Física Computacional Colectânea de problemas baseados em C++

Engª Física-Tecnológica/IST

Fernando Barão

30 Maio 2023

Conteúdo

Introdução **1**

1 Elementos de programação com objectos **3**

1.1 Introdução ao C++ 4
1.2 Gestão de memória e passagem de parâmetros 8
1.3 funções `Lambda` 12
1.4 Números aleatórios 14
1.5 Biblioteca STL (Standard Template Library) 18
1.6 Programaçao com classes 22

2 Representação gráfica e análise de dados **31**

2.1 Análise de dados 32
2.2 Representação gráfica de dados com ROOT 37

3 Representação de números em computador e erros associados **47**

4 Resolução numérica de problemas **51**

4.1 Sistemas Lineares e Matrizes 52
4.2 Interpolação 65
4.3 Derivação e integração numéricas 69
4.4 Raízes de funções 73
4.5 Métodos de monte-carlo 74
4.6 Resolução numérica de equaçoes diferenciais ordinárias 80
4.7 Resolução numérica de equaçoes diferenciais parciais (PDE’s) 84

Introdução

Esta colectânea de problemas de Física Computacional foi desenvolvida para o curso de Engª Física-tecnológica do Instituto Superior Técnico. Visa a resolução de problemas físicos com ferramentas numéricas implementadas com base na linguagem de programação C++.

Os problemas propostos com `fundo azul` seguem o programa ensinado no curso e pretendem ser um guia de aprendizagem da linguagem C++, dos algoritmos numéricos e da resolução dos problemas físicos propostos. Além deste problemas, existem também na colectânea, `Problema mais avançados` que envolvem um trabalho mais vasto de implementação de classes e que são de natureza opcional.

1Elementos de programação com objectos

1.1 Introdução ao C++

Problema 1.1.1

O programa `addnumbers.C` em C++ que se segue é suposto adicionar todos os números inteiros entre dois números introduzidos pelo utilizador no programa.

a. Compile o programa e verifique se existe algum erro.
b. Crie o executável e corra o programa para os pares de valores: $(5, -2)$, $(5, 20)$ e $(10, 55)$.

```
#include <iostream>
using namespace std;
int main() {
  int n, m;
  cout << "Enter integers: " << flush;
  cin >> n >> m ;
  if (n>m) { //swap
    int buf = m;
    m = n;
    n = buf;
  }
  cout << "This program adds all integers between " << n << " and " << m
  ↪  << endl;
  double sum = 0.;
  for (int i=n; i<=m; i++) {
    double += i;
  }
  cout << "Result = " << sum << endl;
}
```

Problema 1.1.2

Considere a seguinte expressão matemática, $z(x) = x + f(x)$, com $f(x) = sin^2(x)$. Realize um programa em C++ que calcule $z(x)$ para os valores de $x = 0.4, 2.1, 1.5$ e imprima os resultados no monitor do computador. A função $f(x)$ deve ser construída autonomamente.

Problema 1.1.3

O espaço em memória ocupado pelas variáveis depende do seu `tipo`. Realize um programa em C++ que calcule o número de bytes ocupado em memória pelas variáveis dos seguintes tipos: `short int`, `int`, `long int`, `unsigned int`, `float`, `double`, `long double`

Problema 1.1.4

Realize um programa em C++ que determine o valor da constante π com precisão `float` e `double` a partir da função `atan()`. Compare o valor obtido com o valor exacto de π que pode consultar na variável `M_PI` existente em `<cmath>`. Determine a precisão obtida em *float* e *double*.

Problema 1.1.5

Aspectos relacionados com a implementação do C++ em cada arquitectura podem ser encontrados na C++ *standard library*, através da inclusão de `<limits` que disponibiliza as seguintes funções:

function	provides
`numeric_limits<type>::max()`	largest type value
`numeric_limits<type>::min()`	smallest type value

Realize um program em C++ que avalie e imprima no monitor do computador os limites máximo e mínimo dos seguintes tipos de variáveis:
`int`, `unsigned int`, `float`, `double`.

Problema 1.1.6

Calcule o quadrado de um número inteiro positivo, x^2, usando somente as operações:

a. adição, subtracção, multiplicação ($\times 2$)
b. junte a hipótese de chamar uma função de forma recorrente (*recursion*)

> solução
>
> $x^2 = (x + 1 - 1)^2 = (x - 1)^2 + 1 + 2\,(x - 1)$

Problema 1.1.7

Tendo como base o programa `addnumbers.C` publicado no problema *1.1*, construa um outro programa `addnumbersS.C` que adiciona os quadrados dos números inteiros compreendidos entre os limites inseridos no programa pelo utilizador. O resultado da soma deve ser calculado como tipo `int` e `double`. Compare os resultados para o caso $(1, 5000)$.

Problema 1.1.8

Realize um programa em C++ composto das funções,

```
int main()
int fact(int)
```

que determine o factorial do número n $(n!)$ introduzido pelo utilizador no programa. Compile o programa na seguinte ordem:

a. obtenha primeiramente o código objecto `.o`
b. obtenha seguidamente o código executável `.exe`

Problema 1.1.9

Realize um programa em C++ que calcule:

$$\sum_{i=0}^{100} \sum_{j=5}^{300} \cos\left(i^2 + \sqrt{j}\right) \tag{1.1}$$

Codifique a soma numa função do tipo,

```
//ilim, jlim = limites de i e j
double Sum(std::array<int,2> ilim, std::array<int,2> jlim);
```

e realize um programa `mainSum.C` de onde chame a função.

```
// function prototype
double Sum(std::array<int,2> ilim, std::array<int,2> jlim);
// main program
int main() {
    double result = Sum({0,100},{5,300});
    std::cout << "resultado da soma: " << result << std::endl;
}

// function Sum
double Sum(std::array<int,2> ilim, std::array<int,2> jlim) {
(...)
}
```

Problema 1.1.10

A função `rand()` declarada em `<cstdlib>` gera um número pseudo-aleatório entre `0` e `RAND_MAX`. Realize um programa em C++ que:

a. gere 1000 números aleatórios x entre $x_{min} = 5$ e $x_{max} = 55$.

b. determine o valor de $y = \frac{x}{x-10}$ para cada aleatório.
c. determine o valor médio de x e o seu desvio padrão.

Problema 1.1.11

Pretende-se calcular a soma dos seguintes valores,

$$0.1 + 0.2 + \cdots + 5.4$$

tendo-se introduzido o seguinte código em C++ num programa:

```
(...)
double sum = 0;
for (double x=0; x!= 5.5; x += 0.1) {
  sum += x;
}
```

Realize um programa inserindo este código e confirme se este realize o que se pretende.

Problema 1.1.12

Uma massa é deixada cair de uma altura h, sem atrito, partindo do repouso ($v_0 = 0$).

a. Determine a a lei física que descreve a altura do corpo em função do tempo, $h(t).
b. Quanto tempo demora a queda para uma altura de 100 metros?
c. Escreva um programa em C++ que receba do utilizador a altura h em metros e calcule e imprima no ecrã o tempo que a massa demora a chegar ao solo.

Problema 1.1.13

Um satélite orbita circularmente em torno da terra a uma dada altitude h e possuindo um período de tempo T.

a. Mostre que a relação entre a altitude h e o período T é dado por:

$$h + R = \left(\frac{GMT^2}{4\pi^2}\right)^{1/3} \tag{1.2}$$

   onde R representa a raio da Terra.
b. Escreva um programa em C++ que receba do utilizador o período de tempo em segundos e calcule e imprima a altitude do satélite em metros.
c. Utilize o programa para calcular as altitudes dos satélites que orbitam a Terra uma vez por dia (geo-síncronos), cada 90 minutos e cada 45 minutos. Comente os resultados.

> constantes:
>
> $G = 6.67 \times 10^{-11}\ m^3\ Kg^{-1}\ s^{-2}$
> $M = 5.97 \times 10^{24}\ Kg$
> $R = 6\,371\ Km$

1.2 Gestão de memória e passagem de parâmetros

Problema 1.2.1

No programa que se segue fazem-se chamadas às funções,

```
// function prototypes
int* fintv(int)
double* fdoublev(int)
```

que retornam ponteiros (variáveis que contêm endereços de memória) para arrays de `int` e `double` respectivamente, cuja dimensão é dada no argumento das funções.

```
 int main() {
  int *a = fintv(100);
  double *b = fdoublev(100);
 }
```

As funções devem ser implementadas autonomamente em ficheiros separados `fintv.C` e `fdoublev.C` . Uma implementação possível da função `fintv` poderia ser a seguinte:

```
int* fintv(int n) {
 int v[n];
 return v;
}
```

Verifique se o exemplo de código está funcional e em caso negativo, corrija-o e complete com a função que falta.

Problema 1.2.2

Realize de seguida novas funções,

```
// function prototypes
int** fintv(...)
double*** fdoublev(...)
```

que permitam a criação dos tensores do programa que se segue.

a. Coloque funcional o seguinte programa `main` .

```
int main() {
 // return matrix of integers (100 x 50) set to 1
 int **a = fintv(100,50);
 // return matrix of integers (100 x 50 x 20) set to 5
 double ***b = fdoublev(100, 50, 20);
 double ***c = fdoublev(100, 50); // default 3rd parameter
}
```

b. Realize as funções que façam o `printout` para o ecrâ dos valores dos tensores

```
void print(int**, ...);
void print(double***, ...);
```

c. Finalmente, no final do programa `main` apague a memória alocada.

