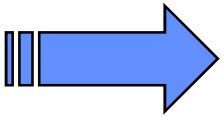
ALCUNI ESEMPI DI ALGORITMI DI CALCOLO NUMERICO

La risoluzione di modelli matematici complessi richiede spesso l'uso di metodi di ***calcolo approssimato*:**



calcolo "numerico"

Riferimenti Bibliografici:

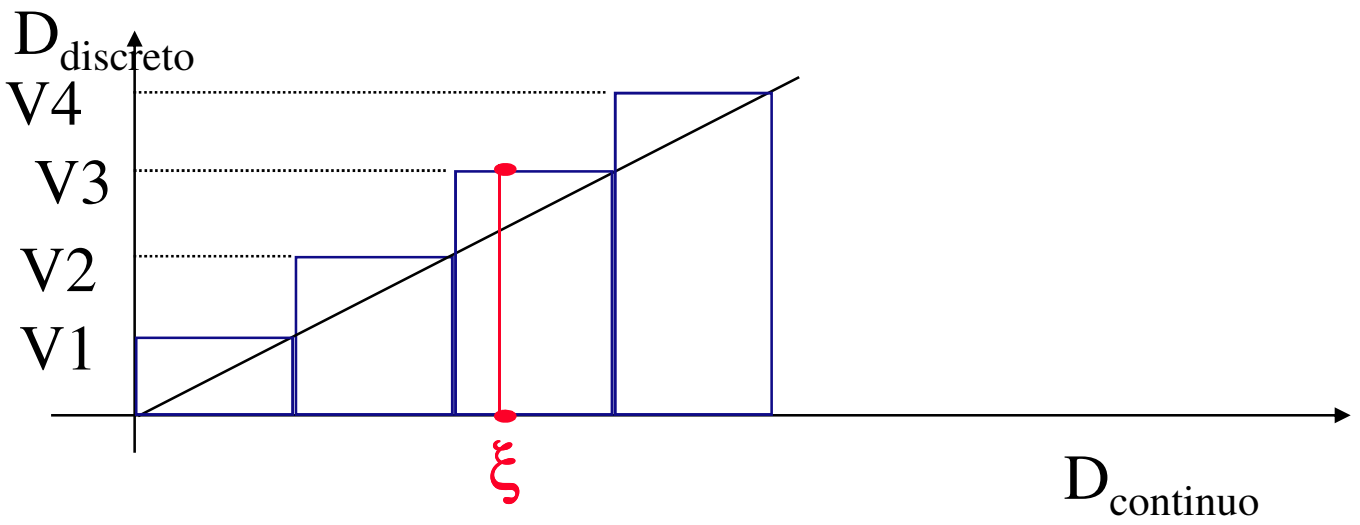
- L.M. Barone, E. Marinari, G. Organtini, F. Ricci-Tersenghi, ***Programmazione Scientifica***, Pearson Education, Milano, 2006
- G. Monegato, ***Fondamenti di Calcolo Numerico,*** Libreria Editrice Universitaria Levrotto&Bella, Torino

CALCOLO NUMERICO

- Si usa quando:
  - ***non*** esistono ***algoritmi esatti di risoluzione***, oppure
  - gli algoritmi esatti ***non sono efficienti*** (tempo, memoria): alto costo computazionale
- Soluzione di problemi attraverso ***algoritmi euristici o approssimanti***:
  - Determinano soluzioni "approssimate" del problema
- Ad esempio, si basano su tecniche del tipo:
  - ***discretizzazione*** di domini continui in domini discreti
  - ***approssimazioni successive*** => metodi ***iterativi***

DISCRETIZZAZIONE

- Valori di un dominio continuo vengono rappresentati da ***elementi di un dominio discreto***:



ξ viene rappresentato da V_3

Con quale approssimazione? Con quali effetti sui risultati successivi (propagazione dell'approssimazione)?

APPROSSIMAZIONI SUCCESSIVE

- valore iniziale V_0 (di primo tentativo)
  - formula iterativa => ottengo V_1
- valore di secondo tentativo V_1
  - formula iterativa => ottengo V_2
- valore di terzo tentativo V_2

  .....

L'algoritmo termina quando due valori consecutivi V_i e V_{i+1} sono ***"sufficientemente vicini"***:

- $| V_{i+1} - V_i | <= \varepsilon$ **scarto assoluto**
- $| (V_{i+1} - V_i) / V_{i+1} | <= \varepsilon$ **scarto relativo**

Il valore ε rappresenta la precisione desiderata e viene stabilito arbitrariamente

APPROSSIMAZIONI SUCCESSIVE

Note:

- Il ***valore di primo tentativo*** dovrebbe essere stimato nel modo migliore possibile *(conoscenza del problema, ordine di grandezza, segno, …)*
- Se ogni successiva approssimazione è più vicina alla risposta corretta e la differenza tende a calare ad ogni passo, l'iterazione converge (come capirlo su un numero finito di iterazioni?)
- Non tutte le iterazioni convergono (divergenti) o ***convergono in modo soddisfacente per essere utili***

ESEMPIO: CALCOLO DELLA RADICE QUADRATA

- Dato un numero reale positivo R, calcolare:

  Q = SQRT(R)

1) Inizializzazione: $Q_1 = R/2$; i=2

2) Formula Iterativa (passo i-esimo, i = 2, 3, ...)

