

Elaborato Calcolo Numerico 2017-2018

Bertini Gabriele - matricola 5793664
Pratesi Mariti Lorenzo - matricola

11 agosto 2018

Indice

1	Errori ed aritmetica finita	2
2	Radici di una equazione	3
3	Sistemi lineari e non lineari	4
4	Approssimazione di funzioni	5
4.1	Esercizio 1	5
4.2	Esercizio 2	5
4.3	Esercizio 3	6
4.4	Esercizio 4	8
4.5	Esercizio 5	9
4.6	Esercizio 6	13
4.7	Esercizio 7	14
4.8	Esercizio 8	14
4.9	Esercizio 9	14
4.10	Esercizio 10	14
5	Formule di quadratura	15
5.1	Esercizio 1	15
5.2	Esercizio 2	15
5.3	Esercizio 3	16
5.4	Esercizio 4	16
5.5	Esercizio 5	17
6	Calcolo del Google pagerank. Risoluzione iterativa di sistemi lineari	18
6.1	Esercizio 1	18
6.2	Esercizio 2	18
6.3	Esercizio 3	19
6.4	Esercizio 4	20
6.5	Esercizio 5	22

Capitolo 1

Errori ed aritmetica finita

Capitolo 2

Radici di una equazione

Capitolo 3

Sistemi lineari e non lineari

Capitolo 4

Approssimazione di funzioni

4.1 Esercizio 1

Prova

```
1 function y = lagrange(xi, fi, x)
2 % y = lagrange(xi,fi,x)
3 % Implementa il calcolo del polinomio interpolante
4 % di grado n in forma di Lagrange
5 % Input:
6 %   xi: vettore delle ascisse di interpolazione
7 %   fi: vettore dei valori della funzione su x
8 %   x: vettore dei punti in cui valutare il polinomio
9 %
10 % Output:
11 %   y: vettore dei valori del polinomio valutato sui punti x
12 if isempty(x), error("Vettore x vuoto!"), end
13 if length(xi)~=length(fi), error("Diversa lunghezza dei vettori xi e fi!"), end
14 n=length(xi)-1;
15 m=length(x);
16 y=zeros(size(x));
17 for i=1:m
18     for j=1:n+1
19         p=1;
20         for k=1:n+1
21             if j~=k
22                 p=p*(x(i)-xi(k))/(xi(j)-xi(k));
23             end
24         end
25         y(i)=y(i)+fi(j)*p;
26     end
27 end
28 return
29 end
```

4.2 Esercizio 2

Prova

```

1 function y = newton(xi,fi,x)
2 % y = newton(xi,fi,x)
3 % Implementa il calcolo del polinomio interpolante
4 % di grado n in forma di Newton
5 % Input:
6 % xi: vettore delle ascisse di interpolazione
7 % fi: vettore dei valori della funzione su x
8 % x: vettore dei punti in cui valutare il polinomio
9 %
10 % Output:
11 % y: vettore dei valori del polinomio valutato sui punti x.
12 if isempty(x), error('Vettore x vuoto!'), end
13 if isempty(xi), error('Vettore xi vuoto!'), end
14 if length(fi)~=length(xi), error('Vettori fi ed xi non compatibili!'), end
15 dd = diffdivN(xi,fi);
16 y = hornerGen(xi,dd,x);
17 return
18 end
19
20 function fi = diffdivN(xi,fi)
21 % f = diffdivN(xi,fi)
22 % Calcolo delle differenze divise per il polinomio di Newton
23 % Questo metodo prende in input:
24 % xi: vettore delle ascisse
25 % fi: vettore dei valori della funzione
26 %
27 % E restituisce:
28 % fi: vettore contenente le differenze divise f[x0],f[x1,x2]...f[x0...xn]
29 % Assumiamo che le ascisse siano tutte distinte.
30 n = length(xi)-1;
31 for j = 1:n
32     for i = n+1:-1:j+1
33         fi(i) = (fi(i)-fi(i-1))/(xi(i)-xi(i-j));
34     end
35 end
36 return
37 end
38
39 function p = hornerGen(xi,fi,x)
40 % p = hornerGen(xi,fi,x)
41 % Algoritmo di horner generalizzato
42 % Questo metodo prende in input:
43 % fi: array dei coefficienti
44 % xi: array degli zeri del polinomio
45 % x: array di punti di valutazione del polinomio
46 % E restituisce la valutazione del polinomio
47 % pr-1(x)+fwr(x) nei punti x0
48 n = length(xi)-1;
49 p = fi(n+1)*ones(size(x));
50 for k = 1:length(x)
51     for i = n:-1:1
52         p(k) = p(k)*(x(k)-xi(i))+fi(i);
53     end
54 end
55 return
56 end

