Exercícios de

Física Computacional

Mestrado em Engenharia Física-Tecnológica (MEFT)

Fernando Barao
Departamento de Física do Instituto Superior Técnico
Ano Lectivo: 2014-15

$2^{\underline{a}}$ série

de problemas de Física Computacional

- ☐ Programação com objectos ROOT
- ☐ Representação dos números, arredondamentos e precisão
- ☐ Interpolação de dados
- ☐ Integração e diferenciação numérica
- ☐ Métodos de monte-carlo
- ☐ Resolução de equações diferenciais (ODEs)

Fernando Barao
Departamento de Física do Instituto Superior Técnico
Ano Lectivo: 2014-15

Exercícios de Física Computacional

Programação utilizando objectos ROOT

Exercício 25 : Lance o root em sessão interactiva e utilize o interpretador de ROOT para correr código C++ que realize as seguintes tarefas:

a) crie um *array* de 2 histogramas *TH1F* utilizando o default constructor.

b) crie um *array* de 2 histogramas *TH1F* com as seguintes características numa só linha de comandos: 10 canais e limites inferior e superior respectivamente, 0.5 e 10.5

c) crie um *array* de 2 histogramas *TH1F* com 5 canais de largura variável dada por: $\mathbf{0.5, 1.5, 4.5, 2.0, 1.0}$

d) crie agora o *array* de 2 histogramas *TH1F* utilizando o default constructor e inicializando-os de seguida com as características da alínea b)

e) crie agora um *array* de 2 ponteiros que aponte para os histogramas com características da alínea b)

f) construa uma macro **mHisto.C** onde reuna o conjunto de operações da alíena d) e execute-a.

Exercício 26 : Lance o root em sessão interactiva e utilize o interpretador de ROOT para correr código C++ que realize as seguintes tarefas:

a) faça um array de dois inteiros sem inicializar os valores e verifique os valores existentes em cada posição do array.

b) liste os objectos existente em memória do ROOT.
Nota: utilize a classe TROOT, consultando a sua documentação em root.cern.ch, e em particular o ponteiro já existente para um objecto TROOT instanciado

c) construa um *array* de três objectos histograma que armazene floats (TH1F) entre os valores -10. e 10, com canais de largura 0.2

```
//a minha tentativa para mostrar a declaração
TH1F h[3]; //que constructor é chamado?
```

d) preencha o primeiro histograma com números aleatórios entre -5 e 5 e o segundo e terceiro histogramas com números aleatórios distribuídos de acordo com as funções

$$\begin{cases} f(x) &= 2x^2 \\ g(x) &= \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \end{cases}$$

Verifique consultando as classes *TF1* e *TFormula* como pode escrever as expressões das funções.

e) Liste agora os objectos existentes em memória de ROOT e verifique que os três histogramas que construiu se encontram lá.

f) Defina agora um *array* de duas funções uni-dimensionais

```
TF1 f[2];
```

onde implemente as seguintes funções:

$$\begin{cases} f_1(x) &= A\ sin(x)/x \quad \text{com } x = [0, 2\pi] \quad \text{e } A = 10. \\ f_2(x) &= Ax^4 + Bx^2 - 2 \quad \text{com } x = [-4, 4] \quad \text{e } A = 4\ B = 2 \end{cases}$$

g) Liste de novo os objectos existentes em memória de ROOT e verifique que os três histogramas que construiu e as duas funções se encontram lá.

h) Procure o ponteiro para o segundo histograma usando a classe TROOT (ou melhor, o ponteiro disponível para o objecto TROOT instanciado).

i) Tendo o ponteiro para o objecto histograma, desenhe-o no ecran, usando o método da class *TH1*, *Draw()*.

j) Obtenha agora o número de canais (bins) do histograma, usando o método da class *TH1*, *GetNbinsX()*. Teve sucesso com esta operação?

k) Desenhemos agora cada um dos outros histogramas e cada uma das funções.

l) Antes de abandonar a sessão de ROOT armazene os objectos construídos num ficheiro ROOT.