Problema 1.2.3

Pretende-se obter o valor da função $f(x) = \sqrt{\sin(2x)}$. Escreva em C++ métodos que permitam o cálculo de $f(x)$, em que x é dado em graus. Teste os diferentes métodos realizando um programa `main.C` onde os referencie.

a. o valor de $f(x)$ é retornado por cópia, pelo método:

```
double func(double);
```

b. o valor de $f(x)$ é retornado por referência:

```
void func(double x, double& f);
```

c. o valor de $f(x)$ é retornado por `pointer` :

```
void func(double x, double* f);
```

Problema 1.2.4

Um método/função em C++ desenvolvido para calcular a soma dos elementos contidos num `array` , possui a seguinte declaração:

```
void sum(const double* const v,  int n);
```

a. escreva o código em C++ que implemente o método
b. diga se é possível retornar a soma dos elementos no 1º elemento do `array` . Justifique.
c. altere a declaração da função de forma a retornar o valor da soma

Problema 1.2.5

Um método/função em C++ desenvolvido para calcular a soma dos elementos contidos num `array` , possui a seguinte declaração:

```
void sum(const std::array<double,100>& a);
```

a. escreva o código em C++ que implemente o método
b. diga se é possível retornar a soma dos elementos no 1º elemento do `array`. Justifique.
c. altere a declaração da função de forma a retornar o valor da soma

`Problema 1.2.6`

Realize o seguinte códigos em C++:

a. uma função que inicialize uma variável inteira com um valor aleatório e retorne o seu `pointer`:

```
int* func1();
```

b. uma função que inicialize uma variável inteira com um valor aleatório e retorne a sua referência:

```
int& func2();
```

Verifique que os endereços da variável `int` interna da função e da variável retornada para o programa `main`, são os mesmos.

c. um programa `main.C` que chame as funções 10^6 vezes. Verifique se tem `memory leakage` no programa. Liberte a memória que eventualmente tenha alocado.

`Problema 1.2.7`

Realize um código C++ no qual se definam métodos que realizem as seguintes tarefas:

a. calcular e retornar o traço da matriz

$$\begin{bmatrix} 2 & 10 \\ 5 & 7 \end{bmatrix} \tag{1.3}$$

```
double Trace(int** mx, int n);
```

b. retorne um `array` com os elementos da linha i da matriz $m \times n$,

$$\begin{bmatrix} 2 & 10 & 5 \\ 3 & 2 & 7 \end{bmatrix} \tag{1.4}$$

```
int* Mrow(int i,  int** mx, int m, int n);
```

c. retorne um array com o resultado da multiplicação de uma matrix $M(n \times m)$ por um vector coluna de $V(m)$ elementos.

$$V(m) = \begin{bmatrix} 2 \\ 5 \\ 7 \end{bmatrix} \tag{1.5}$$

d. aproveitando o resultado da alínea anterior, determine o resultado da multiplicação das seguintes matrizes:

$$\begin{bmatrix} 2 & 10 & 5 \\ 3 & 2 & 7 \end{bmatrix} \times \begin{bmatrix} 5 & 1 & 3 \\ 10 & 1 & 5 \\ 15 & 1 & 4 \end{bmatrix} \tag{1.6}$$

`Problema 1.2.8`

Realize um método em C++, que receba uma matriz de $n \times m$ elementos e um vector coluna de m componentes e calcule o vector produto, usando a seguinte declaração:

```
void Mmultiply(double** mx, double* vr, int n, int m, double*  pt);
```

com:
mx = matriz $n \times m$
vr = vector coluna
pt = vector resultado

Escreva um programa `main.C` que determine o resultado da alínea c) do problema anterior.

1.3 funções `Lambda`

Problema 1.3.1

O modelo de Bohr deve o seu sucesso ao facto de conseguir prever os espectros de emissão e absorção dos átomos, impondo uma quantificação do momento angular orbital do electrões. De facto, somente a utilização da mecânica quântica e a solução da equação de Schrodinger permite obter a solução correcta para os níveis de energia dos átomos. Os níveis de energia *básicos* dos átomos dependem assim da carga eléctrica dos núcleos (Z) e do número quântico $n \quad n = 1, 2, \cdots$,

$$E_n = -\frac{Z^2 m e^4}{8 h^2 {\varepsilon_0}^2} \frac{1}{n^2} \tag{1.7}$$

a. Escreva uma função `lambda` cujo argumento seja o nível n de energia e que retorne o valor da energia associada ao nível n, para átomos de hidrogénio,

```
auto fE = [](int n)->double{
  double En = ...;
  return En;
}
```

Nota: na solução desta alínea e da alínea seguinte, pode definir as constantes físicas necessárias no interior da função. Teste a função para diferentes valores de n, fazendo sair o resultado em unidades de energia Joules (J) e electronVolts (eV).

b. Faça evoluir a implementação da função de forma a aceitar agora dois argumentos: o nível de energia (n) e o número atómico (Z).

```
auto fEZ = [](int n, ...){...}
```

c. Admita agora que o seu programa principal tem as constantes físicas necessárias à solução do problema definidas no programa antes de a função ser chamada. Escreva uma função que calcule os comprimentos de onda dos fotões emitidos nas transições dos níveis de energia,

```
auto fEL = [??](int Z, double n1, double n2){
  (...)
  return En;
}
```

d. Finalmente, escreva um programa que defina uma matriz de níveis de energia E[Z][n], com $Z = 1, \cdots, 10$ e $n = 1, \cdots, 20$, cujos valores de energia sejam calculados por uma função `lambda`.

Problema 1.3.2

Na alínea a) do problema anterior, foi definida uma `Lambda function` para o cálculo dos valores de energia E_n. Gostaríamos que esta função fosse passada por argumento para uma outra função, `test` . Para isso vamos usar o `wrapper std::function` do C++. Complete o programa C++ que se segue:

```
#include <functional>
// function prototype
void test(const std::function<double(int)>& f, int n);
// main program
int main() {
  // define wrapper
  std::function<double(int)> fE = [](int n)->double {
    double En = ...;
    return En;
  }
  // call test: test must print energy value for n
  test(fE,2);
}
```

1.4 Números aleatórios

Problema 1.4.1

O C++ permite a geração de números aleatórios (pseudo-aleatórios) inteiros através da inclusão do módulo `<random>` da *standard library* que possui vários tipos de geradores disponíveis, entre os quais `minstd_rand` e `mt19937`.
Realize um programa em C++ que:

a. Use as funções `min()` e `max()` de cada gerador para imprimir no ecrã o intervalo de valores que podem ser gerados. Compare o máximo do intervalo com o valor máximo que pode armazenar com variáveis do tipo `int` ou `unsigned int`.

b. Crie um objecto do tipo `minstd_rand` e `mt19937` e imprima no ecrã 3 valores aleatórios para cada gerador. Corra o programa várias vezes e verifique se os valores aleatórios são diferentes de cada vez que corre o programa.
Exemplo de geração de de um número aleatório com o gerador `minstd_rand`:

```
minstd_rand generator; //create generator obj
auto r = generator(); // generate random using operator()
```

c. A sequência de números aleatórios é determinada pelo valor da semente definida para o gerador (*seed*). Defina um valor de *seed* para o gerador e imprima no ecrã 3 valores aleatórios. Verifique que os valores dependem da *seed* introduzida.

```
minstd_rand generator(seed_value); //create generator obj with seed
// generator.seed(value); // alternative way to define seed
auto r = generator(); // generate random using operator()
```

Problema 1.4.2

Pretende-se gerar números aleatórios contínuos entre $[x_{min}, x_{max}]$ usando os geradores aleatórios existentes na *standard library*.

a. Obtenha números aleatórios entre $[0, 1]$ usando os geradores `minstd_rand` e `m19937`.

b. Obtenha números aleatórios entre $[5, 55]$ usando os geradores `minstd_rand` e `m19937`
.

c. Obtenha o valor médio dos números aleatórios gerados para uma sequência de 1000 números.

Problema 1.4.3

Usando um gerador de números aleatórios, pode-se gerar números aleatórios segundo distribuições específicas. O C++ disponibiliza na biblioteca `<random>` classes que produzem valores numéricos segundo uma distribuição, utilizando um dado gerador de números aleatórios. Na resolução deste problema, use um gerador e *seed* à sua escolha.

a. No lançamento de uma moeda ao ar, em condições normais, a probabilidade de sair cara ou coroa é de 50% (dois estados possíveis). Este é um processo que segue a distribuição de Bernoulli, neste caso caracterizada por uma probabilidade p de sair cara de 50% (ou identicamente, coroa, de 50%), $B(p = 0.5)$. Podemos de seguida proceder à simulação do lançamento da moeda usando um dos geradores à sua escolha da biblioteca `<random>` da *standard library*. Construa uma função, para uma dada probabilidade de sucesso `p`, que retorna 1 ou 0, quando se obtém respectivamente sucesso ou insucesso,

```
int Bernoulli(double p); // returns 0 or 1
```

Construa de seguida um programa que gere 1000 aleatórios, e obtenha a probabilidade de obter sucesso recorrendo à função definida `Bernoulli`.

b. Em alternativa à função criada anteriormente, poderíamos ter usado um objecto do tipo `bernoulli_distribution` para simular o lançamento de uma moeda ao ar. Gere 1000 aleatórios segundo esta distribuição e imprima no ecrã quantos resultados `true` e `false` obteve. Obtenha a probabilidade de obter `true` e compare com o valor obtido na alínea a.

```
(...)
double p = ...;
std::m19937 generator;
std::bernoulli_distribution D(p);
// example: generating one random
auto x = D(generator);
(...)
```

c. Verifique como varia o valor médio da probabilidade de sucesso com o número de números aleatórios usados $N = 1000, 2000, 3000, 4000, 5000$

Problema 1.4.4

Pretende-se comparar a média e desvio padrão de distribuições diferentes de números aleatórios e ainda o impacto da definição da *seed* na sequência de aleatórios gerada. Utilize um gerador à sua escolha.

a. Gere uma amostra de 1000 números aleatórios x_i no intervalo $[-100, 100]$, utilizando a distribuição do tipo `uniform_real_distribution` e um gerador de números aleatórios da sua escolha, cuja sequência seja a definida pela *seed* $s = 500$. Calcule a média, $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ e o desvio padrão $\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$, da amostra.

b. Gere uma nova amostra de 1000 números aleatórios x_i, utilizando a distribuição do tipo `normal_distribution` de média $\mu = 10$ e $\sigma = 5$, e um gerador de números aleatórios da sua escolha, cuja sequência de númros gerados seja igual ao da alínea anterior (mesmo *seed* $s = 500$). Calcule a média, $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$ e o desvio padrão $\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$, da amostra.

c. Gere agora uma amostra de 2000 números aleatórios x_i, utilizando de forma alternada as distribuições `uniform_real_distribution` no intervalo $[-100, 100]$ e `normal_distribution` de média $\mu = 10$ e $\sigma = 5$, e o gerador de números aleatórios que escolheu com a *seed* $s = 500$. desta forma, produziu duas amostras de 1000 números aleatórios que obedecem às duas distribuições diferentes. Calcule a média e o descio padrão de cada amostra e compare com os valores obtidos anteriormente. Comente o resultado.

