# 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

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

# Exercícios de Física Computacional

## Programação usando 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) 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 26 :** 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 27 :** 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 28 :** 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 ($\lambda$) 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 $\lambda$ em $\mu m$*

**Exercício 29 :** Desenvolvamos agora uma classe **PixelDet** que simule um detector constituído por um conjunto $100 \times 100$ 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 $2 \times 2$ 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.