Generatori di numeri casuali e metodi Montecarlo (parte 2)

Laboratorio Trattamento Numerico dei Dati Sperimentali

Prof. L. Carminati
Università degli Studi di Milano

Generatori di numeri casuali in labTNDS

Nella nostra implementazione ogni numero viene generato da una chiamata ad un certo metodo (non generiamo direttamente una sequenza)

2^{31} (attenzione non 2E31 !)

Possiamo eventualmente conservare il seed originale

Aggiorno `m_seed`

Per la gaussiana `g` possiamo assumere $[-5\sigma;+5\sigma]$ come range di estrazione e calcolare `fmax` come `g(mean)`

Attenzione all'AR : dobbiamo continuare ad estrarre finchè non otteniamo un numero x (!!) buono da restituire

Si potrebbe pensare ad un generico `AcceptReject` che accetti in input `FunzioneBase&, fmax, xmin` e `xmax`

```
class RandomGen {

public:

  RandomGen(unsigned int seed) {
    m_a = 1664525;
    m_c = 1013904223;
    m_m = 1<<31;
    m_seed = seed;
    m_seed_original = seed ;
  }

  unsigned int getSeedOriginal() { return m_seed_original ;} ;

  double Rand( ) {
    m_seed = (m_a*m_seed+m_c)%m_m;
    return double(m_seed)/double(m_m);
  }

  double Uniform( double min, double max ) {
    return min + (max-min)* Rand();
  }

  double GaussAR(double mean, double sigma) {

    double xmin= mean-5*sigma ;
    double xmax= mean+5*sigma;
    double fmax = 1/sqrt( 2*M_PI*sigma* sigma) ;
    double x=0, y=0;

    do {
      x = Uniform( xmin , xmax ) ;
      y = Uniform( 0 , fmax )  ;
    } while (  y > 1/sqrt( 2*M_PI*sigma* sigma) * exp( - pow(x-mean,2)/(2*sigma*sigma) ) );

    return x ;
  }

private:

  unsigned int m_a, m_c, m_m;
  unsigned int m_seed , m_seed_original;

};
```

Generatori di numeri casuali in labTNDS

- Si potrebbe pensare ad un generico `AcceptReject` che accetti `FunzioneBase&, fmax, xmin` e `xmax`

Il massimo della funzione passato dall'esterno

```
double RandomGen::AcceptReject(double xmin, double xmax, const FunzioneBase &f, double fmax) {

  double x = 0 ;
  double y = 0 ;

  do {
    x = Unif( xmin , xmax ) ;
    y = Unif( 0 , fmax )  ;

  } while (  y > f.Eval(x) );

  return x ;
}
```

Generatori di numeri casuali in STL

https://cplusplus.com/reference/random/

```
// ...
#include <random>

// add all ROOT #include

int main() {

  std::mt19937 e1 (2.) ;

  std::uniform_real_distribution<double> uniform_dist(1, 6);
  TH1F hunif("uniform","Uniform distribution",100, 0, 10);
  for ( int k = 0 ; k < 10000 ; k++ ) hunif.Fill( uniform_dist(e1) );

  std::normal_distribution<> normal_dist(2, 1);
  TH1F hgauss("gauss","Normal distribution",100, -10, 10);
  for (int n = 0; n < 10000; n++) hgauss.Fill( normal_dist( e1  ) );

  std::exponential_distribution<> expo_dist(0.5);
  TH1F hexpo("expo","Exponential distribution",100, 0, 20);
  for (int n = 0; n < 10000; n++) hexpo.Fill( expo_dist( e1  ) );

  std::poisson_distribution<> pois_dist(4);
  TH1F hpois("Poisson","Poisson distribution",15, 0, 15);
  for (int n = 0; n < 10000; n++) hpois.Fill( pois_dist( e1  ) ) ;

  // show distributions

  TCanvas * can = new TCanvas("distributions","Random numbers generator");
  can->Divide(2,2);
  can->cd(1);
  hunif.Draw();

  // plot all the others

  return 0;
}
```

Build an engine : a Marsenne Twister uniform generator

Shape the flat generation on the desired output distribution

Generatori di numeri casuali in ROOT

https://root.cern.ch/doc/master/classTRandom.html

```
#include "TRandom3.h"

// add all ROOT #include

int main() {

  TRandom3 rand(2.);

  TH1F Rhunif("Uniform(ROOT)","Uniform(ROOT)",100, 0, 10);
  for (int n = 0; n < 10000; n++) Rhunif.Fill( rand.Uniform(1,6) ) ;
  TH1F Rhgaus("gaus(ROOT)","Normal distribution(ROOT)",100, -10, 10);
  for (int n = 0; n < 10000; n++) Rhgaus.Fill( rand.Gaus(2,1)    ) ;
  TH1F Rhexpo("expo(ROOT)","Exponential distribution(ROOT)",100, 0, 20);
  for (int n = 0; n < 10000; n++) Rhexpo.Fill( rand.Exp(0.5)     ) ;
  TH1F Rhpois("Poisson(ROOT)","Poisson distribution(ROOT)",15, 0, 15);
  for (int n = 0; n < 10000; n++) Rhpois.Fill( rand.Poisson(4)   ) ;

  // show distributions

  TCanvas * Rcan = new TCanvas("distributions(ROOT)","Random numbers generator(ROOT)");
  Rcan->Divide(2,2);
  Rcan->cd(1);
  Rhunif.Draw();

  // plot all the others

  return 0;

}
```

Build a random number generator

Use different implemented methods to get the desired distribution

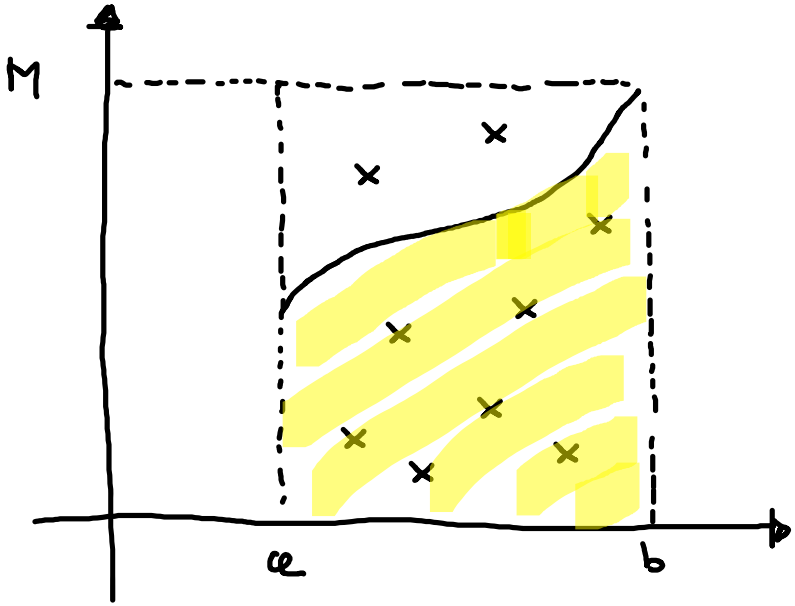
Suggerimento

Usare sempre librerie esistenti !

(eccetto che per l'esame di TNDS)

Integrazione Montecarlo (I): metodo hit or miss

Una interessante applicazione dei numeri casuali: tecniche Montecarlo per l'integrazione numerica



- Estraggo un numero casuale $\bar{x}$ uniformemente distribuito tra a e b
- Estraggo un numero casuale $\bar{y}$ uniformemente distribuito tra 0 e M (M = massimo della f(x))
- Ad ogni estrazione di una coppia ($\bar{x}$, $\bar{y}$) incremento un contatore N_{TOT}
- Se $\bar{y} < f(\bar{x})$ incremento un secondo contatore N_{HIT}

Intuitivamente uno stimatore dell'integrale può essere scritto facilmente

$$\hat{I} = (b-a)M\frac{N_{HIT}}{N_{TOT}}$$

Attenzione nell'implementazione : **rapporto tra interi**

In pratica "peso" l'area del rettangolo R=M(b-a) con la frazione di punti che stanno sotto la funzione

Integrazione Montecarlo (I) : metodo hit or miss

La garanzia che $\hat{I}$ sia un buono stimatore (consistente i.e. converge all'integrale vero per numero di estrazioni crescente) del valore dell'integrale ci viene dal teorema del limite centrale

- Il processo di estrazione del punto può essere rappresentato da una variabile aleatoria x_i che mappi lo spazio degli eventi elementari Ω={successo,insuccesso} in $\mathbb{R}$

$$x_i = \begin{cases} successo & +1 \\ insuccesso & 0 \end{cases}$$

- Assegnamo una probabilità alla variabile aleatoria che abbiamo appena costruito (ad essere precisi la probabilità va assegnata agli eventi di $\mathbb{F}$)

$$x_i = \begin{cases} +1 \ (successo) \rightarrow \ p(+1) = \dfrac{A_{int}}{A_{tot}} \ (= p) \\ 0 \ (insuccesso) \rightarrow \ p(0) = 1 - p(+1) \end{cases}$$

- Calcoliamo il valore di aspettazione e la varianza della variabile aleatoria x_i

$$\mu = \sum x_i p_i = 1p + 0(1-p) = p\left(= \frac{A_{int}}{A_{tot}}\right)$$

$$\sigma^2 = < (x-\mu)^2 > = (1-p)^2 p + (0-p)^2(1-p) = p(1-p)$$

Integrazione Montecarlo (I) : metodo hit or miss

- Se facciamo N estrazioni possiamo rappresentare il risultato con la variabile aleatoria

$$Y_N = \sum x_i \ (= N_{HIT})$$

- Per il teorema del limite centrale

$$Y_N \longrightarrow N\mu = Np = N\frac{A_{int}}{A_{tot}}$$

$$\frac{Y_N}{N} A_{tot} = \frac{N_{HIT}}{N} A_{tot} = \hat{I} \longrightarrow A_{int}$$

Il nostro stimatore dell'integrale converge all'integrale vero !

