329132855468; 280.493377854842

```

110 283.04627596345409; 285.58279031249418; 288.10411560407908; 290.61114637762  
111 293.10405147929356; 295.58230125119843; 298.0450685010922; 300.491720352295  
112 302.92211840461238; 305.33660554850638; 307.73574430079691; 310.11998429502  
113 312.48944283480108; 314.84389733361087; 317.18296107785989; 319.50631665422  
114 321.81387175983622; 324.10576918181607; 326.38227176131744; 328.64360367269  
115 330.88983823711635; 333.1208841764805; 335.33656182845539; 337.536713432915  
116 339.7212828047347; 341.89032766369928; 344.0439698961099; 346.1823204034939  
117 348.30542227073437; 350.41323931302958; 352.50568863290425; 354.58269245062  
118 356.644218368922; 358.69028877861768; 360.72095976245714; 362.7362857810156  
119 364.73629113591937; 366.72096217866738; 368.69026094621796; 370.64414937453  
120 372.58260944851304; 374.50564932624337; 376.41329460098154; 378.30557182913  
121 380.18249432024572; 382.04405732653237; 383.89024360584108; 385.72103465988  
122 387.536420731515; 389.33640448406533; 391.12099749333976; 392.8902126215791  
123 394.64405699673006; 396.3825292087497; 398.10562149814012; 399.813324935127  
124 401.50563434109063; 403.18255039270269; 404.84407827431295; 406.49022417473  
125 408.1209918440062; 409.73638102129644; 411.33638825058244; 412.921009249166  
126 414.49024131283511; 416.044084479198; 417.58254104436963; 419.1056139669702  
127 420.61330519123436; 422.10561478985454; 423.5825412417708; 425.044082506203  
128 426.4902371900024; 427.92100517503803; 429.33638746518653; 430.736385465767  
129 432.12100017236611; 433.490231713411; 434.84407942877209; 436.1825423522358  
130 437.50561977433011; 438.81331157419572; 440.10561818403471; 441.38254026704  
131 442.64407832758076; 443.89023247121247; 445.121002416264; 446.3363877082629  
132 447.53638798954245; 448.7210031720669; 449.89023343843019; 451.044079099539  
133 452.18254040847808; 453.30561743640123; 454.41331006564468; 455.50561808387  
134 456.58254131240341; 457.64407969502105; 458.69023330702885; 459.72100229318  
135 460.73638677939471; 461.73638680930071; 462.72100233509644; 463.69023325827  
136 464.64407949028475; 465.58254099771983; 466.50561781063948; 467.41330999595  
137 468.30561761582442; 469.1825406955432; 470.0440792164261; 470.8902331331069  
138 471.72100240204765; 472.53638700425927; 473.33638695119737; 474.12100227355  
139 474.890233001791; 475.64407915003329; 476.38254071139045; 477.1056176651360  
140 477.81330999001261; 478.50561767556474; 479.18254072584438; 479.84407915481  
141 480.4902329772973; 481.12100220097631; 481.73638682355181; 482.336386835637  
142 482.92100222692193; 483.49023299177946; 484.04407913145218; 484.58254065223  
143 485.10561756129033; 485.61330986272259; 486.10561755591789; 486.58254063666  
144 487.04407910000151; 487.49023294293607; 487.92100216572123; 488.33638677116  
145 488.73638676270332; 489.121002142514; 489.49023291054868; 489.8440790649484  
146 490.18254060332237; 490.50561752409533; 490.81330982719936; 491.10561751385  
147 491.38254058573136; 491.6440790439737; 491.89023288871579; 492.121002119163  
148 492.33638673416181; 492.53638673284894; 492.72100211505534; 492.89023288128  
149 493.04407903233619; 493.18254056888594; 493.30561749113514; 493.41330979879  
150 493.50561749128047; 493.58254056805623; 493.64407902886956; 493.69023287385  
151 493.72100210343126; 493.73638671808629; 493.73638671813666; 493.72100210358  
152 493.69023287410619; 493.64407902922193; 493.58254056850933; 493.50561749183  
153 493.41330979944644; 493.30561749188973; 493.18254056974087; 493.04407903329  
154 492.89023288233608; 492.72100211621063; 492.5363867341041; 492.336386735516  
155 492.12100212061819; 491.89023289026971; 491.64407904562682; 491.38254058748



156 491.10561751570913; 490.81330982914886; 490.50561752614323; 490.18254060546  
157 489.84407906719235; 489.49023291289029; 489.12100214495285; 488.73638676523  
158 488.33638677379349; 487.92100216845029; 487.49023294576131; 487.04407910292  
159 486.58254063968491; 486.1056175590295; 485.61330986592895; 485.105617564591  
160 484.58254065562875; 484.04407913494043; 483.49023299536083; 482.92100223059  
161 482.33638683940364; 481.73638682741006; 481.12100220492596; 480.49023298133  