Exercício 27 : Reúna agora todos os comandos C++ que introduziu linha a linha no exercício anterior, numa macro de nome **mRoot1.C**. Corra a macro de forma interpretada, usando quer os métodos da classe *TROOT*:

a) **Macro("macro-name")**

```
root> gROOT->Macro("mRoot1.C")
```

b) **LoadMacro("macro-name")**

```
root> gROOT->LoadMacro("mRoot1.C")
```

Esta forma permite ter um ficheiro C++ com várias funções que são interpretadas e carregadas em memória e que podem ser chamadas de seguida na linha de comandos ROOT.

c) quer os comandos

```
root> .x mRoot1.C //execute macro
root> .L mRoot1.C //load macro (but not execute it)
```

Exercício 28 : No exercício anterior o código C++ existente na macro **mRoot1.C** foi interpretado. Pretende-se agora compilar este mesmo código usando o compilador ACLIC do ROOT. Para tal execute na linha de comandos ROOT,

```
root> .L mRoot1.C+ //compile and load macro (but not execute it)
```

que produzirá uma biblioteca *shareable* **mRoot1.so**

Exercício 29 : O índice de refracção do material diamante em diferentes comprimentos de onda é dado na tabela que se segue.

color	wavelength (nm)	index
red	686.7	2.40735
yellow	589.3	2.41734
green	527.0	2.42694
violet	396.8	2.46476

Organize um código C++ constituído por uma classe de base, **Material** e por uma classe derivada, **Diamond** que armazene a informação dos materiais (nome, densidade, ...) e possua métodos que permitam:

- definir o índice de refracção
- ajustar por uma lei o índice de refracção em função do comprimento de onda
- desenhar o índice de refracção

```
void SetRefractiveIndex(float*);
float* GetRefractiveIndex();
void FitRefractiveIndex(TF1*);
void DrawRefractiveIndex(); //draw points
void DrawRefractiveIndex(const TF1*); //draw points and function
```

Nota: A lei de variação do índice de refracção (n) *com o comprimento de onda* (λ) *é conhecida como lei de dispersão do material e pode ser ajustada com a fórmula de Sellmeir.*

$$n^2(\lambda) = 1 + \sum_i \frac{B_i \lambda^2}{\lambda^2 - C_i}$$

em que cada termo da série representa uma absorção na região de comprimentos de onda $\sqrt{C_i}$.
Para o ajuste do diamante pode-se usar a expressão:

$$n(\lambda) = A + \frac{B}{\lambda^2 - 0.028} + \frac{C}{(\lambda^2 - 0.028)^2} + D\lambda^2 + E\lambda^4$$

com λ *em* μm

Exercício 30 : Desenvolvamos agora uma classe **PixelDet** que simule um detector constituído por um conjunto 100×100 de píxeis quadrados de dimensão 5 mm. Cada pixel funciona de forma binária, isto é, ou está activo ou inactivo. Os píxeis possuem ruído intrínseco descorrelado cuja probabilidade é de 0.5%. O sinal físico deixado pelo atravessamento de uma partícula de carga eléctrica não nula são 10 pixeis distribuídos aleatoriamente numa região de 2×2 cm^2. Na resolução do problema, podemos associar um sistema de eixos x, y ao detector cuja origem esteja coincidente com o vértice inferior esquerdo do detector. Realize a implementação dos métodos da classe que julgar necessários de forma a simular acontecimentos físicos constituídos por ruído e sinal:

a) **simule o ruído no detector:**
realize um método que simule o ruído e devolva um *array* com o número dos pixeis ruidosos.

```
int* EventNoise(float probability);
```

b) **simule o sinal deixado pela partícula no detector:** realize um método que simule o sinal de uma partícula que passe na posição (x, y) e devolva um *array* com o número dos pixeis activos com sinal.

```
int* EventSignal(float a[2], float signal); //signal=10
```

c) Realize um método que permita visualizar o acontecimento no detector (por exemplo, um histograma bi-dimensional) com uma grelha a definir os pixeis.

```
...? DrawEvent(); //escolha o objecto ROOT a retornar
```

d) Realize um método que em cada acontecimento reconstrua a posição onde a partícula cruzou o detector e devolva ainda o conjunto dos hits associados à reconstrução.

```
// Evt pode ser uma estrutura a definir no ficheiro .h
// que reúna a informação da posição reconstruída do
// evento e ainda quais os pixeis que estão associados
Evt RecEvent();
```

e) Realize ainda um método que permita fazer o *dump* do conteúdo do acontecimento.

