Lezione 7

Enrico Bertolazzi

```
> restart:
> secanti := proc (f, p0, p1, niter, N)
    # f      = funzione assegnata
    # p0, p1 = punti di partenza per il metodo delle secanti
    # niter  = numero di iterazioni assegnato
    #          (in una procedura "GIUSTA" si assegna la tolleranza)
    # N      = cifre usate nei conti
    #
    local i, K, f0, f1, f2, x0, x1, x2 ;
    x0 := evalf(p0,N);
    x1 := evalf(p1,N);
    f0 := evalf(f(x0),N);
    f1 := evalf(f(x1),N);
    for i from 1 to niter do
      K  := evalf((f1-f0)/(x1-x0),N);
      x2 := x1 - f1/K;
      f2 := evalf(f(x2),N);
      # shift dei nomi delle variabili
      x0 := x1 ; f0 := f1 ;
      x1 := x2 ; f1 := f2 ;
    end;
    return x2 ;
  end proc ;
```

$$secanti := \mathbf{proc}(f, p0, p1, niter, N) \tag{1}$$

local $i, K, f0, f1, f2, x0, x1, x2;$

$x0 := evalf(p0, N);$

$x1 := evalf(p1, N);$

$f0 := evalf(f(x0), N);$

$f1 := evalf(f(x1), N);$

for i **to** $niter$ **do**

$K := evalf((f1 - f0)/(x1 - x0), N);$

$x2 := x1 - f1/K;$

$f2 := evalf(f(x2), N);$

$x0 := x1;$

$f0 := f1;$

$x1 := x2;$

```
        f1 := f2
    end do;
    return x2
end proc
```

```
> # uso il metodo delle secanti per approssimare sqrt(2)
  secanti(x -> x^2-2, 3, 2, 5, 20);
```

1.414213575 **(2)**

```
> evalf(sqrt(2),20);
```

1.4142135623730950488 **(3)**

```
> # uso il metodo delle secanti per approssimare root[3](3)
  secanti(x -> x^3-3,3,2,6,20);
```

1.442249574 **(4)**

```
> evalf(root[3](3));
```

1.442249570 **(5)**

```
> # uso il metodo delle secanti per approssimare
  # il reciproco di 6 un numero;
  secanti(x -> 6*x-1,5,4,2,3);
```

0.166666667 **(6)**

```
> evalf (1/6);
```

0.1666666667 **(7)**

```
> # Procedura secanti con grafica per visualizzare la convergenza
  #
  with (plots):
Warning, the name changecoords has been redefined
```

```
> # dati 3 punti su una retta restituisce il segmento piu lungo

  secanti_seg := proc (x0,y0,x1,y1,x2,y2)
    local xx, yy, xxx, yyy ;
    xx := x0 ;
    yy := y0 ;
    if xx > x1 then
      xx := x1 ;
      yy := y1 ;
    end if ;
    if xx > x2 then
      xx := x2 ;
      yy := y2 ;
    end if ;
    xxx := x0 ;
    yyy := y0 ;
    if xxx < x1 then
      xxx := x1 ;
```

```
      yyy := y1 ;
    end if ;
    if xxx < x2 then
      xxx := x2 ;
      yyy := y2 ;
    end if ;
    return [[xx,yy],[xxx,yyy]] ;
  end proc ;
```

$secanti\_seg := \mathbf{proc}(x0, y0, x1, y1, x2, y2)$ **(8)**

local xx, yy, xxx, yyy;

$xx := x0$;

$yy := y0$;

if $x1 < xx$ **then** $xx := x1$; $yy := y1$ **end if**;

if $x2 < xx$ **then** $xx := x2$; $yy := y2$ **end if**;

$xxx := x0$;

