Algortimi e tecniche numeriche

- Calcolo di integrali definiti
- Ricerca degli zeri di una funzione
- Ricerca del minimo di una funzione
- Il metodo dei minimi quadrati
- Manipolazione di vettori e matrici
- Ordinamento dei vettori

Calcolo Numerico

Capita spesso in Fisica di affrontare problemi complessi per i quali non si sa impostare una soluzione analitica, o non si è in grado di risolvere analiticamente le equazioni corrispondenti. In questo caso le tecniche di calcolo numerico consentono di ottenere soluzioni con il grado di precisione richiesto.

Il calcolo numerico è una branchia piuttosto complessa della matematica. Noi ci limiteremo a mostrare alcune tecniche di base, che hanno più che altro utilità dimostrativa.

Ricordatevi, prima di affrontare un problema (serio) di calcolo numerico di consultare la letteratura (ad esempio il classico (”Numerical recipes in C” che trovate disponibile anche su internet) e di verificare se qualcuno che ha già risolto lo stesso problema vi può fornire una libreria di funzioni.

Calcolo di integrali definiti

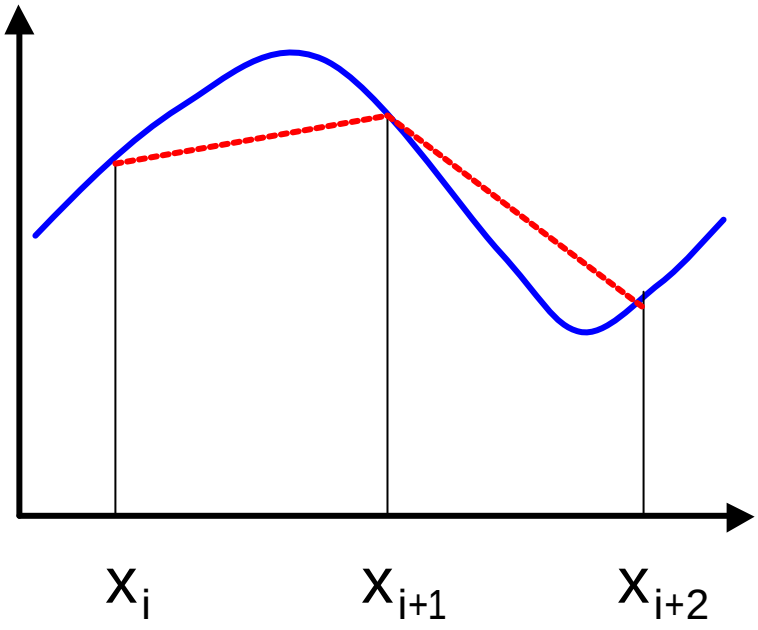
Il metodo più semplice per calcolare un integrale definito consiste nel suddividere l'intervallo di integrazione [a,b] in N parti uguali, delimitate dai punti:

$$x_i = x_0 + w*i; \quad i=0, 1,\ldots,N$$

dove $x_0 = a$ e $w = (b-a)/N$.

In ogni sottointervallo i sostituiremo l'integrale con l'area del trapezio delimitato dalla retta che congiunge i punti $(x_i, f(x_i))$ e $(x_{i+1}, f(x_{i+1}))$.

Questo metodo è esatto per una retta, e in generale l'errore è proporzionale a $w^3 f''(x)$.



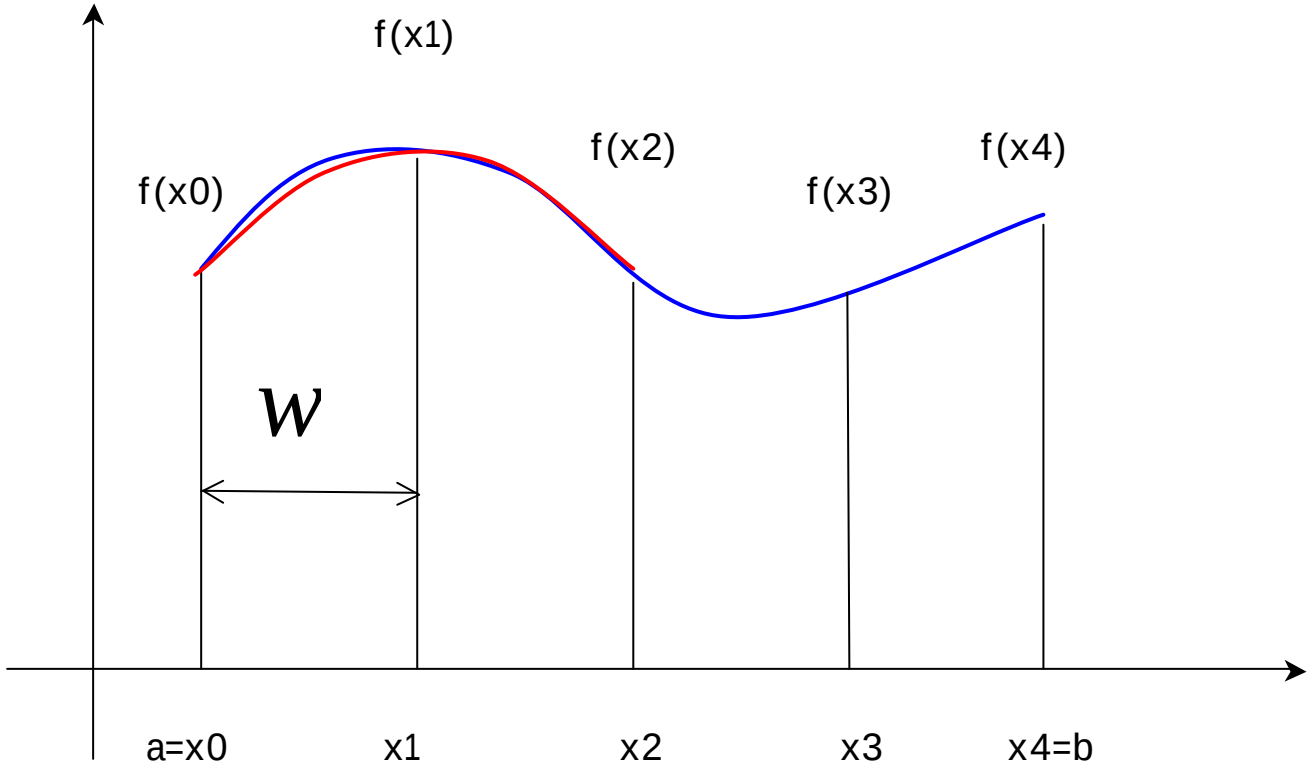
$$\int_a^b f(x)dx = \sum_0^{N-1} \frac{w}{2}\left(f(x_i)+f(x_{i+1})\right)$$