Realize um programa principal **mainPixelDet** onde realize a simulação de 1000 acontecimentos que passem na posição $(4cm, 4cm)$ e obtenha a distribuição da distância reconstruída à verdadeira.

Representação dos números em computador e arredondamentos

Exercício 31 : Considere o número real de precisão simples e 32 bits,

sinal	expoente	mantissa
0	0000 1110	1010 0000 0000 0000 0000 000

a) Determine o valor do expoente verdadeiro.

b) Mostre que a mantissa vale $\mathbf{1.625}$

c) Determine o valor do número real.

Exercício 32 :

a) Escreva uma funçao em C++ que determine os limites *underflow* e *overflow* do seu computador e linguagem de programação, dentro de um factor $\mathbf{2}$.

b) Obtenha os valores limite de underflow e overflow para números reais de precisão simples.

c) Obtenha os valores limite de underflow e overflow para números reais de precisão dupla.

Exercício 33 : Escreva uma funçao em C++ que determine a precisão do computador. Por exemplo, implemente um algoritmo em que se adicione ao número $\mathbf{1.}$ um número cada vez mais pequeno até que este seja inferior à precisão e a soma seja $\mathbf{1}$.

a) para números reais de precisão simples.

b) para números reais de precisão dupla.

Exercício 34 : Habitualmente considera-se que os erros de arredondamento são de natureza aleatória Para verificarmos essa hipótese podemos desenvolver um código em C++ que calcule os erros de arredondamento associados a uma dada operação de cálculo em precisão *float* e usando como referência a representação *double* do resultado. Defina uma classe em C++ de nome **FCtools** onde implemente os seguintes métodos estáticos:

a) Um método que determine o erro de arredondamento relativo à operação $\sqrt{i}$, com $i = 1, \cdots, 1000$.

```
static double RoundOffError(int i);// retorna o erro relativo de arredo
```

b) Um método que retorne um objecto *TGraph* cuja abcissa seja o valor de i e a ordenada o erro de arredondamento.

```
TGraph* RoundOffErrorG(int imin, int imax);
```

c) Um método que retorne um histograma unidimensional *TH1D* com a distribuição dos erros de arredondamento.

```
TH1D* RoundOffErrorH(int imin, int imax);
```

Métodos Numéricos

Exercício 35 : Considere uma matriz triangular superior $\mathbf{U_{3\times3}}$ preenchida com os seguintes números:

$$\begin{array}{ccc} \mathbf{4} & \mathbf{-2} & \mathbf{1} \\ & \mathbf{3} & \mathbf{-3/2} \\ & & \mathbf{3} \end{array}$$

Defina agora uma classe em C++ que manipule esta matriz e defina os métodos necessários na classe que permitam:

a) armazenar a matriz num *array* conventional,

```
double **m; // int m[3][3] (outra forma)
```

b) armazenar a matriz num *array* uni-dimensional e recuperar linhas e colunas

```
double* GetMatrix1D(); //return one-dimensional array with matrix
```

c) armazenar a matriz utilizando a classe Vec definida no Trabalho 1

```
vector<Vec> GetMatrixV(); // return Vec's
```

d) obter *rows* and *columns* a partir das diferentes formas de armazenamento

```
Vec GetRow(vector<Vec>);
Vec GetRow(double*);
Vec GetRow(double**);
Vec GetColumn(vector<Vec>);
Vec GetColumn(double*);
Vec GetColumn(double**);
```

e) obter o determinante da matriz

```
double Determinant(vector<Vec>, const Vec&);
double Determinant(double**, const Vec&);
double Determinant(double*, const Vec&);
```

f) admitindo que a matriz $\mathbf{U}$ resultou do método de eliminação de Gauss simples e que este foi aplicado a um vetor de constantes que após a eliminação de Gauss resultou em $\mathbf{b} = \{\mathbf{11}, -\mathbf{10.5}, \mathbf{9}\}$, elabore um método que resolva as equações,

```
  Vec Solve(vector<Vec>, const Vec&);
  Vec Solve(double**, const Vec&);
  Vec Solve(double*, const Vec&);
```

g) elaborar um método que teste se a solução encontrada é correcta

```
  //input: matrix, constants and variables

  bool Check(vector<Vec>, const Vec&, const Vec&);
  bool Check(double**, const Vec&, const Vec&);
  bool Check(double*, const Vec&, const Vec&);
```

