Algoritmi in C++
(seconda parte)

Introduzione

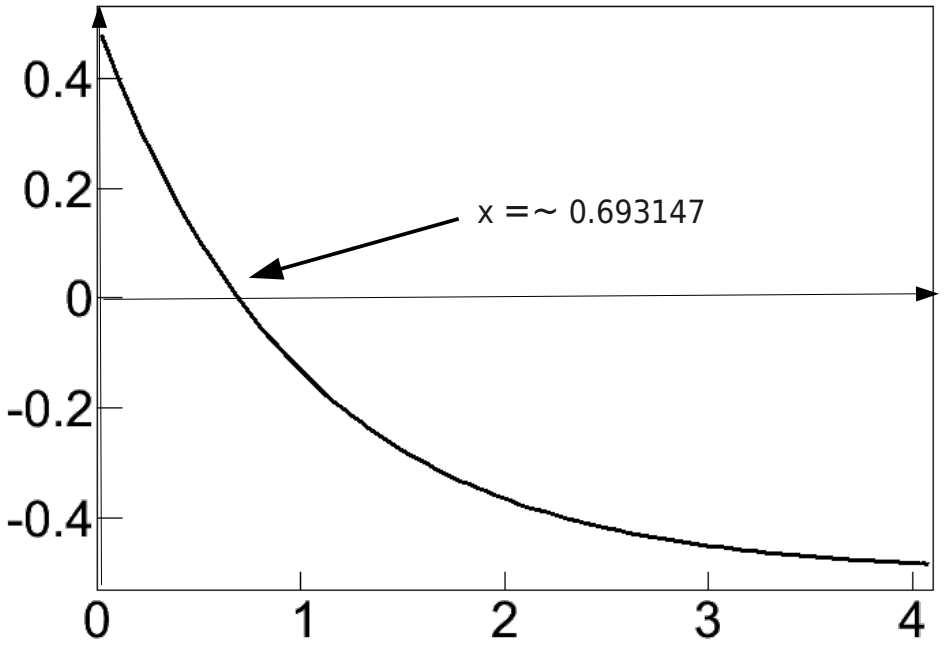
- Obiettivo: imparare a risolvere problemi analitici con semplici programmi in C++.
- Nella prima parte abbiamo imparato:
  - generazione di sequenze di numeri casuali
  - metodi per generare numeri casuali con una distribuzione di probabilità descritta da una funzione (uniforme, gaussiana, esponenziale...)
  - qualche funzionalità di ROOT per creare i grafici degli istogrammi delle distribuzioni
- Problemi che affronteremo in questa lezione:
  - ricerca degli zeri di una funzione
  - integrazione numerica di una funzione
  - ricerca degli estremanti di una funzione

Ricerca degli zeri di una funzione

Zeri di una funzione

- Trovare numericamente gli zeri di funzioni
- Assumiamo funzioni regolari (continue su un compatto)
- Consideriamo funzioni con un solo zero nell'intervallo di studio
- Utilizziamo come esempio la funzione

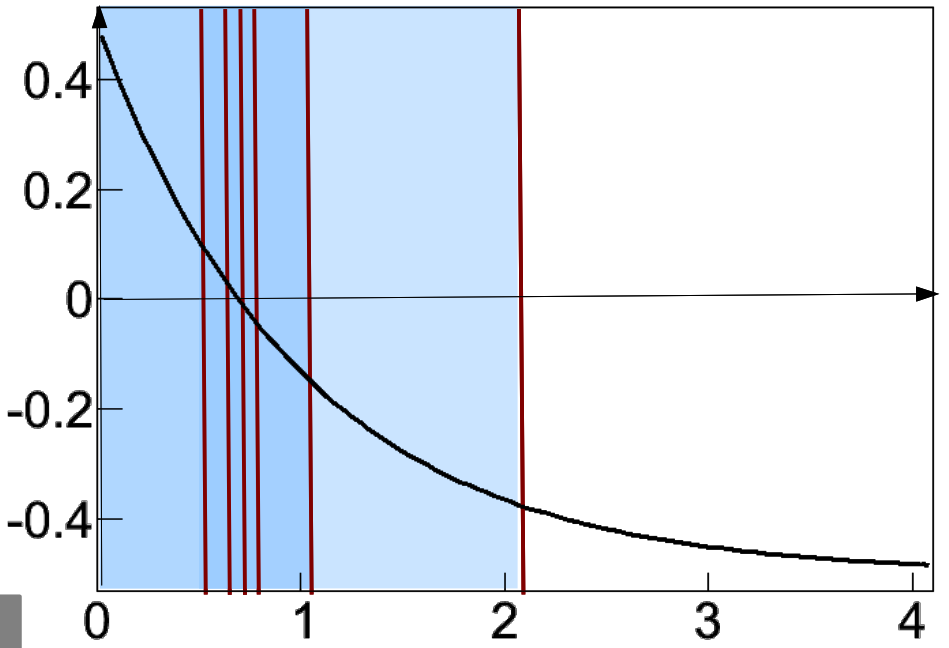
$$f(x) = e^{-x} - 0.5$$



Il metodo della bisezione

Si dimezza l'intervallo x che contiene lo zero finchè non è minore della precisione che interessa

La funzione è approssimata all'ordine zero




```
for (int i = 0 ; i < num_iterations ; ++i)
  {
    double x_m = 0.5 * (xMin + xMax) ;
    if (f (xMin) * f (x_m) < 0)
      xMax = x_m ;
    else
      xMin = x_m ;
  }
return 0.5 * (xMax + xMin) ;
```

L'intervallo che contiene lo zero si riconosce controllando la concordanza delle y