Problema 1.4.5

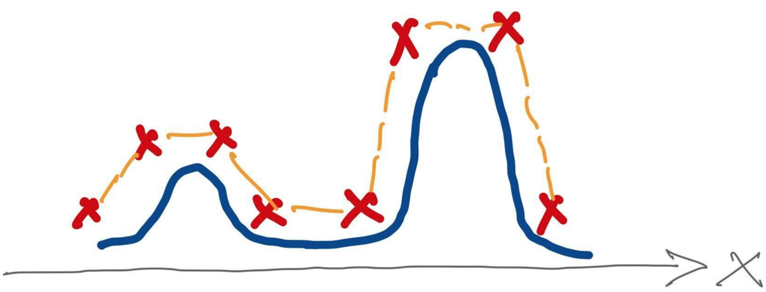
Futuramente, no uso de métodos de Monte-Carlo, aparecerão situações em que será necessário gerar números aleatórios segundo uma função auxiliar. Neste exercício iremos usar a distribuição da *standard library* `piecewise_linear_distribution` , para definirmos uma função auxiliar linear por troços, construída a partir de um conjunto de pontos definidos (x_i, y_i).

a. Defina uma distribuição linear uniforme entre $[-100, 100]$, usando a distribuição `piecewise_linear_distribution` . Gere uma amostra de 1000 números uniforme no intervalo e calcule o valor médio e o desvio padrão da amostra.

```
// defining linear segment
std::array<double, 2> xx {-100.0, 100.0}; // limits
std::array<double, 2> yy {1.0, 1.0}; // arbitrary constant value
// defining distribution
piecewise_linear_distribution distribution(xx.begin(), xx.end(), yy.begin());
```

b. Crie uma distribuição `piecewise_linear_distribution` para aproximar uma função com dois picos, com as seguintes características:

- Pico 1: altura $y_1 = 3$, centro em $x_1 = 10$, largura da base $\ell_1 = 4$ e passa de 0 à altura máxima em $\Delta x_1 = 0.2$
- Pico 2: altura $y_2 = 1$, centro em $x_2 = -5$, largura da base $\ell_2 = 2$ e passa de 0 à altura máxima em $\Delta x_2 = 0.1$ \



c. Gere 10000 números aleatórios segundo a distribuição linear da alínea anterior. Imprima no ecrã quantos resultados obteve no intervalo de cada pico (contagem N_1 em $[x_1 - \ell_1/2, x_1 + \ell_1/2]$ e N_2 em $[x_2 - \ell_2/2, x_2 + \ell_2/2]$).

d. O número de `randoms` gerados na alínea anterior nas regiões dos picos são proporcionais às áreas da função nesses locais. Compare assim o quociente N_1/N_2 com as áreas aproximadas dos picos da função definidas pela aproximação linear, $A_i = y_i \cdot (\ell_i - \Delta x_i)$.

1.5 Biblioteca STL (Standard Template Library)

Problema 1.5.1

Realize um programa `main()` onde se teste a função `rand2vec` cujo objectivo é proceder à geração de n números aleatórios x com valores compreendidos entre 0 e 360. O conjunto de números aleatórios deve ser retornado ao programa principal usando a estructura `vector<double>`. Seguem-se as declarações das funções a utilizar:

```
vector<double> rand2vec(int n);
vector<double>* rand2vecp(int n);
```

Problema 1.5.2

Neste exercício pretende-se explorar os `STL containers` `vector` e `map` da biblioteca STL.

a. O programa `main` (incompleto) que se segue implementa um vector que conterá elementos da tabela periódica. Cada elemento é descrito por uma estructura de dados `ATOM`.

```
int main() {
 // define hydrogen object
 ATOM hydrogen;
 hydrogen.A = 1;
 hydrogen.Z = 1;
 hydrogen.mass = 938.89; //MeV - natural units
 hydrogen.name = "Hydrogen";

 // define other 5 elements
 (...)

 //allocate vector and fill it with ATOM's
 vector<ATOM> vperiodic(6); //6 elems allocated
 (...)

 // print the contents of every element of the vector
 ...

 return 0;
}
```

Complete o programa `main` de forma a executar as acções descritas no programa, nomeadamente:

- escrever uma estructura `ATOM` num ficheiro `header` de nome `atom.h` que contenha os dados de um elemento da tabela periódica

```
struct ATOM {
  int A;
  int Z;
  (..)
};
```

- incluir num vector os primeiros 6 elementos da tabela periódica
- imprimir no ecrâ os detalhes de cada elemento contido no vector

b. Altere agora o programa `main` de forma a trabalhar com um vector com ponteiros para os 6 elementos

```
vector<ATOM*> vperiodic(6);
```

c. Finalmente, realize um novo programa `main` que faça a gestão dos 6 elementos com um `map`, realizando as seguintes acções:

```
int main() {
  // criar mapa
  map< string, ATOM > mperiodic;
  // preencher mapa com os 1os 6 elementos
  ...
  // imprimir no ecrâ todas as entradas do mapa
  ...
}
```

Problema 1.5.3

Pretende-se realizar uma estrutura `map` usando a biblioteca STL do C++ que armazene matrizes de dimensão variável $n \times m$, usando uma chave (`key`) do tipo `string`, emparelhada com uma estructura `vector`,

```
map <string, ...> Mmap;
```

a. Defina uma estrutura STL capaz de armazenar as matrizes que se seguem, definindo uma função que devolva a estrura `STL`

```
---? GetMatrix(int nrows, int mcols, int** M);
```

$$A = \begin{bmatrix} 2 & 10 & 5 \\ 3 & 2 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 1 & 3 \\ 10 & 1 & 5 \\ 15 & 1 & 4 \end{bmatrix} \qquad C = \begin{bmatrix} 5 & 1 \\ 10 & 2 \\ 15 & 1 \end{bmatrix} \tag{1.8}$$

b. Armazene as três matrizes `A` , `B` e `C` sob as chaves "A", "B" e "C" no mapa `Mmap` .

c. Crie uma função `Mmapfind` que procure uma chave (`key`) e retorne a matriz.

```
---? Mmapfind(map<string, ...> , string);
```

Problema 1.5.4

Realize uma função `array2vec` e o respectivo programa `main()` , cujo objectivo é transferir um `array` de números inteiros para uma estructura `STL vector` .

```
vector<int> array2vec(int, int*);
```

a. Aplique a função aos `arrays` seguintes: $a = (1, 10, 5, 6, 9, 3)$ e $b = (2, 5, 5, 7, 3)$

b. Implemente numa função `array2vecsort` , utilizando a biblioteca `<algorithm>` , a seriação dos valores de cada vector, quer na ordem crescente, quer na ordem decrescente. Retorne o vector ordenado.

```
// o=0, crescente; o=1, decrescente
vector<int> array2vecsort(int n, int* a, int o=0);
```

c. Elabore uma função `array2vecmax` , que determine o valor máximo existente em cada um dos `arrays` .

```
int array2vecmax(int n, int* a);
```

d. Elabore uma função `array2vecfind` , que localize a posição do valor 7 em cada um dos `arrays`

```
int array2vecfind(int n, int* a, int value); // value to be found
```

e. Realize as modificações necessárias nos métodos desenvolvidos de forma a impedir que os valores n e a sejam modificados no interior das funções.

f. Realize a desalocação de memória que tenha utilizado, antes do programa terminar.

Problema 1.5.5

Considere os arrays de inteiros $a = [3, 2, 1, 6, 5, 4, 9, 8, 7]$ e $b = [10, 12, 15, 18, 24, 32, 40, 45, 50]$.

a. Armazene-os usando `containers STL` .

b. Construa um novo array `aa`, em que os números ímpares do array `a` sejam transformados em números pares através da multiplicação por 2.

c. Construa um novo array `c`, partindo do array `a` e inserindo após cada valor do array `a`, os valores de `b` que sejam múltiplos do valor de `a`.

d. Realize o sorting do array `c` no sentido crescente.

e. Construa o `array c` invertido (decrescente).

`Problema 1.5.6`

Proceda à geração de um conjunto de 100 números aleatórios no intervalo $[0, 1]$ e guarde-os recorrendo ao `container STL, vector<double>`. Resolva as questões que se seguem recorrendo à biblioteca `STL` e às suas funções.

a. Determine a soma acumulada de todos os valores gerados e a sua média.

b. Determine o desvio padrão da amostra de valores gerados.

c. Proceda a uma transformação do array de forma a obter uma amostra de média nula.

d. Realize o escalonamento dos valores do array de forma crescente e determine as diferenças consecutivas.

1.6 Programaçao com classes

`Problema 1.6.1`

Pretende-se que a classe `Box` descreva um objecto tri-dimensional paralelipipédico reto. A declaração da classe, ainda que incompleta, a ser introduzida no ficheiro `header`, `Box.h`, possui como `data members` as dimensões do paralelipípedo (x, y, z) e seria a seguinte:

```
class Box {
 public:
  Box(); //cubo de lado 1
  Box(float fx, float fy, floatz);
  (...)

 private:
  float x, y, z;
};
\end{Verbatim}
```

a. O programa `main` que se segue, utiliza esta classe e realiza operações sobre objectos por ela descritos. Complete a declaração da classe de forma a realizar as tarefas pedidas no programa e implemente os respectivos métodos num ficheiro `Box.C`. Corra o programa `main` no final.

```
int main() {
  // criar dois cubos de lado 1
  Box B1;
  Box B2 = B1;

  // fazer a soma de dois cubos
  Box B3 = B1 + B2;

  // criar dois paralelipipedos
  Box B5(1,1,2);
  Box B6(B5);

  // somar os dois paralelipipedos
  Box B7;
  B7 = B5 + B6;

  // calcular volumes
  float volume_3 = B3.GetVolume();
  float volume_7 = B7.GetVolume();
```

```
  cout << "volumes: " << volume_3 << " " << volume_7 << endl;
}
```

b. Admita agora que no programa `main` se adicionam objectos `Box`, utilizando ponteiros para estes:

```
(...)
Box* pB2 = new Box();
Box* pBresultado = pBoriginal->Add(pB2);
(...)
```

Complete a declaração da classe `Box` e implemente o código C++ necessário para que estas operações seja possíveis.