- Cosa succede all'errore ? Ancora per il teorema del limite centrale

$$\sigma_{Y_N}^2 = N\sigma_x^2 \Longrightarrow \sigma_{Y_N} = \sqrt{N}\sigma_x$$

$$\sigma_I = \frac{\sigma_{Y_N}}{N} A_{tot} = \frac{\sqrt{N}\sigma_x}{N} A_{tot} = \frac{\sigma_x A_{tot}}{\sqrt{N}} = \frac{k}{\sqrt{N}}$$

Errore di un metodo MC scala come $k/\sqrt{N}$ con N numero di estrazioni che ho effettuato

Integrazione Montecarlo (II) : metodo della media

Vogliamo calcolare l'integrale di una funzione f(x) in [a,b]

- Consideriamo una variabile aleatoria x distribuita secondo una densità di probabilità p(x)
- Estraiamo dei numeri casuali x_i con una distribuzione p(x) e valutiamo $f(x_i)$.
- Costruiamo ora la variabile aleatoria Y_N come somma delle N variabili $f(x_i)$

$$Y_N = \frac{\sum f(x_i)}{N}$$

- Per il teorema del limite centrale

$$Y_N \longrightarrow < f(x) > = \int_{-\infty}^{+\infty} f(x)p(x)dx$$

Law of unconscious statistician (*)

- Se ora scelgo la `p(x)` come distribuzione uniforme tra [a,b]

$$p(x) = \begin{cases} \frac{1}{b-a} & x \in [a,b] \\ 0 & x \notin [a,b] \end{cases} \qquad Y_N \longrightarrow < f(x) > = \int_{-\infty}^{+\infty} f(x)p(x)dx = \int_a^b f(x)\frac{1}{b-a}dx$$

$$\widehat{I_N} = \frac{\sum f(x_i)}{N}(b-a) \longrightarrow \int_a^b f(x)dx \qquad N \to +\infty$$

(*) Law of unconscious statistician

- Vogliamo mostrare che il valore di aspettazione di una funzione g di variabile aleatoria x distribuita con una pdf pari a f(x) è $\int_{-\infty}^{+\infty} g(x)f(x)dx$
- Incominciamo a considerare una variabile aleatoria Y non negativa. Mostriamo che vale

$$< Y > = \int_{y=0}^{+\infty} P(Y \geq y)dy$$

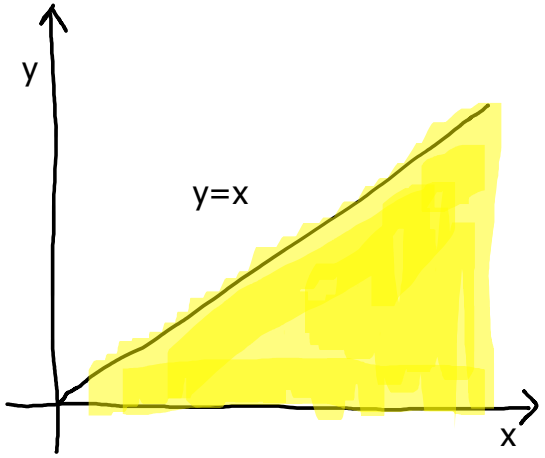
- Infatti possiamo scrivere il membro di destra in questo modo :

Cambio ordine di integrazione, vedi diagramma per il cambio degli estremi

$$\int_{y=0}^{+\infty} P(Y \geq y)dy = \int_{y=0}^{+\infty} \int_{x=y}^{+\infty} f_Y(x)dx\,dy$$

$$\int_{x=0}^{+\infty} \int_{y=0}^{x} f_Y(x)dy\,dx = \int_{x=0}^{+\infty} f_Y(x)\left(\int_{y=0}^{x} dy\right)dx$$

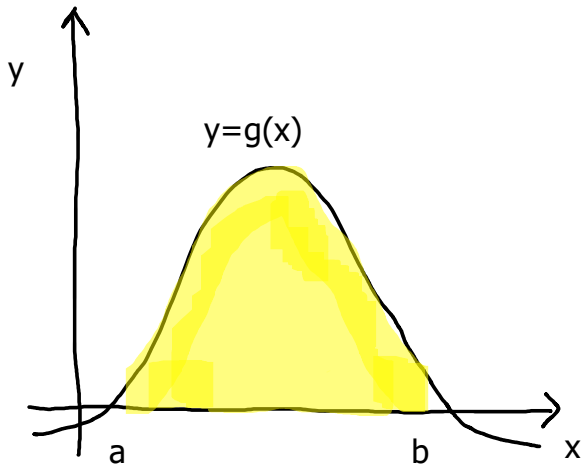
$$= \int_{x=0}^{+\infty} x\, f_Y(x)dx = < Y >$$



(*) Law of unconscious statistician

- Consideriamo una regione di g(x) in cui g(x)≥0 : secondo quando mostrato in precedenza

$$< g(x) >= \int_{y=0}^{+\infty} P(g(x) \geq y)dy$$



Cambio ordine di integrazione, vedi diagramma per il cambio degli estremi

$$\int_{y=0}^{+\infty} \int_{x:g(x)>y} f(x)dxdy = \int_{x=a}^{b} \int_{y=0}^{g(x)} f(x)dydx$$

$$\int_{x=a}^{b} f(x)\left(\int_{y=0}^{g(x)} dy\right)dx = \int_{x=a}^{b} g(x)f(x)dx$$

- Possiamo poi procedere per intervalli in cui la funzione non cambia di segno e concludere che

$$< g(x) >= \int_{+\infty} g(x)f(x)dx$$

Integrazione Montecarlo (II) : metodo della media

Teorema del limite centrale

- L'algoritmo in sintesi :
  - Estraiamo N numeri casuali x_i uniformemente distribuiti [a,b]
  - Calcoliamo i valori della funzione in questi punti $f(x_i)$
  - Calcoliamo lo stimatore $\hat{I}_N = \frac{\sum f(x_i)}{N}(b-a)$: tende proprio all'integrale "vero" per $N \to +\infty$
- Come si comporta l'errore :
  - Dal teorema del limite centrale sappiamo che $\sigma^2_{Y_N} \to \frac{\sigma_f^2}{N}$
  - Quindi poichè $\hat{I}_N = Y_N(b-a)$ allora (propagazione degli errori)

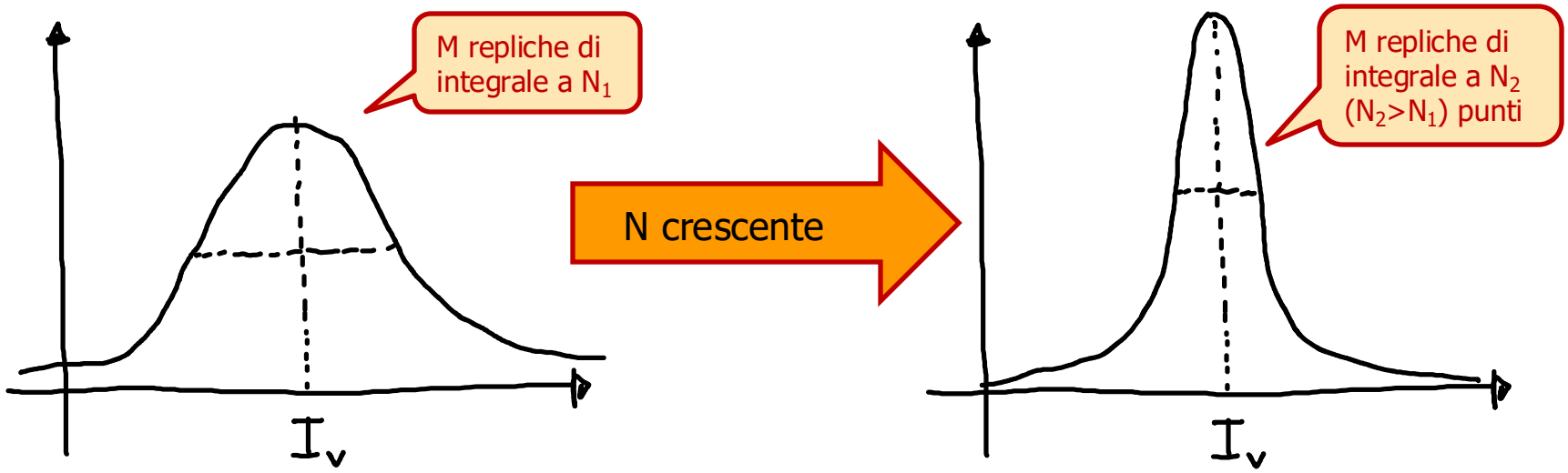
$$\sigma^2_{I_N} = \sigma^2_{Y_N}(b-a)^2 \to \frac{\sigma_f^2}{N}(b-a)^2$$

$$\sigma_I = \frac{\sigma_f(b-a)}{\sqrt{N}}$$

Ancora scaling ~ $k/\sqrt{N}$

Senso dell'errore di un metodo Montecarlo

- Se effettuo il calcolo di un integrale con il metodo Hit or Miss con un set di N punti casuali due volte non otterò lo stesso valore ! (normalmente il set di punti estratti è sempre diverso)
- Il senso dell'errore di un integrale con metodo MC è il seguente : se ripeto il calcolo dell'integrale a N punti M volte e li metto in un istogramma (con M entries) i valori si distribuiranno secondo una gaussiana centrata attorno al valore "vero" dell'integrale e di larghezza σ_I



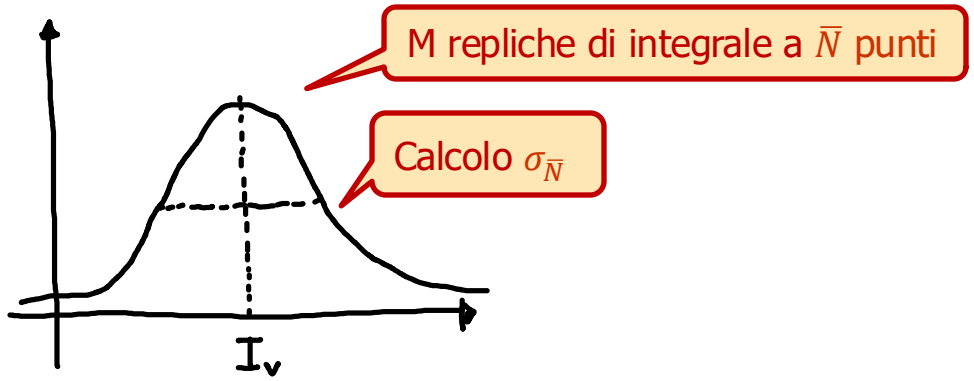
- Per stimare run-time l'errore non ha nessun senso fare I_N-I_{2N} : paradossalmente una stima dell'integrale a N punti può essere più vicina all'integrale vero di una stima a 2N punti !