Exercício 36 : Para a resolução de sistemas de equações é conveniente definirmos uma classe que possua os diferentes métodos de solução.

a) Definamos então a classe **EqSolver**, que tendo em conta os algoritmos definidos no exercício anterior, implemente os seguintes métodos (podendo usar a classe Vec implementada no Trabalho 1):

```
#include "Vec.h"
class EqSolver {
public:
   (...)
   //eliminação de Gauss: matriz M e vector de constantes B
   void GaussElimination(double** M, double* B);
   //resolução do sistema pelo método de eliminação de Gauss
   Vec GaussEliminationSolver();
   //decomposição LU com |L|=1
   void LUdecomposition(double** M);
   Vec LUdecompositionSolver(double** M, double *B);
private:
   double** m; //matriz (ou outra forma)
   double* b; //vector de constantes (ou outra forma)
};
```

b) Resolva o seguinte sistemas de equações lineares por ambos os métodos:

1)

$$\begin{cases} 4x_1 - 2x_2 + x_3 = 11 & (1) \\ -2x_1 + 4x_2 - 2x_3 = -16 & (2) \\ x_1 - 2x_2 + 4x_3 = 17 & (3) \end{cases}$$

2)

$$[\mathrm{A}] = \begin{pmatrix} 2 & -2 & 6 \\ -2 & 4 & 3 \\ -1 & 8 & 4 \end{pmatrix} \qquad [\mathrm{b}] = \begin{pmatrix} 16 \\ 0 \\ -1 \end{pmatrix}$$

Exercício 37 : Para a realização de interpolações, pode-se definir uma classe **DataP**, que conterá os dados respeitantes aos pontos e uma classe que herde desta que se chamará **Interpolator**, onde se definirão os diferentes métodos de interpolação (ver slides das aulas teóricas). Proceda à implementação das classes.

```
public DataP {
public:
private:
   int N; //nb of points
   double *x; //coo x
   double *y; //coo y
};
```

Exercício 38 : Dados os seguintes pontos,

x	-1.2	0.3	1.1
y	-5.76	-5.61	-3.69

determine o valor $y(0)$ usando:

a) o método de Neville

b) o método de Lagrange

c) o método de Newton

d) o método do spline cúbico

Exercício 39 : Para a integração de funções vamos definir a classe **Integrator** onde se definirão os métodos de integração trapezoidal e de Simpson. Implemente os algoritmos e estruture a classe:

```
class Integrator {
public:
   Integrator(TF1* f);
   ...
   void TrapezoidalRule(...);
   void SimpsonRule(...);

private:
   TF1 *func;
};
```

Exercício 40 : Determine o integral $\int_0^{\frac{\pi}{2}} \boldsymbol{cos(x)\ dx}$.

Exercício 41 : Os geradores de números aleatórios usam relações do tipo:

$$\boldsymbol{I_{i+1} = (aI_i + c)\%m}$$

a) Construamos uma classe em C++ **FCrand** que implemente o método das congruências lineares para geração de números aleatórios. Use somente um construtor que possua um parâmetro semente e que possa também funcionar como default constructor a partir da função *time*.

```
class FCrand {
  public:
  //seed number (prototype incompleto)
  FCrand(int seed);
  //generate one random between [0,1]
  float GetRandom();
  //generate one random between [min,max]
  float GetRandom(float min, float max);
  //generate N randoms between [0,1]
  float* GetRandom(int N);
  //generate N randoms between [min,max]
  float* GetRandom(int N, float min, float max);
```

```
  private:
  ...
};
```

Para aferirmos da qualidade do gerador constituído por: $a = 65, c = 319, m = 65537$, vamos gerar 5 números aleatórios e fazer as seguintes distribuições:

b) distribuições de cada um dos números aleatórios para 1000000 de amostragens. Determine o valor médio e o desvio padrão da amostra e compare com os valores esperados.

c) Divida a distribuição em 10 intervalos e coloque num gráfico os valores médios de cada intervalo bem como o erro do valor médio.

d) Um teste à independência dos números aleatórios produzidos por um gerador é o chamado teste de auto-correlação,

$$C_k = \frac{< x_{i+k}\, x_i > - < x_i >^2}{< x_i^2 > - < x_i >^2}$$

onde $k \neq 0$. Este coeficiente deve ser tendencialmente nulo em números aleatórios descorrelados. Produza um gráfico do coeficiente de auto-correlação para diferentes valores de $k = 1, 2, ..., 1000$.

e) distribuição de um número aleatório .vs. outro (escolha quais)

f) distribuição de um número aleatório .vs. outro .vs. outro (escolha quais)

g) Verifique agora o que obteria se utilizasse o gerador *rand()* de números aleatórios existente na biblioteca *<cstdlib>*.

