# Metodi Computazionali della Fisica
# Secondo Modulo: C++

Prima Lezione

# In breve

[fragile]
Obiettivi:

- imparare a sviluppare e gestire un progetto di programmazione;
- conoscere tecniche di integrazione numerica;
- sviluppare un codice funzionante basato su tecniche Monte Carlo.

Contenuti, orario e risorse:
`http://www.ge.infn.it/~piccione/metodicomputazionali/`

Codice di ogni lezione su `PCTEOR1`:
`/home/piccione/metodi_computazionali_2/2008-2009`

# La lezione di oggi

Obiettivo:

- implementare una classe `Integral` che fornisca l'inetgrale di una funzione di una variabile data con diversi metodi.

Contenuti:

- formule di Newton-Cotes;
- formule di integrazione con quadratura gaussiana.

Strumenti:

- le librerie `iostream, iomanip, cmath`;
- la routine `DGAUSS`.

# Trapezoidal rule - 1

- Le formule di Newton-Cotes approssimano un integrale su un intervallo finito con la somma pesata dei valori dell'integranda calcolati a valori equispaziati della variabile di integrazione.
- Il caso più semplice è quello trapezioidale

$$\int_{x_0}^{x_0+\Delta x} dx f(x) = \frac{\Delta x}{2}\left[f(x_0) + f(x_0+\Delta x)\right] + \ldots$$

con $x_0 \leq \xi \leq x_0 + \Delta x$.

# Trapezoidal rule - 2

Per calcolare con questa formula un integrale sull'intervallo $[x_0, x_n]$, ne prendiamo $n$ suddivisioni di larghezza $\Delta x$ e applichiamo la regola trapezioidale su ogni suddivisione. Usando $x_j = x_0 + j \cdot \Delta x$, troviamo

$$\int_{x_0}^{x_n} dx f(x) = \frac{x_n - x_0}{n} \sum_{j=0}^{n} w_j f(x_j) + \mathcal{O}\left(\frac{1}{n^2}\right)$$

con $w_0 = w_n = 1/2$ e $w_j = 1$ per $1 \leq j \leq n-1$.

# Esercizio 1

Implementare la regola trapezioidale e testarla su una funzione nota.

```
#include <iostream>
#include <iomanip>
#include <cmath>

using namespace std;

double f(double x,size_t ifunc_type){
  //Funzione di test per l'integranda
  if(ifunc_type == 1){
    return sin(x);
  }
  else if(ifunc_type == 2){
    return 1/x;
  }
};

double intf(double x,size_t ifunc_type){
  //Funzione di test per la primitiva
  if(ifunc_type == 1){
    return -cos(x);
  }
  else if(ifunc_type == 2){
    return log(x);
  }
};
```

# Esercizio 1

```
int main(int argc,char* argv[]){

  if(argc < 3){
    cerr<<"args missing"<<endl;
    cerr<<" "<<endl;
    cerr<<
      "Usage: ./main x_low x_high ifunc_type"
<<endl;
    cerr<<" "<<endl;
    exit(-1);
  };

  double x1=atof(argv[1]);
  double xn=atof(argv[2]);
  size_t ifunc_type=atoi(argv[3]);

  cout<<"Closed Newton-Cotes Formulas: "<<endl;
  double check=intf(xn,ifunc_type)-intf(x1,ifunc_type);
  cout<<"expected: "<<setw(5)<<" "<<setprecision(8)<<check<<endl;

  double tmp=0.5*f(x1,ifunc_type)+0.5*f(xn,ifunc_type);
  double h=(xn-x1);
  tmp*=h;
  cout<<"trapezoidal: "<<setw(2)<<" "<<setprecision(8)<<tmp<<endl;

  return 0;
};
```

$\longrightarrow$ Generalizzare il codice per il caso di $n$ suddivisioni.