Integrale MC generico a precisione fissata :

Una possibile procedura per un generico metodo di integrazione MC

1. Scelgo un numero di estrazioni (arbitrario) $\bar{N}$ e calcolo M volte (diciamo ~ 10000) l'integrale (ciascun integrale viene dall'estrazione di $\bar{N}$ punti)

2. Dal vettore degli M integrali calcolo $\sigma_{\bar{N}}$



3. Sapendo che l'errore scala come $k/\sqrt{N}$ determino $\tilde{N}$ necessario per ottenere un errore di $\tilde{\sigma}$
   - Posso calcolare la costante $k = \sigma_{\bar{N}} \sqrt{\bar{N}}$
   - Stimo il numero di punti $\tilde{N}$ che sarebbero necessari per avere $\tilde{\sigma}$:

$$\tilde{N} = \frac{k^2}{\tilde{\sigma}^2} = \frac{\sigma_{\bar{N}}^2 \, \bar{N}}{\tilde{\sigma}^2}$$

4. Effettuo il calcolo di un singolo integrale con $\tilde{N}$ estrazioni. Senso di questo integrale: un valore estratto da una distribuzione con larghezza $\tilde{\sigma}$! (non dice quanto sia davvero vicino al valore vero)

Integrale con il metodo della media a precisione fissata :

Per il metodo della media vale la procedura generale della slide precendente ma si può pensare ad un algoritmo più efficiente: non è necessario fare M repliche dell'integrale a $\bar{N}$ punti per stimare $\sigma_{\bar{N}}$!

- Dalle slide precedenti l'incertezza sull'integrale con il metodo della media (dal teo limite centrale):

$$\sigma_I = \frac{\sigma_f (b-a)}{\sqrt{N}}$$

- Posso stimare σ_f da **un singolo integrale** a $\bar{N}$ punti: per calcolare l'integrale infatti devo estrarre punti x_i e calcolare $f(x_i)$. Quindi mentre accumulo le somme di $f(xi)$ posso anche stimare

$$\sigma_f = \sqrt{\frac{1}{\bar{N}-1}\sum \left(f(x_i) - \bar{f}\right)^2}$$

da cui posso calcolare direttamente una stima l'errore sull'integrale

$$\sigma_I = \frac{\sigma_f (b-a)}{\sqrt{\bar{N}}}$$

e σ_I coincide con $\sigma_{\bar{N}}$ (entro gli errori statistici)

- Nella funzione che calcola l'integrale mentre si calcola $\bar{f}$ predisporre anche il calcolo di σ_f

- A questo punto si estrapola a $\tilde{N}$ necessari per ottenere $\tilde{\sigma}$ come nel caso della procedura generale

Implementazione calcolo integrale con metodi MC : opzione 1

- Classe base astratta `IntegraleMC` con generatore `RandomGen` come data membro (oggetto)

```
class IntegraleMC {

public:

  IntegraleMC(unsigned int seed)
    : m_gen(seed)
  {
    m_errore = 0;
  }

  virtual double Integra(const FunzioneBase* f, double inf , double sup , int punti, double fmax ) = 0 ;

  double GetErrore() const {return m_errore;}
  unsigned int GetN() const {return m_punti;}

protected:

  RandomGen m_gen;
  double m_errore;
  unsigned int m_punti;

};
```

Nel costruttore di `IntegraleMC` (che accetta il seme come input) posso inserire una lista di inizializzazione che mi permette di specificare quale costruttore di `RandomGen` invocare

Non necessario per la media

Generatore di numeri casuali come data membro (oggetto)

Implementazione calcolo integrale con metodi MC : opzione 2

- Classe base astratta `IntegraleMC` con generatore `RandomGen` come data membro (pointer)

```
class IntegraleMC {

public:

  IntegraleMC(unsigned int seed)
  {
    m_gen = new RandomGen(seed) ;
    m_errore = 0;
  }

  ~IntegraleMC() { delete m_gen ; } ;

  virtual double Integra(const FunzioneBase* f, double inf , double sup , int punti, double fmax ) = 0 ;

  double GetErrore() const {return m_errore;}
  unsigned int GetN() const {return m_punti;}

protected:

  RandomGen * m_gen;
  double m_errore;
  unsigned int m_punti;

};
```

Costruisco un generatore e faccio in modo che `m_gen` ci punti

Nel distruttore devo deallocare la memoria assegnata a `m_gen`

Generatore di numeri casuali come data membro (puntatore)

Implementazione calcolo integrale con metodi MC

- Classe derivate `IntegraleMedia` implementa un metodo `Integra(…)` concreto (quello della media)

```
class IntegraleMedia : public IntegraleMC {

public:

  IntegraleMedia(unsigned int seed) : IntegraleMC(seed) { ; };

  virtual ~IntegraleMedia() { ; } ;

  virtual double Integra (const FunzioneBase* f, double inf , double sup , int punti, double fmax = 0) {

    // ... implementare il metoro

  };

};
```

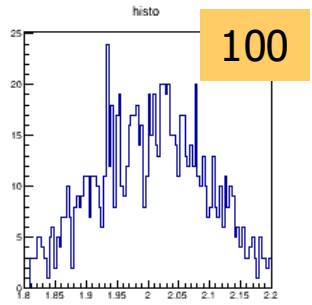
In questo modo posso ometterlo nella chiamata

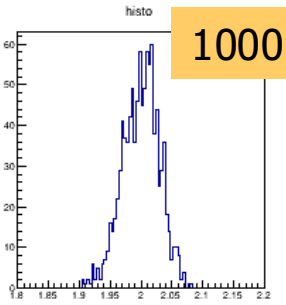
- Fare la stessa cosa con `IntegraleHoM`

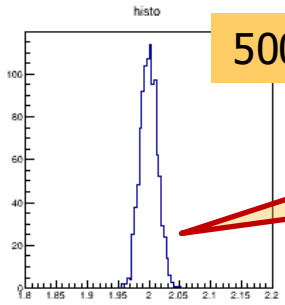
Verifica andamento errore metodi integrazione MC : esercizio10.2

1. Calcolare 10000 volte il valore dell'integrale di xsin(x) tra [0,π/2] utilizzando il metodo della media a 100 punti e fare un grafico (istogramma) della distribuzione dei valori ottenuti.
2. Estendere il punto precedente calcolando 10000 volte il valore dell'integrale di xsin(x) tra [0,π/2] utilizzando il metodo della media a N punti con N = 100, 500, 1000, 5000, 10000 punti. Per ogni valore di N produrre il grafico della distribuzione dei 10000 valori ottenuti.
   - [NOTA: poichè il calcolo degli integrali con N molto elevato potrebbe richiedere un certo tempo, potrebbe essere utile salvare in diversi files i valori degli integrali calcolati per un determinato N e svolgere i punti successivi con un secondo programma che utilizzi come input i files di integrali del programma precedente.]
3. Stimare l'errore sul calcolo dell'integrale a 100, 500, 1000, 5000, 10000 punti come deviazione standard dei 10000 valori calcolati per ogni N. Far un grafico di questo errore in funzione di N.
4. Assumendo che l'andamento dell'errore sia noto (del tipo k/√N) si determini quanti punti sono necessari per ottenere una precisione di 0.001.
5. Si ripeta lo stesso lavoro con il metodo hit-or-miss.

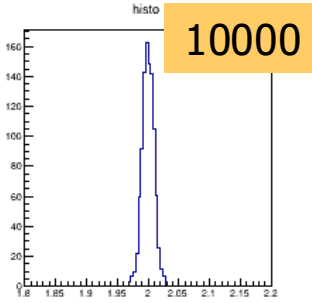
Verifica andamento errore metodi integrazione MC : esercizio10.2

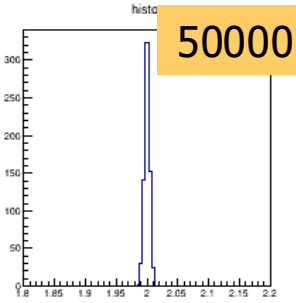


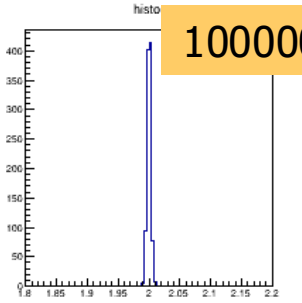


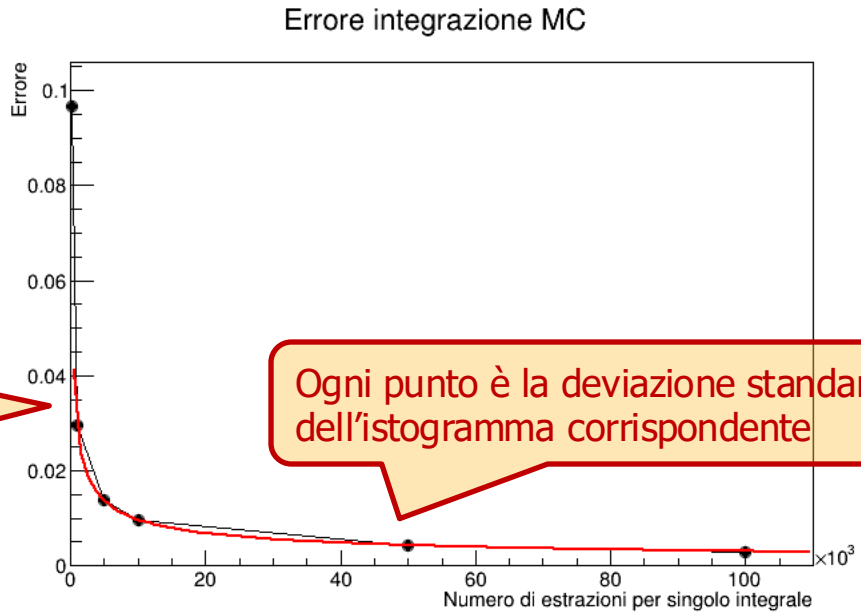


Istogramma con 1000 entries : ogni entry qui per esempio è un integrale calcolato con 5000 punti.









Curva rossa : fit con un modello
$f(N) = k/\sqrt{N}$

Ogni punto è la deviazione standard dell'istogramma corrispondente

Esercizio10.2 : implementazione in due steps

Questo è un caso tipico in cui il codice è costituito da una parte lenta (il calcolo ripetuto degli integrali) e una parte veloce (l'estrazione della deviazione standard dagli istogrammi e la creazione del grafico finale). In genere si tende a separare le due parti in codici diversi.