Problema 1.6.2

Realizou-se uma classe genérica `pessoa` que pretende descrever um conjunto de características associadas às pessoas (aqui tratadas como objectos!). A declaração da classe é a seguinte:

```
  class pessoa {

  public:
  //constructor (nome do aluno, data de nascimento)
   pessoa(string, unsigned int);
   void SetName(string); //set name
   void SetBornDate(unsigned int); //nascimento
   string GetName(); //get name
   unsigned int GetBornDate();
   virtual void Print(); // print

  private:
   string name; //nome
   unsigned int DataN; //data de nascimento
}
```

a. Implemente o código associado aos `method members` da classe escrevendo sempre em cada método o código necessário que imprima o nome da classe e do método `[class::method]` de forma a sabermos quando é chamado. Compile o código e veja se não existem erros.

b. Para testar o código da classe realize um programa `main` onde proceda à construção de um `array` de 10 objectos pessoa:

```
pessoa P[10];
```

Que constructor é chamado? Corrija o código e declaração da classe caso existam erros.

c. Admita agora que pretendia construir um `array` de N ponteiros para objectos `pessoa`. Construa uma função que retorne o ponteiro para o `array`.

```
pessoa** DoArray(int N);
```

Inclua a informação do nome dos alunos e a sua data de nascimento.

`Problema 1.6.3`

Construa agora uma classe `alunoIST` que derive da classe `pessoa`. A nova classe deverá ter novos `data members` como por exemplo:

- Número mecanográfico do aluno: `int number`
- Curso frequentado: `string branch` e novas funções que interajam com os novos `data members`.

```
class alunoIST : public pessoa {
  public:
   //constructor (numero e nome do aluno)
   alunoIST(int number, string curso);
   void SetNumber(int);
   int GetNumber();
   void Print();
   (...)

  private:
   int number;
   string branch;
}
```

a. Implemente o código da nova classe.

b. Construa um `array` de objectos `alunoIST` com conteúdo.

c. Construa a seguinte função (ou método),

```
//function prototype
void Dummy(pessoa**, int);  //int has the number of array entries
```

que receba um ponteiro genérico para um `array` de ponteiros de objectos `pessoa`. No interior da função circule sobre todos os objectos e chame a função membro `Print()`. A função que é chamada pertence a que class? `pessoa` ou `alunoIST`?

`Problema 1.6.4`

Na sequência das classes anteriores podemos prosseguir o exercício criando agora a classe `Turma` ,
que não necessita de derivar de nenhuma das classes anteriores, antes usando como `data members` objectos da classe `alunoIST` . Uma declaração ainda que incompleta da classe seria:

```
class Turma {
  public:
   Turma(string, int n); //nome da turma, num de alunos
   ~Turma(); //destructor

  private:
   alunoIST **va; //pointer to array of pointers
   int Nalunos; // number of alunoIST
}
```

a. Complete a declaração da classe de forma a incluir os seguintes métodos:

   - `default constructor`

   ```
   Turma();
   Turma(string s="", int=0,  alunoIST** p=nullptr);
   ```

   - `copy constructor`

   ```
   Turma(const Turma&);
   ```

   - `copy assignment`

   ```
   Turma& operator=(const Turma&);
   ```

   - outros métodos

   ```
   void AddAluno(alunoIST* const);
   alunoIST* FindAluno(int numero);
   int GetNalunos(); // returns number of alunoIST
   ```

b. Implemente o código da classe e em particular o método,

   ```
   alunoIST* FindAluno(int);
   ```

   Procure realizar uma implementação eficiente da procura (dicotómica).

c. Construa um programa `main()` onde possa testar a classe definindo uma dada turma de LEFT.

Problema 1.6.5

O movimento de um corpo a uma dimensão pode ser descrito pela classe `Motion1D` onde se registam as N posições do corpo e os tempos. Apresenta-se de seguida a declaração da classe:

```
class Motion1D {
public:
  Motion1D(int N=0);
  virtual ~Motion1D();

  void SetMotion(float* t, float* x, int);

  int GetN(); //returns number of points
  float* GetTimes(); // returns array of times
  float* GetPositions(); //returns array of positions

  virtual void Print();
  virtual float TotalDistance(); //total distance
  virtual float MeanVelocity(); //mean velocity

protected:
  int N; //number of points
  float* t; //time array
  float* x; //position array
}
```

O movimento uniforme a uma dimensão pode ser descrito por uma classe `Uniform1D` que derive da classe `Motion1D`, cuja declaração a ser colocada no ficheiro `Uniform1D.h` seria a seguinte:

```
class Uniform1D : public Motion1D {
public:
  Uniform1D( // default constructor
          int fN=0, // number of points (number of steps + 1)
          float ti=0., // initial time
          float xi=0., // initial position
          float dt=0., // time range
          float vel=0.); // velocity
  ~Uniform1D();

  void Print();
private:
  float ti; // initial time
```

```
  float dt; // time duration
  float xi; // initial position
  float vel; // velocity (m/s)
};
```

Por sua vez, a implementação da classe a ser colocada no ficheiro `Uniform1D.C` , seria a seguinte:

```
#include "Uniform1D.h"

Uniform1D::Uniform1D(int fN, float ti, float xi, float dt, float vel) :
 ↪  Motion1D(fN) {
  // t and x arrays are created by Motion1D constructor
  for (int i=0; i<N; i++) {
     ... //fill here t and x arrays
  }
}

void Uniform1D::Print() {
  Motion1D::Print(); //call Print from base class
  printf("ti=%f, xi=%f, vel=%f \n", ti, xi, vel);
}
```

Produza um programa C++ de nome `Runiform1D.C` onde realize as seguintes acções:

a. Instancie um objecto `Uniform1D` na memória `heap` com 100 pontos discretos, durante 1000 segundos de duração e a uma velocidade de 10 m/s. Imprima os valores usando o método Print().

```
// instantiate object Uniform1D
Uniform1D *p1D = new Uniform1D(100, 0., 0., 1000., 10.); // 1000 sec
p1D->Print();
```

b. Construa um array de dois ponteiros do tipo `Motion1D` que contenha os seguintes objectos `Uniform1D` e `Motion1D` . Inicialize os valores de `Motion1D` com 400 pontos de tempo e distância, percorrida por um corpo em queda livre.

```
// make an array with Motion1D derived objects
Motion1D* pm[2] = {
  new Uniform1D(100, 0., 0., 500., 20.),
  new Motion1D(400)
};
```

```
// fill Motion1D object with values
(...)
```

c. Imprima através do método `Print()` os valores contidos em ambos os objectos.

d. Construa agora um array de dois objectos `Motion1D`, com 400 pontos. Inicialize os valores de um objecto `Motion1D` com 400 pontos de tempo e distância percorrida por um corpo em queda livre e o outro com movimento de um corpo atirado ao ar na vertical com velocidade inicial de $v_0 = 1$ m/s.

```
Motion1D m[2] = {
  (...)
};
(...)
```

> solução
>
> ```
> Motion1D m[2] = { Motion1D(400), Motion1D(400)};
> //create arrays t and x and fill
> m[0].SetMotion(...);
> m[1].SetMotion(...);
> ```

e. Remova os objectos criados da memória.

> solução
>
> ```
> // delete objects
> delete p1D;
> delete p[0];
> delete p[1];
> ```

Problema 1.6.6

Um polígono pode ser definido a partir de um conjunto de segmentos de recta. Neste problema desenvolva a classe `segment`, que armazenará o conjunto de dois pontos que constituem o segmento e cuja estrutura mínima se mostra de seguida,

```
class segment {
  public:
  (...)
  private:
  vector<pair<float,float>> SEG;
};
```

de forma a que esta permita correr sem erros o programa que se segue.

```
int main() {
  vector<pair<float,float>> V{{2.3,5.2},{2.8,4.5}};
  segment S1(V);
  S1.Dump();

  segment S2(S1);

  segment S3;
  S3 = S1;

  // create a segment
  segment Sx;
  Sx.Add(1,2);
  Sx.Add(5,8.1);
  Sx.Replace(1, 0.25, 0.5867); // replace 1st point

  vector&lt;segment&gt; VS;
  VS.push_back(segment(V));

  segment Y1;
  Y1 = std::move(VS[0]);
}
```

O uso de *const* em variáveis retornadas por referência

Porquê usar o `const` nas variáveis retornadas por referência? Como exemplo para uso do `const`, utilizemos o problema 1.6.6 onde foi implementada a classe `segment` e adicionemos a esta o seguinte método que retorna por referência o vector de pontos bi-dimensionais,

```
std::vector<std::pair<float,float>>& GetSEG();
const std::vector<std::pair<float,float>>& GetSEG() const;
```

Implementem agora o código que se segue e verifiquem os efeitos.

a. Criamos um objecto `segment` que é `const` (ou seja, imutável) e vamos tentar modificá-lo.

```
const segment S(...);
S.GetSEG() = std::vector<std::pair<float,float>>(); // ERROR: Cannot
↪  modify
// or
S.GetSEG().clear(); // ERROR: Cannot modify
```

b. Criamos um objecto `segment` sem o qualificativo `const` (ou seja, mutável) e vamos tentar modificá-lo.

```
segment S(...);
S.GetSEG() = std::vector<std::pair<float,float>>(); // MODIFIES SEG in
↪  S
// or
S.GetSEG().clear(); // clears SEG
```

Neste exemplo vemos como se conseguiu alterar o conteúdo do membro da classe `SEG` dada a existência do método,

```
std::vector<std::pair<float,float>>& GetSEG();
```

Atenção por isso, à passagem de referências não protegidas para membros da classe.