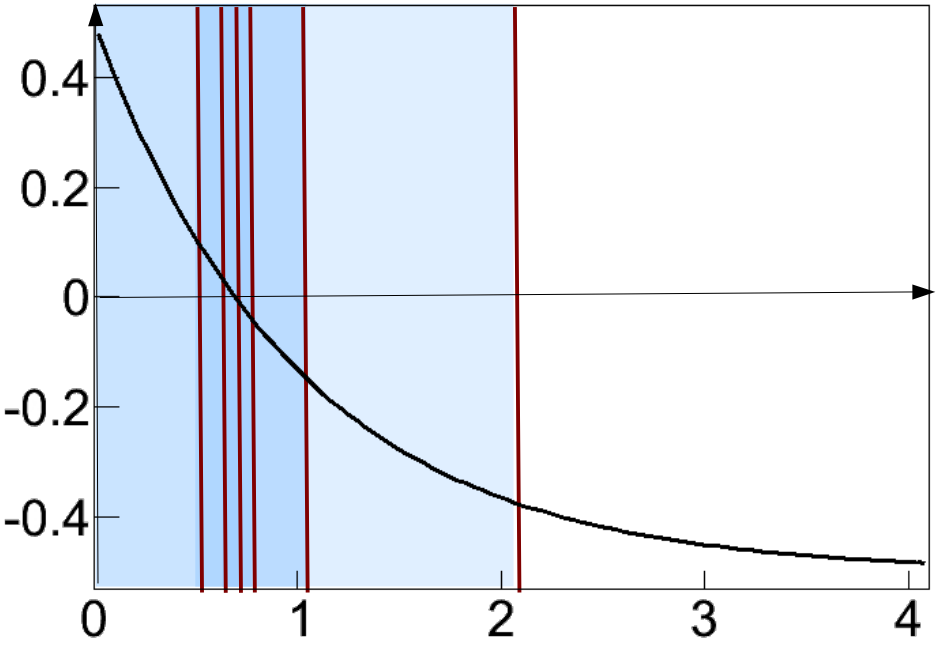
Si restituisce il punto medio dell'intervallo dopo un certo numero di iterazioni

La bisezione ricorsiva

L'algoritmo è ricorsivo, cioè ad ogni iterazione si ripete la stessa procedura

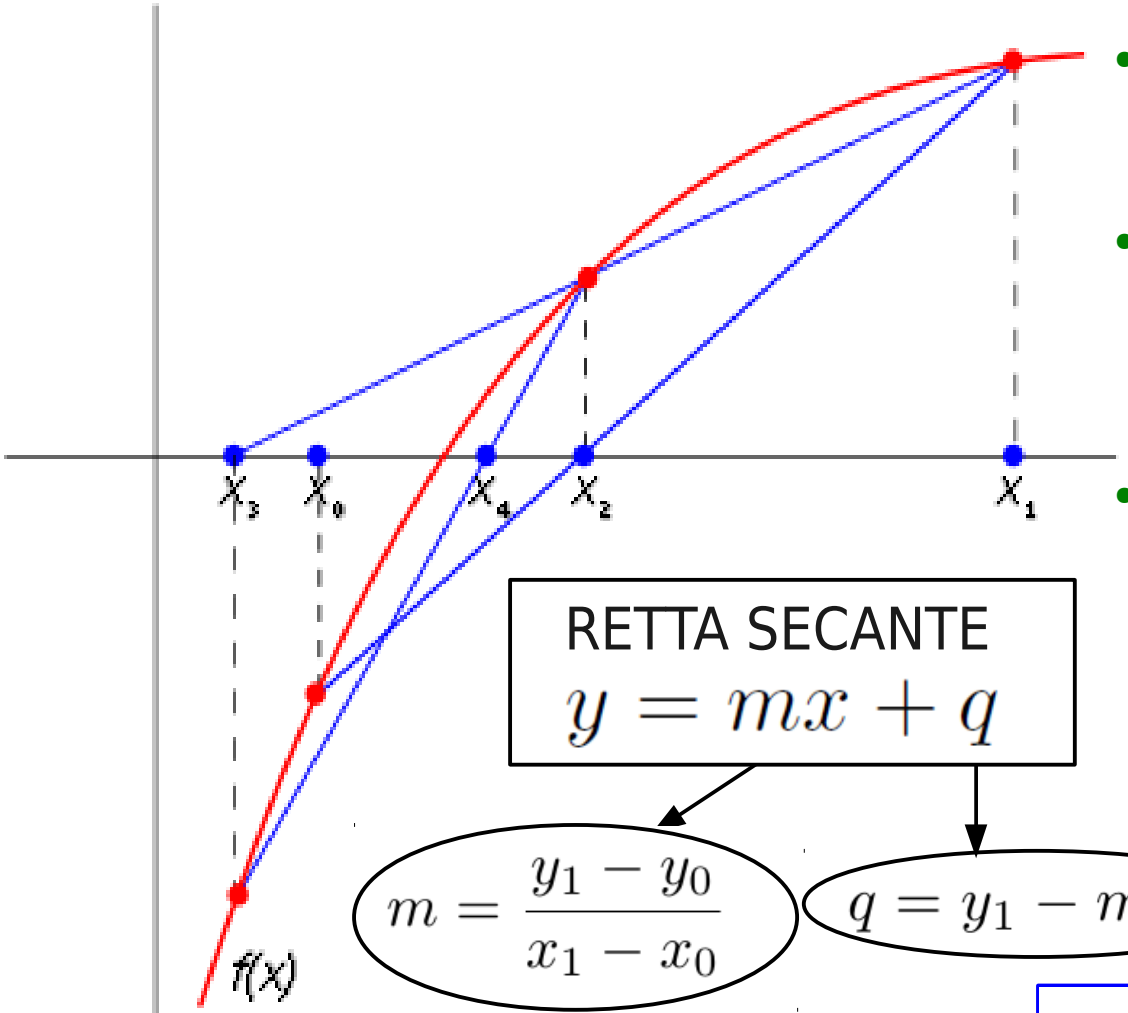
Si può implementare sotto forma di funzione ricorsiva, cioè che richiama se stessa

Per interrompere la ricorsione si mette un **if** di controllo




```
double x_m = 0.5 * (xMin + xMax) ;
if (xMax - xMin < precision) return x_m ;
if (f (xMin) * f (x_m) < 0)
  return zero_R_BIS (f, xMin, x_m, precision) ;
return zero_R_BIS (f, x_m, xMax, precision) ;
```

Il metodo delle secanti



- Si ricerca lo zero nell'intervallo $[x_0,x_1]$.
- Si trova lo zero della retta secante passante per i punti (x_0,y_0) e (x_1,y_1).
- Indichiamo con x_2 lo zero della secante e si ripete la procedura precedente usando i punti (x_2,y_2) e (x_1,y_1).