$Q_i = (Q_{i-1} + R/Q_{i-1})/2$

3) Condizione di arresto:

se $|(Q_i - Q_{i-1})/Q_i)| <= \varepsilon$ or i >=MAXITER

altrimenti

i = i+1, torna al passo 2

- Infatti:

Supponiamo Q_1 circa uguale (≈) a Q_0, allora:

$(Q_1 - Q_0)^2 \approx 0$; $(Q_1^2 - 2Q_1 * Q_0 + Q_0^2) \approx 0$

$2Q_1 * Q_0 \approx Q_0^2 + R$ $(Q1^2 \approx R)$

$Q_1 \approx (Q_0 + R/Q_0)/2$

CALCOLO DELLA RADICE QUADRATA

Esempio:

- R = 9
- Q1 = 9/2 = 4.5
- Q2 = (4.5 + 9/4.5)/2 = (6.5)/2 = 3.25
- Q3=(3.25+9/(3.25))=(3.25+ 2.769)/2=(6.019)/2=3.0095
- .....

Radice Quadrata: ALGORITMO IN C

```
#include <stdio.h>
#include <math.h>
#define MAX 1000
#define EPS 0.00001

double radice_q(double R)
{  double Q, Qold;
   int i, fine=0;
   Qold=R/2;
   for (i=0; i<MAX && !fine; i++)
      {     Q=(Qold + R/Qold)/2;
            if (abs((Q-Qold)/Q)<EPS)
                        fine=1;
            else Qold=Q;
      }
   if (i>=MAX)    printf("Numero iterazioni eccessivo\n");
   return Q;

}
```

INVOCAZIONE

```
main()
{
  double V;
  printf("dammi il valore:  ");
  scanf("%lf", &V);
  printf("\nRisultato:%lf (valore atteso:
         %f)\n", radice_q(V), sqrt(V));

}
```

ESEMPIO: seno() DI UN ANGOLO

- Dato un angolo X espresso in radianti, calcolare:

S = seno(X)

- Sviluppo in Serie di Taylor:

$$S = X - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \ldots = \sum_{i=0}^{\infty} (-1)^i . \frac{X^{2i+1}}{(2i+1)!}$$

$$\Rightarrow S = \sum_{i=0}^{\infty} T_i$$

$$T_i = (-1)^i \frac{X^{2i+1}}{(2i+1)!} =$$

$$= -(-1)^{i-1} \frac{X^{2(i-1)+1}}{(2(i-1)+1)!} \cdot \frac{X^2}{(2i+1)2i} =$$

$$= -T_{i-1} \cdot \frac{X^2}{(2i+1)2i}$$

SENO DI UN ANGOLO

Si arresta lo sviluppo in serie dopo M termini,
con M (<= MAXITER) tale che:

$$se \qquad S = \sum_{i=0}^{M} (-1)^i \frac{X^{2i+1}}{(2i+1)!}$$

$$\frac{X^{2M+1}}{(2M+1)!} \leq \varepsilon$$

Passo iterativo:

- **Termine i-esimo:**

$T_i = (-1)^i \; X^{2i+1} / (2i+1)!$

$= -(-1)^{i-1} \; (X^{2(i-1)+1}/(2(i-1)+1)!) \; * \; X^2/((2i+1)2i)$

$= - X^2/((2i+1)2i)* T_{i-1}$

UNA POSSIBILE IMPLEMENTAZIONE

```
#include <stdio.h>
#include <math.h>
#define EPS 0.0000001
#define MAX 10000

double seno(double x){
   double T, Told, sum;
   int i, fine=0;
   Told=x; sum=x;
   for (i=1; i<MAX && !fine; i++) {
      T= -Told*x*x/((2*i+1)*2*i);
      sum = sum + T;
      if (abs(T-Told)<EPS) fine=1;
      else Told=T;
    }
   if (i==MAX) printf("troppe iterazioni\n");
   return sum;
}
```

INVOCAZIONE

Ad esempio:

```
main()
{  double v;
   printf("Dammi x:  ");
   scanf("%lf", &v);
   printf("Seno di %lf: %lf (valore atteso %lf)\n",
           v, seno(v), sin(v));
}
```

EQUAZIONI: SOLUZIONI APPROSSIMATE

- In problemi di ingegneria è necessario spesso trovare i valori delle radici di un'equazione

- Per equazioni lineari o quadratiche la soluzione è semplice *(esiste forma analitica chiusa)*:

  - $aX + b = 0 \Rightarrow \mathbf{X = -b/a}$

  - $aX^2 + bX + c = 0 \Rightarrow$
    $\mathbf{X_1} = -(b + SQRT(b^2-4ac))/2a$
    $\mathbf{X_2} = -(b - SQRT(b^2-4ac))/2a$