Objectos da classe `segment` do tipo `const` só poderão recorrer a métodos da classe que sejam `const` , porque estes asseguram que os membros da classe não serão modificados. Vejamos o exemplo seguinte:

```
// define const segment
const segment S(...);
// get a copy of SEG
auto SEG_copy = S.GetSEG(); // requires method [GetSEG() const]
```

2Representação gráfica e análise de dados

2.1 Análise de dados

`Problema 2.1.1`

O conjunto de dados por país em relação à pandemia COVID-19 podem ser encontrados sob a forma de ficheiro `CSV, comma separated values` no `site` do Centro Europeu de Europeu de Prevenção de Doenças. Estamos perante uma série temporal de dados de COVID e é importante fazer a sua caracterização em termos de auto-correlação, estacionaridade (constância temporal) e periodicidade.

O armazenamento de dados em ficheiros deste tipo é frequente e por isso torna-se útil desenvolvermos uma ferramenta de leitura de dados que permita a sua posterior análise. Deixando para uma fase posterior o desenvolvimento de uma classe de leitura flexível de ficheiros CSV, vejamos agora a leitura e análise deste ficheiro cujo formato é o seguinte:

A	B	C	D	E	F	G	H	I	J
17/09/2020	17,9,2020	17	0	Afghan	AF	AFG	38041757	Asia	1.65344624
16/09/2020	16,9,2020	40	10	Afghan	AF	AFG	38041757	Asia	1.70864874
15/09/2020	15,9,2020	99	6	Afghan	AF	AFG	38041757	Asia	1.62715933
14/09/2020	14,9,2020	75	0	Afghan	AF	AFG	38041757	Asia	1.45629446
13/09/2020	13,9,2020	35	0	Afghan	AF	AFG	38041757	Asia	1.3090878
12/09/2020	12,9,2020	34	0	Afghan	AF	AFG	38041757	Asia	1.22496971
11/09/2020	11,9,2020	28	0	Afghan	AF	AFG	38041757	Asia	1.16450983
…	…	…	…	…	…	…	…	…	…

> onde:
>
> - A: date
> - B: day, month, year
> - C: cases
> - D: deaths
> - E: country or territory
> - F: geoId
> - G: country code
> - H: population 2019
> - I: continent
> - J: cumulative number for 14 days per 10^5

a. Desenvolva a classe em C++ `DataReader` que proceda à leitura do ficheiro. Identifique o número de colunas que terá que ler, e armazene a informação lida na classe.

```
int main() {
  DataReader D("COVID19.csv");
  (...)
}
```

b. Implemente o método `DisplayData` que permita fazer o display de:

- evolução do número total de mortes por dia
- evolução do número total de casos por dia
- evolução do número de casos diários para um dado país
- sobreposição das curvas de número de casos diários para um conjunto de países

```
class DataReader {
  public:
   // constructor and destructor
   DataReader(string filename);
   ~DataReader();

   // other methods
   TMultiGraph* DisplayData(
                              string KIND,   // KIND: "cases", "deaths"
                              vector<string> COUNTRIES, // COUNTRIES:
 ↪  "ALL" or {"PT", "FR", ...}
                              string OPTION="daily"   // OPTION:
 ↪  "daily", "cumulative"
                             );

   void Print(); // print class elements

   (...)

private:
   (...)
};
```

Problema 2.1.2

O teste de Kolmogorov-Smirnov permite em estatística a comparação das formas de uma distribuição contínua de uma dada variável com uma distribuição de referência (*template function*). Este teste permite assim validar o grau de acordo de uma distribuição com a sua referência, através da determinação da diferença máxima entre a distribuição acumulada,

$$F(x) = \int_{x_{min}}^{x} f_{distrib}(x)\, dx \tag{2.1}$$

e a distribuição acumulada de referência. Utilizando esta ideia, neste problema pretende-se estimar a qualidade de um gerador aleatório, que na essência deve permitir gerar números descorrelados entre si, gerando desta forma uma distribuição *plana*.

Realize então o seguinte código:

a. uma função `GetRandom` que retorne um número aleatório no intervalo $[x_{min}, x_{max}]$

```cpp
// returns random number between [xmin, xmax]
// xmin = minimal value
// xmax = maximal value

double GetRandom(double xmin, double xmax);
```

b. uma função `Fobs` que retorne a distribuição acumulada da variável aleatória x num número de intervalos N,

$$F(x_m) = \sum_{i=1}^{m} x_i \qquad m = 1, 2, ..., N \tag{2.2}$$

e que receba o número de vezes que deve chamar a função geradora da variável x, os limites dos intervalos de x e um ponteiro para a função geradora. "'cpp // returns accumulated distribution on x intervals // NCALLS = number of times generation function is called // vector x = x boundaries on the N intervals // (N+1 boundaries) // double (*f) (double, double)) = pointer to generating function

vector Fobs(int NCALLS, vector x, double (*f) (double, double)); "' Nota: garanta que a distribuição acumulada é normalizada a 1.

c. uma função `KolmogorovTest` que retorne a diferença máxima entre a distribuição acumulada e a distribuição de referência

```cpp
// returns maximal distance (F distance) between template
// and observations
// vector<double> x = x boundaries on the N intervals (N+1
↪ boundaries)
// vector<double> Fx = accumulated distribution
// vector<double> Fref = reference distribution

double KolmogorovTest(vector<double> x,
                      vector<double> Fx,
                      vector<double> Fref);
```

d. Realize agora um programa `main` que faça a geração de 10000 números aleatórios no intervalo $[0, 5]$ (dividido em bins de 0.1) e calcule no final a diferença de kolmogorov.

e. Repita o procedimento para 1000 amostras de 100000 números aleatórios e faça um *plot* com a distribuição das diferenças de kolmogorov.

`Problema 2.1.3`

A actividade solar varia no tempo, possuindo uma correlação directa com um indicador que corresponde ao número de manchas solares que se observam na sua superfície. O número de manchas solares ao longo do tempo pode ser obtido no sidc.

O formato dos dados é o seguinte:

```
Column 1-3: Gregorian calendar date
            - Year
            - Month
            - Day
 Column 4: Date in fraction of year
 Column 5: Daily total sunspot number.
           A value of -1 indicates that no number is
           available for that day (missing value).
 Column 6: Daily standard deviation of the input sunspot numbers from
           individual stations.
 Column 7: Number of observations used to compute the daily value.
 Column 8: Definitive/provisional indicator. A blank indicates that
           the value is definitive. A '*' symbol indicates that the
           value is still provisional and is subject to a possible revision
           (Usually the last 3 to 6 months)
```

a. Utilizando a classe `DataReader`, proceda à leitura da coluna 4 do ficheiro e à leitura do número de spots do sol (sunspot number), que se encontra na coluna 5 e armazene esta informação.
b. Com base num método da classe `DataReader`, realize um gráfico mostrando a evolução do número de *sunspots* no tempo.

Elabore agora uma nova classe `DataManip` que herde de `DataReader` e na qual implemente métodos que permitam realizar as operações que se seguem:

a. Ordene os valores de sunspot por ordem crescente e por ordem decrescente.

b. Obtenha as derivadas temporais de sunspot em função da data.

c. Determine a média deslizante do sinal para 3 e 5 valores. Compare sob a forma gráfica.

d. Implemente uma função, que actue no `array smoothed`,

```
void LocalMaxima(vector<pair<double,double> >& SS,
                 vector<pair<double,double> >& LMAX)
```

que determine todos os pares de valores (data, sunspot) correspondentes aos valores máximos locais do valor do sunspot.

e. Realize uma outra função que encontre os mínimos locais.

f. Determine de forma aproximada a periodicidade do sinal de sunspot.

2.2 Representação gráfica de dados com ROOT

`Problema 2.2.1`

Lance o root em sessão interactiva e utilize o interpretador de ROOT.

```
root -l
```

Escreva no interpretador o código C++ que realize as seguintes tarefas:

a. crie um *array* de 2 histogramas uni-dimensionais `TH1F` utilizando o `default constructor` .

b. crie um *array* de 2 histogramas uni-dimensionais `TH1F` com as seguintes características: 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 definidos pelo conjunto de limites: $0.5, 1.5, 4.5, 2.0, 1.0$

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.

```
root -l
root> .x mHisto.C
```

`Problema 2.2.2`

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.

```
root -l
root> gROOT->ls()
```

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

```
TH1F h[3]; //what constructor is called?
```

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

$$\begin{cases} f(x) & = 2x^2 \\ g(x) & = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x}{\sigma}\right)^2} \end{cases} \tag{2.3}$$

Verifique consultando a classe `TF1` (ver documentação da classe) como pode escrever as expressões das funções. As duas formas mais simples que podem ser utilizadas são, a escrita da função como uma fórmula `string` que é interpretada por ROOT e a outra mais flexível que é através da utilização de funções *Lambda*. Por exemplo para se criar o objecto `TF1 fsum` , contendo a fórmula $y(x) = ax + b$, onde a e b são parâmetros da função, faz-se:

```
// lambda function
auto lambda = [](double *x, double *p) { return p[0]*x[0] + p[1];}
// TF1
TF1 fsum("fsum", lambda, 0, 10, 2);
fsum.SetParameters(1, 1);

// TF1 using tsring method
TF1 fsum2("fsum2", "[0]*x + [1]", -3, 3);
```

e. Defina agora um *array* de duas funções uni-dimensionais

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

onde implemente as seguintes funções:

$$\begin{cases} f_1(x) & = A\ sin(x)/x \quad \text{com } x = [0, 2\pi] \quad \text{e } A = 10. \\ f_2(x) & = Ax^4 + Bx^2 - 2 \quad \text{com } x = [-4, 4] \quad \text{e } A = 4\ B = 2 \end{cases} \tag{2.4}$$

f. Desenhe no ecran os histogramas, usando o método da class `TH1` , `Draw()` .

g. Obtenha agora o número de canais (bins) do histograma, usando o método da class `TH1` , `GetNbinsX()` .

h. Desenhe cada uma das funções $f(x)$ e $g(x)$.

i. Desenhe as funções $f_1(x)$ e $f_2(x)$. Verifique o impacto que o método `TF1::SetNpx(int N)` tem no desenho da função, para as opções $N = 5, 10, 50$.