RETTA SECANTE

$$y = mx + q$$

$$m = \frac{y_1 - y_0}{x_1 - x_0} \qquad q = y_1 - mx_1$$

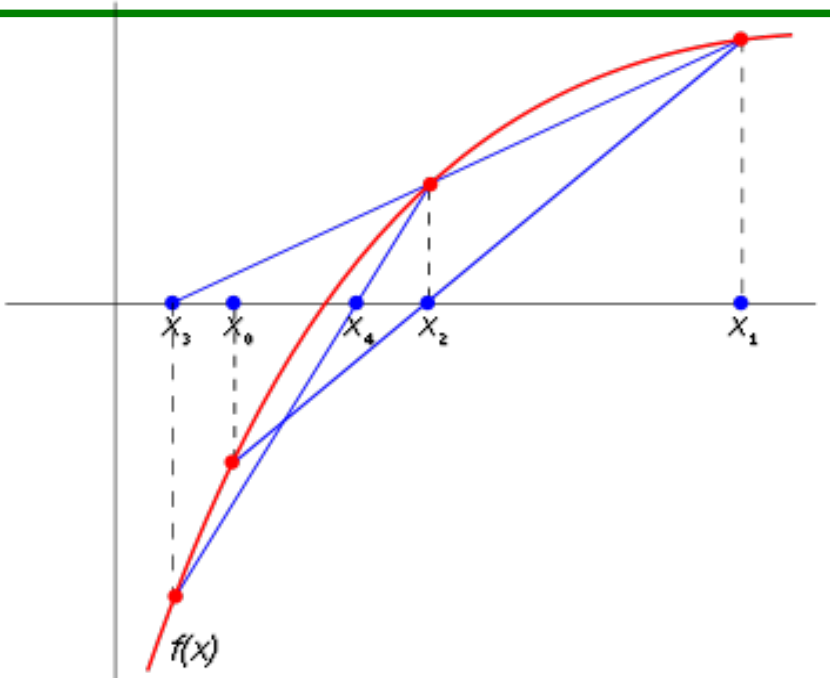
$$x_2 = x_1 - y_1 \frac{x_1 - x_0}{y_1 - y_0}$$

Il metodo delle secanti

Ci si avvicina al punto di zero sfruttando un'approssimazione della derivata della funzione

La derivata viene calcolata come la retta per gli estremi dell'intervallo di studio (corda)

Il nuovo punto – più vicino allo zero – è dato dall'intersezione della corda con l'asse x




```
double x3 = x2 - f(x2)
     * (x2 - x1) / (f(x2) - f(x1)) ;
if (fabs (x3 - x2) < prec)
  return x3 ;
return zero_R_SEC (f, x2, x3, prec) ;
```

La formula per il punto successivo è:

$$x_{n+1} = x_n - y_n \times \frac{x_n - x_{n-1}}{y_n - y_{n-1}}$$

punto di partenza

nella direzione data dal rapporto incrementale, verso opposto alla y

La funzione è ricorsiva

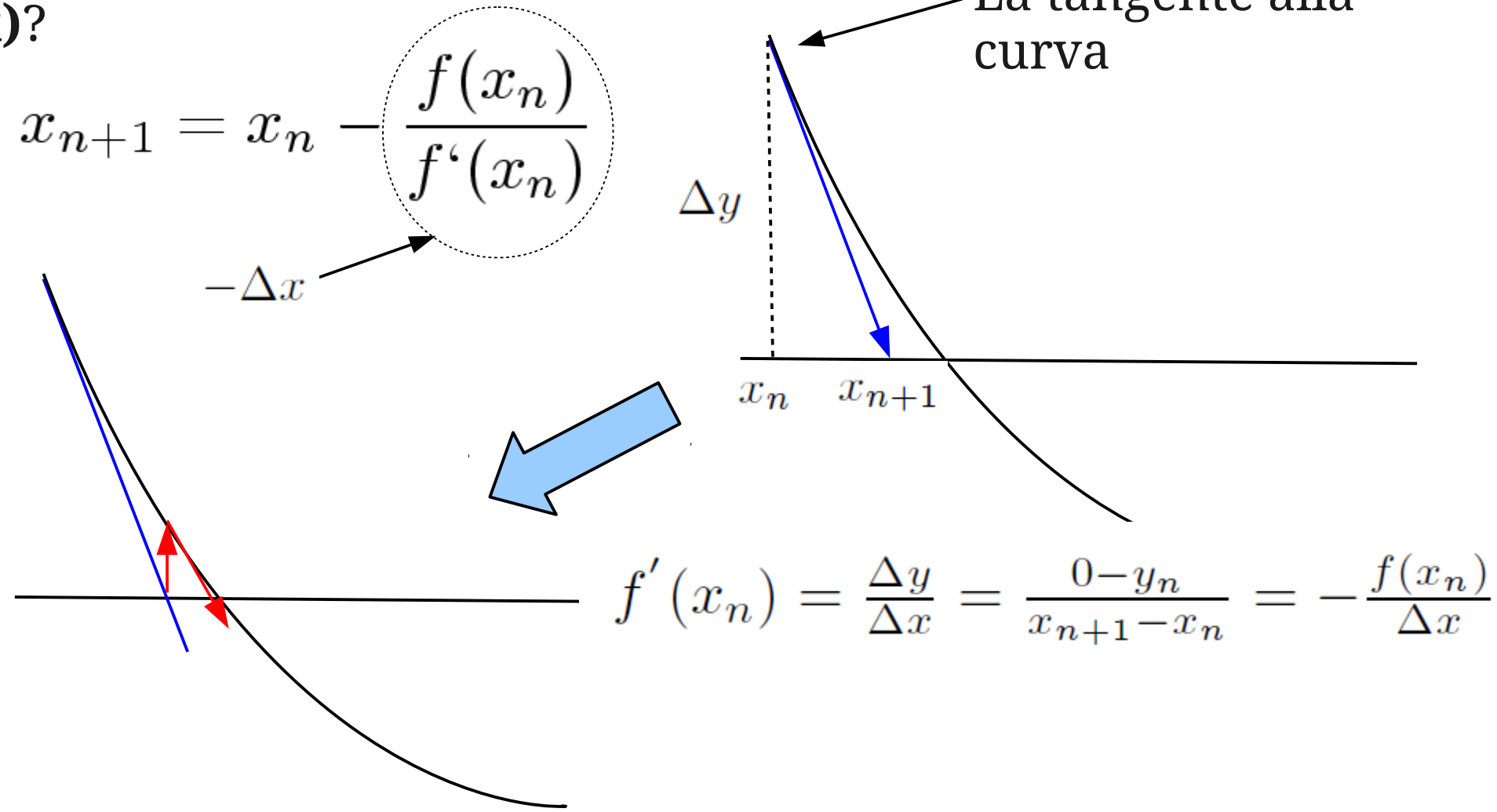
Il metodo delle tangenti (Newton)

Se si conosce in maniera analitica la funzione **f(x)** e la sua derivata prima **f'(x)**?

