

```

1 # Nome del file: equazioni_non_lineari.mpl6
2 # Autore: Claudio Marsan
3 # Ultima revisione: 28 novembre 2001
4 # Realizzato per Maple V release 6.02
5 # Testato su MS-Windows 2000
6 # Contenuto: alcuni metodi per la risoluzione di equazioni non lineari
7 # (1) Il metodo di bisezione
8 # (2) Il metodo di falsa posizione (regula falsi)
9 # (3) Il metodo delle secanti
10 # (4) Il metodo delle tangenti (metodo di Newton)
11 # (5) Il metodo di iterazione
12
13
14 #####
15 # (1) Il metodo di bisezione #
16 #####
17
18 metodo_di_bisezione := proc(f::algebraic,      # la funzione di cui si cerca uno zero
19                             a,                # estremo sinistro dell'intervallo
20                             b,                # estremo destro dell'intervallo
21                             max_passi::posint, # numero massimo di passi da eseguire
22                             epsilon)         # precisione richiesta
23   local F, incognite, x0, x1, x2, n;
24   incognite := op(select(type, indets(f), name));
25   F := unapply(f, incognite);
26   x0 := evalf(a); x1 := evalf(b);
27   if (x0 >= x1) then
28     ERROR(`Intervallo inesistente!`);
29   end if;
30   if evalf(F(x0)*F(x1)) >= 0.0 then
31     ERROR(`Intervallo non valido!`);
32   end if;
33   n := 0;
34   while (n < max_passi) do
35     n := n + 1;
36     x2 := 0.5*(x0 + x1);
37     if evalf(F(x0)*F(x2)) < 0.0 then
38       x1 := x2;
39     else
40       x0 := x2;
41     end if;
42     lprint(n, x2, F(x2));
43     if abs(F(x2)) < epsilon then
44       break;
45     end if;
46   end do;
47 end;
48
49
50 #####
51 # (2) Il metodo di falsa posizione (regula falsi) #
52 #####
53
54 regula_falsi := proc(f::algebraic,      # la funzione di cui si cerca uno zero
55                       a,                # una ascissa
56                       b,                # un'altra ascissa
57                       max_passi::posint, # numero massimo di passi da eseguire
58                       epsilon)         # precisione richiesta
59   local F, incognite, x0, x1, x2, y0, y1, y2, n;
60   incognite := op(select(type, indets(f), name));
61   F := unapply(f, incognite);
62   x0 := evalf(a); x1 := evalf(b);
63   if (x0 >= x1) then
64     ERROR(`Intervallo inesistente!`);
65   end if;
66   y0 := evalf(F(x0)); y1 := evalf(F(x1));
67   if evalf(F(x0)*F(x1)) >= 0.0) then
68     ERROR(`Intervallo non valido!`);
69   end if;
70   n := 0;
71   while (n < max_passi) do

```

```

72     n := n + 1;
73     x2 := x1 - y1*(x1 - x0)/(y1 - y0);
74     y2 := evalf(F(x2));
75     if y0*y2 < 0.0 then
76         x1 := x2;
77         y1 := y2;
78     else
79         x0 := x2;
80         y0 := y2;
81     end if;
82     lprint(n, x2, y2);
83     if abs(y2) < epsilon then
84         break;
85     end if;
86 end do;
87 end;
88
89
90
91 #####
92 # (3) Il metodo delle secanti #
93 #####
94
95 metodo_delle_secanti := proc(f::algebraic,      # la funzione di cui si cerca uno zero
96                             a,                  # estremo sinistro dell'intervallo
97                             b,                  # estremo destro dell'intervallo
98                             max_passi::posint, # numero massimo di passi da eseguire
99                             epsilon)           # precisione richiesta
100 local F, incognite, x0, x1, x2, y0, y1, y2, n;
101 incognite := op(select(type, indets(f), name));
102 F := unapply(f, incognite);
103 x0 := evalf(a); x1 := evalf(b);
104 y0 := evalf(F(x0)); y1 := evalf(F(x1));
105 n := 0;
106 while (n < max_passi) do
107     n := n + 1;
108     x2 := x1 - y1*(x1 - x0)/(y1 - y0);
109     y2 := evalf(F(x2));
110     x0 := x1;
111     y0 := y1;
112     x1 := x2;
113     y1 := y2;
114     lprint(n, x2, y2);
115     if abs(y2) < epsilon then
116         break;
117     end if;
118 end do;
119 end;
120
121
122
123 #####
124 # (4) Il metodo delle tangenti (metodo di Newton) #
125 #####
126
127 metodo_di_Newton := proc(f::algebraic,      # la funzione di cui si cerca uno zero
128                          a,                  # valore iniziale
129                          max_passi::posint, # numero massimo di passi da eseguire
130                          epsilon)           # precisione richiesta
131 local F, incognite, x0, xn, yn, n;
132 incognite := op(select(type, indets(f), name));
133 F := unapply(f, incognite);
134 xn := evalf(a);
135 n := 0;
136 while (n < max_passi) do
137     n := n + 1;
138     x0 := xn;
139     xn := x0 - F(x0)/(D(F)(x0));
140     yn := F(xn);
141     lprint(n, xn, yn);
142     if abs(yn) < epsilon then

```

```

143     break;
144   end if;
145 end do;
146 end;
147
148
149 #####
150 # (5) Il metodo di iterazione #
151 #####
152
153 metodo_di_iterazione := proc(f::algebraic,      # la funzione di cui si cerca uno zero
154                               a,                # valore iniziale
155                               max_passi::posint, # numero massimo di passi da eseguire
156                               epsilon)          # precisione richiesta
157   local F, incognite, x1, x2, n;
158   incognite := op(select(type, indets(f), name));
159   F := unapply(f, incognite);
160   x1 := evalf(a); x2 := F(x1);
161   n := 0;
162   while (n < max_passi) do
163     n := n + 1;
164     x1 := x2;
165     x2 := F(x1);
166     lprint(n, x1, x2);
167     if abs(x2 - x1) < epsilon then
168       break;
169     end if;
170   end do;
171 end;
172

```