

```

1 #####
2 #
3 #          Studio di funzioni razionali fratte
4 #          -----
5 #
6 # Autore: Claudio Marsan
7 # Ultima modifica: 16 marzo 2003
8 # Versione: Maple V Release 6.02 for Windows 2000
9 #
10 #####
11
12 studio_funzione_razionale := proc(funz_rat :: ratpoly, x, x1, x2, y1, y2)
13   local load_lib, f, num, den, poli, num_poli, zeri, pn, num_pn, i, grv, t,
14     as_or, grh, m_obl, q_obl, gro, fl, f2, aux_crit, crit, num_crit,
15     flessi_or, k, cresc, num_cresc, aux_conc, conc, num_conc, conc2,
16     num_conc2, grf;
17
18   # Controllo input
19   if x1 >= x2 then
20     ERROR(`Intervallo delle ascisse non valido!`);
21   end if;
22   if y1 >= y2 then
23     ERROR(`Intervallo delle ordinate non valido!`);
24   end if;
25
26   # Carica librerie ausiliarie
27   if load_lib <> 1 then
28     with(plots):
29     load_lib := 1;
30   end if;
31
32   # Semplificazione della funzione data
33   f := unapply(simplify(funz_rat(x)), x);
34
35   # Numeratore e denominatore di f(x)
36   num := numer(f(x));
37   den := denom(f(x));
38   lprint(`-----`);
39
40   # Zeri del denominatore di f(x) (poli)
41   poli := {fsolve(den = 0, x, fulldigits)};
42   num_poli := nops(poli);
43   if num_poli = 0 then
44     lprint(`Il dominio della funzione e' l'insieme dei numeri reali`);
45   else
46     lprint(`Punti estranei al dominio della funzione: `);
47     print(poli);
48   end if;
49   lprint(`-----`);
50
51   # Zeri del numeratore di f(x)
52   zeri := {fsolve(num = 0, x, fulldigits)};
53   if nops(zeri) = 0 then
54     lprint(`La funzione non ha zeri reali`);
55   else
56     lprint(`Zeri della funzione:`);
57     print(zeri);
58   end if;
59   lprint(`-----`);
60
61   # Ordinata all'origine
62   if {0.0} intersect poli = {0.0} then
63     lprint(`Il grafico della funzione non interseca l'asse y`);
64   else
65     lprint(`Ordinata all'origine:`);
66     print(f(0));
67   end if;
68   lprint(`-----`);
69
70   # Pari o dispari
71   if eval(f(x)-f(-x)) = 0 then
72     lprint(`La funzione e' pari`);
73   else
74     lprint(`La funzione non e' pari`);
75   end if;
76   if eval(f(x)+f(-x)) = 0 then
77     lprint(`La funzione e' dispari`);
78   else
79     lprint(`La funzione non e' dispari`);
80   end if;
81   lprint(`-----`);
82
83   # Segno della funzione negli intervalli determinati dai punti notevoli
84   pn := sort(convert(poli union zeri, list));
85   num_pn := nops(pn);
86   if num_pn = 0 then
87     if f(0) > 0 then
88       lprint(`La funzione e' sempre positiva`);
89     else
90       lprint(`La funzione e' sempre negativa`);
91     end if;
92   else
93     if f(pn[1]-1) < 0 then
94       lprint(`La funzione e' negativa nell'intervallo aperto:`);

```

```

95     print(-infinity..pn[1]);
96     else
97         lprint(`La funzione e' positiva nell'intervallo aperto:`);
98         print(-infinity..pn[1]);
99     end if;
100    if num_pn <> 1 then
101        for i from 1 to num_pn-1 do
102            if f(0.5*(pn[i] + pn[i+1])) < 0 then
103                lprint(`La funzione e' negativa nell'intervallo aperto:`);
104                print(pn[i]..pn[i+1]);
105            else
106                lprint(`La funzione e' positiva nell'intervallo aperto:`);
107                print(pn[i]..pn[i+1]);
108            end if;