$yyy := y0$;

if $xxx < x1$ **then** $xxx := x1$; $yyy := y1$ **end if**;

if $xxx < x2$ **then** $xxx := x2$; $yyy := y2$ **end if**;

return $[[xx, yy], [xxx, yyy]]$

end proc

```
> # procedura secanti con grafica

  secanti := proc (f, p0, p1, niter, N)
    # f      = funzione assegnata
    # p0, p1 = punti di partenza per il metodo delle secanti
    # niter  = numero di iterazioni assegnato
    #          (in una procedura "GIUSTA" si assegna la tolleranza)
    # N      = cifre usate nei conti
    #
    local i, K, f0, f1, f2, x0, x1, x2, L ;
    x0 := evalf(p0,N) ;
    x1 := evalf(p1,N) ;
    f0 := evalf(f(x0),N) ;
    f1 := evalf(f(x1),N) ;
    # primo segmento verticale
    L := [ [[x0,0],[x0,f0]] ] ;
    for i from 1 to niter do
      K  := evalf((f1-f0)/(x1-x0),N) ;
      x2 := x1 - f1/K ;
      f2 := evalf(f(x2),N) ;
      # segmento verticale
      L := [op(L), [[x1,0],[x1,f1]] ] ;
```

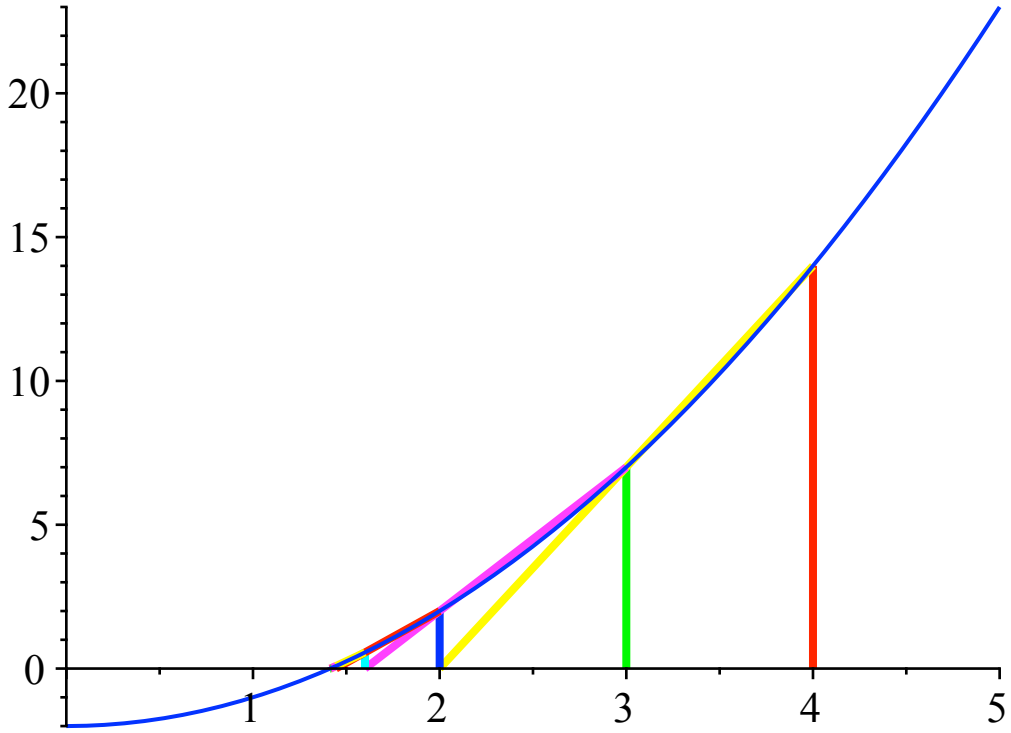
```
      L := [op(L), secanti_seg(x0,f0,x1,f1,x2,0)] ;
      # shift dei nomi delle variabili
      x0 := x1 ; f0 := f1 ;
      x1 := x2 ; f1 := f2 ;
    end ;
    # L = lista dei segmenti che passano per
    # [x[k-1],y[k-1]], [x[k],y[k]] e intersecano
    # l'asse delle x.
    return x2, L;
  end proc ;
```