1. Codice che calcola gli integrali e li scrive su files: per esempio un file con i famosi 1000 numeri per ogni valore di N (N=100, 1000, 5000, 10000, 50000, 100000). Questa parte di codice potrebbe durare ordine minuti se per esempio spingiamo N fino al milione

2. Codice che apre i files e produce i grafici finali separato :
    1. Potremmo volerlo modificare molte volte a seconda dell'aspetto che vogliamo dare ai grafici
    2. Potremmo volerlo implementare usando altri tools o linguaggi (c++ e ROOT, python e matplotlib, gnuplot …)

Esercizio10.2 (I) : calcolo e scrittura su files

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <iostream>

using namespace std;

int main() {

  vector<int> cases { 500, 1000, 5000, 10000, 50000, 100000 } ;

  IntegratoreMedia MCIntegrator ;
  xsinx mysin;
  fstream f ;

  for ( auto i = 0 ; i < cases.size() ; i++ ) {

    cout << "==>> Running integration with N = " << cases[i] << " points" << endl;

    int nmax = 10000 ;

    string nomefile = ....   ;

    f.open ( nomefile, ios::out );

    for ( int k = 0 ; k < nmax ; k++ )  {

      f << MCIntegrator.Integra( mysin , /* ... */, cases[i], /* ... */ )  << endl;

      if ( ! (k%1000 )) cout << "Working " << double(k)/nmax * 100 << "%" << endl;

    }

    f.close() ;

  }

}
```

Casi che voglio esplorare : numero di estrazioni per il calcolo del singolo integrale

Questa è la parte computazionalmente 'lunga' se `nextractions[j]` è molto grande

Esercizio10.2 (II): lettura e plots

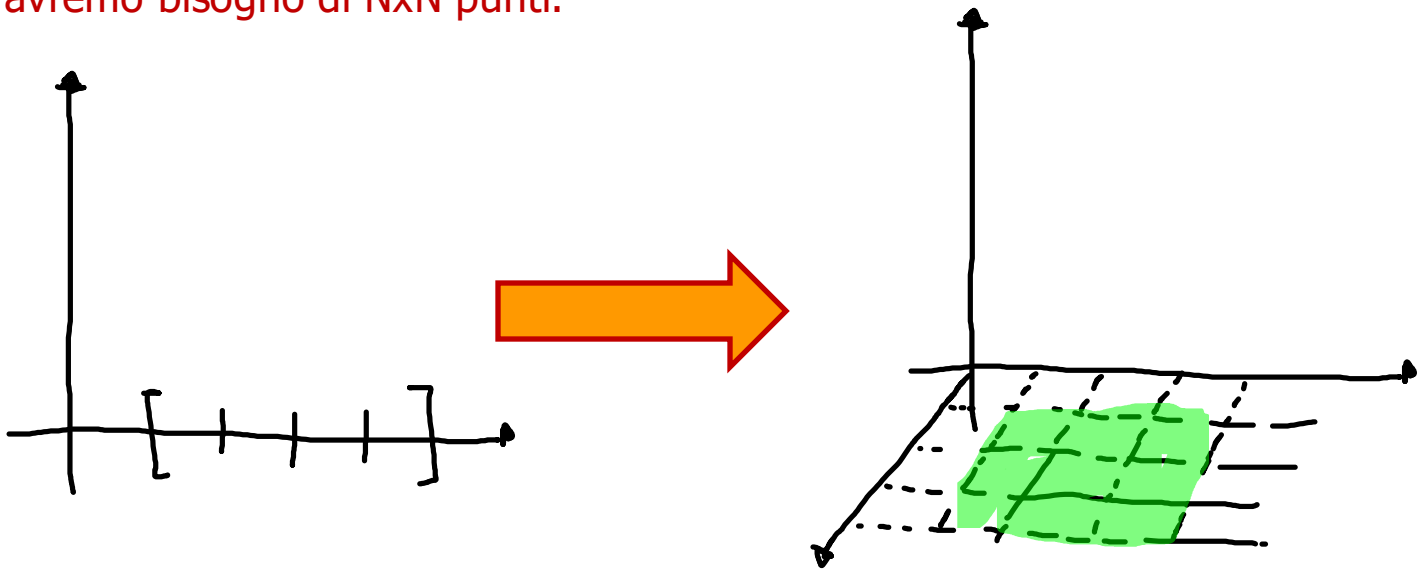
```
#include <vector>
#include <iostream>
#include <fstream>

#include "TApplication.h"
#include "TH1F.h"
#include "TH1.h"

// ...

using namespace std;

int main() {

  TApplication app("app",0,0) ;

  vector<int> cases { 100, 1000, 5000, 10000, 50000, 100000 } ;

  vector<TH1F*> vhistos ;

  // ....

  for ( auto i = 0 ; i < cases.size() ; i++ ) {

    // open file and read integrals values

    // Fill histograms

    // Compute stddev and fill TGraph

  }

  // ....

  mygraph.Draw("ALP");

  TF1 fun("fun","[0]/sqrt(x)",10,100000);
  mygraph.Fit(&fun);

  app.Run();

  return 0 ;