109        end do;
110    end if;
111    if f(pn[num_pn]+1) < 0 then
112        lprint(`La funzione e' negativa nell'intervallo aperto:`);
113        print(pn[num_pn]..+infinity);
114    else
115        lprint(`La funzione e' positiva nell'intervallo aperto:`);
116        print(pn[num_pn]..+infinity);
117    end if;
118 end if;
119 lprint(`-----`);
120
121 # Asintoti verticali
122 if num_poli = 0 then
123     lprint(`Non ci sono asintoti verticali`);
124     grv := plot(0, x1..x2, color=BLACK);
125 else
126     lprint(`Asintoti verticali: `);
127     for i from 1 to num_poli do
128         print(`x` = poli[i]);
129         lprint();
130     end do;
131     grv := seq(plot([poli[i], t, t=y1..y2], color=BLUE), i=1..num_poli);
132 end if;
133 lprint(`-----`);
134
135 # Asintoti orizzontali
136 as_or := limit(f(x), x = infinity);
137 if {as_or} intersect {infinity, -infinity, undefined} = {} and type(as_or, realcons) then
138     lprint(`Asintoto orizzontale:`);
139     print(`y` = as_or);
140     grh := plot(as_or, x1..x2, color=BLUE);
141 else
142     lprint(`Non ci sono asintoti orizzontali`);
143     grh := plot(0, x1..x2, color=BLACK);
144 end if;
145 lprint(`-----`);
146
147 # Asintoti obliqui
148 m_obl := limit(f(x)/x, x = infinity);
149 if {m_obl} intersect {0, infinity, -infinity, undefined} = {} and type(m_obl, realcons) then
150     q_obl := limit(f(x) - m_obl*x, x = infinity);
151     lprint(`Asintoto obliquo:`);
152     print(`y` = m_obl*x + q_obl);
153     gro := plot(m_obl*x + q_obl, x=x1..x2, color=BLUE);
154 else
155     lprint(`Non ci sono asintoti obliqui`);
156     gro := plot(0, x1..x2, color=BLACK);
157 end if;
158 lprint(`-----`);
159
160 # Calcolo della prima derivata di f(x);
161 f1 := D(f);
162 lprint(`La derivata prima della funzione:`);
163 print(factor(f1(x)));
164 lprint(`-----`);
165
166 # Calcolo della seconda derivata della funzione
167 f2 := D(f1);
168 lprint(`La derivata seconda della funzione:`);
169 print(factor(f2(x)));
170 lprint();
171 lprint(`-----`);
172
173 # Punti critici
174 aux_crit := {fsolve(numer(factor(f1(x))) = 0, x, fulldigits)};
175 crit := sort(convert(aux_crit, list));
176 num_crit := nops(crit);
177 flessi_or := {};
178 if num_crit = 0 then
179     lprint(`Non ci sono punti critici`);
180 else
181     lprint(`Punti critici:`);
182     print(seq([crit[i], f(crit[i])], i = 1..num_crit));
183     lprint(`-----`);
184     for i from 1 to num_crit do
185         k := 2;
186         while subs(x = crit[i], diff(f(x), x$k)) = 0 do
187             k := k + 1;
188         end do;

```

```

189     if type(k, odd) then
190         flessi_or := flessi_or union {crit[i]};
191         lprint(`Flesso orizzontale in:`);
192         print([crit[i], f(crit[i])]);
193     else
194         if subs(x = crit[i], diff(f(x), x$k)) > 0 then
195             lprint(`Minimo locale in:`);
196             print([crit[i], f(crit[i])]);
197         else
198             lprint(`Massimo locale in:`);
199             print([crit[i], f(crit[i])]);
200         end if;
201     end if;
202 end do;
203 end if;
204 lprint(`-----`);
205
206 # Intervalli in cui la funzione e' crescente o decrescente
207 cresc := aux_crit union poli;
208 cresc := sort(convert(cresc, list));
209 num_cresc := nops(cresc);
210 if num_cresc = 0 then