# Simpson's rule

La regola di Simpson calcola l'integrale in tre punti:

$$\int_{x_0}^{x_2} dx f(x) = \frac{\Delta x}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] + \dots$$

quindi su $n$ suddivisioni:

$$\int_{x_0}^{x_n} dx f(x) = \frac{x_n - x_0}{n}\sum_{j=0}^{n} w_j f(x_j) + \mathcal{O}\left(\frac{1}{n^4}\right)$$

con $n$ pari, $w_0 = w_n = 1/3$, e per $1 \leq j \leq n$ abbiamo $w_j = 4/3$ se $j$ è dispari e $w_j = 2/3$ se $j$ è pari.

# Esercizio 2

- Implementare la regola di Simpson e testarla su una funzione nota.
- Generalizzare il codice per il caso di $n$ suddivisioni.

# Definizioni - 1

1. Il prodotto scalare di due funzioni $f$ e $g$ rispetto ad una funzione peso $W$ si definisce come

$$\langle f|g\rangle = \int_a^b dx\ W(x)f(x)g(x)\,.$$

2. Due funzioni sono *ortogonali* se il loro prodotto scalare è zero.
3. Una funzione è *normalizzata* se il prodotto scalare con se stessa è uguale a 1.
4. Un set di funzioni è *ortonormale* se ogni funzione è normalizzata e se le funzioni sono tra di loro ortogonali.

# Definizioni - 2

Esistono set di polinomi che

1. hanno un solo elemento per ogni valore di $j$, $p_j(x)$, per ogni $j = 0, 1, 2, \ldots$;
2. sono ortogonali rispetto ad una data funzione peso $W(x)$.

Il set di polinomi si può costruire con la relazione di ricorrenza:

$$\begin{aligned} p_{-1}(x) &\equiv 0 \\ p_0(x) &\equiv 1 \\ p_{j+1}(x) &= (x - a_j)p_j(x) - b_j p_{j-1}(x) \end{aligned}$$

con

$$a_j = \frac{\langle x p_j | p_j \rangle}{\langle p_j | p_j \rangle} \qquad b_j = \frac{\langle p_j | p_j \rangle}{\langle p_{j-1} | p_{j-1} \rangle}$$

# Gauss-Legendre

L'integrale di una funzione $f$ si può scrivere come

$$\int_a^b dx f(x) \simeq \sum_{i=1}^{n} w_i f(x_i)\,,$$

con

$$w_j = \frac{\langle p_{n-1}|p_{n-1}\rangle}{p_{n-1}(x_j)p'_n(x_j)}$$

dove $p'_n(x_j)$ è la derivata del polinomio $n$-esimo e $x_1, \ldots, x_n$ sono le soluzioni di $p_{n+1}(x) = 0$. Nel caso di Gauss-Legendre

$$\begin{aligned} W(x) &= 1 \qquad -1 < x < 1 \\ (j+1)p_{j+1} &= (2j+1)xp_j - jp_{j-1}\,. \end{aligned}$$

# DGAUSS

DGAUSS è una routine delle librerie del CERN che utilizza una quadratura guassiana *adattabile*.

Dato un integrale $\int_a^b dx f(x)$

1. calcola l'integrale con una quadratura di Gauss-Legendre a 8 e a 16 punti;
2. se la differenza tra le due quadrature è minore dell'accuratezza richiesta, il risultato è l'integrale a 16 punti, altrimenti prosegue;
3. divide l'intervallo in due e riparte da 1 fino a quando la somma degli intervalli non e' uguale a quello iniziale.

# Esercizio 3

Implementare la classe definita dal seguente header file

```
#ifndef _INTEGRAL_HPP
#define _INTEGRAL_HPP

#include <iostream>
#include <iomanip>
#include <cmath>

enum {NOMETH,TRAP,SIMP,DGAU};
enum {NOFUNC,LIN,SIN,LOG};

using namespace std;

class Integral1d {

public:

  Integral1d();
  Integral1d(size_t imethod,size_t ifunc_type,double EPS);
  Integral1d(const Integral1d&);
  ~Integral1d(){};

  double integrate(double xmin,double xmax);
  double check(double xmin,double xmax);

private:

  double func(double x);
  double primitive(double x);
  double dgauss(double xmin,double xmax);

  size_t _imethod, _ifunc_type;
  double _EPS;

};
#endif
```