}
```

Casi che voglio esplorare : numero di estrazioni per il calcolo del singolo integrale

Lettura da file e calcolo della deviazione standard

Perchè si utilizzano tecniche MC ? Integrali multidimensionali

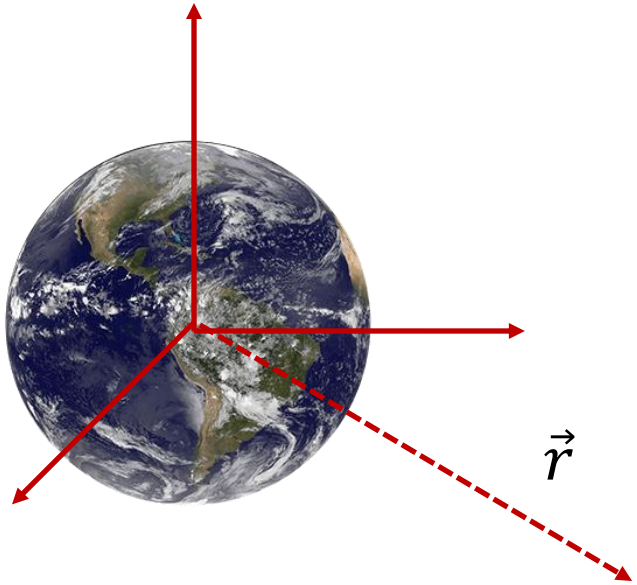
- Applichiamo il metodo dei trapezoidi ad un integrale monodimensionale : ci aspettiamo che l'errore scali come $1/N^2$ (potete riprendere le trasparenze della lezione dedicate)
- Cosa succeede se calcoliamo un integrale doppio con il medesimo metodo ? Per avere la stessa precisione avremo bisogno di NxN punti.



  - Per N fissato la precisione del metodo è ridotta in funzione del numero di dimensioni : $\frac{1}{N^{\frac{2}{d}}}$ (trapezodi mentre va come $\frac{1}{N^{\frac{4}{d}}}$ nel caso del metodo di Simpson

- Nei metodi MC errore ~ $\frac{1}{\sqrt{N}}$ indipendentemente dal numero di dimensioni del problema !

Un esempio di integrale multidimensionale



$$V(\vec{r}) = -\int dx'dy'dz' G \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|}$$

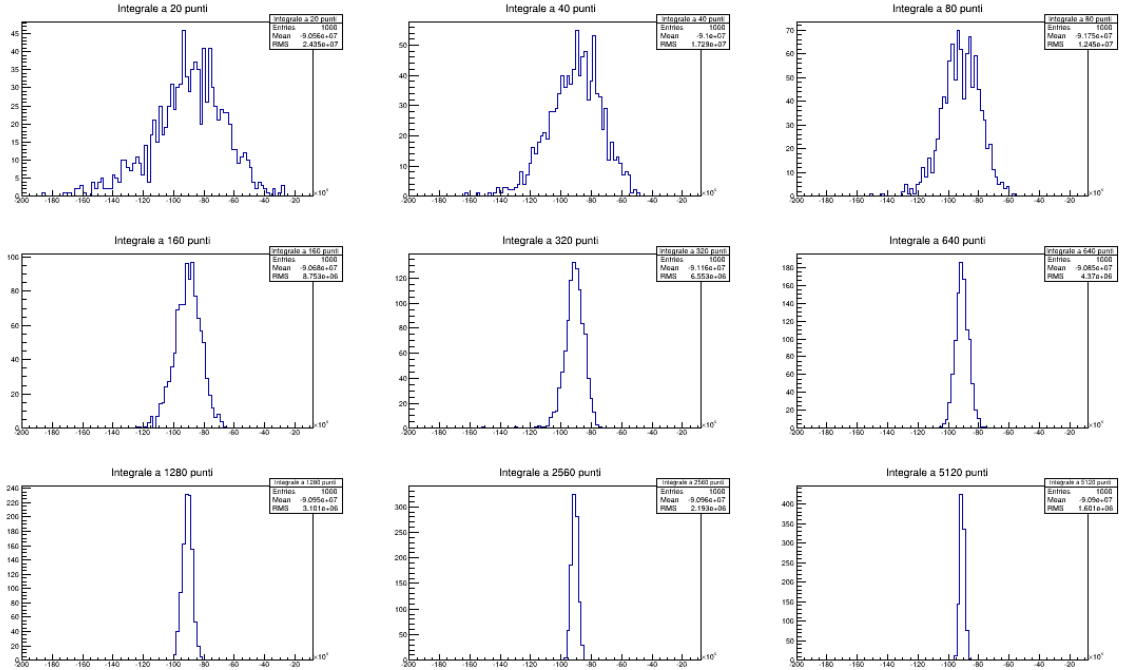
$$V(\vec{r}) = -\int_{-R}^{R} dx' \int_{-R}^{R} dy' \int_{-R}^{R} dz' G \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|} f(\vec{r}')$$

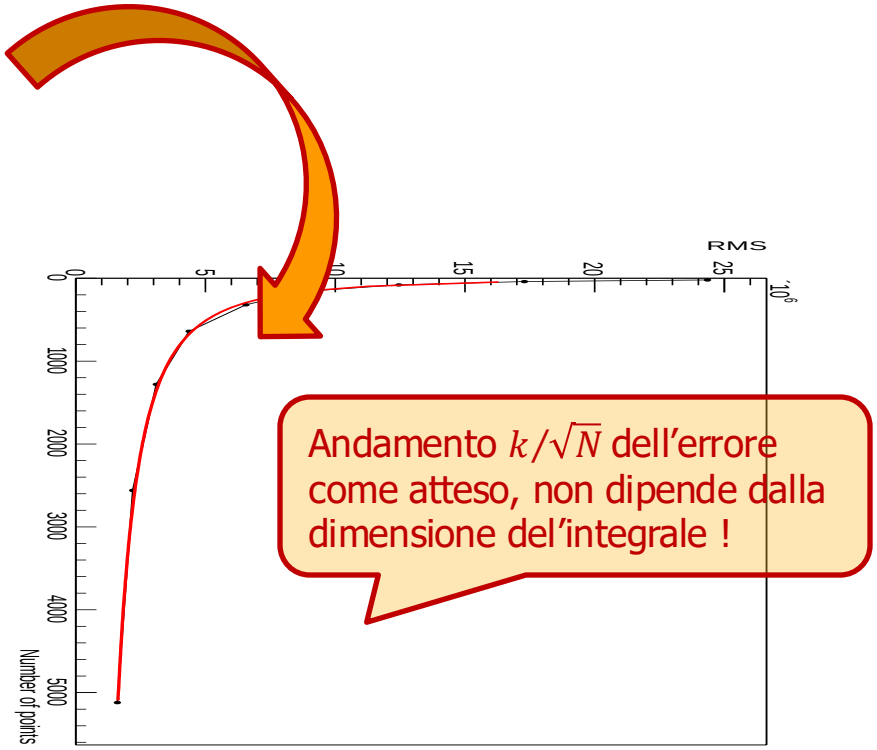
$$f(\vec{r}') = \begin{cases} 0 \; se \; |\vec{r}'| > R \\ 1 \; se |\vec{r}'| < R \end{cases}$$

Posso utilizzare il metodo della media :

- Estraggo (uniformemente) una terna di punti in un cubo di lato 2R che contenga la terra
- Per ogni terna calcolo il valore della funzione (che in questo caso è una funzione scalare)
- Eseguo i punti precedenti un numero N di volte e calcolo la media della funzione nei punti estratti
- Moltiplico questo numero per il volume di estrazione $(2R)^3$

Un esempio di integrale multidimensionale





Una funzione integrale !

$$V(\vec{r}) = -\int dx'dy'dz' G \frac{\rho(\vec{r}')}{|\vec{r} - \vec{r}'|}$$

Ripetiamo il calcolo dell'integrale 3D variando la posizione del punto (in funzione della distanza dal centro della sfera)

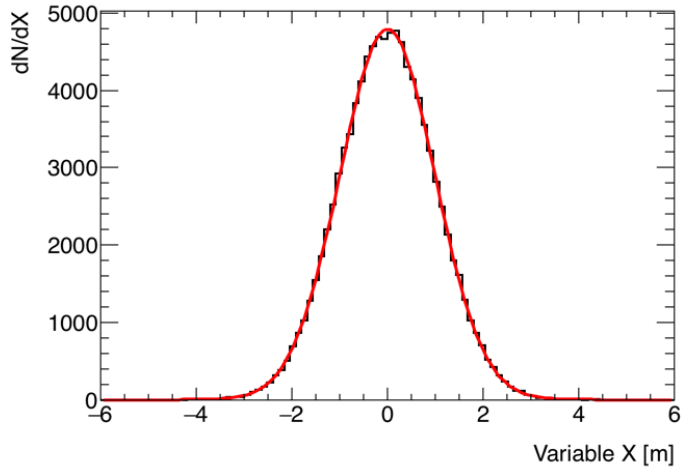
$|\vec{r}|$

Esercizio10.2 : eseguire un fit

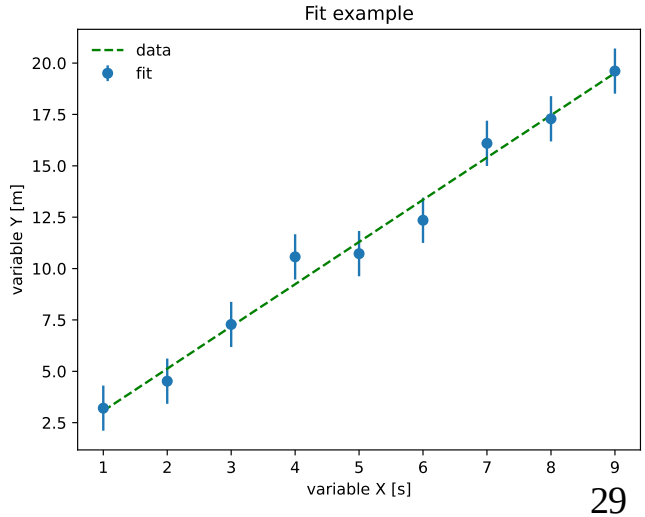
Approfondimento 1

L'esercizio 10.2 ci da lo spunto per un primo approfondimento su cosa significa fare un fit. Possiamo considerare due casi :

1. Abbiamo effettuato N misure di una certa quantità X e vogliamo capire se un certo modello teorico (funzione) descrive la distribuzione di X.



2. Abbiamo N coppie di misure (X, Y) e vogliamo capire se un certo modello teorico (funzione) descriva l'andamento osservato di Y al variare di X



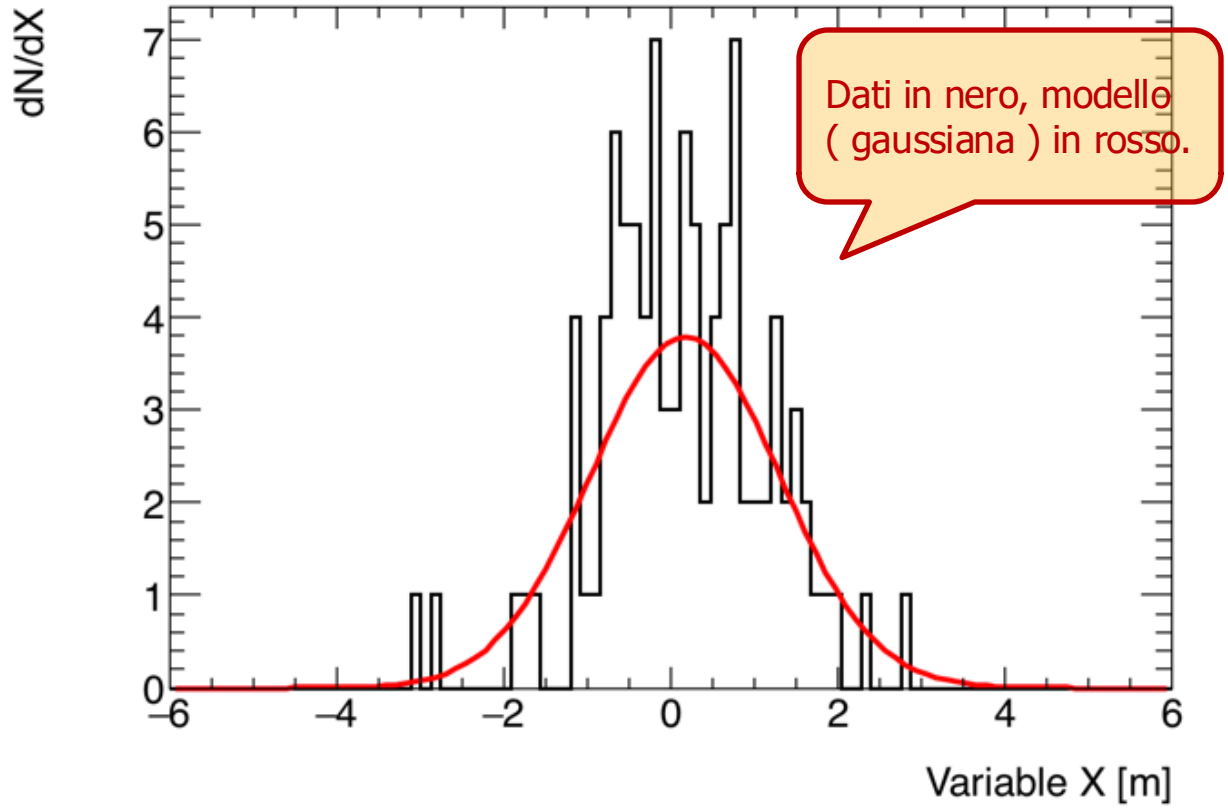
Eseguire un fit : caso 1

Approfondimento 1

- Si ha a disposizione una serie di misurazioni di una variabile e tipicamente si studia la distribuzione di questa variabile mediante un istogramma.
- Si vuole capire se il set di dati potrebbe provenire da una data distribuzione teorica (tipicamente descritta da una funzione):
  - Primo: si vorrebbe verificare se il modello (funzione) che si sta assumendo è accettabile: il modello rappresenta davvero i miei dati?
  - Secondo: per una dato modello (funzione) si vogliono estrarre i valori dei parametri del modello che danno il miglior accordo con i dati
- Fare un fit significa cercare i valori dei parametri tali che il modello abbia il miglior accordo con i dati
  - Definire uno stimatore (figura di merito per decidere cosa è meglio)
  - Minimizzatore (algoritmo per minimizzare la figura di merito): varia in modo intelligente i valori dei parametri fino a trovare il minimo valore dello stimatore
- In genere, sul mercato sono disponibili pacchetti per eseguire queste operazioni facilmente: ROOT (ad esempio) fornisce l'infrastruttura utilizzando Minuit come minimizzatore

Eseguire un fit : caso 1

Approfondimento 1



Quale figura di merito minimizzare ?

Approfondimento 1

- Si può costruire uno stimatore a piacere ma ce n'è uno che ha proprietà interessanti, il χ^2

$$\chi^2 = \sum_{i=1}^{n} \frac{(o_i - e_i)^2}{e_i}$$

Dove O_i è il numero osservato di eventi in un bin mentre e_i è il numero previsto (dal modello) di eventi in un bin (ed n è il numero di bin riempiti dell'istogramma).

- Perché il χ^2? Perché conosciamo la distribuzione di questa variabile!
  - Si può usare la distribuzione per stimare la bontà del fit

- Lo stimatore costruito sopra è distribuito come la seguente variabile (k qui è n-# di parametri della funzione di fit):

$$\chi^2(k) = \sum_{i=1}^{k} x_i^2 = x_1^2 + \ldots + x_k^2$$

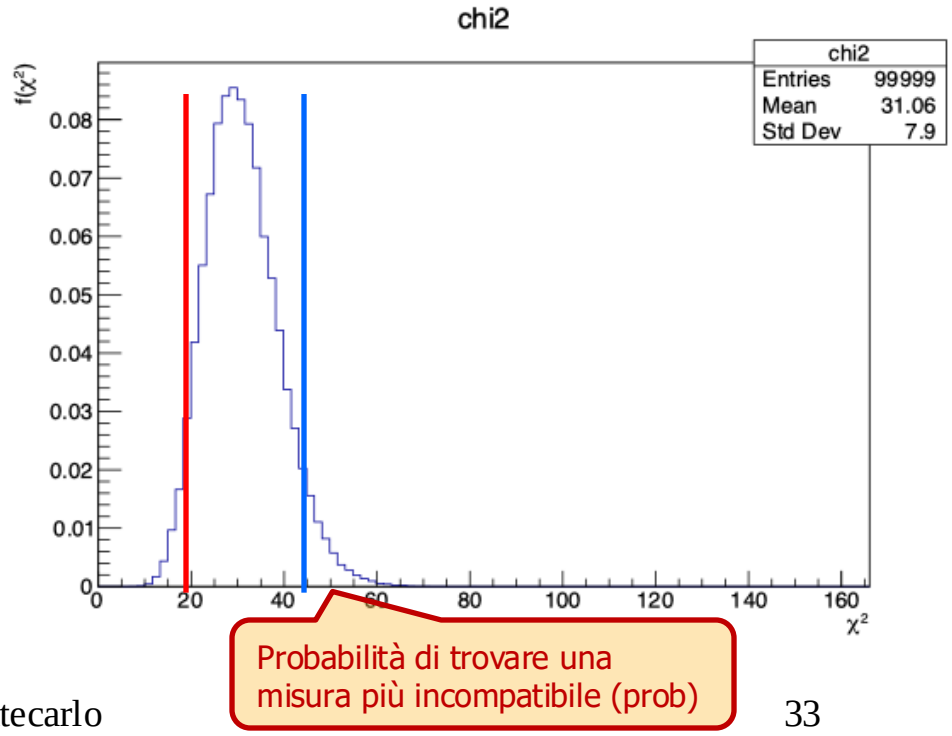
dove x_i sono variabili indipendenti distribuite come N(0,1)

Il test di χ^2 (test di ipotesi)

Approfondimento 1

- Definire un'ipotesi nulla : il dataset è una rappresentazione di una variabile distribuita secondo la distribuzione teorica m(x,**p**) dove **p** è un set di parametri
- Definire un livello di confidenza che siamo disponibili a tollerare (tipicamente 5%).
- Considerare la distribuzione $f(\chi^2)$ del χ^2 per uno specifico numero di gradi di libertà e calcolare la soglia χ^2_α per cui $P(\chi^2 > \chi^2_\alpha) = 0.05$
- Calcolare il valore χ^2_m sui dati a rigettare l'iptotesi nulla $\chi^2_m > \chi^2_\alpha$

- Assumiamo un fit con 31 DoF : si trova χ^2_m = 20 (rosso) che è chiaramente molto più piccolo di χ^2_α (blu): non posso rigettare l'ipotesi nulla quindi il modello deve essere accettato
- Se $\chi^2_m >> \chi^2_\alpha$ allora è molto improbabile trovare un altro dataset più incompatibile con il modello quindi posso rigettare l'ipotesi nulla
- Operativamente si usa la funzione `Prob()` per calcolare $P(\chi^2 > \chi^2_m)$ e poi si compara questo valore con 0.05 : se $P(\chi^2 > \chi^2_m) > 0.05$ non posso rigettare l'ipotesi nulla



Esempio di fit usando ROOT (Minuit)

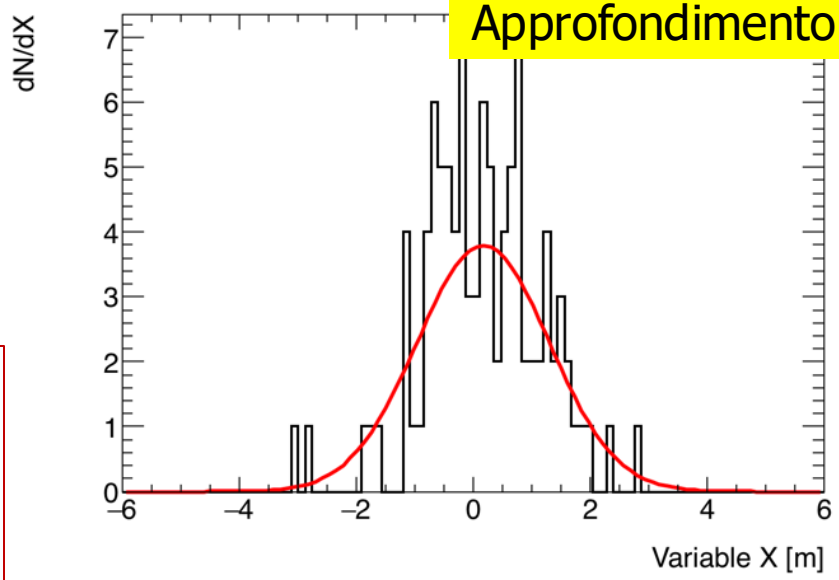
Approfondimento 1

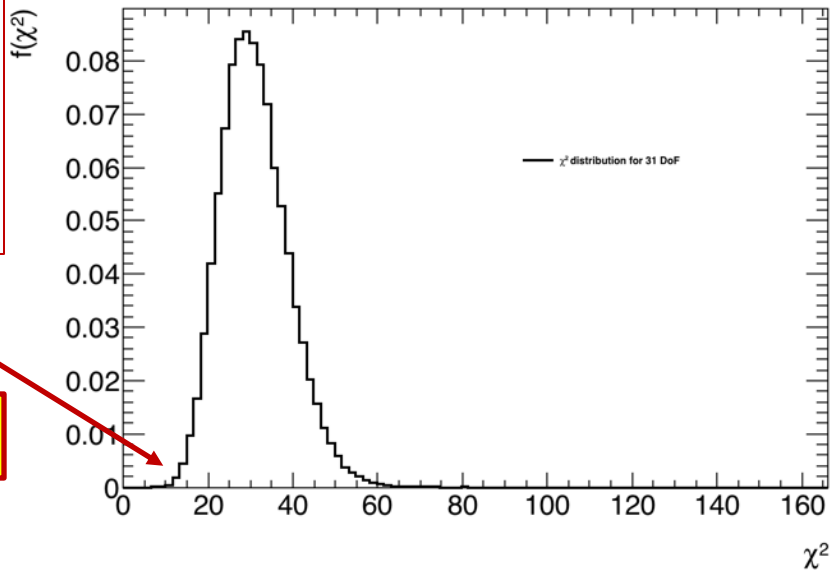
```
// put data into an histogram