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

$-\Delta x$

La tangente alla curva



$$f'(x_n) = \frac{\Delta y}{\Delta x} = \frac{0 - y_n}{x_{n+1} - x_n} = -\frac{f(x_n)}{\Delta x}$$

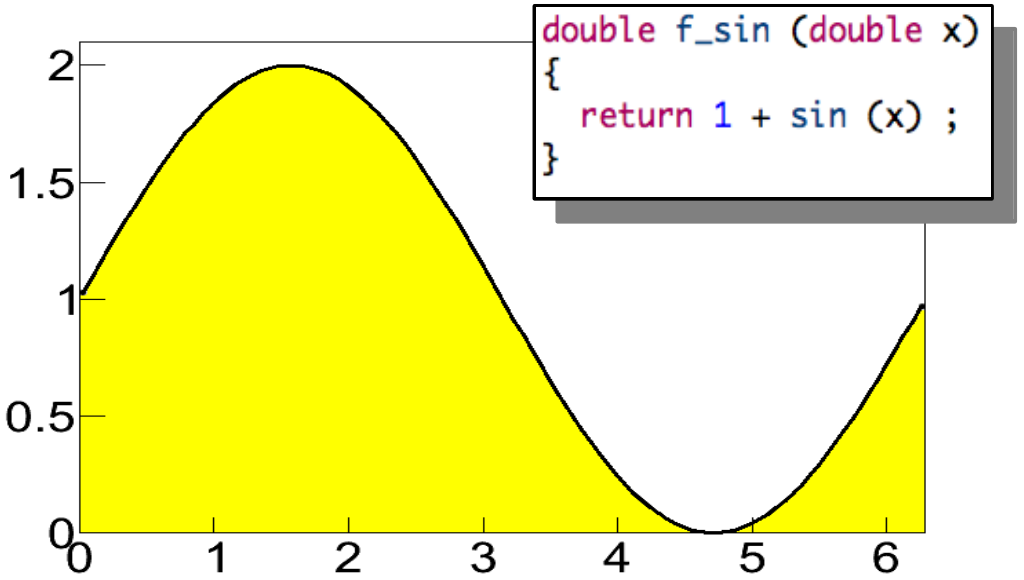
Esercizi

- **Esercizio 1:** scrivere un programma che calcoli trovi lo zero della funzione f(x) con le tecniche descritte, alla precisione di 1/10,000
  - … scrivendo gli algoritmi che trovano gli zeri come funzioni del C++
- **Esercizio 2:** modificare la funzione della bisezione in modo che calcoli una volta sola il valore di **f** in ogni punto considerato
- **Esercizio 3:** Calcolare la radice cubica di un numero qualsiasi col metodo della bisezione

Integrazione numerica di una funzione

Integrazione numerica

- Calcolare numericamente integrali di funzioni
- Assumiamo funzioni regolari (continue su un compatto)
- Consideriamo funzioni definite positive (l'integrale è definito a meno di una costante)
- Utilizziamo come esempio la funzione $f(x) = \sin(x) + 1$

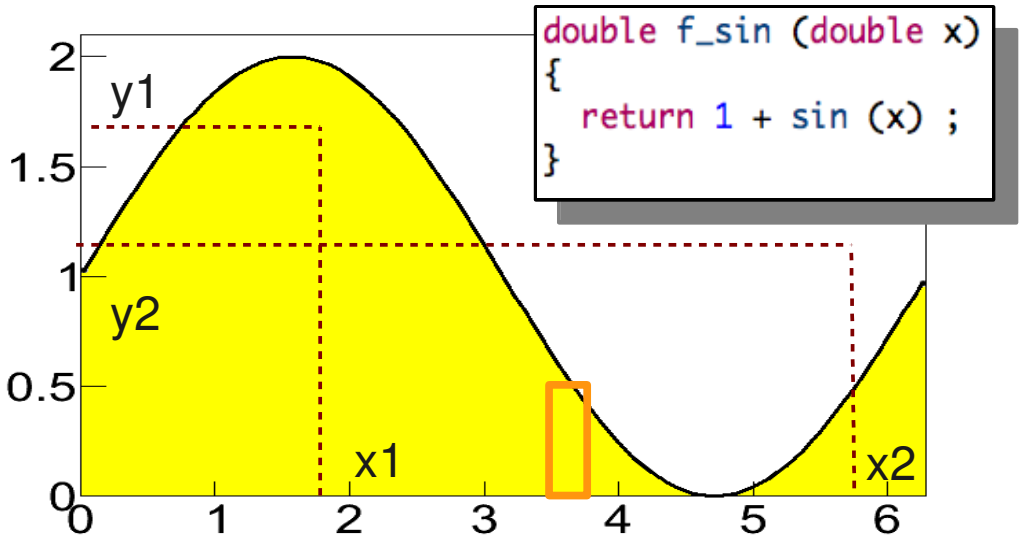


L'integrale di questa funzione, nell'intervallo [0, 1] è circa:

1.4597

Il metodo montecarlo

Contare il numero di punti che cascano al di sotto di una funzione, in proporzione al totale




```
double integral_MC (double f (double),
                    double xMin, double xMax,
                    double yMin, double yMax,
                    int max_num = 10000000)
{
  int N_integral = 0 ;
  for (int it = 0; it < max_num; ++it)
    {
      double x = rand_range (xMin, xMax) ;
      double y = rand_range (yMin, yMax) ;
      if (y < f (x)) ++N_integral ;
    }
  return (yMax - yMin) * (xMax - xMin) *
                    N_integral / max_num ;
}
```

Si generano uniformemente coppie di numeri casuali sull'asse x ed y.