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

Problema 2.2.3

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

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

> `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.

b. quer os comandos:

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

`Problema 2.2.4`

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`

`Problema 2.2.5`

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

Neste exercício pretende-se estruturar a informação relacionada com os materiais num código C++. Para isso, podemos imaginar uma hierarquia de classes constituída por uma classe de base `Material` , que contenha as características básicas de um material, como sejam o seu nome e a sua densidade, e classes derivadas onde sejam implementados outras características dos materiais.

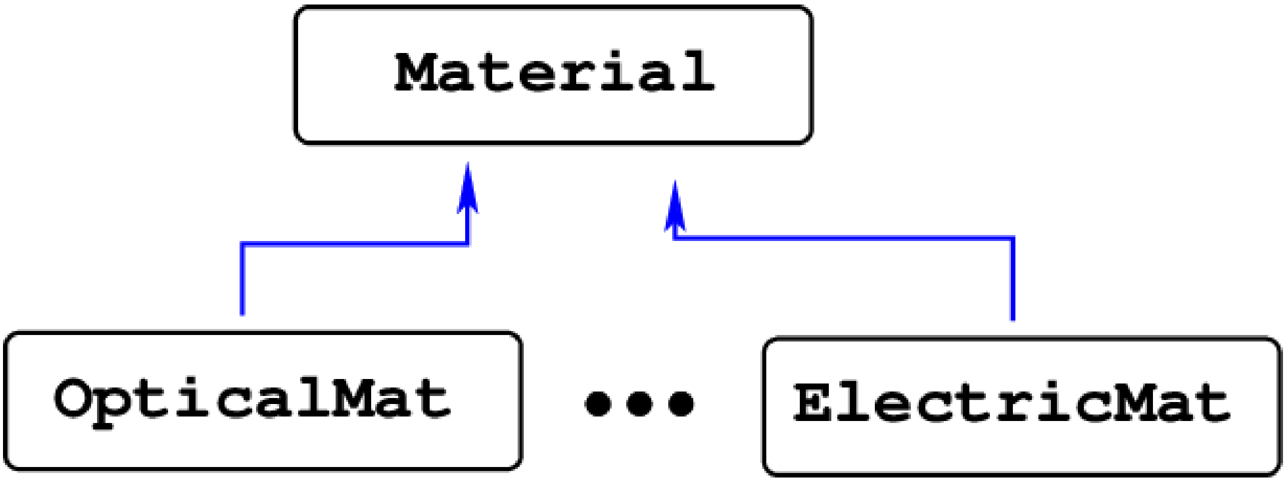


Figura 2.1: esquema das classes

A classe `Material` :

```
class Material {
 public:
 Material(string fname="", Double_t fdens=0): name(fname),
 ↪  density(fdens){;}
 string GetName() {return name;}
 Double_t GetDensity() {return density;}
 virtual void Print();

 protected:
 string name;
 Double_t density;
};
```

Podemos, por exemplo, agrupar os materiais ópticos numa classe `OpticalMat` que derive da classe `Material` e onde iremos colocar as características ópticas do material como por exemplo o índice de refração. A classe `OpticalMat` deve possuir métodos que permitam:

- definir o índice de refracção
- ajustar por uma lei o índice de refracção em função do comprimento de onda
- fazer um gráfico com o índice de refracção

```
class OpticalMat : public Material {
public:
  ...
  void SetRefIndex(vector<pair<float,float> >); //pair(wavelength, ref
   ↪  index)
  vector<pair<float,float> > GetRefIndex();
  void SetFitRefIndex(TF1*); //provide function to be fitted through TF1
  TF1* GetFitRefIndex(); //return TF1 pointer to fit function
  void DrawRefIndexPoints(); //draw points
  void DrawFitRefIndex(); //draw points and function
  void Print(); //define print for this class

private:
  // method with the fit function
  double FitRefIndex(double* x, double* par);
  // we need to store the refractive index characteric of the material
  ...
  // we need to store a TF1 pointer to the fit Ref Index function
  TF1* f;
  };
```

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 Sellmeier.

$$n^2(\lambda) = 1 + \sum_i \frac{B_i \lambda^2}{\lambda^2 - C_i} \tag{2.5}$$

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:

$$n(\lambda) = A + \frac{B}{\lambda^2 - 0.028} + \frac{C}{(\lambda^2 - 0.028)^2} + D\lambda^2 + E\lambda^4 \tag{2.6}$$

com λ em μm

Problema 2.2.6

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 can be a structure to be defined on header file .h
//     that will gather reconstruction information of event
//     and idetify which pixels are associated

Evt RecEvent();
```

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

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

Problema 2.2.7

Vimos na secção "Elementos de progamação por objectos" (sub-secção 1.4) a possibilidade de geração de números aleatórios utilizando a biblioteca `random` . Neste problema iremos usar o gerador `m19937` para construirmos uma distribuicão bi-dimensional uniforme de números aleatórios.

b. Com recurso ao gerador `m19937` , gere uma amostra de 10000 valores aleatórios (x, y) no intervalo $x, y \in [0, 50]$. Para tal instancie dois geradores com uma *seed* igual e utilize-os para gerar x e y. Construa um histograma de frequências bi-dimensional com os números aleatórios, utilizando intervalos em x e y, respectivamente $\Delta x = 1$ e $\Delta y = 1$. Faça *display* do histograma.

c. De novo com recurso a um só gerador `m19937`, gere uma amostra de 50000 valores aleatórios (x, y) no intervalo $x, y \in [0, 50]$, gerando alternadamente os valores de x e y. Realize um diagrama de frequências bi-dimensional com os valores dos números aleatórios, utilizando intervalos em x e y, respectivamente $\Delta x = 1$ e $\Delta y = 1$. Faça *display* do histograma.

d. Calcule o valor médio da frequência dos canais do histograma da alínea anterior e represente num histograma uni-dimensional (`TH1F`) a diferença do valor de cada canal e a média, $z_i - \langle z \rangle$.

`Problema 2.2.8`

Um dado é um cubo (6 faces) que dada a sua simetria é usado em jogos para se obter um número aleatório com igual probabilidade entre 1 e 6, dependendo da face visível no topo. Neste problema, iremos contruir uma classe que permite simular o lançamento de um dado.

a. Construa a seguinte classe `dice` ,

```
class dice {
 public:
  dice() = default;
  dice(m19937 generator, uint64_t seed=0);

  int launch(); // throw the dice once
  std::vector<int> launch(int n); // throw n times

  void SetSeed(uint64_t);

 private:
  m19937 G; //generator
  uint64_t seed;
};
```

b. Com recurso à classe anteriormente definida, construa um programa em C++ em que gere uma amostra de 1000 valores de face do cubo e construa um histograma com os resultados. Verifique que a probabilidade de saída de cada uma das faces é semelhante e compare com o valor esperado.

c. Admita agora que o dado está viciado e que a face 3 tem uma probabilidade de ser amostrada de 25% enquanto que as outras possuem a mesma probabilidade de sair. Adicione uma nova função `launch` que permita definir esta situação,

```
int launch(std::array<std::pair<int, float>, 6>); // pair contains
 ↪  (face, probability)
```

Proceda agora ao lançamento do dado 1000 vezes e construa um histograma com os resultados. Verifique que a probabilidade de saída de cada uma das faces está de acordo com o esperado.

Utilização da biblioteca ROOT para display de resultados

ROOT, é um *package* de software desenvolvido no CERN em C++. Para mais detalhes ver:

- reference documentation: link
- histogram classes: link
- about colors in ROOT: link

Representação gráfica de um histograma bi-dimensional

O exemplo que se segue mostra como usar ROOT para construir gráficos, usando um programa C++, onde se constói um histograma bi-dimensional, um Canvas onde o desenhar, e se recorre a um `TApplication` para o mostrar.

```
#include "TCanvas.h"  // include class TCanvas from ROOT library
#include "TRootCanvas.h"
#include "TH2F.h"     // histogram 2D
#include "TApplication.h"

int main() {

  // build 2D histogram
  // TH2F (float precision), TH2D (double precision)
  TH2F *h2 = new TH2F("h2", "histograma 2D", 50, 0, 50, 50, 0, 50);

  // now we have to fill every histogram cell with the calculated power
  (...)