```

4.3 Esercizio 3

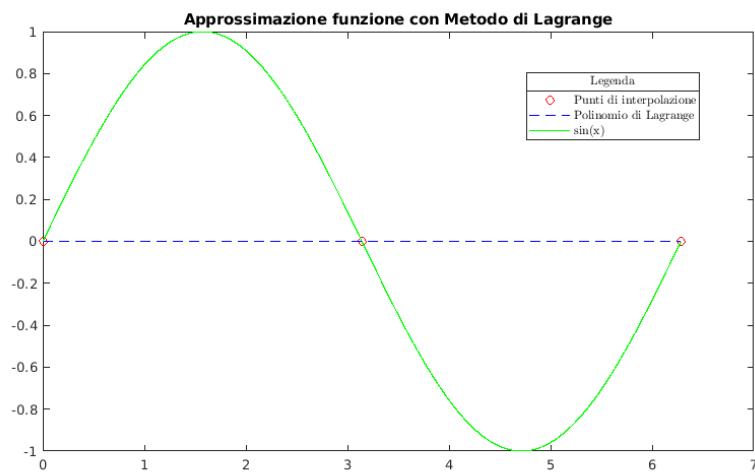
Prova

```

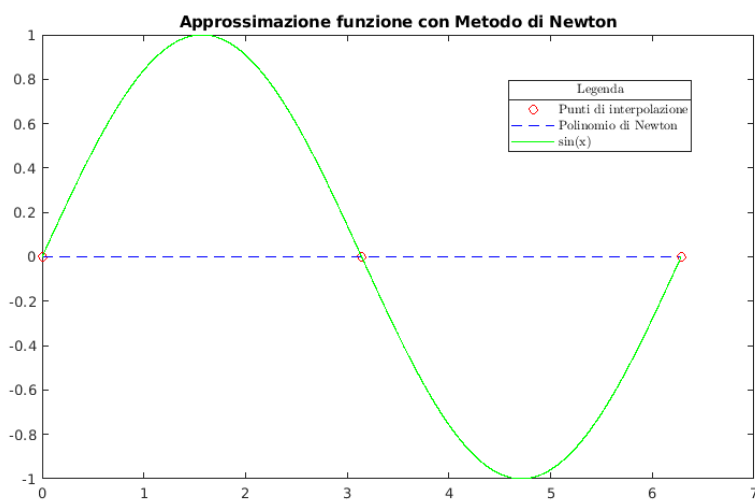
1 function y = hermite(xi,fi,fil,x)
2 % y = hermite(xi,fi,fil,x)
3 % Calcola il polinomio interpolante di Hermite
4 % Input:
5 % xi: vettore delle ascisse di interpolazione
6 % fi: vettore dei valori della funzione su x
7 % fil: vettore della derivata su x
8 % x: vettore dei punti in cui valutare il polinomio
9 %
10 % Output:
11 % y: vettore dei valori del polinomio valutato sui punti x.
12 if isempty(x), error('Vettore x vuoto!'), end
13 if isempty(xi), error('Vettore xi vuoto!'), end
14 if length(xi)~=length(fi) || length(xi)~=length(fil)
15     error('Vettori xi, fi e fil di lunghezza diversa!')
16 end
17 n = length(xi)-1;
18 xi = reshape([xi; xi], [], 1)';
19 fi = reshape([fi; fil], [], 1)';
20 dd = diffdivH(xi,fi);
21 y = hornerGen(xi,dd,x);
22 return
23 end
24
25 function fi = diffdivH(xi,fi)
26 % fi = diffdivH(xi,fi)
27 % Calcola differenze divise per polinomio di Hermite
28 % Questo metodo prende in input:
29 % xi: vettore delle ascisse
30 % fi: vettore dei valori della funzione
31 %
32 % E restituisce:
33 % fi: vettore contenente le differenze divise f[x0],f[x0,x0],...,f[x0,...,xn]
34 n = length(xi)-1; %2*m+1
35 for i = n:-2:3
36     fi(i) = (fi(i)-fi(i-2))/(xi(i)-xi(i-1));
37 end
38 for j = 2:n
39     for i = n+1:-1:j+1
40         fi(i) = (fi(i)-fi(i-1))/(xi(i)-xi(i-j));
41     end
42 end
43 return
44 end
45
46 function p = hornerGen(xi,fi,x)
47 % p = hornerGen(xi,fi,x)
48 % Algoritmo di horner generalizzato
49 % Questo metodo prende in input:
50 % fi: array dei coefficienti
51 % xi: array degli zeri del polinomio
52 % x: array di punti di valutazione del polinomio
53 % E restituisce la valutazione del polinomio
54 % pr-1(x)+fwr(x) nei punti x0
55 n = length(xi)-1;
56 p = fi(n+1)*ones(size(x));
57 for k = 1:length(x)
58     for i = n:-1:1
59         p(k) = p(k)*(x(k)-xi(i))+fi(i);
60     end
61 end
62 return