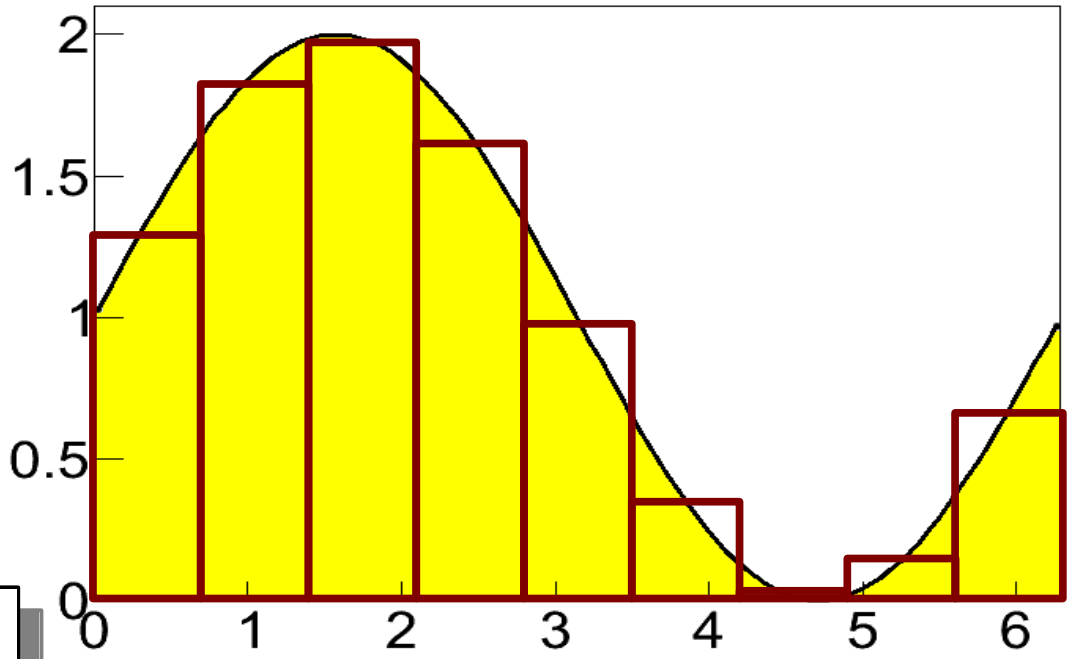
Se la coppia di punti si trova sotto la linea (x, f(x)) si incrementa il conto dei punti sotto la curva.

La frazione di x presi in un certo intervallo dipende dall'area di f(x) in quell'intervallo.

Il metodo dei rettangoli

Approssima l'integrale con la somma delle aree di rettangoli costruiti con il valore della funzione nel punto medio dell'intervallo.

La funzione è approssimata all'ordine zero.




```
double step = (xMax - xMin) / nBins ;
double x = xMin ;
double integral = 0 ;
while (x < xMax)
  {
    integral += f (0.5 * (x+x+step))
                  * step ;
    x += step ;
  }
return integral ;
```

L'intervallo è diviso in **nBins** passi.

La lunghezza di un passo è **step**, l'area del rettangolo è base x altezza.

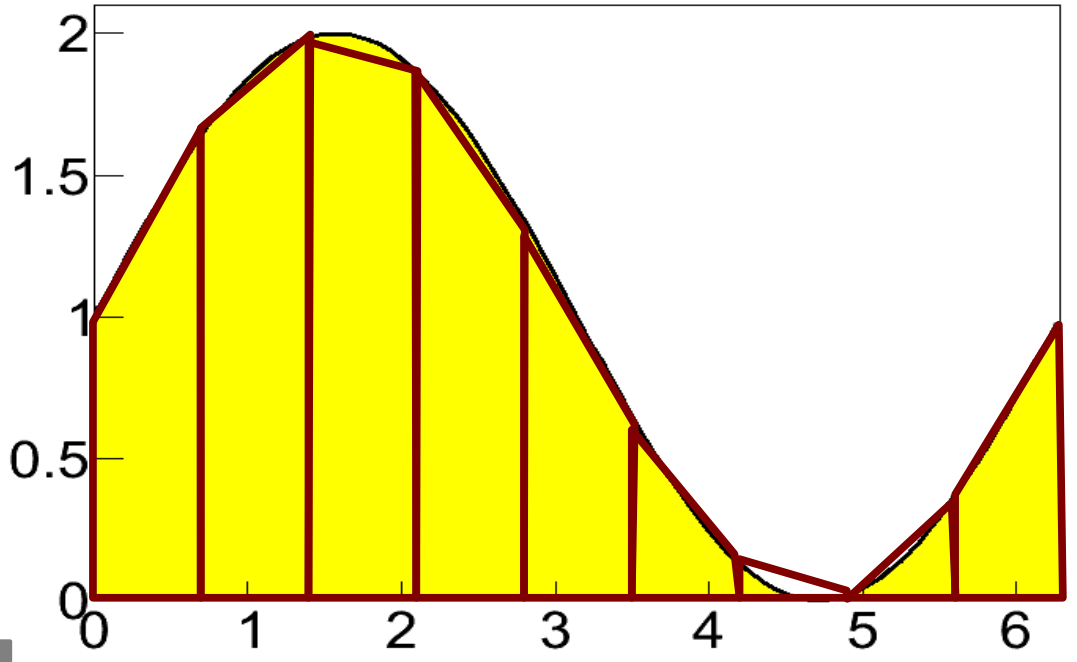
L'integrale è la somma delle aree.

Non è necessario conosce il massimo della funzione.

Il metodo dei trapezi

Approssima l'area della funzione con la somma delle aree di trapezi costruiti con due estremi della funzione.