[**Hp:** $\Delta = (b^2-4ac) >= 0$]

CALCOLO DELLE RADICI DI UNA FUNZIONE

- Ricerca delle (eventuali) ***radici reali di una funzione*** che supporremo ***definita e continua*** in un certo intervallo dell'asse delle ascisse

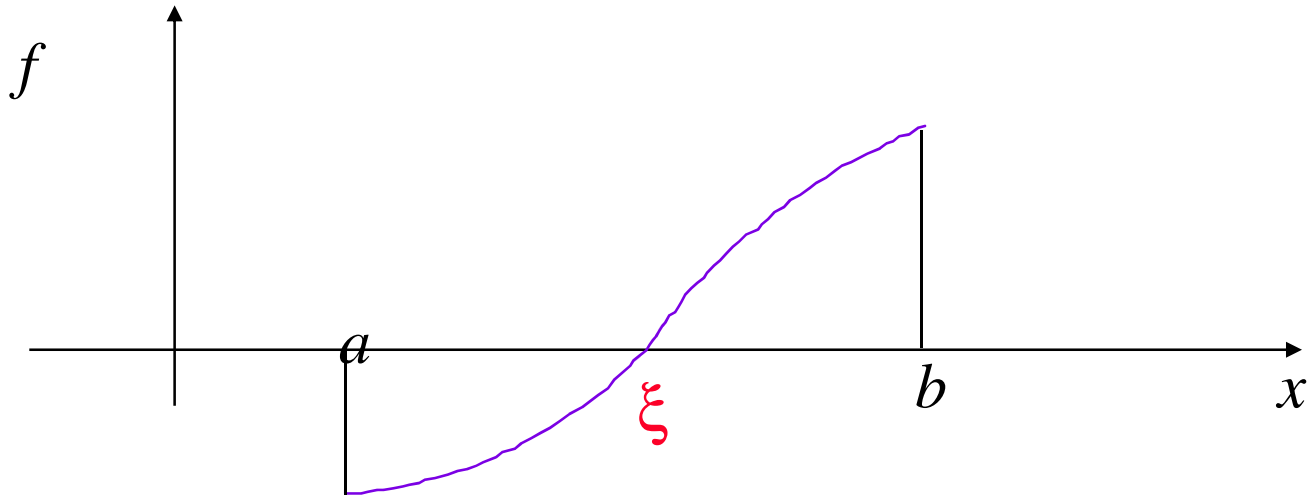
- La ***ricerca delle radici approssimate*** si compone di due fasi:

  1) ***separazione delle radici***, che consiste nel determinare degli intervalli [a, b] ***contenenti una sola radice***
  2) calcolo di un ***valore approssimato della radice*** e miglioramento di tale valore fini ad ottenere la precisione desiderata (tramite iterazioni successive)

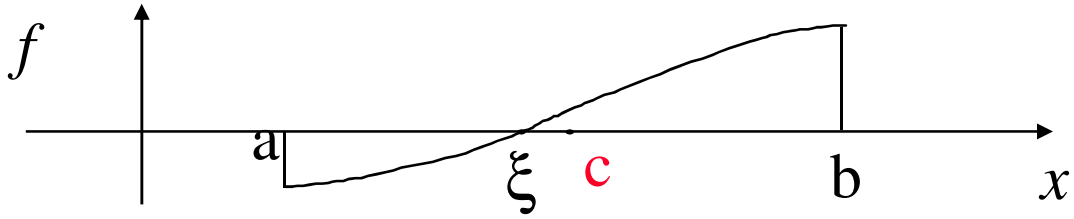
FUNZIONI CONTINUE: PROPRIETÀ

Proprietà

Se una funzione continua *f(x)* assume in due punti *a* e *b* ***valori di segno opposto***, ***esiste almeno un valore ξ*** (o un numero dispari di punti) ***compreso fra a e b in cui f(x)=0***



METODO DELLA BISEZIONE



- Dividiamo l'intervallo [a,b] a metà **c = (a + b)/2**

  -> calcoliamo **f(c)**
- Se f(c) = 0, la radice è trovata (procedimento terminato)
- Se f(c) > 0, trascuriamo l'intervallo destro [c, b], poniamo *b = c* e **ripetiamo la procedura nel nuovo intervallo [a, b]**
- Se f(c) < 0, trascuriamo l'intervallo sinistro [a, c], poniamo *a = c* e **ripetiamo la procedura nel nuovo intervallo [a, b]**
- Proseguendo in questo modo o si trova la radice esatta o si raggiunge un intervallo **[a, b] < 2^{ε}**

  dove *ε* è la precisione desiderata
- In pratica, **si approssima la funzione con la retta che passa per i punti (a, sign(f(a))), (b, sign(f(b)))**