```

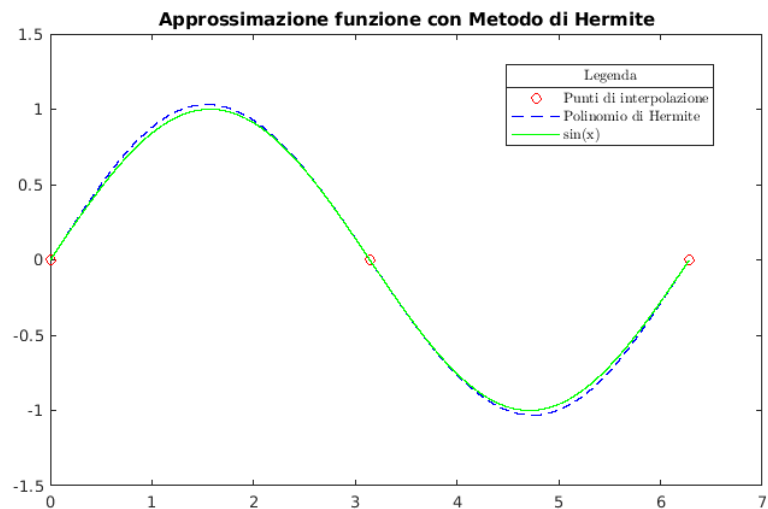

4.4 Esercizio 4



Prova



Prova



Prova

4.5 Esercizio 5

Prova

```

1 function y = spline3( xi, fi, x, tipo )
2 % y = spline3( xi, fi, x, tipo )
3 % Calcola e valuta i valori della spline cubica naturale o not-a-knot
4 % (rispettivamente per tipo uguale a 0 o 1) delle coppie di dati assegnati.
5 % Input:
6 %   xi: ascisse di interpolazione
7 %   fi: valori della funzione, valutati nelle ascisse xi
8 %   x: punti di valutazione della spline
9 %   tipo:
10 %       - nat identifica la spline naturale
11 %       - nak identifica la spline not-a-knot
12 %       - un valore diverso restituisce errore
13 % Output:
14 %   y: valutazione dei punti x calcolati sulla spline richiesta
15 if tipo == "nat"
16     mi = sistemaSplineNaturale(xi, fi);
17     y = valutaSplineNaturale(xi, fi, mi, x);
18 elseif tipo == "nak"
19     mi = sistemaSplineNotAKnot(xi, fi);
20     y = valutaSplineNotAKnot(xi, fi, mi, x);
21 else
22     error('Il tipo indicato non   corretto.');
```

```

23 end
24 return
25 end
26
27 function fin = differenzeDivise(x, f)
28 % Input:
29 % - f: funzione;
30 % - x: vettore delle ascisse.
31 % Output:
32 % - fin: l'ultima differenza divisa calcolata, corrispondente a quella che
33 %       include tutte le ascisse.
```

```

34     n = length(x);
35     for i = 1:n-1
36         for j = n:-1:i+1
37             f(j)=(f(j)-f(j-1))/(x(j)-x(j-i));
38         end
39     end
40     fin = f(end);
41 end
42
43 function m = sistemaSplineNaturale( xi , fi )
44 % Input:
45 %     xi: ascisse di interpolazione
46 %     fi: valori della funzione, valutati nelle ascisse xi
47 % Output:
48 %     m: coefficienti necessari a calcolare l'espressione della spline
49 %     cubica
50
51 % calcolo delle phi, zi e diff_div (lunghi n-1)
52 % n:= grado del polinomio interpolante
53 n = length(xi)-1;
54 phi = zeros(n-2, 1);
55 zi = zeros(n-2, 1);
56 dd = zeros(n-1, 1);
57 hi = xi(2) - xi(1);
58 h1l = xi(3) - xi(2);
59 phi(1) = hi/(hi+h1l);
60 zi(1) = h1l/(hi+h1l);
61 dd(1) = differenzeDivise(xi(1:3), fi(1:3));
62 for i = 2:n-2
63     hi = xi(i) - xi(i-1);
64     h1l = xi(i+1) - xi(i);
65     phi(i) = hi/(hi+h1l);
66     zi(i) = h1l/(hi+h1l);
67     dd(i) = differenzeDivise(xi(i-1:i+1), fi(i-1:i+1));
68 end
69 dd(i+1) = differenzeDivise(xi(i:i+2), fi(i:i+2));
70 % calcolo delle li ed ui che completano il sistema lineare della spline
71 % naturale
72 u(1) = 2;
73 for i = 2:n-1
74     l(i) = phi(i-1)/u(i-1);
75     u(i) = 2-l(i)*zi(i-1);
76 end
77 % risoluzione del sistema lineare LU per trovare gli m
78 % 1) L*y = 6*b
79 % il vettore dd delle diffDiv viene sovrascritto
80 dd(1) = 6*dd(1);
81 dd(2:n-1) = 6*dd(2:n-1)-l(2:n-1)*dd(1:n-2);
82 % 2) U*m = y
83 m(n-1) = dd(n-1)/u(n-1);
84 m(n-2:-1:1) = (dd(n-2:-1:1)-zi(n-2:-1:1) * m(n-1:-1:2))/u(n-2:-1:1);
85 end
86
87 function y = valutaSplineNaturale(xi, fi, mi, x)
88 % Input:
89 %     xi: ascisse di interpolazione
90 %     fi: valori della funzione, valutati nelle ascisse xi
91 %     mi: coefficienti necessari a calcolare l'espressione della spline
92 %     x: punti in cui valutare la spline
93 % Output:
94 %     y: valutazioni nei punti x della spline.
95