L'errore sul calcolo dell'integrale e proporzionale al cubo della larghezza dell'intervallo, per cui, come facilmente intuibile, intervalli piccoli significa errori piccoli (e, naturalmente, tempi di calcolo lunghi).

Un ulteriore miglioramento del metodo consiste nell'utilizzare un doppio intervallo (cioè tre punti) ed approssimare l'integrale nel doppio intervallo con l'area sotto la parabola che passa per i tre punti. La formula da usare, detta formula di Simpson, è:

$$\int_{x}^{x+2w} f(x)dx = \frac{w}{3}\left(f(x_i)+4f(x_{i+1})+f(x_{i+2})\right)$$

In questo caso l'errore è proporzionale a $w^5 f^{(4)}(x)$.



$$\int_{x}^{x+2w} f(x)dx = \frac{w}{3}\left(f(x_i)+4f(x_{i+1})+f(x_{i+2})\right)$$

```
double integr(double func(double x), double a, double b, int n) {
  double x, w = (b-a)/n;
  double integral=0;
  for (x=a; x<b; x+=w) {
    integral += w*(func(x) + func(x+w))/2;
  }
  return integral;
}

double interg_err(double func(double x), double a, double b, double err) {
  int n = 100;
  double rel, integral = integr(func,a,b,n);
  do {
    n*=2;
    double new = integr(func,a,b,n);
    rel = fabs(integral-new)/integral;
    integral = new;
  } while (rel > err);
  cout << "Numero intervalli = " << n << endl;
  return integral;
}
```

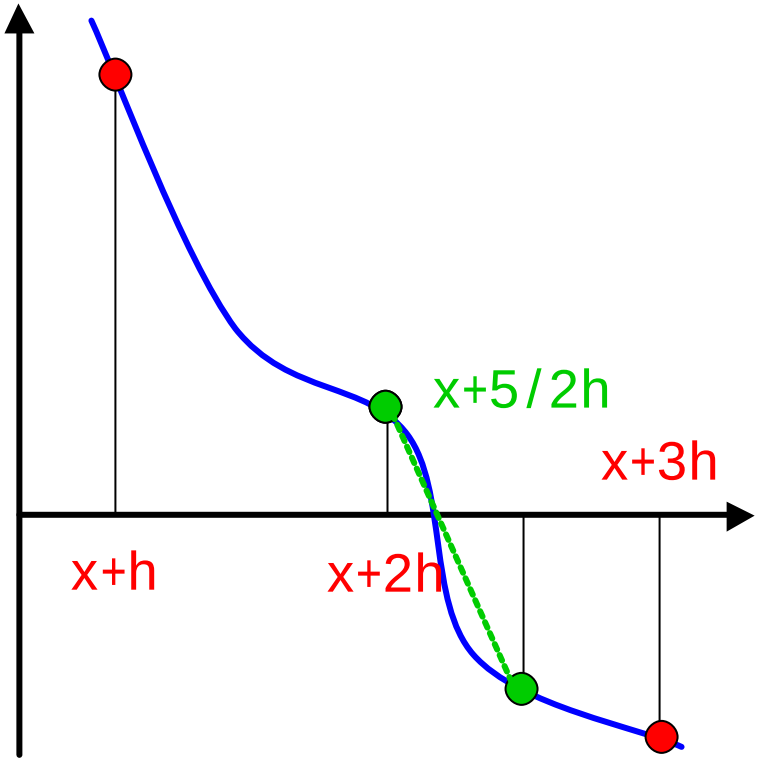
Calcolo con larghezza di intervallo fissata

Si dimezza la larghezza dell'intervallo fino ad ottenere una variazione relativa rispetto all'iterazione precedente al di sotto di un valore prefissato.

Zeri di una funzione

Se una funzione continua assume, agli estremi di un intervallo, valori discordi in segno si annulla in almeno un punto all'interno dell'intervallo.

Una tecnica di ricerca degli zeri consiste quindi nel fissare un punto di inizio della ricerca x_0 e un passo di campionamento h; si calcola la funzione nei punti x_0, x_1, con $x_1 = x_0 + h$. Se i valori $f(x_0)$ e $f(x_1)$ sono concordi si passa all' intervallo $[x_1, x_2]$ e così via fino a quando si trovano valori discordi. A questo punto si cambia segno al passo di campionamento e lo si dimezza. L'iterazione continua fino a che il passo di campionamento non scende sotto un valore prefissato.



La miglior stima dello zero è data dallo zero della retta passante per gli estremi dell'intervallo

Osservazioni

- Il metodo funziona solo con funzioni continue e in intervalli limitati.
- La condizione di cambio segno è sufficiente ma non necessaria. Diventa necessaria solo nel caso in cui la funzione coincida con la spezzata passante per gli estremi degli intervalli.
- Di fatto il metodo funziona solo a patto che l'intervallo di campionamento sia sufficientemente piccolo da rendere accettabile l'approssimazione rettilinea.

Osservazioni su metodo della bisezione

Quello che abbiamo appena accennato è detto metodo di bisezione. Per permettere un' implementazione efficiente si devono fare alcune osservazioni e controlli sulla convergenza del metodo.