METODO DELLA BISEZIONE

```
#include <stdio.h>
#include <math.h>
#define JMAX 1000
#define EPS 0.00001

double myfun(double z) {  // ad es. per myfun= z^3/2
   return z*z*z/2;   }

double bisez(double a, double b, double xacc)

{  int i, fine=0;
   double  Xa, Xb, Fa, Fb, Xm, Fm, tmp;

   if (a>b)     // estremi non ordinati
      {tmp=a; a=b;    b=tmp;} // scambio

   Fa=myfun(a); Fb=myfun(b);
   if (Fa*Fb>=0) { printf("intervallo non valido\n");
       exit();   }
   else {  Xa=a; Xb=b;     }
…
```

METODO DELLA BISEZIONE

```
…
for (i=0; i<JMAX && !fine; i++){
     Fa=myfun(Xa); Fb=myfun(Xb);
     Xm=(Xa+Xb)*0.5;
     Fm=myfun(Xm);
     if (Fm*Fa<0) Xb=Xm;
          else Xa=Xm;
     if ((abs(Xb-Xa)< xacc) || (Fm==0)) fine=1;
     }
if (fine) return Xm;
else
     { printf("troppe iterazioni\n");
       exit();  }
}
```

CALCOLO DELLE RADICI: CORDE, SECANTI, NEWTON-RAPHSON

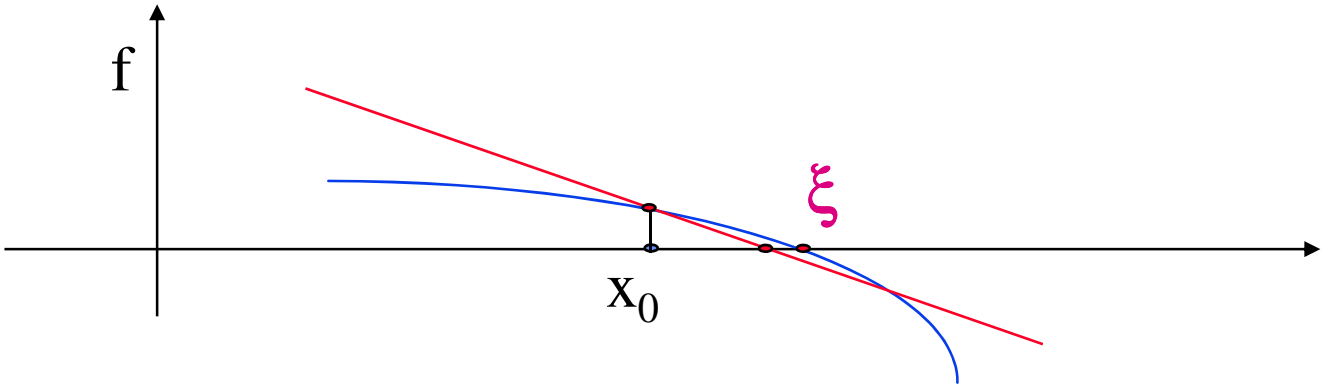
Idea di Base:

Detto ξ lo zero di f (appartenente all'intervallo [a,b]), sia x_0 una arbitraria approssimazione dello zero ξ nell'intervallo [a,b]:

- Si approssima la funzione con una retta passante per $(x_0, f(x_0))$ la cui equazione è

$$y = K_0(x - x_0) + f(x_0)$$

- L'intersezione tra la retta e l'asse delle ascisse dà origine alla nuova approssimazione x_1 della radice ξ



CALCOLO DELLE RADICI: CORDE, SECANTI, NEWTON-RAPHSON

Quindi, si ripete **iterativamente**:

Ad ogni iterazione *i* si ottiene una nuova approssimazione x_{i+1} dello zero, risolvendo l'equazione:

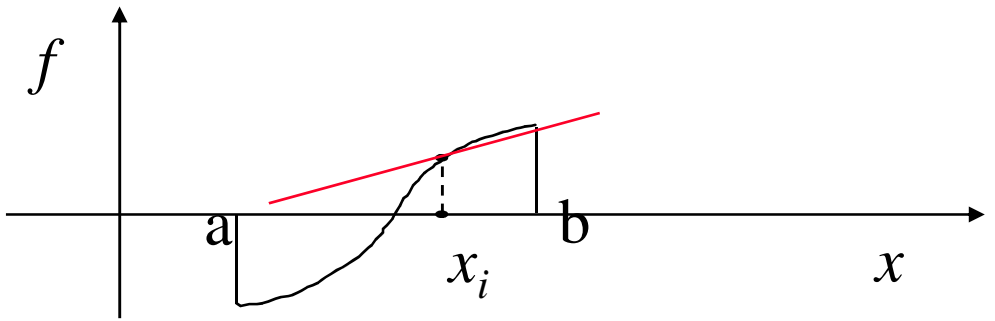
$$K_i(x-x_i) + f(x_i) = 0$$

dove x_i, soluzione approssimata dell'iterazione precedente

Questi metodi si basano tutti su ***approssimazioni successive della funzione f con rette*** del tipo:

$$y = K_i(x-x_i) + f(x_i)$$

Ogni metodo differisce dagli altri per la ***scelta del coefficiente angolare*** K_i



METODO DELLE CORDE (Regula Falsi)