TH1F data_small ("data_small","data_small",100, -6, 6 ) ;
for ( int k = 1 ; k < 100 ; k ++ ){ data_small.Fill( ... ); };

// create the fitting function ( already in ROOT )

TF1 * gaus_small = new TF1("gaus","gaus",-10,10);

// setup plots

TCanvas * can_small = new TCanvas("test_small","test_small");
can_small->cd();

// set cosmetics

data_small.Draw();

// Perform the fit and access the results

data_small->Fit(gaus_small);

cout << "Fit small data : Prob = " << gaus_small->GetProb() << endl;
```

Esempio di fit usando ROOT (Minuit)

Approfondimento 1

```
 FCN=19.9441 FROM MIGRAD    STATUS=CONVERGED      76 CALLS          77 TOTAL
                     EDM=4.0448e-07    STRATEGY= 1      ERROR MATRIX ACCURATE
  EXT PARAMETER                                   STEP         FIRST
  NO.   NAME      VALUE            ERROR          SIZE      DERIVATIVE
   1  Constant     3.79521e+00   6.17374e-01   9.53237e-04  -1.07965e-03
   2  Mean         1.71244e-01   1.58176e-01   3.51747e-04  -1.27424e-03
   3  Sigma        1.13353e+00   2.03254e-01   9.86157e-05   4.06094e-03

****************************************
Minimizer is Minuit / Migrad
Chi2                      =      19.9441
NDf                       =           31
Edm                       =   4.0448e-07
NCalls                    =           77
Constant                  =      3.79521   +/-   0.617374
Mean                      =     0.171244   +/-   0.158176
Sigma                     =      1.13353   +/-   0.203254     (limited)
Fit small data : Prob = 0.936994
```



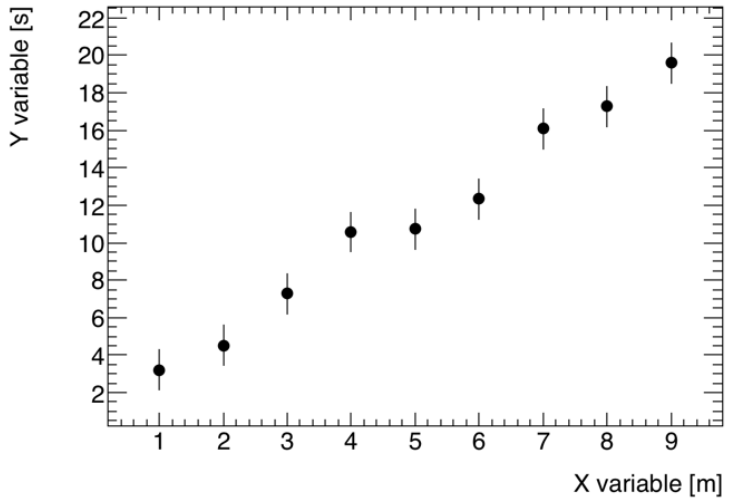


Good fit !

Eseguire un fit : caso 2

Approfondimento 1

- In laboratorio si misura il valore di una variabile Y in funzione di un'altra variabile X e si disegna un grafico (assumiamo di avere incertezze solo su Y) :



- Vorrei controllare se il mio modello (una funzione) è in accordo con le misure
  - Come nel caso precedente il modello è parametrico (si conosce la forma funzionale ma non i valori esatti dei parametri che verranno determinati dal fit sui dati)

$$\chi^2 \equiv \sum_{i=1}^{n} \frac{(x_i - x_{ti})^2}{\sigma_i^2}$$

- Come prima si costruisce questo stimatore che si distribuisce come un χ^2 con (N_{points}-N_{para}) DoF !

Eseguire un fit : caso 2

Approfondimento 1

```
int main() {

  TApplication myApp("myApp",0,0);

  // open file

  ifstream file ;
  file.open("data.dat") ;

  // define TGraphErrors

  TGraphErrors * mygr = new TGraphErrors();

  // read values from file and fill the TGraphErrors

  double x,y,err;

  for ( int k =0 ; k < 9 ; k++ ) {
    file >> x >> y >> err ;
    mygr->SetPoint(k, x, y );
    mygr->SetPointError(k,0, err);
  }

  file.close();

  // Draw

  TCanvas * can_tgraph = new TCanvas("can_tgraph","can_tgraph");
  can_tgraph->cd();
  mygr->SetTitle("Measurement");
  mygr->GetXaxis()->SetTitle("X variable [m]");
  mygr->GetYaxis()->SetTitle("Y variable [s]");
  mygr->Draw("A*");