  // Draw
  // - we need to instatiate TApplication to have a graphics display
  // - produce a canvas (tela gráfica) where graphics objects will be
  ↪  placed
  // - draw histogram: check the many options you have available;
  //   here we choose a colored gradient representation
  // - save plot to file
  // - Run application

  TApplication app("app", nullptr, nullptr);
  auto c = new TCanvas("canvas", "lightmap canvas", 0, 0, 800, 800); //
  ↪  size 800x800
  h2->Draw("COLZ");
  c->Update(); // update display canvas
  c->SaveAs("lightmap.pdf"); // save graphics in pdf format (eps, png,
↪  ...)
```

```
  // this gives you control of graphics window buttons
  TRootCanvas *rc = (TRootCanvas *)c->GetCanvasImp();
  rc->Connect("CloseWindow()", "TApplication", gApplication,
↪  "Terminate()");
  // without next code line ( very last line of your program), nothing is
   ↪  displayed
  app.Run(); // could be: gSystem->ProcessEvents()
}
```

3Representação de números em computador e erros associados

`Problema 3.1.1`

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 1.625
c. Determine o valor do número real.

`Problema 3.1.2`

a. Escreva uma funçao em C++ que determine os limites *underflow* e *overflow* do seu computador e linguagem de programação, dentro de um factor 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.

`Problema 3.1.3`

Escreva uma função em C++ que determine a precisão do computador. Por exemplo, implemente um algoritmo em que se adicione ao número $1.$ um número cada vez mais pequeno até que este seja inferior à precisão e a soma seja 1.

a. para números reais de precisão simples.
b. para números reais de precisão dupla.

`Problema 3.1.3`

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:

```
class FCtools {
  public:
  (...)
};
```

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

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

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

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

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

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

Problema 3.1.4

A resolução da equação quadrática $x^2 - 2bx + c = 0, \quad b^2 > c$ pode ser feita com recurso à formula resolvente dando lugar à seguinte solução:

$$x_{1,2} = b \pm \sqrt{b^2 - c} \tag{3.1}$$

a. Mostre que o produto das duas soluções nos dá a seguinte equação:

$$\begin{aligned} x_{1,2} &= b \pm \sqrt{b^2 - c} \\ x_1 \times x_2 &= c \end{aligned} \tag{3.2}$$

b. As soluções da equação podem ser dadas pelos seguintes algoritmos:

b.1) $x_1 = b + \sqrt{b^2 - c}$
$x_2 = b - \sqrt{b^2 - c}$

b.2) $if\ b > 0$

$x_1 = b + \sqrt{b^2 - c}$

$x_2 = c/x_1$

$else$

$x_2 = b - \sqrt{b^2 - c}$

$x_1 = c/x_2$

$endif$

Qual dos algoritmos tem menor erro? Porquê? Crie um código C++ em que resolve o sistema para $b = 0.03,\ c = 0.0008$ e verifique a sua conclusão anterior.

> `solução`
>
> a segunda é melhor porque reduz o número de subtracções; as multiplicações introduzem menos erros de arredondamento do que as subtracções

`Problema 3.1.5`

A derivada numérica da função $\cos(x)$ pode ser calculada recorrendo à expansão em série de Taylor de primeira ordem.

a. Mostre que se pode escrever a seguinte igualdade numérica:

$$\frac{\cos(x+\delta) - \cos(x)}{\delta} + \sin(x) \simeq 0 \tag{3.3}$$

b. Crie um programa em C++ que verifique a igualdade anterior para $x = 3$ e $\delta = 10^{-11}$.

c. É possível reescrever a diferença entre dois cosenos, usando a seguinte identidade trigonométrica:

$$\cos(\alpha) - \cos(\beta) = -2\sin(\frac{\alpha+\beta}{2})\ \sin(\frac{\alpha-\beta}{2}) \tag{3.4}$$

Assim, a equação demonstrada na alínea a) pode ser rescrita como:

$$-\frac{2}{\delta}\sin\left(\frac{2x+\delta}{2}\right)\ \sin\left(\frac{\delta}{2}\right) + \sin(x) \simeq 0 \tag{3.5}$$

Crie uma nova função em C++ que permita avaliar esta expressão e comprare com o valor obtido b). Justifique.

4Resolução numérica de problemas

4.1 Sistemas Lineares e Matrizes

Problema 4.1.1

Considere a seguinte matriz:

$$M = \begin{bmatrix} 1.0 & 7.0 & 5.0 & 3.0 & -3.0 \\ 5.0 & 2.0 & 8.0 & -2.0 & 4.0 \\ 1.0 & -5.0 & -4.0 & 6.0 & 7.6 \\ 0.0 & -5.0 & 3.0 & +3.2 & 3.3 \\ 1.0 & 7.0 & 2.0 & 2.1 & 1.2 \end{bmatrix} \tag{4.1}$$

Para a representação de matrizes usaremos a biblioteca C++ *Eigen* cuja documentação pode encontrar aqui. Desenvolva um programa principal `tEigen.C` onde execute as seguintes tarefas:

a. Instancie uma matriz de tamanho fixo que armazena a matriz do enunciado.
b. Troque a linha 1 com a linha 5 usando o método `swap()` .
c. Determine o vector coluna cujos elementos correspondem aos valores máximos absolutos de cada linha da matriz.

Problema 4.1.2

A resolução de sistemas lineares de equações envolva a solução da equação matricial $\mathbf{A}\,\mathbf{x} = \mathbf{b}$. Para tal iremos desenvolver as classes `FCmatrixAlgo` e `EqSolver` . Na primeira, implementam-se os métodos (algoritmos) de redução da matriz de coeficientes (A) como são o da eliminação de Gauss e o da decomposição LU. Na segunda, implementam-se as soluções do sistema de equações.

Consideremos então o sistema linear de equações,

$$\begin{cases} 4x_1 - 2x_2 + x_3 = 11 \quad (1) \\ -2x_1 + 4x_2 - 2x_3 = -16 \quad (2) \\ x_1 - 2x_2 + 4x_3 = 17 \quad (3) \end{cases} \tag{4.2}$$

a. Escreva o sistema de equações na forma matricial, $\mathbf{A}\,\mathbf{x} = \mathbf{b}$.

b. Desenvolva a classe `FCmatrixAlgo` e um conjunto de métodos estáticos de acordo com a seguinte declaração:

```
#include <Eigen/Core>

class FCmatrixAlgo {
  public:
```

```
  FCmatrixAlgo() = default; // compiler do it
  ~FCmatrixAlgo() = default;

  /*
  Implements Gauss elimination
  */
  static void GaussElimination(
              Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic>&,
↪  // matrix coeffs
              Eigen::Matrix<double,Eigen::Dynamic,1>&        //
↪  vector of constants
                            ); //no pivoting
  static void GaussEliminationPivot(
              Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic>&,
↪  // matrix coeff
              Eigen::Matrix<double,Eigen::Dynamic,1>&,       //
↪  vector of constants
              Eigen::Matrix<double,Eigen::Dynamic,1>&        // row
↪  order indexing
                              ); //make pivoting

  /*
  Implements LU decomposition (Doolitle)
  */
  static void LUdecomposition(
              Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic>&,
↪  // matrix coeff
              Eigen::Matrix<int,Eigen::Dynamic,1>&, // row order
↪  indexing
              bool bpivot=false // activate pivoting
                           );

}
```

c. Desenvolva a classe `EqSolver` , de acordo com a seguinte declaração:

```
#include <Eigen/Dense>

class EqSolver {
  public:
  // constructors and destructor
  EqSolver();
  EqSolver(
```

```
            const
            ↪  Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic>&,
            ↪  // matrix coeffs
            const Eigen::Matrix<double,Eigen::Dynamic,1>& // vector of
            ↪  constants
           );
  ~EqSolver() = default;

  // output (optional)
  friend ostream& operator<<(ostream&, const EqSolver&);

  // solvers
  const Eigen::Matrix<double,Eigen::Dynamic,1>& GaussSolver(bool
   ↪  pivot=false);
  const Eigen::Matrix<double,Eigen::Dynamic,1>& LUSolver(bool
   ↪  pivot=false);
  void IterativeJacobiSolver(
       Eigen::Matrix<double,Eigen::Dynamic,1>&, // starting solution
       int& itmax,  //nb of max iterations
       double tol=1.E-3); // tolerance on convergence
  void IterativeGaussSeidelSolver(
       Eigen::Matrix<double,Eigen::Dynamic,1>&,
       int& itmax,
       double tol=1.E-3);

  private:
  Eigen::Matrix<double,Eigen::Dynamic,Eigen::Dynamic> M; //
 ↪  coefficients matrix
  Eigen::Matrix<double,Eigen::Dynamic,1> b; // constants vector
};
```

d. Desenvolva um programa principal onde obtenha as soluções do sistema de equações, aplicando os vários métodos de solução:

- `GaussSolver`
- `LUSolver`
- `IterativeJacobiSolver`
- `IterativeGaussSeidelSolver`

Problema 4.1.3

O circuito que se segue possui oito resistências alimentadas por dois potenciais eléctricos, -10 V e $+10$ V.

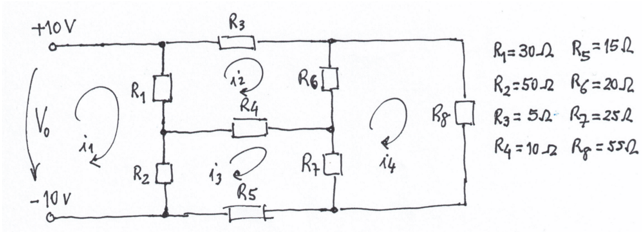


Figura 4.1: Circuito eléctrico

a. Estabeleça as equações do circuito usando as leis de Kirchoff, das malhas $\sum_i V_i = 0$ e dos nós $\sum_j I_j = 0$.
b. Escreva as equações em termos matriciais, identificando a matriz **A** e os vectores **X** e **b**.
c. Resolva o sistema de equações usando os diferentes métodos de solução.

Problema 4.1.4

Resolva o seguinte sistemas de equações lineares:

$$[\mathbf{A}] = \begin{pmatrix} 2 & -2 & 6 \\ -2 & 4 & 3 \\ -1 & 8 & 4 \end{pmatrix} \qquad [\mathbf{b}] = \begin{pmatrix} 16 \\ 0 \\ -1 \end{pmatrix} \tag{4.3}$$

Problema 4.1.5

A definição de uma matriz é mais facilmente implementável usando uma classe que armazene os elementos lineares da matriz, linha ou coluna. Neste problema pretende-se desenvolver a classe vetor `Vec` que depois posteriormente poderá ser usada como objecto na manipulação de matrizes. A declaração da classe que se segue mostra os *data members* que esta deve possuir:

```
class Vec {
  public:
   (...)

  private:
   int N; //number of elements
   double *entries; // pointer to array of doubles
};
```

Proceda então à implementação dos métodos da classe num ficheiro `Vec.C` e à sua respectiva declaração num ficheiro `Vec.h` , de forma
a que a classe possa realizar as operações que se enunciam de seguida:

a. Os construtores desta classe devem ser tais que permitam a construção dos vectores usando as seguintes formas:

```
Vec v1(10); //array with 10 values set to zero
Vec v2(10,5.); //array with 10 values set to 5.