$$secanti := \mathbf{proc}(f, p0, p1, niter, N) \tag{9}$$

```
    local i, K, f0, f1, f2, x0, x1, x2, L;
    x0 := evalf(p0, N);
    x1 := evalf(p1, N);
    f0 := evalf(f(x0), N);
    f1 := evalf(f(x1), N);
    L := [[[x0, 0], [x0, f0]]];
    for i to niter do
        K := evalf( (f1 - f0) / (x1 - x0), N);
        x2 := x1 - f1 / K;
        f2 := evalf(f(x2), N);
        L := [op(L), [[x1, 0], [x1, f1]]];
        L := [op(L), secanti_seg(x0, f0, x1, f1, x2, 0)];
        x0 := x1;
        f0 := f1;
        x1 := x2;
        f1 := f2
    end do;
    return x2, L
end proc
```

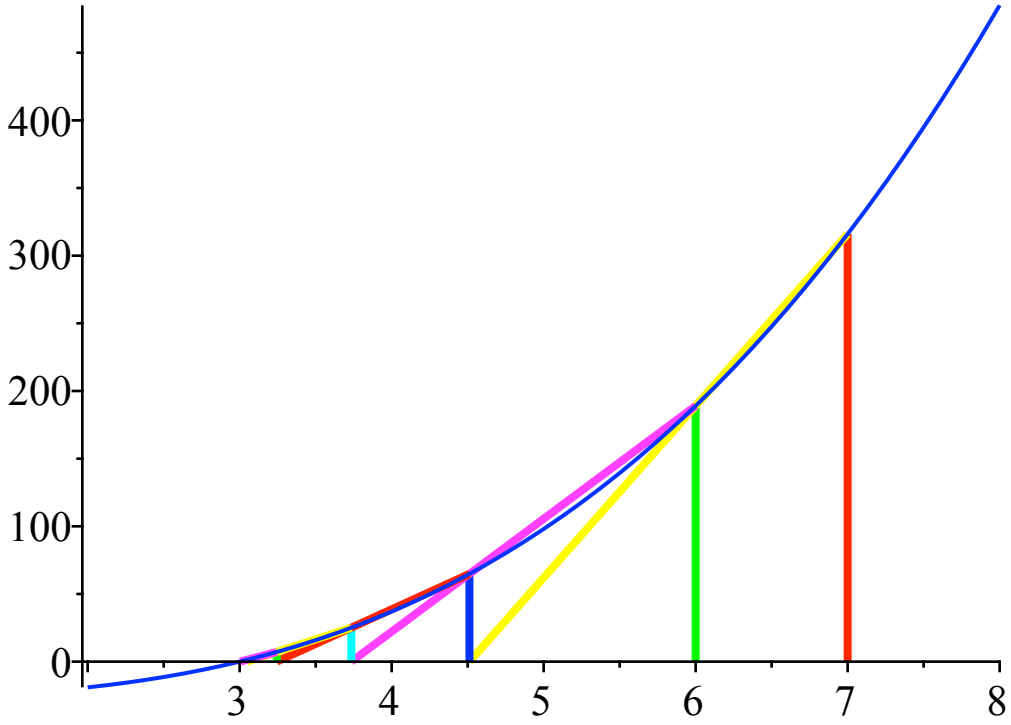
```
> # esempio 1:
  # andamento del metodo delle secanti
  # per il calcolo di sqrt(2)
  fun    := x -> x^2 - 2 ;
  sol, L := secanti(fun,4,3,7,200):
  A := plot(L,thickness=3):
  B := plot(fun,0..5,color=blue):
  display({A,B});
```