La funzione è approssimata al primo ordine.




```
double step = (xMax - xMin) / nBins ;
double x = xMin ;
double integral = 0 ;
while (x < xMax)
  {
    integral += (f (x) + f(x+step))
                * 0.5 * step ;
    x += step ;
  }
return integral ;
```

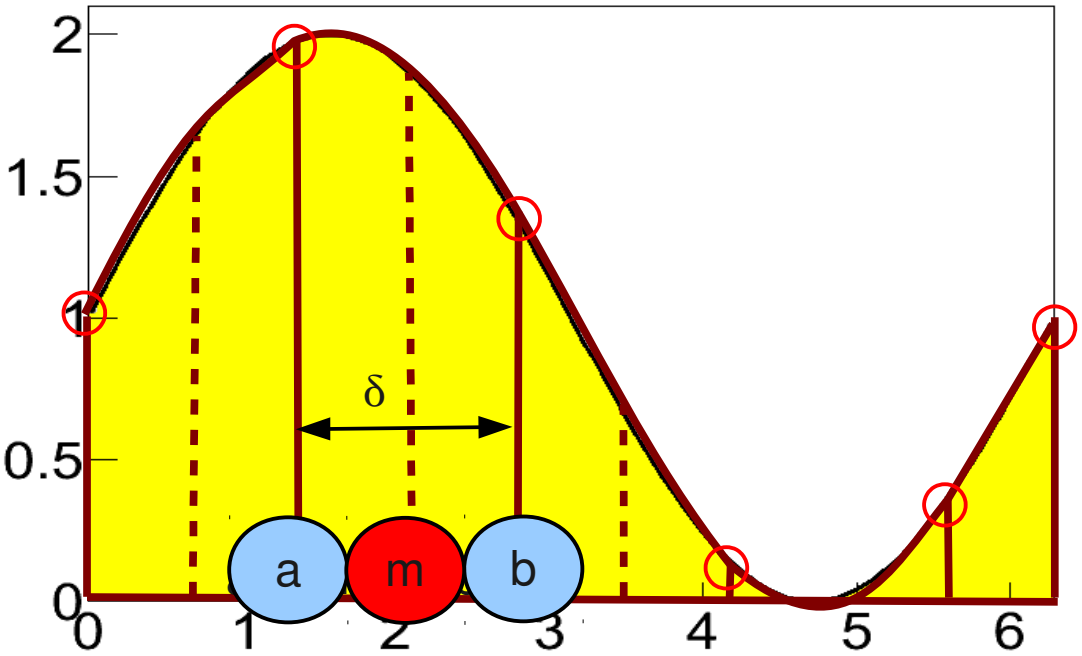
La singola area è la superficie del trapezio.

Il metodo di Simpson

Approssima l'area della funzione con la somma delle aree di parabole costruite con due estremi della funzione ed il loro punto medio.

La funzione è approssimata al secondo ordine.

```
double integral_SIM (double f (double),
                     double xMin, double xMax,
                     int nBins = 1000)
{
  double step = (xMax - xMin) / nBins ;
  double x = xMin ;
  double integral = 0 ;
  while (x < xMax)
    {
      integral += step * (f (x) + f(x + step) +
              4 * f (0.5 * (x + x + step))) / 6. ;
      x += step ;
    }
  return integral ;
}
```



La singola area è l'integrale della parabola per i tre punti, che risulta:

$$I = \frac{\delta}{6}\left(f(a) + f(b) + 4f(m) \right)$$

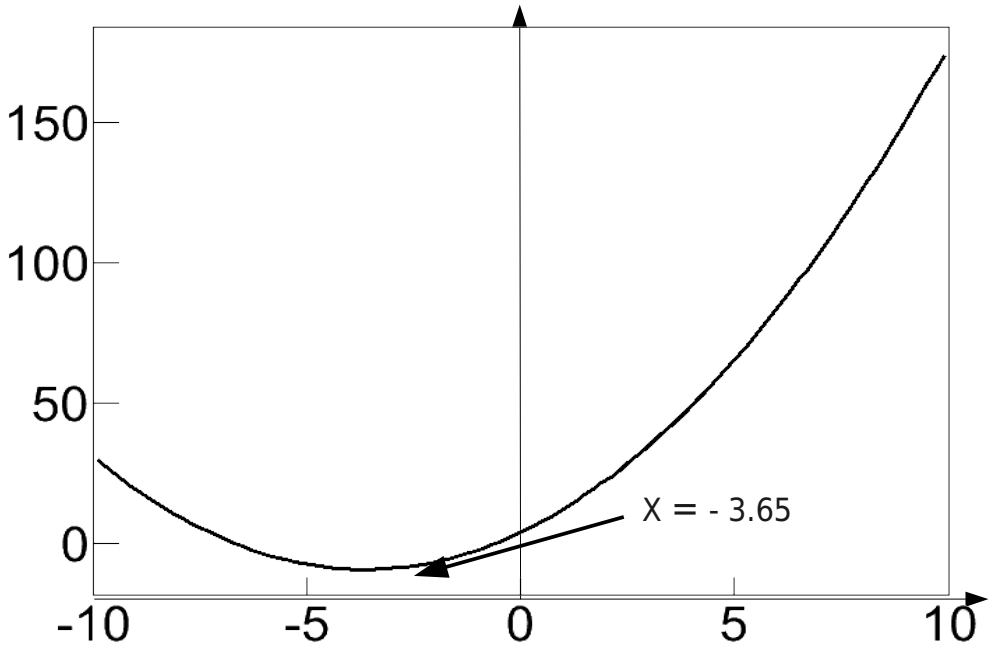
Esercizi

- **Esercizio 4:** scrivere un programma che calcoli l'integrale della funzione f(x) con le varie tecniche descritte, alla precisione di 1/10,000
  - ... scrivendo gli algoritmi che calcolano gli integrali come funzioni del C++
  - confrontare le prestazioni dei vari algoritmi al variare del numero di iterazioni effettuate (precisione, velocità)