Dati x_a, x_b, due valori reali appartenenti ad [a, b] tali che ***sign(f(x_a))!=sign(f(x_b))***

- Retta passante per $f(x_a)$, $f(x_b)$

  ⇒ **intersezione con l'asse *x*: x_1**

- Ad ogni iterazione, il coefficiente angolare K_i della retta che approssima f è calcolato come:

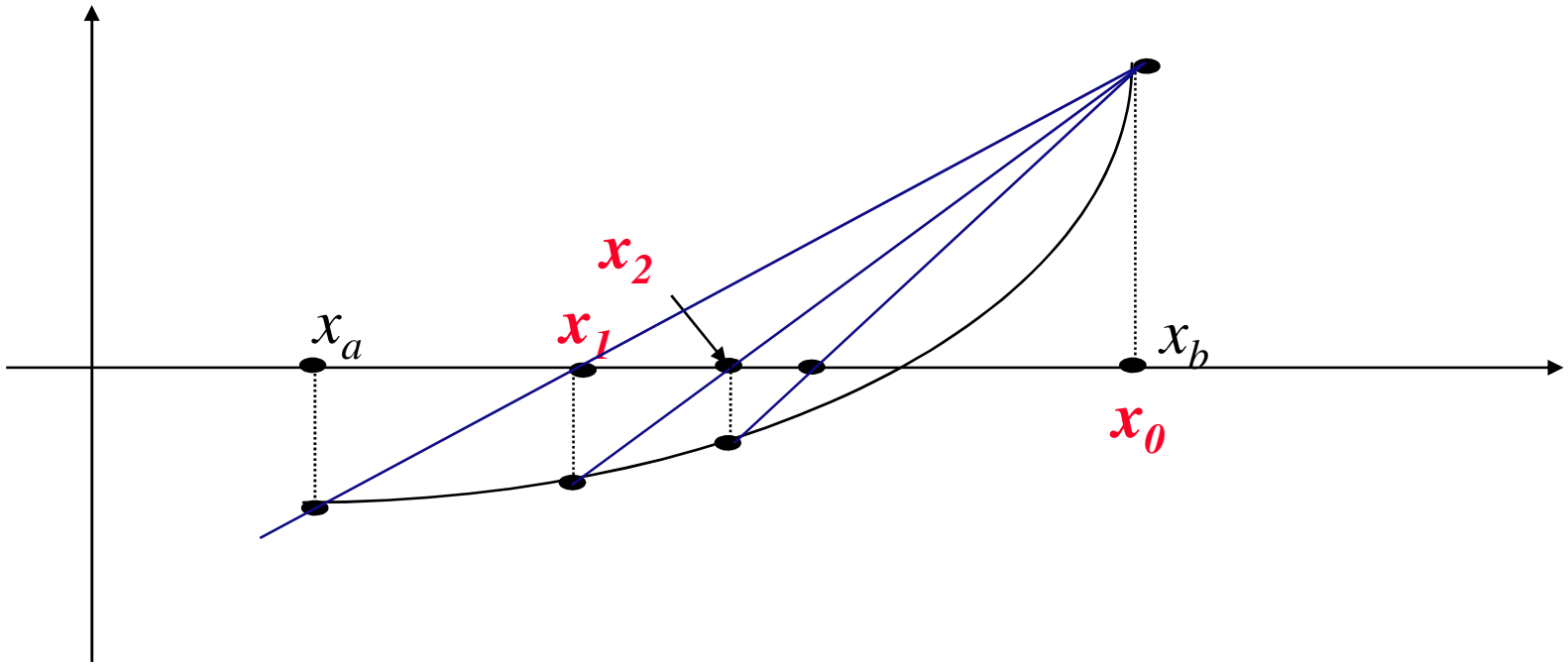
$$K_i = \frac{f(x_i) - f(x_{\lim})}{x_i - x_{\lim}}$$

dove:

- $x_{lim}=x_a$ se $sign(f(x_a))!=sign(f(x_i))$
- $x_{lim}=x_b$ se $sign(f(x_a))=sign(f(x_i))$

METODO DELLE CORDE (Regula Falsi)

Coefficiente angolare: $$K_i = \frac{f(x_i) - f(x_{\lim})}{x_i - x_{\lim}}$$



METODO DELLE CORDE (Regula Falsi)

```
…
#define JMAX 1000
#define EPS 0.00001

double myfun(double z) {  // ad es. per myfun= z^3/2
   return z*z*z/2;   }

double K (double xi, double xa, double xb)
{double lim, cond;
   cond=myfun(xi)*myfun(xa);
   if (cond<0) lim=xa; else lim=xb;
   return (myfun(xi)-myfun(lim))/(xi-lim);
}

double corde(double a, double b, double xacc)
{  int i, fine=0;
   double  Fa, Fb, Xi, Fi, Xold, tmp;
   if (a>b) /* estremi non ordinati */
      {tmp=a; a=b; b=tmp; }
```

METODO DELLE CORDE (Regula Falsi)