162 479.8440791589473; 479.18254073006534; 478.50561767987512; 477.813309994411  
163 477.10561766962383; 476.38254071596612; 475.64407915469633; 474.89023300654  
164 474.12100227839431; 473.336386956119; 472.53638700926592; 471.7210024071387  
165 470.89023313828181; 470.04407922168429; 469.18254070088392; 468.30561762124  
166 467.41331000146334; 466.50561781622417; 465.58254100338445; 464.64407949602  
167 463.69023326409308; 462.72100234099673; 461.73638681527814; 460.73638678544  
168 459.72100229931419; 458.69023331323342; 457.6440797012998; 456.582541318755  
169 455.50561809029932; 454.41331007214131; 453.30561744296892; 452.18254041511  
170 451.04407910624735; 449.89023344520632; 448.72100317891073; 447.53638799645  
171 446.33638771523977; 445.12100242330609; 443.89023247831892; 442.64407833475  
172 441.38254027427638; 440.10561819132892; 438.81331158155075; 437.50561978174  
173 436.18254235970966; 434.8440794363039; 433.49023172099976; 432.121000180011  
174 430.73638547346752; 429.33638747294071; 427.92100518284536; 426.49023719786  
175 425.04408251411388; 423.58254124973172; 422.10561479786452; 420.61330519929  
176 419.10561397507547; 417.582541052521; 416.0440844873944; 414.49024132107536  
177 412.9210092574499; 411.3363882589075; 409.73638102966214; 408.1209918524116  
178 406.490224183178; 404.84407828279427; 403.18255040122034; 401.5056343496434  
179 399.813324943714; 398.10562150676009; 396.38252921740144; 394.6440570054124  
180 392.890212630291; 391.12099750207983; 389.33640449283257; 387.5364207403083  
181 385.72103466870777; 383.89024361468279; 382.04405733539664; 380.18249432913  
182 378.30557183803842; 376.41329460990562; 374.50564933518496; 372.58260945747  
183 370.64414938350592; 368.69026095520479; 366.72096218766683; 364.73629114493  
184 362.73628579003645; 360.72095977148672; 358.69028878765477; 356.64421837796  
185 354.58269245967682; 352.50568864195628; 350.41323932208394; 348.30542227978  
186 346.18232041254913; 344.04396990516346; 341.89032767275; 339.72128281378127  
187 337.53671344195618; 335.33656183748974; 333.12088418550667; 330.88983824613  
188 328.64360368169679; 326.38227177031087; 324.10576919079585; 321.81387176880  
189 319.50631666317651; 317.18296108679118; 314.84389734252306; 312.48944284369  
190 310.11998430389968; 307.73574430964351; 305.33660555732843; 302.92211841340  
191 300.491720361065; 298.04506850983273; 295.58230125990883; 293.1040514879722  
192 290.61114638627146; 288.10411561269018; 285.58279032106952; 283.04627597199  
193 280.49337786334297; 277.923291337015; 275.33618770470434; 272.7333154673607  
194 270.11643414581619; 267.486781439016; 264.84416414745414; 262.1868172952824  
195 259.51222378838042; 256.81853127520742; 254.10584275950649; 251.37658853587  
196 248.6344932954826; 245.88241185491523; 243.12026753920617; 240.344605278006  
197 237.55016872433723; 234.73276722052151; 231.89206352678252; 229.03270262561  
198 226.16253864975653; 223.28810515032833; 220.4098888076583; 217.52116755745  
199 214.61107861464566; 211.6703053599073; 208.69694150558055; 205.699619794079  
200 202.69485353158012; 199.69758121979734; 196.7100152799344; 193.719097460228  
201 190.70267282062977; 187.64043410257491; 184.5257773451882; 181.373981517120

```

202 178.21967446620621; 175.09735092080797; 172.01349356235016; 168.94253418623
203 165.84008110835873; 162.662980887804; 159.39175225864656; 156.0508544661194
204 152.71251189836684; 149.45865443787054; 146.30407386535666; 143.19746461688
205 140.05213654158356; 136.77965608439436; 133.32739633817172; 129.72517219297
206 126.120864225845; 122.72068106946588; 119.54668284677876; 116.4911138767665
207 113.39134435903917; 110.07611057230066; 106.41300481888547; 102.40157374393
208 98.325091744656945; 94.7565227693993; 91.670862577501424; 88.82896368628462
209 85.946016447371051; 82.720437028687414; 78.846938622874347; 74.143026220864
210 69.021406241044616; 65.309455687509242; 62.551281009030234; 60.207861844938
211 57.8433207168606; 55.012682473530994; 51.111286424314009; 45.30218627677344
212 37.308051312224677; 33.90861273159512; 32.016943927165855; 30.6078819685360
213 29.206722102516835; 27.375685725123112; 24.283338324860996; 17.646381150357
214 ];

```