211     if f1(0) > 0 then
212         lprint(`La funzione e' sempre crescente`);
213     else
214         lprint(`La funzione e' sempre decrescente`);
215     end if;
216 else
217     if f1(cresc[1]-1) < 0 then
218         lprint(`La funzione e' decrescente nell'intervallo aperto:`);
219         print(-infinity..cresc[1]);
220     else
221         lprint(`La funzione e' crescente nell'intervallo aperto:`);
222         print(-infinity..cresc[1]);
223     end if;
224     if num_cresc <> 1 then
225         for i from 1 to num_cresc-1 do
226             if f1(0.5*(cresc[i] + cresc[i+1])) < 0 then
227                 lprint(`La funzione e' decrescente nell'intervallo aperto:`);
228                 print(cresc[i]..cresc[i+1]);
229             else
230                 lprint(`La funzione e' crescente nell'intervallo aperto:`);
231                 print(cresc[i]..cresc[i+1]);
232             end if;
233         end do;
234     end if;
235     if f1(cresc[num_cresc]+1) < 0 then
236         lprint(`La funzione e' decrescente nell'intervallo aperto:`);
237         print(cresc[num_cresc]..infinity);
238     else
239         lprint(`La funzione e' crescente nell'intervallo aperto:`);
240         print(cresc[num_cresc]..infinity);
241     end if;
242 end if;
243 lprint(`-----`);
244
245 # Flessi obliqui
246 aux_conc := {fsolve( numer(factor(f2(x))) = 0, x, fulldigits)};
247 aux_conc := aux_conc minus flessi_or;
248 conc := sort(convert(aux_conc, list));
249 num_conc := nops(conc);
250 if num_conc = 0 then
251     lprint(`Non ci sono flessi obliqui`);
252 else
253     lprint(`Flessi obliqui:`);
254     print(seq([conc[i], f(conc[i])], i = 1..num_conc));
255 end if;
256 lprint(`-----`);
257
258 # Intervalli nei quali la funzione e' convessa/concava
259 conc2 := poli union aux_conc union flessi_or;
260 conc2 := sort(convert(conc2, list));
261 num_conc2 := nops(conc2);
262 if num_conc2 = 0 then
263     if f2(0) > 0 then
264         lprint(`La funzione e' convessa`);
265     else
266         lprint(`La funzione e' concava`);
267     end if;
268 else
269     if f2(conc2[1]-1) > 0 then
270         lprint(`La funzione e' convessa nell'intervallo aperto:`);
271         print(-infinity..conc2[1]);
272     else
273         lprint(`La funzione e' concava nell'intervallo aperto:`);
274         print(-infinity..conc2[1]);
275     end if;
276     if num_conc2 <> 1 then
277         for i from 1 to num_conc2 - 1 do
278             if f2(0.5*(conc2[i] + conc2[i+1])) > 0 then
279                 lprint(`La funzione e' convessa nell'intervallo aperto:`);
280                 print(conc2[i]..conc2[i+1]);
281             else
282                 lprint(`La funzione e' concava nell'intervallo aperto:`);

```

```
283         print(conc2[i]..conc2[i+1]);
284     end if;
285 end do;
286 end if;
287 if f2(conc2[num_conc2]+1) > 0 then
288     lprint(`La funzione e' convessa nell'intervallo aperto:`);
289     print(conc2[num_conc2]..infinity);
290 else
291     lprint(`La funzione e' concava nell'intervallo aperto:`);
292     print(conc2[num_conc2]..infinity);
293 end if;
294 end if;
295 lprint(`-----`);
296
297 # Visualizzazione grafica
298 grf := plot(f(x), x=x1..x2, y=y1..y2, discont=true, color=RED);
299 display([grh, gro, grf, grv]);
300
301 end;
```