```

```

96     if xi(1)>x(1) || xi(end)<x(end)
97         error('I punti di valutazione non rientrano nel dominio della spline.');
```

98 end

99 if length(xi)~=length(fi)

100 error('I vettori xi e fi devono avere la stessa lunghezza.');

101 end

102 % raccolgo tutti i sottoinsiemi di punti da valutare con le relative

103 % functions, i punti x vengono ordinati

104 mi = [0 mi 0];

105 sort(x);

106 y = zeros(length(x),1);

107 lastIndex = 1;

108 k = 2;

109 for j = 1 : length(x)

110 if x(j) <= xi(k-1)

111 j = j + 1;

112 else

113 if j ~= lastIndex

114 % calcolo la spline relativa al k-1° intervallo

115 hi = xi(k) - xi(k-1);

116 ri = fi(k-1) - ((hi^2)/6) * mi(k-1);

117 qi = (fi(k) - fi(k-1))/hi - (hi/6) * ...

118 (mi(k) - mi(k-1));

119 spline = @(x) (((x - xi(k-1)).^3) .* mi(k) + ((xi(k) - x).^3) .* mi(k-1))./(6*hi) +

120 qi .* (x - xi(k-1)) + ri;

121 % calcolo le valutazioni della spline

122 y(lastIndex : j-1) = spline(x(lastIndex : j-1));

123 lastIndex = j;

124 end

125 k = k+1;

126 end

127 end

128 % valutazione degli ultimi punti

129 if j ~= lastIndex

130 hi = xi(end) - xi(end-1);

131 ri = fi(end-1) - ((hi^2)/6) * mi(end-1);

132 qi = (fi(end) - fi(end-1))/hi - (hi/6) * (-mi(end-1));

133 spline = @(x) (((x - xi(end-1)).^3) .* mi(end) + ((xi(end) - x).^3) .* mi(end-1))./(6*hi) +

134 qi .* (x - xi(end-1)) + ri;

135 y(lastIndex:j-1) = spline(x(lastIndex:j-1));

136 end

137

138 end

139

140 function m = sistemaSplineNotAKnot(xi, fi)

141 % Calcola i coefficienti m da applicare all'espressione della spline cubica

142 % not-a-knot.

143 % Input:

144 % xi: ascisse di interpolazione

145 % fi: valori della funzione, valutati nelle ascisse xi

146 % Output:

147 % m: coefficienti necessari a calcolare l'espressione della spline

148 % not-a-knot

149

150 % calcolo delle phi, zi e diff.div

151 % n:= grado del polinomio interpolante

152 n = length(xi)-1;

153 hi = xi(2) - xi(1);

154 hi1 = xi(3) - xi(2);

155 phi(1) = hi/(hi+hi1);

156 zi(1) = hi1/(hi+hi1);

157 diffDiv(1) = differenzeDivise(xi(1:3), fi(1:3));

```

158     for i = 2:n-1
159         hi = xi(i) - xi(i-1);
160         hi1 = xi(i+1) - xi(i);
161         phi(i) = hi/(hi+hi1);
162         zi(i) = hi1/(hi+hi1);
163         diffDiv(i) = differenzeDivise(xi(i-1:i+1), fi(i-1:i+1));
164     end
165     diffDiv(i+1) = differenzeDivise(xi(i:i+2), fi(i:i+2));
166     % espando diffDiv a vettore di lunghezza n+1
167     diffDiv = [6*diffDiv(1), 6*diffDiv, 6*diffDiv(end)];
168     % i = 1
169     u(1) = 1;
170     % i = 2
171     w(1) = 0;
172     l(1) = phi(1) / u(1);
173     u(2) = 2 - phi(1);
174     % i = 3
175     w(2) = zi(1) - phi(1);
176     l(2) = phi(2) / u(2);
177     u(3) = 2 - l(2)*w(2);
178     % i = 4:n-1
179     for i = 4:n-1
180         w(i-1) = zi(i-2);
181         l(i-1) = phi(i-1) / u(i-1);
182         u(i) = 2 - l(i-1)*w(i-1);
183     end
184     % i = n
185     w(n-1) = zi(n-2);
186     l(n) = (phi(n-1) - zi(n-1)) / u(n-1);
187     u(n) = 2 - zi(n-1) - l(n)*w(n-1);
188     % i = n+1
189     w(n) = zi(n-1);
190     l(n+1) = 0;
191     u(n+1) = 1;
192     % 1) Ly = 6*diffDiv
193     y(1) = diffDiv(1);
194     for i = 2:n+1
195         y(i) = diffDiv(i) - l(i-1)*y(i-1);
196     end
197     % 2) Um = y
198     m = zeros(n+1, 1);
199     m(n+1) = y(n+1) / u(n+1);
200     for i = n:-1:1
201         m(i) = (y(i) - w(i) * m(i+1)) / u(i);
202     end
203     m(1) = m(1) - m(2) - m(3);
204     m(end) = m(n+1) - m(n) - m(n-1);
205     return
206 end
207
208 function y = valutaSplineNotAKnot(xi, fi, mi, x)
209 % Valuta i punti della spline cubica not-a-knot nei punti assegnati x
210 % Input:
211 %     xi: ascisse di interpolazione
212 %     fi: valori della funzione, valutati nelle ascisse xi
213 %     mi: coefficienti necessari a calcolare l'espressione della spline
214 %     x: punti in cui valutare la spline
215 % Output:
216 %     y: valutazioni nei punti x della spline.
217
218 if xi(1)>x(1) || xi(end)<x(end)
219     error('I punti di valutazione non rientrano nel dominio della spline.');
```

