Homework 4: Integração numérica e Random Walk

Fis. Computacional 2022-23 (P4)

Fernando Barão

6 de junho de 2023

Integração de funções

Pretende-se integrar a seguinte função uni-dimensional entre os limites $[0, 2]$.

$$f(x) = x^4 \log(x + \sqrt{x^2 + 1})$$

Para tal propomos que use as classes `Functor` , classe de base das funções e a classe `MyFunction` onde deve definir a função integranda.

classe Functor

```
#ifndef __FUNCTOR__
#define __FUNCTOR__

#include <string>

#include "TCanvas.h"

class Functor {

 public:
  Functor(std::string s="Functor") : name(s) {;}
  ~Functor() = default;

  virtual double operator()(double x);

  // args:
  // xi, xf ........... xmin and xmax limits for function display
  // num .............. number of sampling points to be used o TGraph
  // xtitle, ytitle ... axis titles
  virtual void Draw(double xi, double xf, int num, std::string
   ↪  xtitle="x", std::string ytitle="y");

protected:
  static TCanvas *c;
  std::string name;
};

#endif
```

classe MyFunction

```
#ifndef __MYFUNCTION__
#define __MYFUNCTION__

#include "Functor.h"

class MyFunction : public Functor {

 public:
  MyFunction(std::string s="MyFunction") : Functor(s) {;}
  ~MyFunction() = default;

  double operator()(double x) {
     // implement here the function f(x)
     (...)
  }
};

#endif
```

1. Determine o integral da função com os métodos Trapezoidal e Simpson.

   A classe `IntegDeriv` deverá ser implementada com os métodos de derivação e integração.

   classe Derivator/Integrator

```
#ifndef __INTEGDERIV__
#define __INTEGDERIV__

#include "Functor.h"

class IntegDeriv {

public:
 IntegDeriv(Functor&);
 ~IntegDeriv() = default;

 // integration methods

 void TrapezoidalRule(double xi, double xf, double& Integral, double&
  ↪  Error);
 void simpsonRule(double xi, double xf, double& Integral, double&
  ↪  Error);
```

```
 // derivative methods

 (...)

private:
  Functor& F;
};

#endif
```

Random walk e difusão

A difusão corresponde ao movimento aleatório de corpos num dado meio físico e em resultado da sua interação com o meio. Este fenómeno foi identificado pela primeira vez pelo botânico escocês Robert Brown no século 19, ao observar o movimento de partículas de pólen na água. Daí que este tipo de movimento aleatório tenha passado a designar-se como movimento **Browniano**. Este fenómeno permaneceu inexplicado…

O ano de 1905 é considerado um ano de referência para a Física. Porquê? Einstein deu contributos para três problemas abertos da Física e que exigiam uma visão radicalmente diferente. Concretamante, a explicação do movimento browniano, a introdução da dos quanta de luz e a Relatividade. A publicação de Einstein *"On the movement of small particles suspended in a stationary liquid demanded by the molecular-kinetic theory of heat"* refere o movimento browniano como uma marcha aleatória no espaço resultado das transferências de momento linear ($\Delta\vec{p}$) para as partículas de pólen em direções aleatórias, por parte das moléculas de água. Esta explicação foi pioneira e abriu definitivamente a via experimental para a explicação da matéria como sendo constituída por átomos e moléculas.

Random walk uni-dimensional (1D)

No sentido de extraírmos as características do movimento difusivo, vamos proceder à simulação a uma dimensão da marcha aleatória. Vamos considerar partículas que se podem deslocar ao longo do eixo do xx, partindo no instante $t = 0$, de $x = 0$ e que obedecem às seguintes regras:

- cada partícula pode deslocar-se para a direita ($-\delta$) ou para a esquerda ($+\delta$), em cada intervalo de tempo τ.
- a probabilidade em cada passo de a partícula se deslocar para a esquerda ou direita, é $1/2$.
- cada passo é independente do anterior (não há memória)
- cada partícula move-se de forma totalmente independente das outras

1. Comece por implementar a classe `Rwalk1D` que simula as trajectórias das partículas na marcha aleatória no método `Run`, armazenando as trajectórias das partículas num `std::map` **mT**

   declaração da classe Rwalk1D (ficheiro: src/Rwalk1D.h)

```
class Rwalk1D {

 public:

 Rwalk1D(int N=1, double x=0., // N=nb of particles, x=x(0)
       double pL=0.5, double pR=0.5, // probabilities Left Right
       double dt=1, double dx=1 // time and space steps
       );
```

```
 ~Rwalk1D();

 // particle simulation

 void Run(int nsteps); // number of steps

 // getters

 const std::vector<double>& GetTrajectory(int n=1); // particle
  ↪  number
 double GetTimeStep();
 double GetSpaceStep();

private:

 double x0; // init coo
 int N; // number of particles

 double pL, pR; // probabilities  (left, same, righ)
 double dt, dx; // steps (time, space)

 std::map<int, std::vector<double> > mT; // trajectories

};
```

2. Realize um programa principal `main/tRwalk.C` em que obtenha,

   a) a trajectória (x, t) de 5 partículas

   b) o valor médio do deslocamento de 200 partículas ao fim de 10 e 100 passos, $< x(n = 10, 100) >$.

   c) os histogramas das posições das 200 partículas ao fim de 10 e 100 passos.

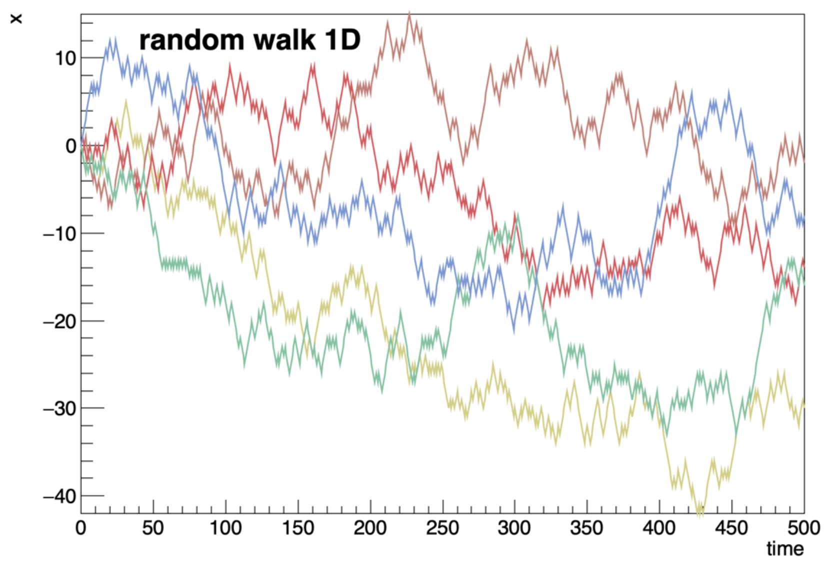

Figura 1: trajectórias $x(t)$ obtidas do random walk uni-dimensional para as cinco primeiras partículas