Ricerca degli estremanti di una funzione

Estremanti di una funzione

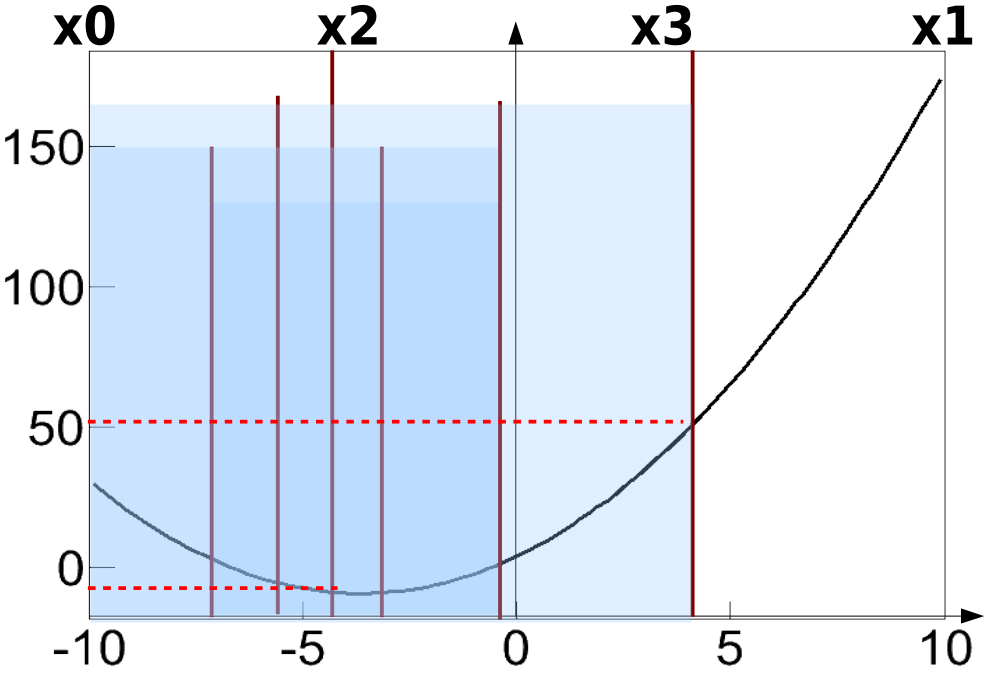
- Trovare numericamente estremanti di funzioni
- Assumiamo funzioni regolari (continue su un compatto)
- Consideriamo funzioni con un solo minimo nell'intervallo di studio.
- Utilizziamo come esempio la funzione $f(x) = x^2 + 7.3\,x + 4$



Il metodo della trisezione

Per trovare il minimo di una funzione servono abbastanza punti da capirne la pendenza.

L'intervallo si stringe eliminando il tratto dove il minimo di sicuro non c'è.




```
if (fabs (x1-x0) < prec)
  return 0.5 * (x1 + x0) ;
double x2 = x0 + 0.3 * (x1 - x0) ;
double x3 = x1 - 0.3 * (x1 - x0) ;
if (f(x2) > f(x3))
  return min_R_TRI (f, x2, x1, prec) ;
else
  return min_R_TRI (f, x3, x0, prec) ;
```

La funzione è ricorsiva.

A seconda di **f(x2)>f(x3)** si sceglie su che sotto-intervallo iterare.

Le differenze sono fatte con segno.

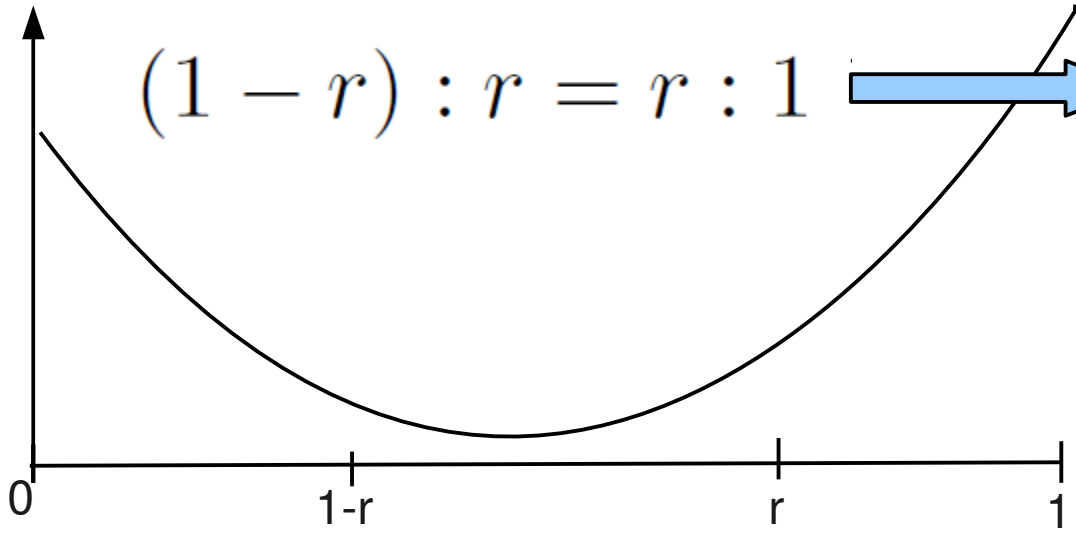
Il metodo della sezione aurea

Metodo “più intelligente” per tripartire l'intervallo in cui si cerca il minimo.

Si consideri per comodità l'intervallo [0,1], suddiviso in tre parti dipendenti dal valore di *r*.

Come per la trisezione, all'iterazione successiva si considera [0,r] se f(1-r)<f(r) viceversa si sceglie l'intervallo [1-r,1]

All'iterazione successiva si vuole “riciclare” uno dei punti considerati in quella precedente, perciò si impone la condizione:



$$(1-r) : r = r : 1 \Rightarrow r^2 + r - 1 = 0$$

r>0

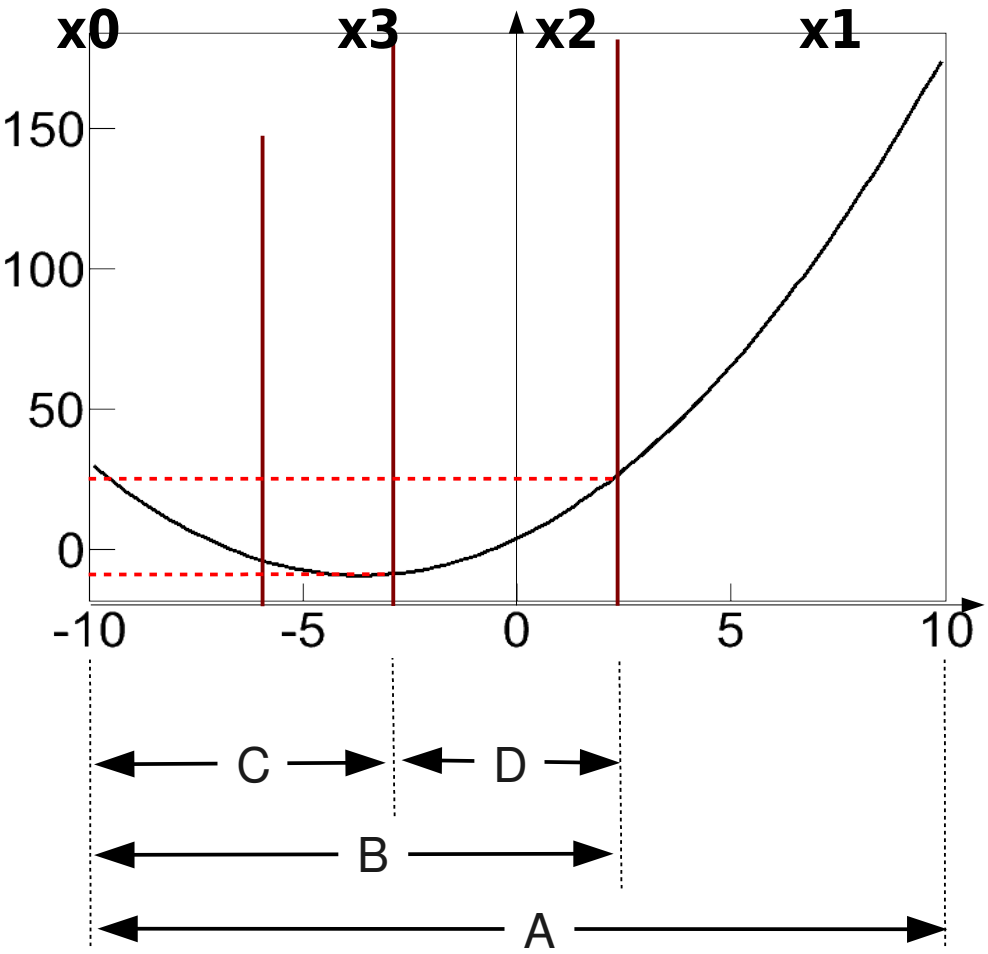
$$r = \frac{\sqrt{5}-1}{2} \backsimeq 0.618$$

Il metodo della sezione aurea

La velocità dipende dal tempo di calcolo del valore della funzione.

Riutilizzando uno dei punti generati in un ciclo successivo, si risparmia (molto) tempo.

Bisogna scegliere correttamente la dimensione dei sotto-intervalli.



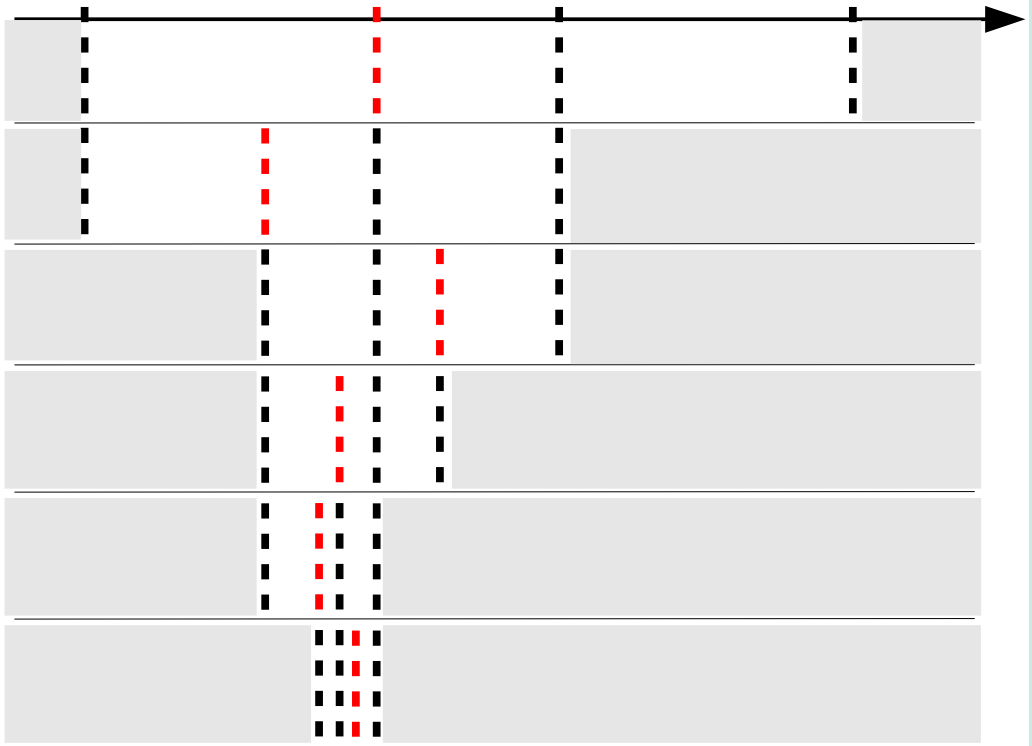

```
if (fabs (x1-x0) < prec)
  return 0.5 * (x1 + x0) ;
double x3 = x0 + (x2 - x0) * 0.618 ;
double f3 = f(x3) ;
if (f2 > f3)
  return min_R_GOL (f, x0, x2, x3, f3, prec) ;
else
  return min_R_GOL (f, x1, x3, x2, f2, prec) ;
```

$$\frac{B}{A} = \frac{C}{B}$$

B=0.618*A

Attenzione

Il C++ non vede la funzione, conosce solo i punti dove la valuta



ciclo	precisione
0	1 Δx
1	0.62 Δx
2	0.38 Δx
3	0.24 Δx
4	0.15 Δx
5	0.09 Δx

$$R = \frac{\sqrt{5}-1}{2} = 0.618$$

Esercizi

- **Esercizio 5:** trovare il minimo della funzione f(x) con il metodo della trisezione e della sezione aurea, con la precisione di 1/10,000
  - … scrivendo gli algoritmi che trovano gli estremanti come funzioni del C++
  - confrontare i due algoritmi: quale dei due è più veloce? (se non si vedono differenze, provate a rallentare la funzione f(x), introducendo il comando **sleep(1)** prima del **return**)