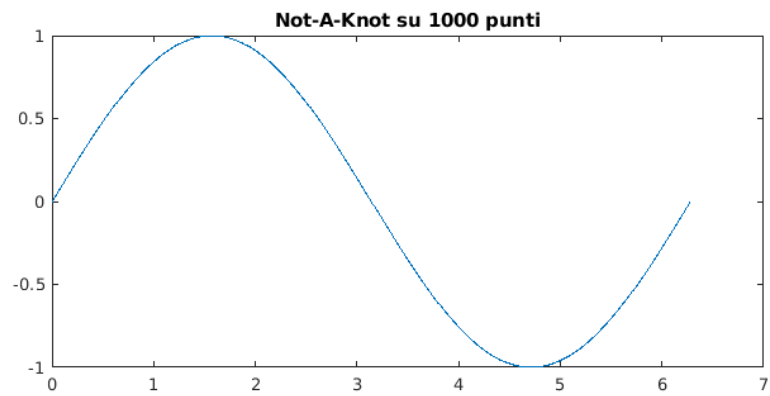
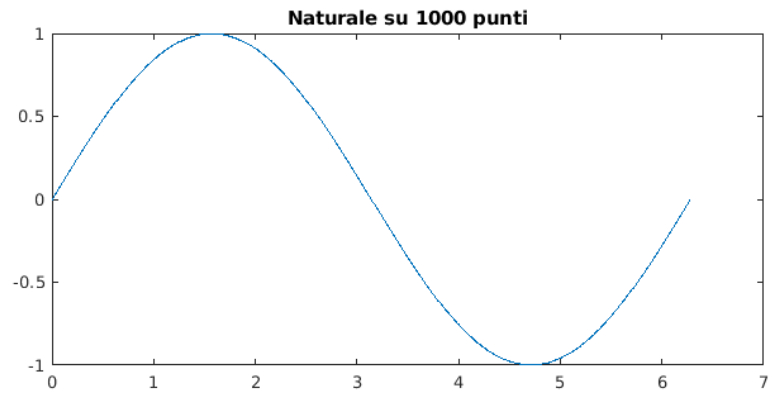
```

220     end
221     if length(xi)~=length(fi)
222         error('I vettori xi e fi devono avere la stessa lunghezza.');
```

```

223     end
224     % raccolgo tutti i sottoinsiemi di punti da valutare con le relative
225     % functions, i punti x vengono ordinati
226     sort(x);
227     y = zeros(length(x),1);
228     lastIndex = 1;
229     k = 2;
230     for j = 1 : length(x)
231         if x(j) <= xi(k-1)
232             j = j + 1;
233         else
234             if j ~= lastIndex
235                 % calcolo la spline relativa al k-1° intervallo
236                 hi = xi(k) - xi(k-1);
237                 ri = fi(k-1) - ((hi^2)/6) * mi(k-1);
238                 qi = (fi(k) - fi(k-1))/hi - (hi/6) * ...
239                     (mi(k) - mi(k-1));
240                 spline = @(x) (((x - xi(k-1)).^3) .* mi(k) + ((xi(k) - x).^3) .* mi(k-1))./(6*hi) .
241                     + qi .* (x - xi(k-1)) + ri;
242                 % calcolo le valutazioni della spline
243                 y(lastIndex : j-1) = spline(x(lastIndex : j-1));
244                 lastIndex = j;
245             end
246             k = k+1;
247         end
248     end
249     % valutazione degli ultimi punti
250     if j ~= lastIndex
251         hi = xi(end) - xi(end-1);
252         ri = fi(end-1) - ((hi^2)/6) * mi(end-1);
253         qi = (fi(end) - fi(end-1))/hi - (hi/6) * (-mi(end-1));
254         spline = @(x) (((x - xi(end-1)).^3) .* mi(end) + ((xi(end) - x).^3) .* mi(end-1))./(6*hi) .
255             + qi .* (x - xi(end-1)) + ri;
256         y(lastIndex:j-1) = spline(x(lastIndex:j-1));
257     end
258 end