Uno zero (la radice) di una funzione si dice "bracketed" nell'intervallo (a,b) se f(a) e f(b) hanno segno opposto. Il metodo della ricerca di tale intervallo è detto bracketing (root bracketing)

Se abbiamo una funzione continua sappiamo che esiste almeno uno zero nell'intervallo determinato; se la funzione non è continua, ma è limitata, invece di uno zero possiamo trovare un punto di discontinuità.

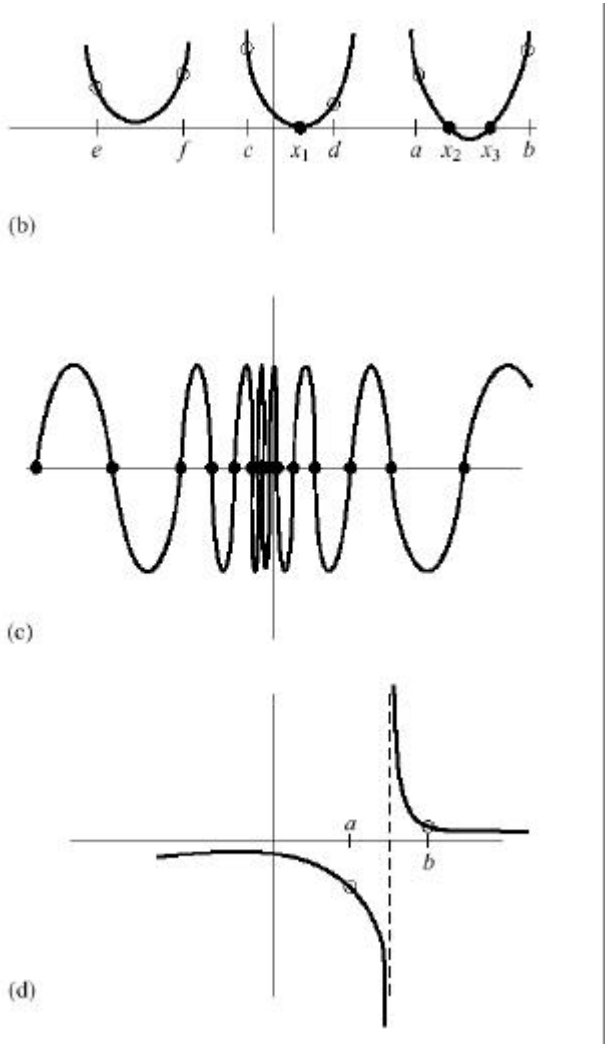
Solo per funzioni con singolarità c'è la possibilità che uno zero non sia nell'intervalllo trovato con il bracketing.

Osservazioni su metodo della bisezione

Bracketing non riuscito di zeri

Numero infinito di zeri. Caso patologico per la ricerca

Funzione con singolarità



Implementazione

Una buona implementazione dovrebbe prevedere tutti i casi possibili di applicazione.

Nel caso di ricerca degli zeri di una funzione è difficile controllare tutti i casi patologici. Considerare la funzione come una scatola nera e non fare alcuna considerazione su di essa, può portare a volte a risultati completamente errati (come per esempio nel caso di singolarità)

Quindi le considerazioni seguenti si applicano a funzioni "ragionevoli", sta all'utente poi verificare i risultati

- dobbiamo per prima cosa controllare che l'intervallo scelto abbia le caratteristiche richieste.

- controllare la convergenza del metodo

Implementazione

Descriviamo l'algoritmo in un modo più semplice per permettere una implementazione più efficiente.

Fino a che l'intervallo di campionamento è > accuratezza

1. Fissato l'intervallo di partenza (x1,x2) con f(x1)*f(x2) <0.
2. Prendiamo il punto medio xmed=(x1+x2)/2
3. Verifichiamo in quale dei due intervalli (x1,xmed) e (xmed,x2) si ha il cambiamento di segno, si prende in considerazione questo nuovo intervallo e si torna al punto 1.
4. Alla fine dell'iterazioni (cioè quando si è raggiunta l'accuratezza richiesta) il punto cercato è dato da xmed.

Convergenza

Ad ogni iterazione l'intervallo che contiene lo zero è dimezzato. Dopo n iterazioni si è quindi raggiunta una dimensione dell'intervallo pari a Δ_n , così si può determinare in anticipo il numero di iterazioni che ci permettono di raggiungere una determinata accuratezza.

$$n = \log_2 \frac{\Delta_0}{\Delta_n}$$

dove Δ_0 è l'intervallo iniziale. Quindi fissati tutti i parametri il metodo di bisezione converge in n passi: se l'intervallo di partenza contiene più zeri l'agolritmo converge ad uno di questi e se la funzione ha delle singolarità, l'algoritmo converge ad esse.

Convergenza -applicaizoni

Tenendo sempre presente che il computer lavora con un numero finito di bit e quindi con precisione finita, resta da determinare un controllo sulla convergenza “pratica”.

Se poniamo il nostro limite di accuratezza a 10^{-6} questo valore è più che realistico se lo zero si trova in un intorno di 1, ma diventa irraggiungibile se la radice ha un valore intorno a 10^{26}.

Dato che una tolleranza relativa non avrebbe senso per zeri vicino all’origine, si può pensare di limitare il numero di iterazioni.