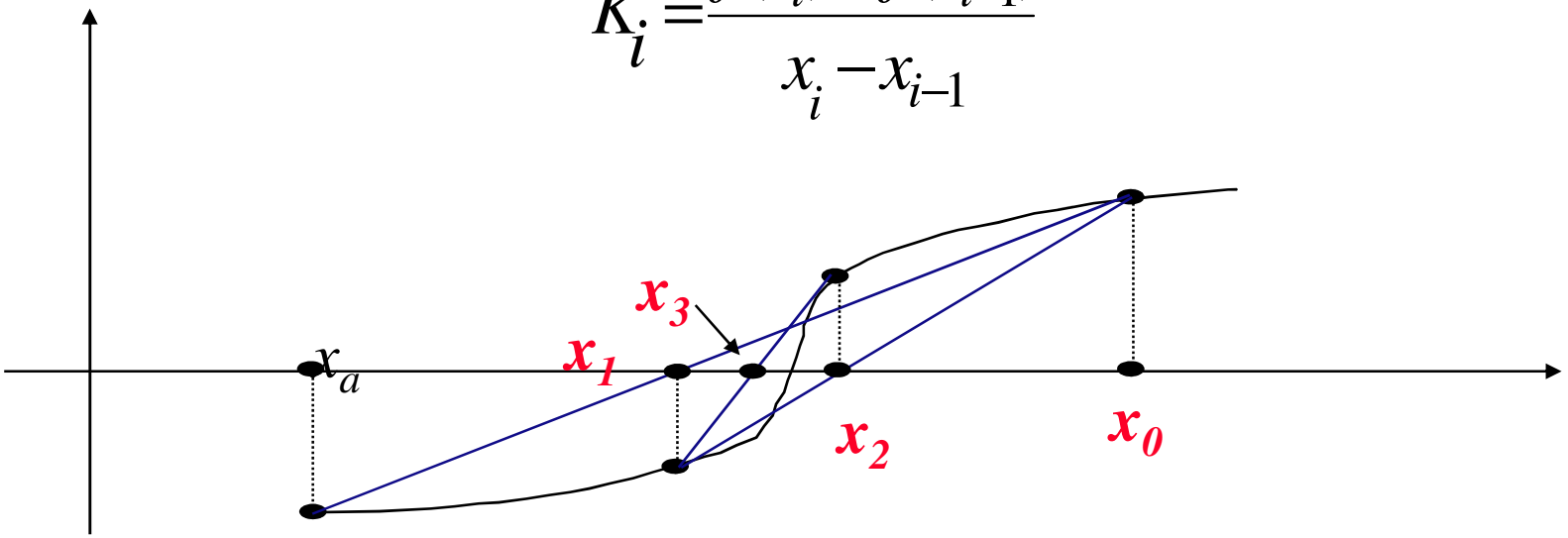
```
…
   Fa=myfun(a);
   Fb=myfun(b);

   if (Fa*Fb>=0)
      {printf("intervallo non valido\n");
       exit();
       }
   Xold=(a+b)*0.5; /*inizializzazione*/

   for (i=0; i<JMAX && !fine; i++) {
      Xi=-myfun(Xold)/K(Xold,a,b) + Xold;
      Fi=myfun(Xi);
      if ((abs(Xi-Xold)< xacc) || (Fi==0)) fine=1;
      else Xold=Xi; }

   if (fine) {printf("eseguite %d iterazioni\n",i); return Xi;}
   else { printf("troppe iterazioni\n");
      exit(); }
   }
```

METODO DELLE SECANTI

- Sia x_0, un valore reale appartenente ad [a, b]
- Si traccia la retta passante per
  - $f(x_0)$ e $f(x_a)$ se $sign(f(x_0)) != sign(f(x_a))$
  - $f(x_0)$ e $f(x_b)$ se $sign(f(x_0)) != sign(f(x_b))$