```



Prova

4.6 Esercizio 6

Prova

```

1 function xi = ceby(n, a, b)
2 % xi = ceby(n, a, b)
3 % Implementa il calcolo delle ascisse di Chebyshev
4 % per il polinomio di grado n su [a,b]
5 % Input:
6 %   n: grado del polinomio interpolante
7 %   a: estremo sinistro dell'intervallo
8 %   b: estremo destro dell'intervallo
9 %
10 % Output:
11 %   xi: ascisse di Chebyshev
12     if n<=0
13         error('Inserire numero maggiore di 0!')
14     end
15     xi = cos((2*[0:n]+1)*pi/(2*n+2));
16     xi = ((a+b)+(b-a)*xi)/2;
```

```
17     return
18 end
```

4.7 Esercizio 7

Prova

4.8 Esercizio 8

Prova

4.9 Esercizio 9

Prova

4.10 Esercizio 10

Capitolo 5

Formule di quadratura

5.1 Esercizio 1

Prova

```
1 function If = trapcomp(n, a, b, fun)
2 % If = trapcomp(n, a, b, fun)
3 % Calcola l'integrale della funzione, nell'intervallo prescelto,
4 % usando la formula dei trapezi composta.
5 % Input:
6 %   n: intero positivo, indica num intervalli in [a,b]
7 %   a: estremo sinistro
8 %   b: estremo destro
9 %   fun: funzione integranda
10 % Output:
11 %   If: approssimazione dell'integrale definito della funzione
12     if n <= 0, error('Numero intervalli non valido. '), end
13     if a >= b, error('Intervallo di integrazione non valido. '), end
14     x = linspace(a, b, n+1);
15     f = feval(fun, x);
16     If = ((b-a)/n)*(f(1)/2 + sum(f(2:n)) + f(n+1)/2);
17 end
```

5.2 Esercizio 2

Prova

```
1 function If = simpcomp(n, a, b, fun)
2 % If = simpcomp(n, a, b, fun)
3 % Calcola l'integrale della funzione, nell'intervallo prescelto,
4 % usando la formula di Simpson composta.
5 % Input:
6 %   n: intero positivo pari, indica num intervalli in [a,b]
7 %   a: estremo sinistro
8 %   b: estremo destro
9 %   fun: funzione integranda
10 % Output:
11 %   If: approssimazione dell'integrale definito della funzione
12     if mod(n,2) ~= 0
```

```

13         error('Il numero di intervalli deve essere pari.')
14     end
15     if n <= 0
16         error('Numero di intervalli non valido.')
17     end
18     if a >= b, error('Intervallo di integrazione non valido. '), end
19     x = linspace(a, b, n+1);
20     f = feval(fun, x);
21     If = f(1) + f(n+1);
22     If = (If + 4*sum(f(2:2:n)) + 2*sum(f(3:2:n-1)))*(b-a)/(3*n);
23 end

```

5.3 Esercizio 3

Prova

```

1 function If = trapad(a, b, fun, tol, fa, fb)
2 % If = trapad(a, b, fun, tol)
3 % Calcola ricorsivamente l'integrale della funzione, nell'intervallo prescelto,
4 % usando la formula dei trapezi adattiva.
5 % Input:
6 %   a: estremo sinistro
7 %   b: estremo destro
8 %   fun: funzione integranda
9 %   tol: tolleranza
10 % Output:
11 %   If: approssimazione dell'integrale definito della funzione
12     if a >= b, error('Intervallo di integrazione non valido. '), end
13     if nargin <= 4
14         fa = feval(fun, a);
15         fb = feval(fun, b);
16     end
17     h = b - a;
18     x1 = (a+b)/2;
19     f1 = feval(fun, x1);
20     I1 = (h/2)*(fa + fb);
21     If = (I1 + h*f1)/2;
22     err = abs(If - I1)/3;
23     if err > tol
24         If = trapad(a, x1, fun, tol/2, fa, f1) ...
25             + trapad(x1, b, fun, tol/2, f1, fb);
26     end
27 end

```

5.4 Esercizio 4

Prova

```

1 function If = simpad(a, b, fun, tol, fa, f1, fb)
2 % If = simpad(a, b, fun, tol)
3 % Calcola ricorsivamente l'integrale della funzione, nell'intervallo prescelto,
4 % usando la formula di Simpson adattiva.
5 % Input:
6 %   a: estremo sinistro
7 %   b: estremo destro

```

```

8  % fun: funzione integranda
9  % tol: tolleranza
10 % Output:
11 % If: approssimazione dell'integrale definito della funzione
12 if a >= b, error('Intervallo di integrazione non valido. '), end
13 x1 = (a+b)/2;
14 if nargin <= 4
15     fa = feval(fun, a);
16     fb = feval(fun, b);
17     f1 = feval(fun, x1);
18 end
19 h = (b - a)/6;
20 I1 = h*(fa + 4*f1 + fb);
21 f2 = feval(fun, (a+x1)/2);
22 f3 = feval(fun, (x1+b)/2);
23 If = .5*h*(fa + 4*f2 + 2*f1 + 4*f3 + fb);
24 err = abs(If - I1)/15;
25 if err > tol
26     If = simpad(a, x1, fun, tol/2, fa, f2, f1)...
27         + simpad(x1, b, fun, tol/2, f1, f3, fb);
28 end
29 end