  // define the fitting function, set initial values and fit. Then see results

  TF1 * fit_fun = new TF1("fit_fun","[0]*x + [1]",0,10);
  fit_fun->SetParameter(0,1.5);
  mygr->Fit( fit_fun );
  cout << "First parameter " << fit_fun->GetParameter(0) << endl;
  cout << "Second parameter " << fit_fun->GetParameter(1) << endl;
  cout << "Chi2 prob = " << fit_fun->GetProb() << endl;

  myApp.Run();

}
```

Assegna valori iniziali ragionevoli al parametro 0

Lancia il fit

Modello da verificare è la dipendenza lineare

Eseguire un fit : caso 2

Approfondimento 1

```
{

  ifstream file ;
  file.open("data.dat") ;

  TGraphErrors * mygr = new TGraphErrors();

  double x,y,err;

  for ( int k =0 ; k < 9 ; k++ ) {
    file >> x >> y >> err ;
    mygr->SetPoint(k, x, y );
    mygr->SetPointError(k,0, err);
  }

  file.close();

  // Draw

  TCanvas * can_tgraph = new TCanvas("can_tgraph","can_tgraph");
  can_tgraph->cd();
  mygr->GetXaxis()->SetTitle("X variable [m]");
  mygr->GetYaxis()->SetTitle("Y variable [s]");
  mygr->Draw("AP");

  // define the fitting function, set initial values and fit. Then see results

  TF1 * myfit = new TF1("myfit","[0]*x + [1]",0,10);
  cout << "First parameter " << myfit->GetParameter(0) << endl;
  myfit->SetParameter(0,1.5);
  TFitResultPtr res1 = mygr->Fit(myfit,"S","",1,9);
  cout << "First parameter " << myfit->GetParameter(0) << endl;

  res1->Print();
  cout << myfit->GetProb() << endl;

}
```

Si può anche eseguire in forma di script interpretato (cling) direttamente da shell senza compilazione :

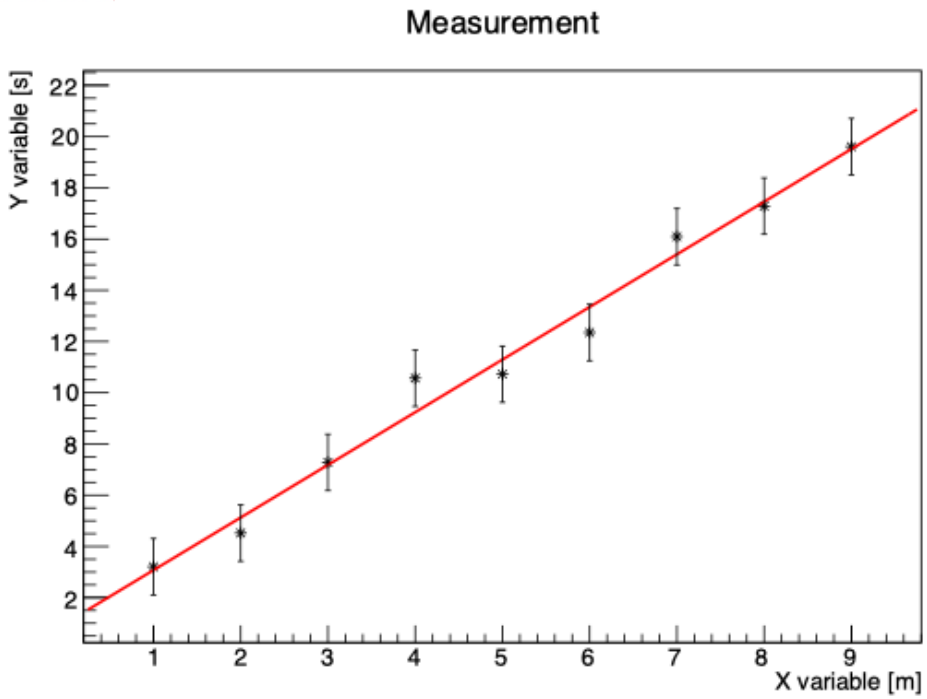
```
root test_root.C
```

Eseguire un fit : caso 2

Approfondimento 1

```
lcarmina@Leonardos-MBP-4 fit % ./fit_root
****************************************
Minimizer is Minuit2 / Migrad
Chi2                      =      3.30522
NDf                       =            7
Edm                       =   1.5023e-22
NCalls                    =           32
p0                        =      2.05528   +/-   0.142009
p1                        =      1.01647   +/-   0.799131
First parameter 2.05528
Second parameter 1.01647
Chi2 prob = 0.855405
```

La prob è grande : non posso rigettare l'ipotesi nulla, accetto il modello lineare



Attenzione : la minimizzazione è un'arte

Approfondimento 1

- Normalmente non possiamo semplicemente lanciare ciecamente un fit e prendere sempre per buoni i risultati. Molto spesso l'algoritmo va "imboccato" con un set di valori iniziali ragionevoli per i parametri che stiamo cercando
- Questo non significa che stiamo barando, in questo caso l'algoritmo convergerà più velocemente al minimo vero (se esiste) e si riduce la probabilità che finisca in un minimo secondario

```
TF1 * fit_fun = new TF1("fit_fun","[0]*x + [1]",0,10);
fit_fun->SetParameter(0,1.5);
mygr->Fit( fit_fun );
```

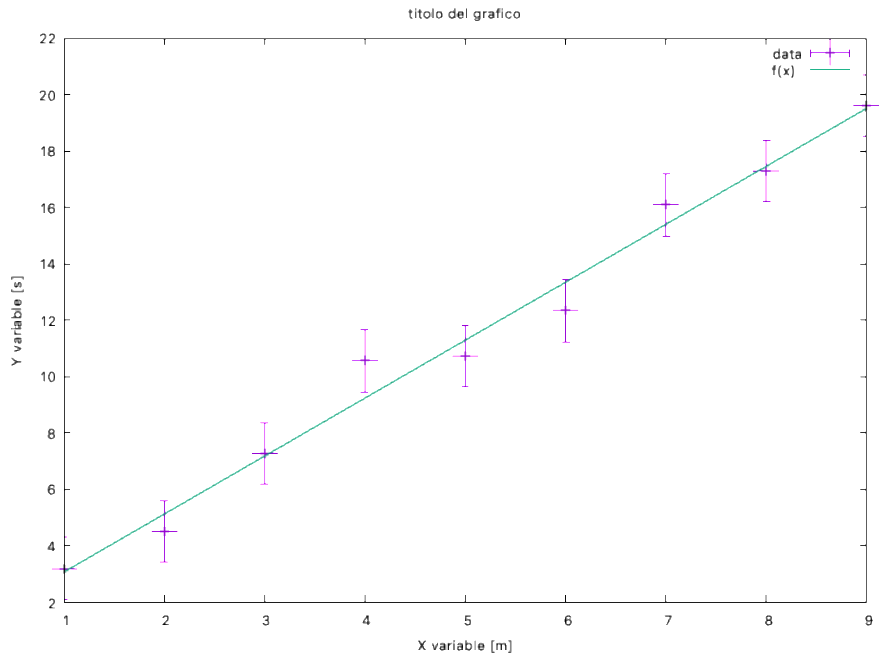
Flessibilità : vi piace gnuplot ?

Approfondimento 1

```
#definisce il terminale
set term qt

# definizione dei titoli per gli assi e per il grafico
set title  "titolo del grafico"
set xlabel "X variable [m]"
set ylabel "Y variable [s]"

# definizione della funzione
f(x) = a *x + b

# valori delle costanti
a=1.5; b=4.5;

fit f(x) 'data.dat' u 1:2:3 yerr via a,b

pl 'data.dat' u 1:2:3 title 'data' w ye points 3, f(x)
```

Da eseguire con i seguenti comandi :

```
gnuplot
gnuplot>load "test_gnuplot.gnu"
```



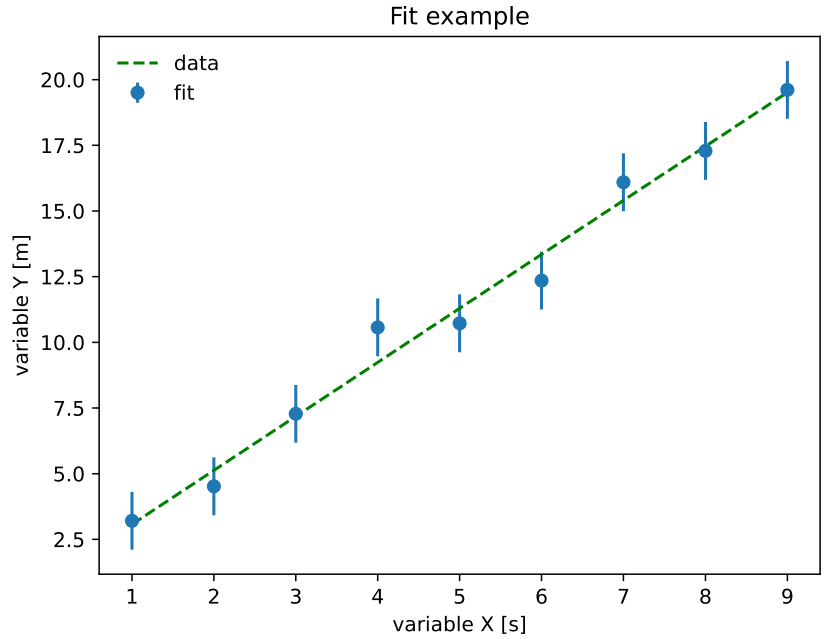
Flessibilità : vi piacciono python/matplotlib ?

Approfondimento 1

```
import numpy as np
from numpy import pi, r_, loadtxt
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

data = loadtxt('data.dat')
x = data[:, 0]
y = data[:, 1]
err = data[:, 2]

def func(x, a, b ):
    return a * x + b

print (x)
print (y)
print (err)

plt.errorbar(x, y, yerr=err , label='data' , fmt='o')

popt, pcov = curve_fit(func, x, y, sigma=err)
print (popt)

chisq = sum(((y-func(x,*popt))/err)**2)
print (chisq)

plt.plot(x, func( x , *popt), 'g--', label='fit')

plt.title("Fit example")
plt.xlabel("variable X [s]")
plt.ylabel("variable Y [m]")
plt.legend(('data', 'fit'))

plt.show()
```

Da eseguire con il seguente comando :