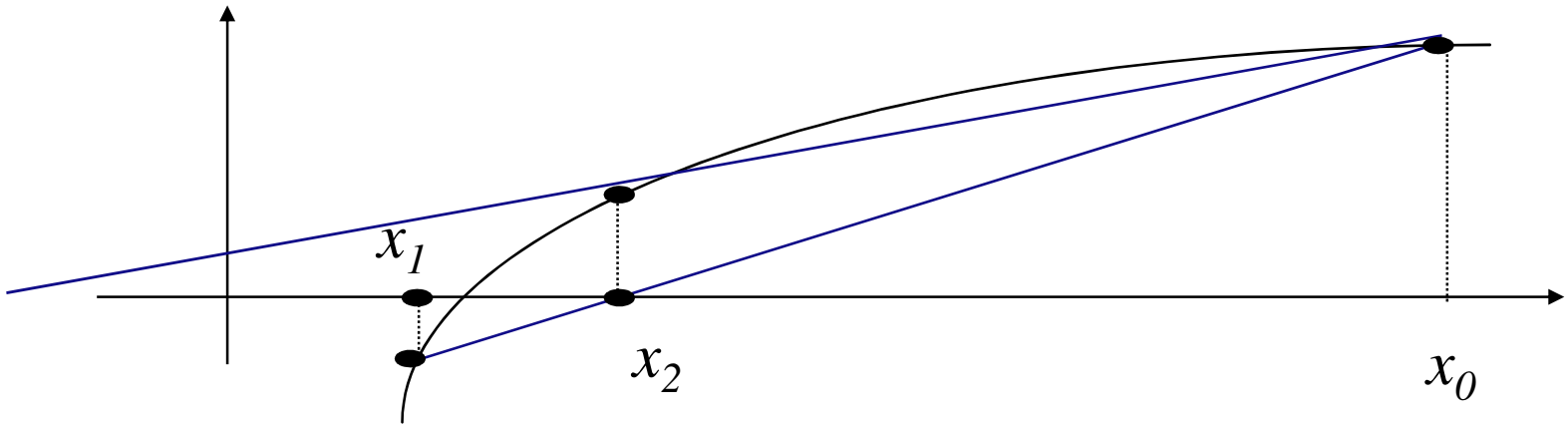
  individuo l'intersezione x_1 con l'asse x
- Ad ogni iterazione i, il **coefficiente angolare K_i** viene calcolato come:

$$K_i = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$



Metodo delle Secanti

- Diversamente dal metodo di bisezione, ***il metodo delle secanti può non convergere***:



in caso di convergenza, il metodo delle secanti è più efficiente di quello delle corde

METODO DELLE SECANTI

```
…
#define JMAX 1000
#define EPS 0.00001

double myfun(double z) {  // ad es. per myfun= z^3/2
      return z*z*z/2;  }

double Ks(double Xnew, double Xold) {
      return (myfun(Xnew)-myfun(Xold))/(Xnew-Xold);
}

double secanti(double a, double b, double xacc) {
      int i, fine=0;
      double  Fa, Fb, Xi, Fi, Xnew,Xold, tmp;
      if (a>b) /* estremi non ordinati */
            {tmp=a; a=b; b=tmp;}
      Fa=myfun(a);
      Fb=myfun(b);
      if (Fa*Fb>=0) {
            printf("intervallo non valido\n");
            exit(); }
```

METODO DELLE SECANTI

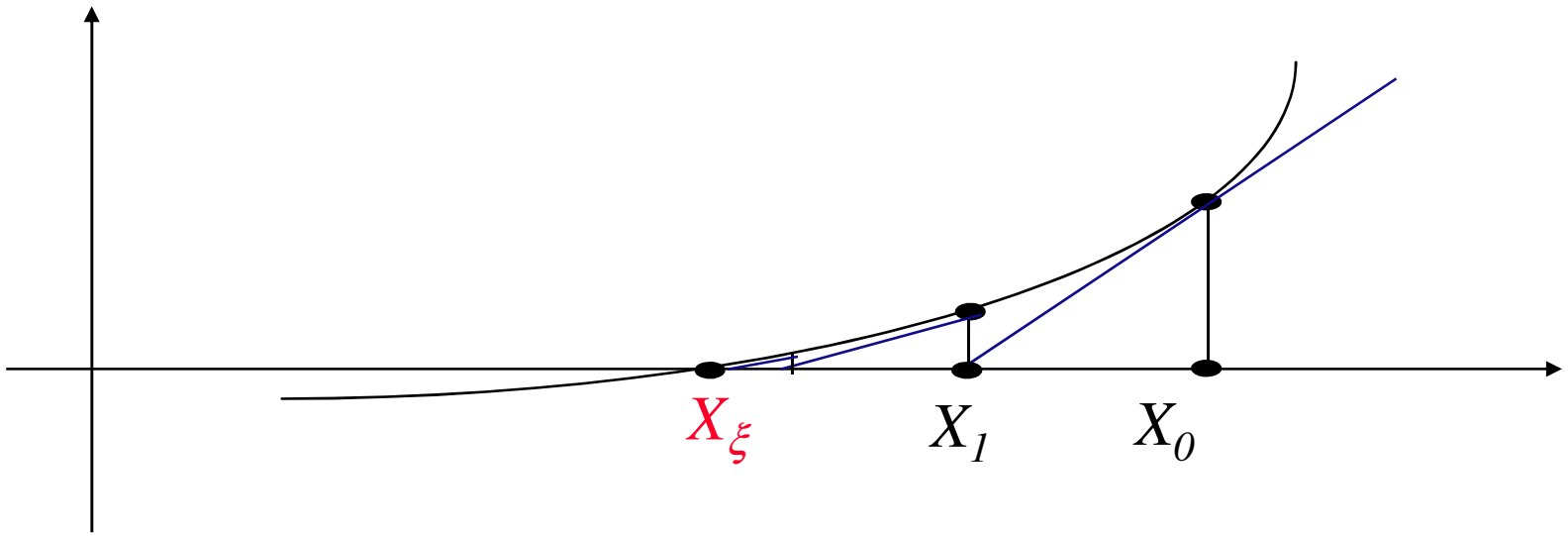
```
…
      Xold=a;
      Xnew=b;

for (i=0; i<JMAX && !fine; i++) {
      Xi=-myfun(Xold)/Ks(Xnew,Xold) + Xold;
      Fi=myfun(Xi);
      if ((abs(Xi-Xold)< xacc) || (Fi==0)) fine=1;
      else {Xold=Xnew;Xnew=Xi;}
      }
if (fine) {printf("eseguite %d iterazioni\n",i); return Xi;}
else {
      printf("troppe iterazioni\n");
      exit(); }
}
```