```

5.5 Esercizio 5

Prova

Capitolo 6

Calcolo del Google pagerank. Risoluzione iterativa di sistemi lineari

6.1 Esercizio 1

Prova

```
1 function A = matriceSparsa(n)
2 % A = matriceSparsa(n)
3 % Genera la matrice quadrata sparsa nxn, con n maggiore di 10.
4 %
5 % Input:
6 %   - n: numero di righe/colonne della matrice quadrata sparsa
7 % Output:
8 %   - A: matrice quadrata nxn sparsa
9 %n deve essere maggiore di 10
10 if n <= 10
11     error('n deve essere maggiore di 10.');

---


```

6.2 Esercizio 2

Prova

```

1 function [lambda, i] = potenze(A, tol, x0, maxit)
2 % [lambda, i] = metodoPotenze(A, tol, [x0, maxit])
3 % Restituisce l'autovalore dominante della matrice A e il numero di
4 % iterazioni necessarie per calcolarlo.
5 %
6 % Input:
7 %   - A: matrice utilizzata per il calcolo
8 %   - tol: tolleranza dell' approssimazione
9 %   - [x0]: vettore di partenza
10 %   - [maxit]: numero massimo di iterazioni
11 % Output:
12 %   - lambda: matrice quadrata nxn sparsa
13 %   - i: numero di iterazioni
14 [m,n] = size(A);
15 if m ~= n
16     error('La matrice deve essere quadrata.');
```

```

17 end
18 if x0(:)==0
19     error('Il vettore x0 non puo avere esclusivamente elementi nulli.');
```

```

20 end % if da eliminare per calcolo con matrice
21
22 if nargin <= 2
23     x = rand(n, 1);
24 else
25     x = x0;
26 end
27 if nargin <= 3
28     maxit = 100*2*round(-log(tol));
29 end
30 x = x0; % da eliminare per calcolo con matrice
31 x = x / norm(x);
32 lambda = inf;
33 for i=1:maxit
34     lambda0 = lambda;
35     v = A * x;
36     lambda = x' * v;
37     err = abs(lambda - lambda0);
38     if err <= tol
39         break
40     end
41     x = v/norm(v);
42 end
43 if err > tol
44     warning("Raggiunto maxit.");
45 end
46 return
47 end

```

6.3 Esercizio 3

Prova

```

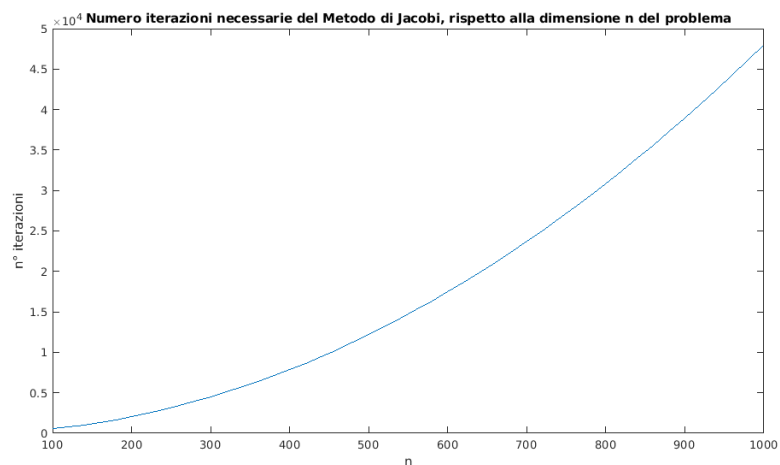
1 function [x,i,nr] = jacobi(A, b, tol, x0, maxit)
2 % [x,i] = jacobi(A, b, tol, [x0, maxit])
3 % Restituisce la soluzione del sistema lineare Ax=b approssimata con il
4 % metodo di Jacobi e il numero di iterazioni eseguite.
5 % Input:
6 %   - A: matrice utilizzata per il calcolo

```

```

7 % - b: vettore dei termini noti
8 % - tol: tolleranza dell' approssimazione
9 % - [x0]: vettore di partenza
10 % - [maxit]: numero massimo di iterazioni
11 % Output:
12 % - x: soluzione approssimata del sistema
13
14 % - A non deve avere elementi nulli sulla diagonale
15 D = diag(A);
16 if ~all(D) % all(D) ritorna vero se tutti elementi !=0
17     error('La diagonale di A non deve avere elementi nulli');
18 end
19 n = length(b);
20 if nargin <= 3
21     x = rand(n,1);
22 else
23     x = x0;
24 end
25 if nargin <= 4
26     maxit = 100*n*round(-log(tol));
27 end
28 for i = 1:maxit
29     r = A * x - b;
30     nr = norm(r, inf);
31     if nr <= tol
32         break;
33     end
34     r = r./D;
35     x = x - r;
36 end
37 if nr > tol
38     warning('Raggiunto maxit. ');
39 end
40 return
41 end