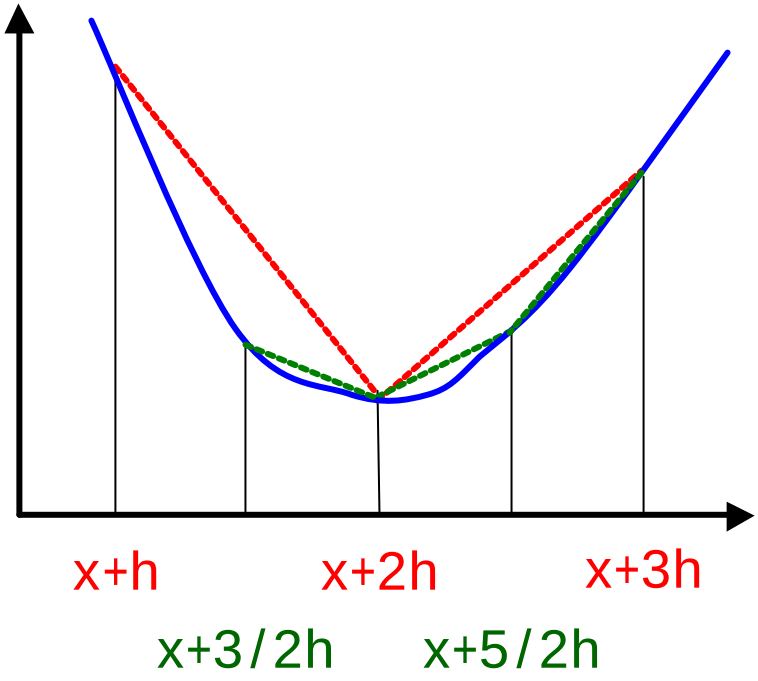
Tutte queste considerazioni permettono di scrivere il seguente codice

(Adattato da Numerical Recipes cap. 9)

```
#define JMAX 40
double rtbis(double func(double), double x1, double
x2, double xacc) {
double dx,f,fmid,xmid,rtb;
  f = func(x1);
  fmid = func(x2);
  if (f*fmid >= 0.0 ) {
    cout << "Root must be bracketed !" << endl;
    return 0.0 ;
  }
  // Orient the search so that f >0 lies at x+dx
  if (f <0.0)  {
    dx= x2-x1;
    rtb =x1;
  }
  else {
    dx= x1-x2;
    rtb =x2;
  }
for (int j=0; j< JMAX;j++){
    dx*=0.5;
    xmid= rtb+dx;
    fmid=func(xmid ); //Bisection loop.
    if (fmid <= 0.0) rtb=xmid;
    if (fabs(dx) < xacc || fmid == 0.0)
            return rtb;
     }
   cout << " Too many bisections " << endl;
  return 0.0;
}
```

Minimizzazione di funzioni

Un metodo per la ricerca del minimo di una funzione in casi non patologici è il seguente: si fissa un punto di inizio della ricerca x_0 e un passo di campionamento h; si calcola la pendenza della retta congiungente i punti della funzione agli estremi dell' intervallo $[x_0, x_1]$, con $x_1 = x_0 + h$. Si passa quindi all' intervallo $[x_1, x_2]$ e così via fino a quando la pendenza cambia segno. A questo punto si cambia segno al passo di campionamento e lo si dimezza. L'iterazione continua fino a che il passo di campionamento non scende sotto un valore prefissato.

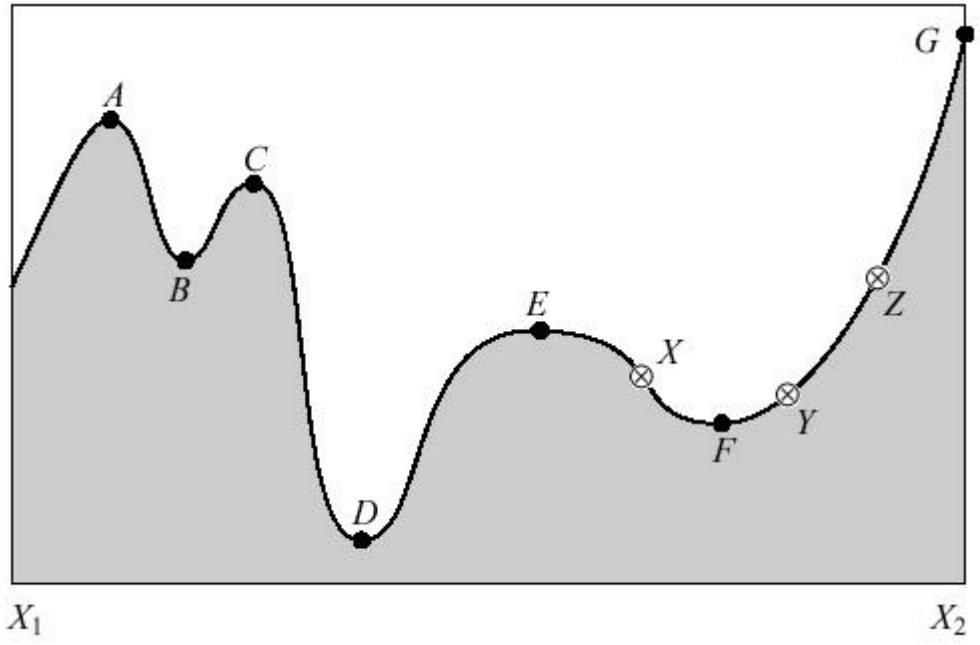


La miglior stima del minimo è x_j se $[x_{j-1},x_j]$ e $[x_j,x_{j+1}]$ sono gli ultimi intervalli in cui c'è stato un cambio di pendenza.

Osservazioni

Se osserviamo la funzione qui sotto, abbiamo massimi locali (A,C,E), minimi locali (B,F) e massimi e minimi globali (G e D).

Il massimo globale G è agli estremi dell'intervallo e la derivata non si annulla, questo può essere una difficoltà per molti algoritmi



I punti X,Y,Z identificano un intervallo di bracketing del minimo F, perchè Y ha un valore più basso di X,Z.

Bracketing di un minimo

Nel metodo di bisezione per trovare gli zeri di una funzione, abbiamo visto che uno zero è localizzato dagli estremi di un intervallo dove la funzione assume valori con segno opposto.