Exercício 42 : Determinemos a superfície de um de círculo de raio 1, isto é, o valor de π. Consideremos um grande número N de pares de números aleatórios (r_1, r_2), tirados a partir de uma distribuição aleatória entre 0 e 1. Construa um algoritmo que determine o valor de π e coloque o valor calculado bem como o seu erro, em função do número de amostragens N (10000, 100000, 1000000).

Exercício 43 : A classe **Integrator** pode ser extendida de forma a incluir os métodos de integração de Monte-Carlo, simples, *importance sampling* e de *aceitação-rejeição*.

```
class IntegratorMC : public Integrator {
public:
   IntegratorMC(TF1*, TF1*); //function and pdf
   ...
   // methods need number of samplings (N) and shall return
   // the integral value and error
   void UniformRandom(...);
   void ImportanceSampling(...);
   void Rejection(...);

private:
   TF1 *func;
   TF1 *pdf;
};
```

Exercício 44 : Determine o integral,

$$\boldsymbol{F = \int_0^1 \frac{dx}{1+x^2}}$$

usando:

a) o método trapezoidal utilizando um passo $\boldsymbol{h = 0.2}$

b) o método de Simpson usando o mesmo passo

c) o método de monte-carlo com variável aleatória uniforme, usando 100, 1000 e 10000 amostragens. Determine o erro associado ao cálculo do integral em cada um dos casos.

Exercício 45 : A resolução de equações diferenciais ordinárias (ODEs) por via numérica, exige a implementação dos diferentes métodos iterativos (Euler, Runge-Kutta, etc.) de forma a se obter a solução das equações. Implemente uma classe cujo nome seja **ODEsolver** que vise a solução de ume equação genérica de primeira-ordem, $\boldsymbol{\frac{dy}{dt} = f(t,y)}$
Implemente os construtores que achar convenientes mas inclua também o construtor que permita instanciar a classe da seguinte forma:

```
//definir função f(t,y)
TF1 *f = new TFD1(...);

ODEsolver S(f, ti, yi, step); //ti,yi (initial values)
```

Defina os seguintes métodos na classe, que retornam o valor y_{n+1} iterando a partir de t_n, y_n:
- *IterEuler*
- *IterPredictorCorrector*
- *IterRK2*
- *IterRK4*

Exercício 46 : Considere a equação diferencial $\frac{dy}{dx} = 3x - y + 8$ cujo valor inicial é $y(0) = 3$. Determine a solução da equação $y(x)$ no intervalo $x \in [0, 2.0]$ e usando um passo de 0.1, com os diferentes métodos numéricos.

Exercício 47 : Considere o sistema de duas equações diferenciais de primeira-ordem,

$$\begin{cases} \frac{dz}{dx} &= sin(x) + y \\ \frac{dy}{dx} &= cos(x) - z \end{cases}$$

cujos valores iniciais são: $z(0) = y(0) = 0$. Determine a solução das equações $y(x)$ e $z(x)$ no intervalo $x \in [0, 2.0]$ e usando um passo de 0.1, com o método Runge-Kutta de ordem-4.

Exercício 48 : Um corpo de massa m sujeito ao campo gravítico encontra-se em queda livre, possuindo ainda uma força de travagem proporcional à velocidade ($\propto kv$), devido à resistência do ar.

a) Determine a equação do movimento.

b) Resolva numericamente a equação do movimento tendo em conta os seguintes valores iniciais e características do corpo:
   $z(0) = 2$ Km
   $\dot{z}(0) = 0$ m/s
   $m = 80$ Kg
   $k = 0.3$ Kg/s
   $g = 9.81$ m/s^2

b.1) Reduza a equação do movimento a um sistema de equações de 1ª ordem

b.2) Obtenha a solução $v(t)$ e determine a velocidade limite da queda

b.3) Quanto tempo demora a queda?