```



6.4 Esercizio 4

Prova

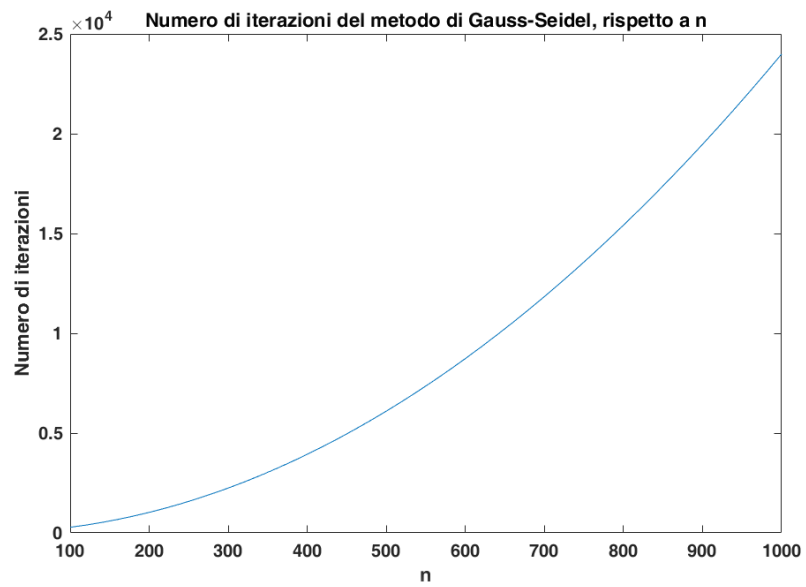
```

1 function [x,i,nr] = gauss_seidel(A, b, tol, x0, maxit)
2 % [x,i] = jacobi(A, b, tol, [x0, maxit])
3 % Restituisce la soluzione del sistema lineare Ax=b approssimata con il
4 % metodo di Gauss-Seidel e il numero di iterazioni eseguite.
5 % Input:
6 %   - A: matrice utilizzata per il calcolo
7 %   - b: vettore dei termini noti
8 %   - tol: tolleranza dell' approssimazione
9 %   - [x0]: vettore di partenza
10 %   - [maxit]: numero massimo di iterazioni
11 % Output:
12 %   - x: soluzione approssimata del sistema
13     n = length(b);
14     if nargin <= 3
15         x = rand(n,1);
16     else
17         x = x0;
18     end
19     if nargin <= 4
20         maxit = 100*n*round(-log(tol));
21     end
22     for i = 1:maxit
23         r = A * x - b;
24         err = norm(r,inf);
25         nr(i) = err;
26         if err<=tol
27             break;
28         end
29         r = Msolve(A,r);
30         x = x-r;
31     end
32     if err>tol
33         warning('Non raggiunta tolleranza richiesta.');
```

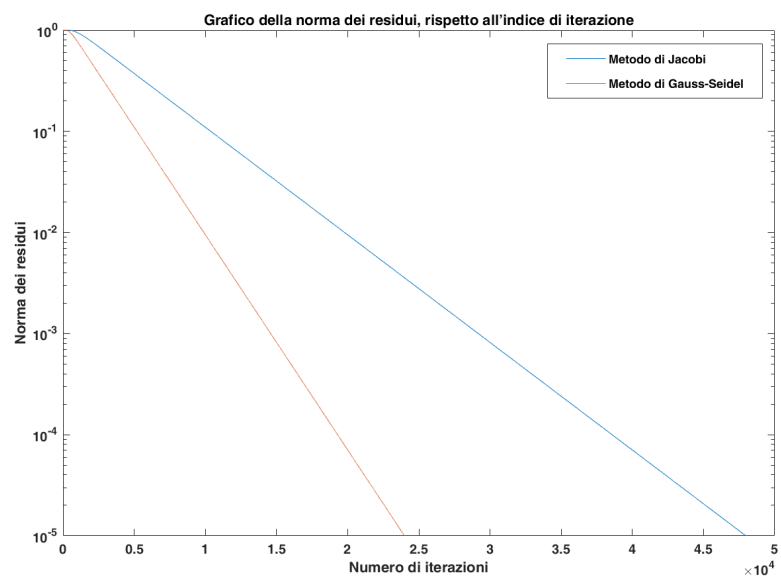
```

34     end
35     return
36 end
37
38 function u = Msolve(M, r)
39     % u = Msolve(M, r)
40     % Restituisce la soluzione del sistema lineare Mx=r.
41     u = r;
42     n = length(u);
43     for i = 1:n
44         u(i) = u(i)/M(i,i);
45         u(i+1:n) = u(i+1:n) - M(i+1:n,i)*u(i);
46     end
47     return
48 end

```



6.5 Esercizio 5



Prova