Per identificare la posizione di un minimo (massimo) occorre una terna di punti tali che per esempio se $a<b<c$

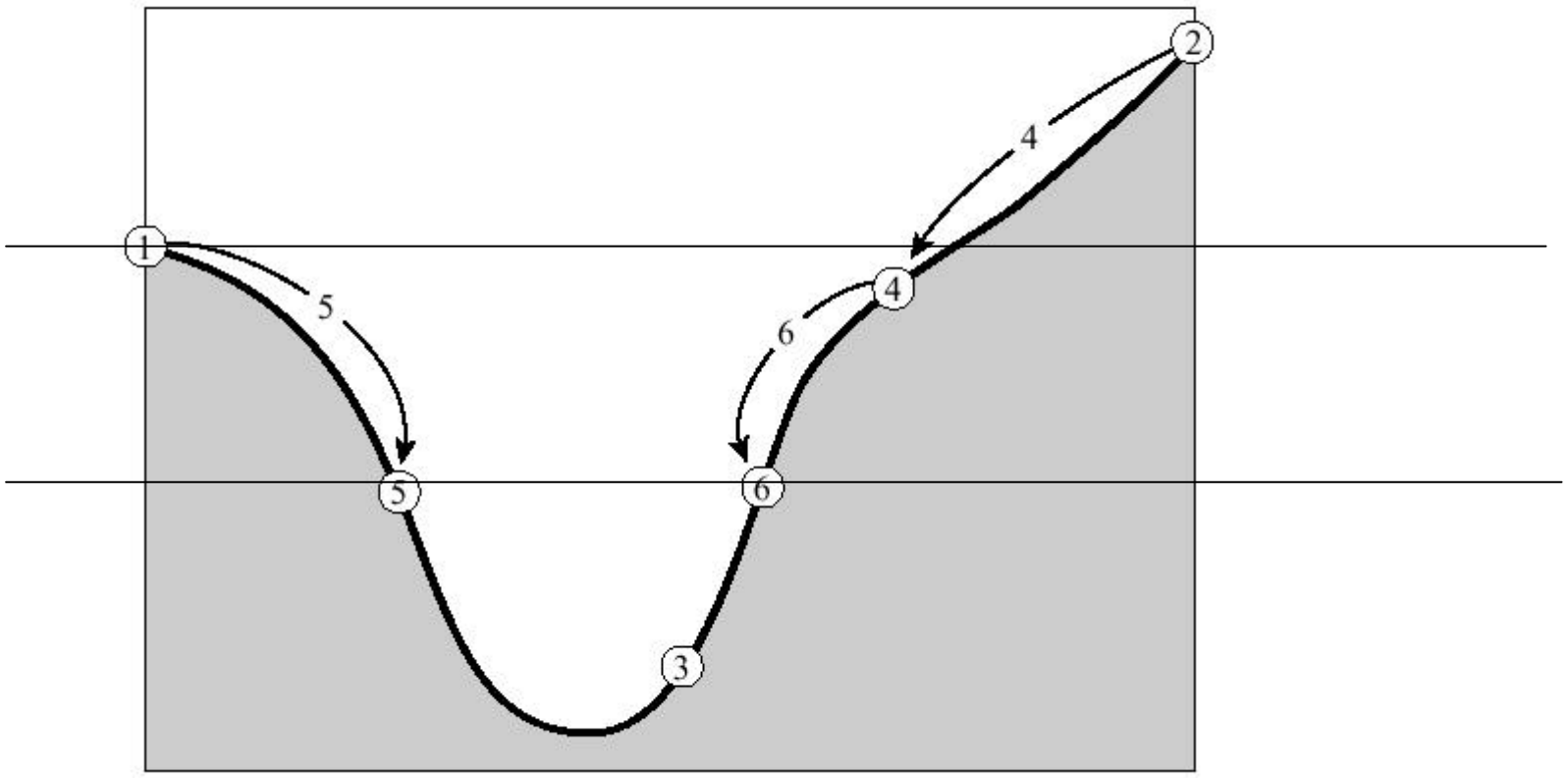
$f(b)$ è minore sia di $f(a)$ sia di $f(c)$.

In questo caso sappiamo che il minimo sarà nell'intervalo (a,c). Il metodo, analogo a quello di bisezione è scegliere un nuovo punto x, per esempio in (b,c) valutare $f(x)$

se $f(b)<f(x)$ allora la nuova terna di punti è (a,b,x) altrimenti è (b,x,c)

Si continua il processo finchè la distanza dei due punti più esterni è minore della precisione richiesta.

Bracketing di un minimo



1,2,3 sono i punti iniziali, il nuovo punto scelto è il 4 che sostituisce il 2, poi viene scelto il 5 che sostituisce il punto 1, poi il 6 sostituisce il 4. La nuova terna è 5,3,6. Ad ogni passo si sceglie il punto centrale di uno dei due intervalli se la funzione ha un valore minore degli estremi.

Ricerca del minimo

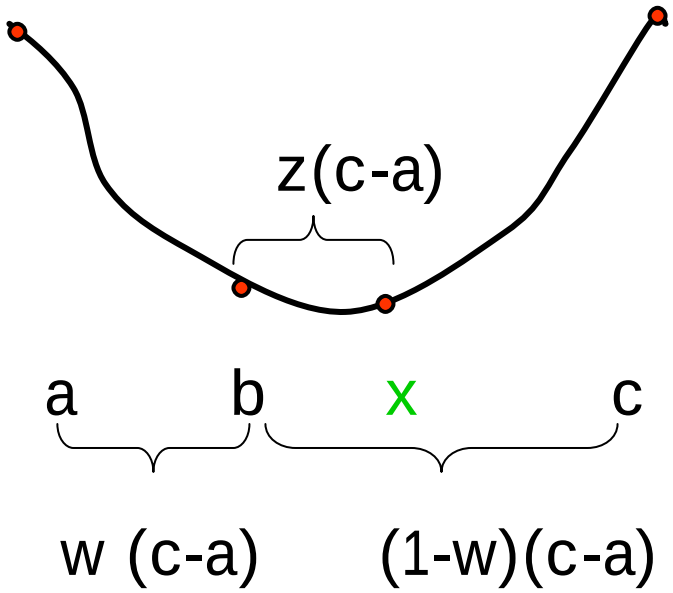
Il metodo fino ad ora accennato può essere migliorato, nel senso dell'efficienza, se si segue una buona strategia nella scelta del nuovo punto ad ogni passo.