$$fun := x \to x^2 - 2$$



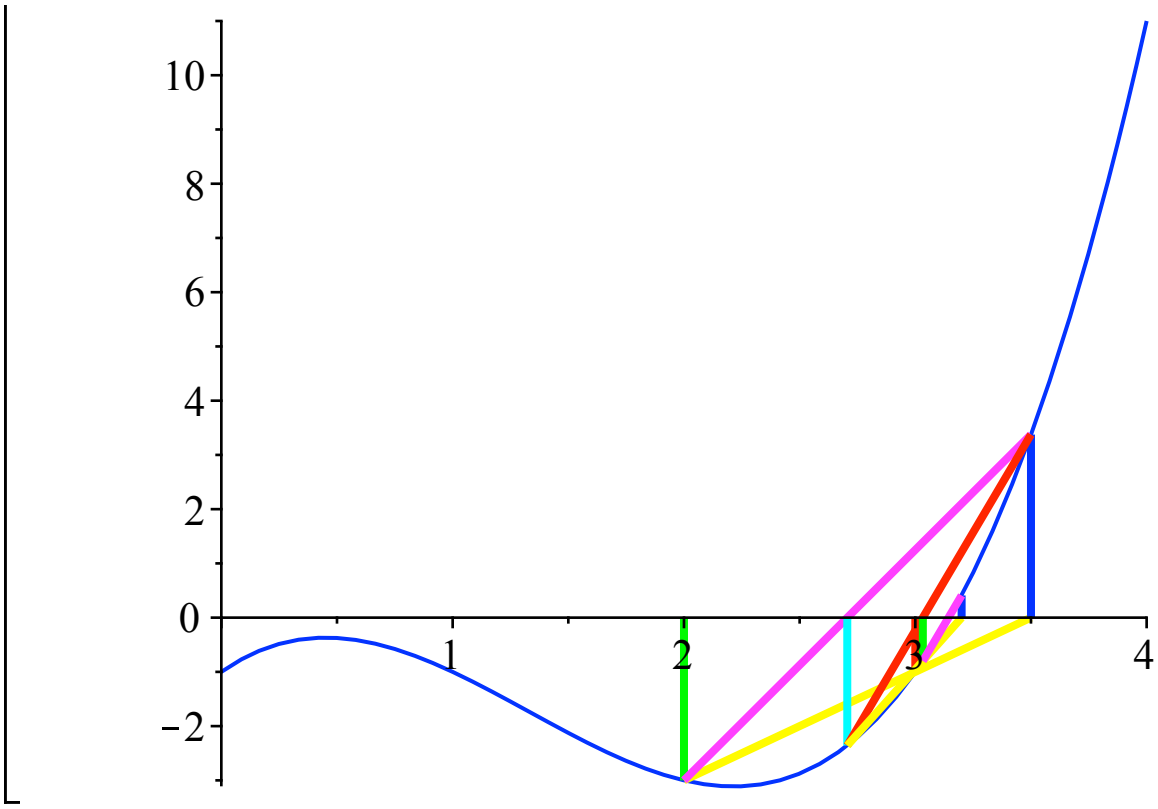

```
> # esempio 2:
  # andamento del metodo delle secanti
  # per il calcolo di root[3](27)
  fun    := x -> x^3-27 ;
  sol, L := secanti(fun,7,6,5,200):
  A := plot(L,thickness=3):
  B := plot(fun,2..8,color=blue):
  display({A,B});
```

$$fun := x \to x^3 - 27$$




```
> # esempio 3:
  # andamento del metodo delle secanti
  # per il calcolo di 1/sqrt(3).
  # esempio di andamento complesso
  fun    := x -> x*(x-1)*(x-3)-1 ;
  sol, L := secanti(fun,3,2,5,200):
  A := plot(L,thickness=3):
  B := plot(fun,0..4,color=blue):
  display({A,B});
```

$$fun := x \rightarrow x\,(x-1)\,(x-3)-1$$


10
8
6
4
2
0
−2
1
2
3
4