METODO DELLE TANGENTI (Newton-Raphson)

- Ad ogni iterazione i, il **coefficiente angolare K_i** della retta che approssima f è la ***derivata della funzione nel punto x_i***

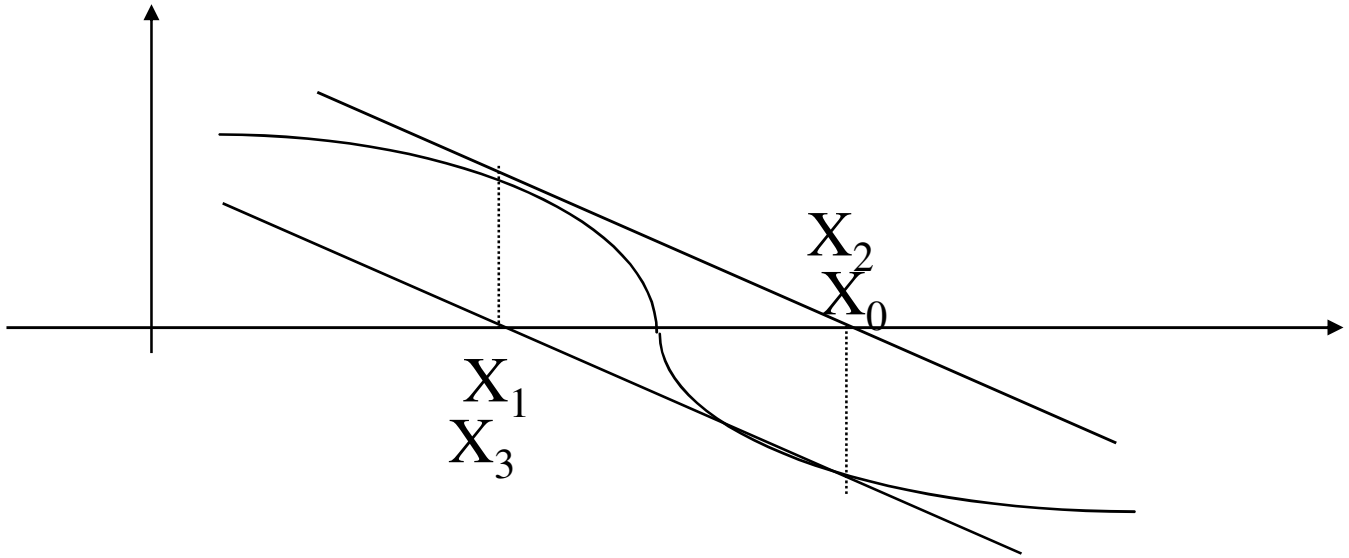
$$K_i = f'(x_i)$$

- Ovviamente è necessario che la funzione sia ***derivabile*** nell'intervallo considerato



METODO DELLE TANGENTI (Newton-Raphson)

- Anche il metodo delle tangenti ***può non convergere***:



- imponiamo anche la condizione che ***derivata seconda sia continua e mantenga segno costante*** nell'intervallo: $f(X)*f''(X) > 0$

➡ ***condizione sufficiente per la convergenza*** del metodo

METODO DELLE TANGENTI (Newton-Raphson)

```
…
#define JMAX 1000
#define EPS 0.00001
#define DELTA 0.001

double myfun(double z) {  // ad es. per myfun= z^3/2
      return z*z*z/2;  }

double Kt(double xi){
      return (myfun(xi+DELTA)-myfun(xi))/DELTA;}

double tangenti(double a, double b, double xacc) {
      int i, fine=0;
      double  Fa, Fb, Xi, Fi, Xold, tmp;
      if (a>b) /* estremi non ordinati */
            {tmp=a; a=b; b=tmp;}
      Fa=myfun(a);
      Fb=myfun(b);
      if (Fa*Fb>=0) {
            printf("intervallo non valido\n");
            exit(); }
```

METODO DELLE TANGENTI (Newton-Raphson)

```
…
        Xold=(a+b)*0.5; /*inizializz. */
       for (i=0; i<JMAX && !fine; i++) {
             Xi=-myfun(Xold)/Kt(Xold) + Xold;
             Fi=myfun(Xi);
             if ((abs(Xi-Xold)< xacc) || (Fi==0)) fine=1;
             else Xold=Xi;
             }

       if (fine) {
             printf("eseguite %d iterazioni\n",i);
             return Xi;}
       else {
             printf("troppe iterazioni\n");
             exit(); }

}
```