Dati (a,b,c) supponiamo che b sia posto tra a e c in modo che

$$\frac{b-a}{c-a}=w \qquad \frac{c-b}{c-a}=1-w$$

Supponiamo ora che il nuovo punto x sia sistemato ad una frazione z di (a,c) oltre b

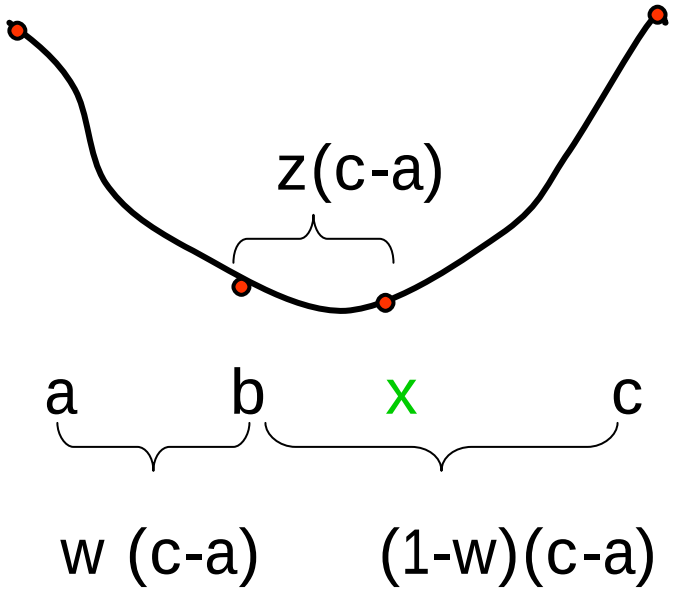
$$\frac{x-b}{c-a}=z$$



Ricerca del minimo

Il nuovo intervallo di bracketing potrà avere lunghezza w+z oppure 1-w. Per minimizzare il caso peggiore poniamo queste due lunghezze uguali; si deduce che

$$w + z = 1 - w \qquad z = 1 - 2w$$



Notiamo che x giacerà nel segmento tra (a,b) e (b,c) più largo, ma dove? Se si vuole mantenere la scala della scelta precedente, possiamo porre x ad una frazione w di (b,c) (+ grande) ovvero

$$\frac{z}{1-w} = w$$

x sarà quindi il simmetrico di b nell'intervallo originale (a,c)

Ricerca del minimo

Dalle due equazioni in z si ottiene una equazione di secondo grado in w (con la seguente soluzione <1)

$$w^2 - 3w + 1 = 0 \qquad w = \frac{3-\sqrt{5}}{2} \cong 0.38197$$

In altre parole si può definire un intervallo di bracketing ottimo quando il punto intermedio dista 0.38197 da un estremo e 0.61803 dall'altro.

Questo metodo è chiamato "golden section" e si può riassumere: ad ogni iterazione il nuovo punto della terna di bracketing è scelto quello posizionato nell'intervallo più largo ed ad una distanza pari a 0.38197 della larghezza dell'intervallo stesso rispetto a punto centrale della terna.

Questo porta al seguente codice (Numerical Recipes cap.10)

```
#define R 0.61803399 //The golden ratios.
#define C (1.0-R)
#define SHFT2(a,b,c) (a)=(b);(b)=(c);
#define SHFT3(a,b,c,d) (a)=(b);(b)=(c);(c)=(d);


double golden(double ax, double bx, double cx,
double f(double), double tol, double &xmin)
{
  double f1,f2,x0,x1,x2,x3;
  x0=ax;
  x3=cx;
  if (fabs(cx-bx) > fabs(bx-ax))
    {
      x1=bx;
      x2=bx+C*(cx-bx);
    } else {
      x2=bx;
      x1=bx-C*(bx-ax);
    }
 f1=f(x1);
  f2=f(x2);
  while (fabs(x3-x0) > tol*(fabs(x1)+fabs(x2))) {
    if (f2 < f1) {
      SHFT3(x0,x1,x2,R*x1+C*x3)
            SHFT2(f1,f2,f(x2))
            } else {
              SHFT3(x3,x2,x1,R*x2+C*x0)
                SHFT2(f2,f1,f(x1))
                }
  }
  if (f1 < f2) {
    xmin=x1;
    return f1;
  } else {
    xmin=x2;
    return f2;
    }
}
```

Sviluppo in serie di una funzione

Come abbiamo visto nell'illustrazione dei semplici metodi di integrazione e ricerca di zeri, l'approccio del calcolo numerico consiste (spesso) nel sostituire una funzione complicata con una sua approssimazione più semplice. Si può dimostrare che, data una funzione $f(x)$ di classe $C(8)$ si può scrivere:

$$f(x) = f(0) + f'(0)x + \frac{1}{2} f''(0)x^2 + \ldots + \frac{1}{n!} f^{(n)}(0)x^n + \ldots$$

Questa serie, detta serie di Taylor, ci dice che, noti i valori di tutte le derivate di una funzione in un punto è possibile ricostruire la funzione in un punto qualsiasi.

Osservazioni sulla serie di Taylor

$$f(x) = f(0) + f'(0)x + \frac{1}{2} f''(0)x^2 + \ldots + \frac{1}{n!} f^{(n)}(0)x^n + \ldots$$

- Lo sviluppo di Taylor è polinomiale: gli $f^{(n)}(0)$ sono costanti numeriche.
- Si può partire in un punto diverso dall'origine; basta fare una traslazione dell'asse x.
- Se si è interessati all'intorno dell'origine (o del punto di sviluppo) si avrà $x \ll 1$, e quindi i contributi dei termini di ordine n andranno progressivamente decrescendo. Ciò consente di troncare lo sviluppo ad un dato termine n trascurando i contributi di ordine x^{n+1}.