```
python test_python.py
```



Flessibilità

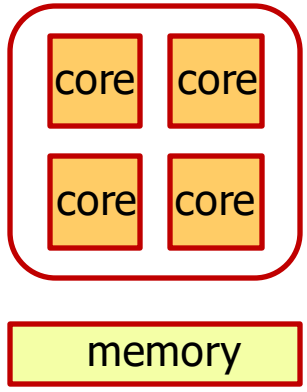
Approfondimento 1

- In linea di principio potremmo continuare con molte altre opzioni : si potrebbe guardare a pyROOT o matlab (con licenza)
- Il linguaggio perfetto o lo strumento perfetto per tutto non esistono, diffidate dei fondamentalisti
- Il segreto è essere flessibili, intelligenti ed eclettici: scegliere il linguaggio, progettare il codice e utilizzare gli strumenti che meglio si adattano alle esigenze del caso.

Esercizio10.2 : velocizzare la generazione

Approfondimento 2

- I moderni processori sono costituiti da più computing units (cores) in ogni CPU : i nostri programmi sono tipicamente eseguiti da un solo core mentre normalmente gli altri stanno a guardare.
  - È possible scrivere codice in modo che il carico sia distrubuito su tutti i cores di una macchina ?

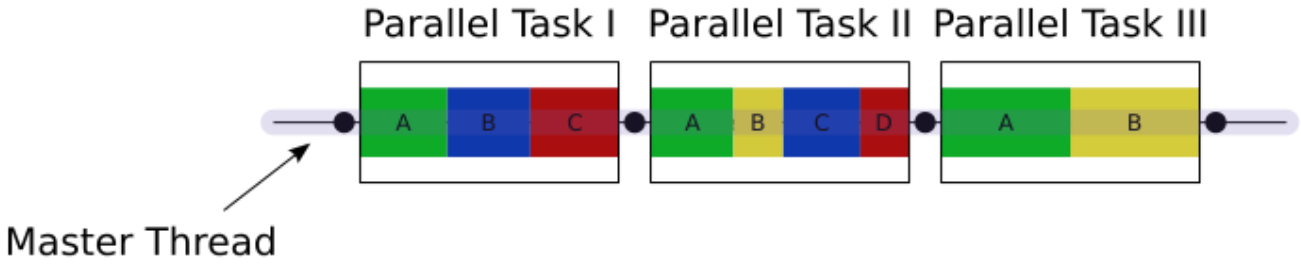


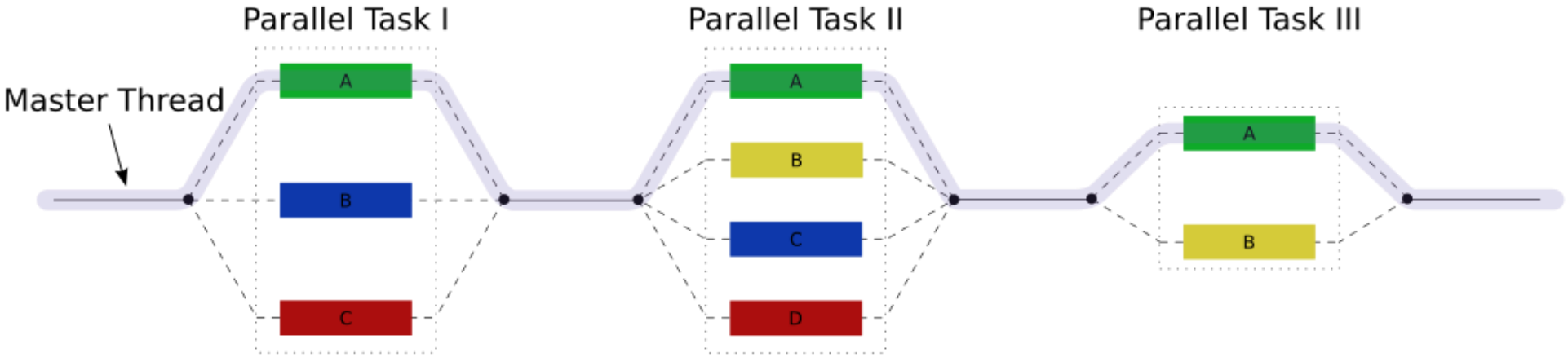
  - Si può utilizzare per esempio `openMP` che permette di generare diversi threads che vengono eseguiti in parallelo su cores diversi

The fork/join model

Approfondimento 2

- I programmi con OpenMP partono con un singolo thread, il *master thread*
- All'inizio della parte parallela il master crea (FORK) un insieme di thread paralleli
- Le istruzioni nella regione parallela vengono eseguiti (in parallelo) da ogni thread
- Alla fine della regione parallela tutti i threads si sincronizzano e ritornano (JOIN) nel master





A simple parallelization example : openMP

Approfondimento 2

OpenMC viene fornito come API standard (*) per scrivere applicazioni parallele a memoria condivisa in C, C++ e Fortran

L'API OpenMP è composta da:

- Direttive del compilatore
  - Ex `#pragma omp parallel` :
    - per definire la parte parallela del codice
  - Subroutine/funzioni runtime
    - `omp_get_num_procs()` (restituisce il numero di processori nella macchina)
    - `omp_get_thread_num()` (restituisce il numero del thread)
  - Variabili d'ambiente
    - export OMP_NUM_THREADS=4 (*)

(*) Nella programmazione informatica, una Application Programming Interface (API) è un insieme di definizioni, di subroutine, protocolli e strumenti per la creazione di software applicativo.

Threads

Approfondimento 2

- Il thread è una sequenza indipendente di esecuzione del codice
  - Blocco di codice con una entrata e una uscita
- Per i nostri scopi il thread è un concetto più astratto
- I thread non sono correlati a core/CPU
- I thread OpenMP sono mappati su core fisici
- È possibile mappare più di 1 thread su un core
- In pratica è meglio avere una mappatura uno a uno.

Parallelizzazione codice

Approfondimento 2

```
  double IntegraleAVE(double xmin, double xmax, FunzioneBase * f, int punti, int ncores , unsigned int seed )
  {

    double I = 0;
    double startTime = omp_get_wtime();

    // here begins the part of the code which will be executed in parallel

#pragma omp parallel num_threads( ncores )
    {
      double fmed=0;
      // note that I generate separate generators per thread. Each generators
      // needs a different seed
      unsigned int local_seed = seed + omp_get_thread_num();
      Random myGen (local_seed) ;
      unsigned int counter = 0; // this is to check how omp for works

#pragma omp critical  // just a debugging thing, should not slow down too much
      {
        cout << "  [debug] Seed in thread " << omp_get_thread_num() << " is " << local_seed << endl;
      }

      // this is the real magic
#pragma omp for
      for (int i=0; i<punti;  i++ ) {
        fmed += f->Eval( myGen.Rand(xmin,xmax)  ) ;
        counter++;
      } ;

      // put everything together. critical is not needed, just to see correctly the output messages
#pragma omp critical
      {
        I = I + fmed;
        cout << "  [debug] counter in thread " << omp_get_thread_num() << " ==>> " << counter << endl;
      }
    }
    I= I/ double(punti) * (xmax-xmin);

    double stopTime = omp_get_wtime();
    double secsElapsed = stopTime - startTime;

    cout << "It took me " << secsElapsed
         << " seconds to compute the integral as " << I
         << " with " << ncores << " threads " << endl;

    return I ;

  };
```

Parallelizzazione codice

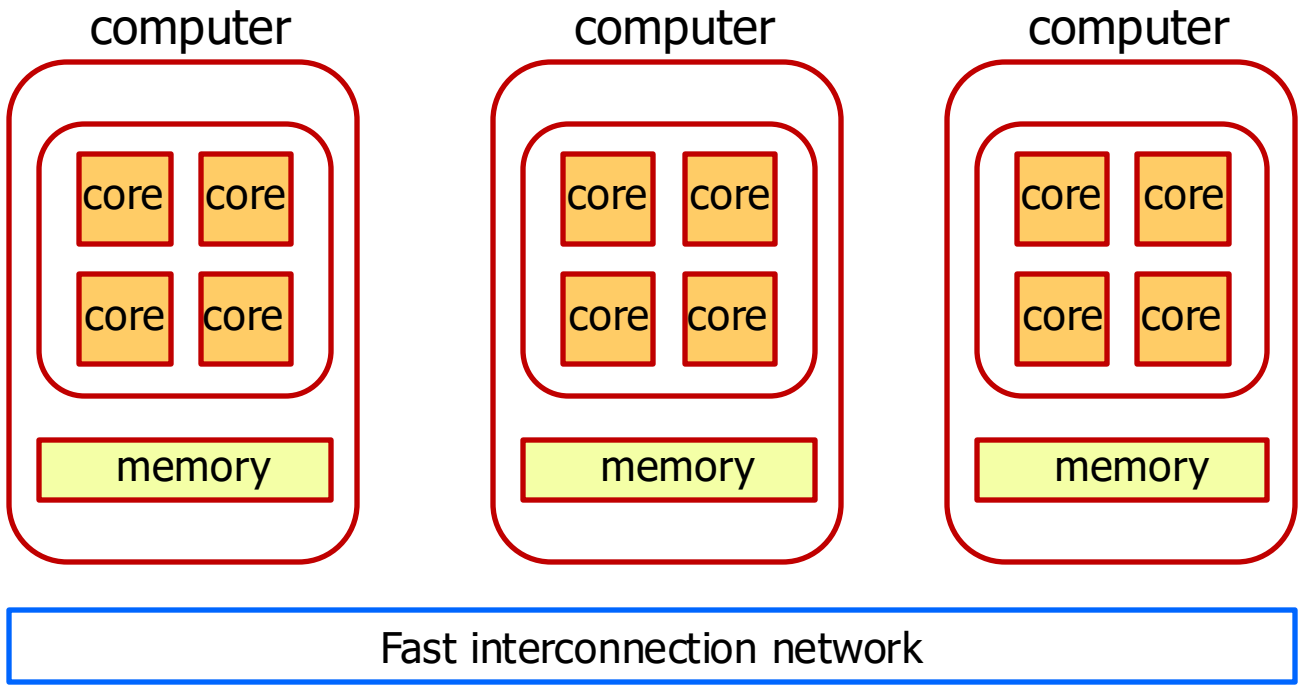
Approfondimento 2

See a couple of online examples on replit https://replit.com/join/vefatiqcan-lcarmina

Modelli complessi : generazione parallela

Approfondimento 2

Caso in cui i vari processi di soluzione di un problema complesso comunicano tra loro (Message Passing Interface – MPI)



Generazione parallela e batch system

Caso di parallelismo imbarazzante : le parti non comunicano tra loro (caso tipico l'analisi dei dati di un esperimento) si può usare un batch system (eg condor)