Metodo dei minimi quadrati

Supponiamo di avere un set di misure $(x_i, y_i \pm \sigma_i)$ con $i=1,\ldots N$ e di una relazione funzionale teorica che lega le grandezze x e y

$$Y = f(x, p_1, p_2, \ldots, p_M)$$

dove i p_j sono parametri liberi della teoria. Si pone il problema di determinare quale tra le possibili scelte dei p_j si adatti meglio ai punti misurati. Il metodo dei minimi quadrati consiste nell'affermare che la miglior scelta è quella che rende minima la funzione:

$$c^2(p_1, p_2, \ldots, p_M) = \sum_{i=1}^{N} \frac{(y_i - f(x_i, p_1, p_2, \ldots, p_M))^2}{s_i^2}$$

Metodo dei minimi quadrati

Per fissare le idee possiamo pensare che la funzione sia una retta, cioè

$$f(x, p_1, p_2) = p_1 x + p_2$$

Il problema del best-fit consiste nello scegliere, tra le infinite possibili rette, quella che meglio si adatta alle nostre misure.

Per fare questo calcoliamo la funzione $\chi^2(p_1,p_2)$, che consiste nella somma dei quadrati delle distanze lungo y tra i punti e la retta diviso per il quadrato dell'errore. Ovviamente tale funzione è nulla se la retta passa per tutti i punti, e comunque è tanto più piccola quanto maggiore è l'accordo tra la retta e i punti, o quanto maggiore è l'errore di misura.

Metodo dei minimi quadrati

Il problema della minimizzazione del χ^2 nel caso di una retta possiede una soluzione analitica. Se però la funzione non è una retta ci si deve affidare ad un metodo numerico di ricerca del minimo. Se la funzione ha molti parametri la ricerca sarà complicata, e quindi è conveniente usare un algoritmo "serio" e non tentare il fai-da-te.

Il χ^2, oltre a indicarci i valori dei parametri di f che danno il migliore accordo ci fornisce altre informazioni:

- indica la qualità dell'accordo; intuitivamente χ^2 piccolo indica buon accordo, ma vedrete nel corso di Lab1 b che il χ^2 fornisce anche un indicazione statistica quantitativa.
- fornisce una stima dell'errore sui parametri.

Errori sui parametri

Visto che le misure sono affette da errore anche la determinazione dei parametri della funzione avrà una indeterminazione.

Se siamo nel caso di una retta e, quindi, possiamo calcolare i parametri analiticamente, anche la valutazione dell'errore su di essi può essere calcolata con le formule che ci permettono di trattare gli errori statistici (par. 11.1 dispense di Laboratorio di Fisica).

Se, d'altra parte, il minimo è trovato in modo numerico, occorre trovare un altro metodo per la valutazione dell'errore sui parametri.

Errori sui parametri

La regola dice che l'intervallo corrispondente all'errore sui parametri è quello in cui il χ^2 varia, rispetto al minimo, di 1

$$\chi^2(p_1,\ldots,p_M) - \chi^2_{min} < 1$$

Non possiamo dimostrare questa affermazione in generale. Se però si considera il caso particolare di una funzione con un solo parametro e un solo punto sperimentale si può fare una verifica diretta.

Infatti se abbiamo una sola misura diciamo y_0 e un solo parametro p, possiamo scegliere la funzione teorica in modo che il χ^2 sia 0 nel minimo.

Errori sui parametri

Supponiamo ora di ripetere la misura e di trovare il valore $y_0+\sigma_y$ il nuovo valore del parametro p che permetterà di annullare χ^2 sarà t.c.

$$c^2(p_0+\Delta p) = \frac{\left(y_0 - f(y_0 + s_y)\right)^2}{s_y^2}$$

ovvero

$$c^2(p_0+\Delta p) = \frac{\left(y_0 - y_0 - s_y\right)^2}{s_y^2} = 1$$

Cioè la variazione di p, in corrispondenza della variazione di σ sul dato, corrisponde ad una variazione su χ^2 di 1.

Minimizzazione del c^2

Il caso di una misura e un parametro è poco interessante, ma può servire come esempio in laboratorio per utilizzare la funzione di minimizzazione vista in precedenza.

Supponiamo di eseguire una serie di misure (i=1,...,N) di una stessa quantità, diciamo y_i con errori σ_i e vogliamo valutare la migliore stima della misura. Dalla teoria degli errori sappiamo che questa stima è data dalla media pesata

$$c^2(p_1, p_2, \ldots, p_M) = \sum_{i=1}^{N} \frac{(y_i - f(x_i, p_1, p_2, \ldots, p_M))^2}{s_i^2}$$

se

$$f(p) = p$$

$$c^2(p) = \sum_{i=1}^{N} \frac{(y_i - p)^2}{s_i^2}$$

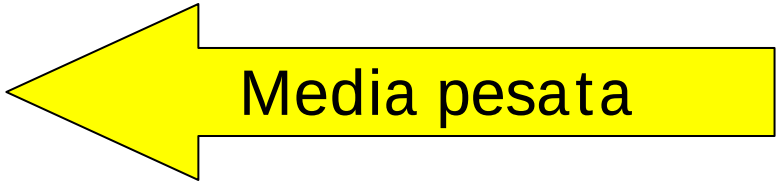
Minimizzazione del c^2

Calcoliamo il minimo del χ^2

$$\frac{\partial c^2}{\partial p} = -2\sum_{i=1}^{N} \frac{(y_i - p)}{s_i^2} = 0$$

$$\sum_{i=1}^{N} \frac{y_i}{s_i^2} - p\sum_{i=1}^{N} \frac{1}{s_i^2} = 0$$

$$p = \frac{\sum_{i=1}^{N} \frac{y_i}{s_i^2}}{\sum_{i=1}^{N} \frac{1}{s_i^2}}$$



Calcolo dell'errore sul parametro

Calcoliamo ora l'errore su p imponendo una variazione di 1 sul χ^2

$$c^2(p+\Delta p) - c^2(p) = 1$$

$$\sum_{i=1}^{N} \frac{(y_i - (p+\Delta p))^2}{s_i^2} - \sum_{i=1}^{N} \frac{(y_i - p)^2}{s_i^2} = 1$$

svolgendo i calcoli

$$\Delta p^2 \sum_{i=1}^{N} \frac{1}{s_i^2} + 2p\Delta p \sum_{i=1}^{N} \frac{1}{s_i^2} - 2\Delta p \sum_{i=1}^{N} \frac{y_i}{s_i^2} = 1$$

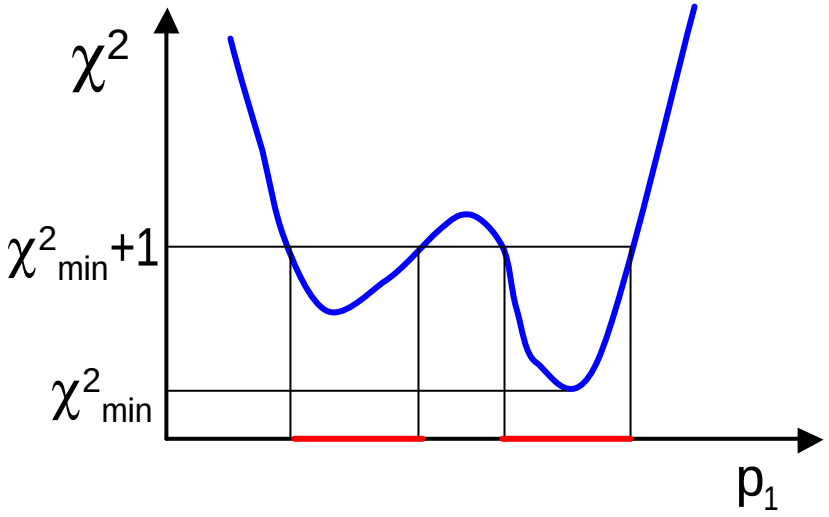
sostituendo il valore di p e semplificando

$$\Delta p^2 = \frac{1}{\sum_{i=1}^{N} \frac{1}{s_i^2}} \text{ che per errori tutti uguali diventa } \Delta p^2 = \frac{s^2}{N}$$

Sviluppo di c^2 intorno al minimo

In generale la forma della regione corrispondente all'errore sui parametri è complessa, e non è necessariamente connessa, nel senso che può consistere di diverse regioni separate.

Possiamo però pensare di eseguire uno sviluppo in serie intorno al minimo (ad es. con due parametri):



$$c^2(p_1,p_2) = c^2_{min} + \frac{\partial c^2}{\partial p_1}\Delta p_1 + \frac{\partial c^2}{\partial p_2}\Delta p_2 + \frac{1}{2}\frac{\partial^2 c^2}{\partial p_1^2}\Delta p_1^2 + \frac{\partial^2 c^2}{\partial p_1 \partial p_2}\Delta p_1 \Delta p_2 + \frac{1}{2}\frac{\partial^2 c^2}{\partial p_1^2}\Delta p_1^2 + O(\Delta p^3)$$

Siccome le derivate prime nel minimo sono nulle, restano solo le derivate seconde, e quindi la forma della regione corrispondente alla variazione di χ^2 uguale a uno è ellittica, e le lunghezze degli assi sono legate agli errori sui parametri.

$$c^2(p_1,p_2) - c^2_{min} = 1 = \frac{1}{2}\frac{\partial^2 c^2}{\partial p_1^2}\Delta p_1^2 + \frac{\partial^2 c^2}{\partial p_1 \partial p_2}\Delta p_1 \Delta p_2 + \frac{1}{2}\frac{\partial^2 c^2}{\partial p_1^2}\Delta p_1^2$$

La libreria Minuit

Minuit è una pacchetto specializzato per la minimizzazione di funzioni a molti parametri. Nella libreria Root c'è un'interfaccia semplificata a questo pacchetto. Vedremo i dettagli in laboratorio. Ciò che dovete fornire alla funzione che trovate in Root è:

- la funzione da minimizzare (il χ^2, che dipende da M parametri dati sotto forma di vettore)
- il valore iniziale dei parametri
- il valore iniziale del passo di ricerca del minimo (che può essere zero se volete bloccare un parametro al valore iniziale)
- il massimo e il minimo per ciascun parametro

In output avrete:

- il valore dei parametri corrispondenti al minimo
- gli errori sui parametri

Ordinamento di un vettore

L'ordinamento di un vettore di n elementi si può realizzare nel modo seguente: si cerca l'elemento più piccolo, lo si scambia con quello nella prima posizione, quindi si ripete la procedura nel vettore di n-1 elementi da 1 a n-1; la procedura va iterata fino a quando non si eseguono più scambi o si resta con un vettore di un solo elemento.

```
void sort(int *vect, int n) {
  int i,swap,top,temp;

  if (n < 2) return;

  top = 0;
  do {
    swap = 0;
    for (i=n-1; i>top; i--) {
      if (vect[i] < vect[i-1]) {
        swap = 1;
        temp = vect[i];
        vect[i] = vect[i-1];
        vect[i-1] = temp;
      }
    }
    top++;
  } while(swap != 0 && top < n-1);
}
```

Aggiunta di elementi a un vettore

Per aggiungere un elemento ad un vettore si devono fare due cose: variarne la dimensione e spostare tutti gli elementi posizionati dopo quello aggiunto:

```
int *vect = NULL;
int n_el = 0, max_el = 0;

void increase_size(int siz) {
  int i,*temp;
  /* Crea un nuovo vettore */
  max_el = max_el + siz;
  temp = new int[max_el]
  /* Copia il vecchio vettore nel nuovo */
  for (i=0; i<n_el; i++) temp[i] = vect[i];
  /* Cancella il vecchio vettore e riassegna il puntatore */
  if (vect != NULL) delete[] vect;
  vect = temp;
}
```