double a[]={1.2, 3.0, 5.4, 5.2, 1.0};
Vec v1(5,a); //array with 5 values given by "a" pointer

Vec v2(v1); //define a vector by using another one
```

b. Defina o método `SetEntries` de forma a permitir redefinir um objecto `Vec` de n elementos, com o conteúdo de um *array*

```
void  SetEntries (int n, double*);
```

Escreva um pequeno programa `main` onde usando o método `SetEntries` copie o conteúdo da matriz C para um array de objectos `Vec`

```
int main() {
  // matrix 5x5
  double cm[][5] = {...};

  //array of Vec's for storing matrix rows
  Vec cv[5];

  //copy rows as arrays into Vecs
  for (int i=0; i<5; i++) {
    cv[i].SetEntries(...);
  }
}
```

$$C = \begin{bmatrix} 1.0 & 7.0 & 5.0 & 3.0 & -3.0 \\ 5.0 & 2.0 & 8.0 & -2.0 & 4.0 \\ 1.0 & -5.0 & -4.0 & 6.0 & 7.6 \\ 0.0 & -5.0 & 3.0 & +3.2 & 3.3 \\ 1.0 & 7.0 & 2.0 & 2.1 & 1.2 \end{bmatrix} \tag{4.4}$$

c. Adicione ao programa a possibilidade de ler a matriz a partir de um ficheiro *matrix.txt* cujo conteúdo seria, para a matriz anterior:

```
// matrix elements
1.0   7.0   5.0    3.0   -3.0
5.0   2.0   8.0   -2.0    4.0
```

```
1.0  -5.0  -4.0    6.0    7.6
0.0  -5.0   3.0   +3.2    3.3
1.0   7.0   2.0    2.1    1.2
```

Utilize a classe auxiliar `FCtools` já construída no secção da Representação dos números, onde implemente os seguintes métodos que permitam fazer a leitura de matrizes escritas em ficheiro:

```
class FCtools {
  public:
   (...)
   //file name, returns lines
   static vector<string> ReadFile2String(string);
   //file name, returns vectors of Vec's
    ↪
   static vector<Vec> ReadFile2Vec(string);
   //file name, returns pointer to array of Vec's, int provides
    ↪  number of lines
   static Vec* ReadFile2Vecp(string, int&);
};
```

A leitura da matrix existente no ficheiro de texto seria então feita da seguinte forma:

```
int main() {
   (...)
   int n=0;
   Vec* cvp = ReadFile("matrix.txt", n);
   (...)
}
```

d. Complete agora o programa `main` anterior de forma a fazer um histograma bi-dimensional e fazer o seu *plot* (método `Draw()`) no ecran. As opções mais frequentemente utilizadas no desenho de histogramas bi-dimensionais são: `{COL, COLZ, LEGO, SURF}` (SURF possui variantes como SURF1, etc…). Para mais detalhes ver a documentação da classe THistPainter. Nota: para aceder aos elementos da classe `Vec` necessita de definir o método `At(int)` .

```
int main() {
  (...)
  // instantiate 2-dim histogram
  TH2F *h2 = new TH2F(...);
  // fill histogram with matrix values
  for (int i=0; i<...) { //loop on rows
```

```
    for (int j=0; j<...) { loop on columns
      h2->Fill(i,j,cv[i].At(j));
    }
  }
  // make plot
  (...)
}
```

e. Para completar a classe `Vec` , devem ser ainda definidos os seguintes *overloading de operadores* de forma que possamos:

- igualar dois vectores (=)
- somar dois vectores (+=, +)
- subtrair dois vectores (-=, -)
- aceder a um elemento i do vector através de $v[i]$
- poder fazer o negativo (-) ou o positivo (+) do vector
- multiplicar dois vectores (a[i] = b[i]*c[i])
- multiplicar um vector por um escalar (a[i] = b[i]*λ)

f. Devem ser também definidos os métodos `size, dot` que permitirão respectivamente:

- `size` : obter a dimensão do vector
- `dot` : fazer o produto interno com outro vector

g. Defina, por último, os métodos `void Print()` e `void swap(int,int)` que permita, respectivamente, imprimir o conteúdo de um vector e trocar dois elementos de ordem.

`Problema 4.1.6`

Neste problema iremos utilizar a classe `Vec` para manipular a matriz **C** dada no problema anterior. Escreva um programa `main` onde realize as seguintes acções:

a. Recupere num *array* de 5 objectos `Vec` as linhas (rows) da matriz **C** e imprima com a ajuda da função `Print()` os valores no ecrã.

b. Obtenha um objecto `Vec` que resulte da multiplicação da constante 2 pela primeira linha da matriz.

c. Obtenha a nova matriz **D** sob a forma de um *array* de 5 objectos `Vec` , que resulte da seguinte operação entre as duas primeiras linhas da matriz **C**:
$L_2 \leftarrow L_2 - \frac{C_{21}}{C_{11}} \times L_1$

d. Multiplique as duas primeiras linhas da matriz **C** e obtenha um novo objecto `Vec` com o resultado.

e. Implemente a função,

```
void swap(Vec&, Vec&);
```

que troque o conteúdo de dois vectores `Vec` . Utilize esta função para trocar linhas da matriz **C**. Por exemplo, troque a 4^a linha com a 5^a linha da matriz.

Problema 4.1.7

Considere a seguinte matriz $\mathbf{M}_{3\times3}$ preenchida com os seguintes números:

$$\begin{pmatrix} 4 & -2 & 1 \\ 3 & 3 & -3/2 \\ 1 & 0 & 3 \end{pmatrix} \tag{4.5}$$

a. Defina a classe em C++ `FCmatrixT` que manipule esta matriz e que armazene o seu conteúdo nas diferentes formas expressas na seguinte definição da classe:

```
class FCmatrixT {
  public:
    FCmatrixT(); // flag=0
    FCmatrixT(double** fM, int fm, int fn); //matrix fm x fn, flag=1
    FCmatrixT(double* fM, int fm, int fn); // flag=2
    FCmatrixT(const vector<Vec>&); // flag=3

    void Print(); // print matrix (use M3 to do it)

  private:
    double** M1;
    double* M2;
    int m; //nb rows
    int n; //nb cols
    int flag; // integer with a definition of which constructor was
     ↪  used

    // store matrix in M3 independently of which constructor is used
    vector<Vec> M3;
};
```

b. Implemente os seguintes métodos da classe que permitem obter os conteúdos das linhas (rows) e colunas (columns) das matrizes.

```
Vec GetRow(int i); // row i
Vec GetCol(int i); // column i
```

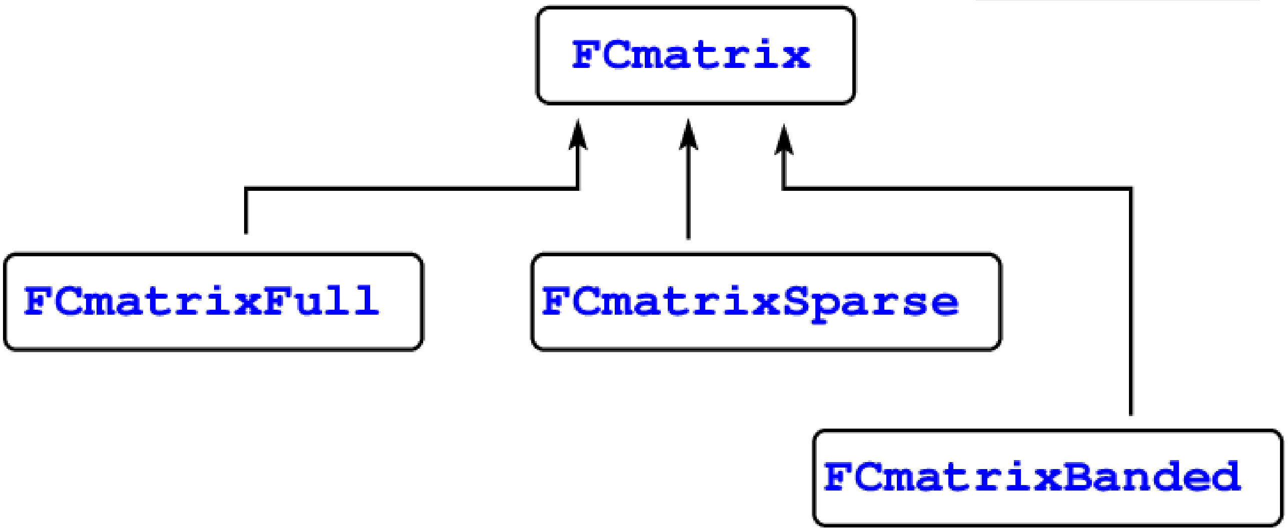
c. Implemente agora o `operador[]` de forma a aceder a cada um dos elementos da matriz.

```
int main() {
  (...)
  // define matrix
  FCmatrixT A ...;
  // print element (1,4)
  cout << A[1][4] << endl;
  // change value of element (1,4) and print it
  A[1][4] = 3.;
  cout << A[1][4] << endl;
}
```

Realize a implementação deste método para os três casos de armazenamento.

`Problema 4.1.8`

O armazenamento e manipulação de matrizes pode ser feito com o auxílio de uma classe genérica de base `FCmatrix` e de classes derivadas que tenham em conta as particularidades dos conteúdos das matrizes. Existem as matrizes que necessitam de um armazenamento integral de todos os elementos (`FCmatrixFull`), outras matrizes que possuem muitos zeros entre os seus elementos (`FCmatrixSparse`) e ainda matrizes que possuem estruturas em banda como por exemplo as matrizes tridiagonais (`FCmatrixBanded`).



a. A classe de base pode ser feita a partir da classe desenvolvida no exemplo precedente. Nela devem ser declarados e implementados os seguintes métodos:

   a.1) os construtores que permitam armazenar os elementos necessários e suficientes para a reconstrução da matriz, na classe "'cpp classe FCmatrix { public: FCmatrix(); FCmatrix(double** fM, int fm, int fn); //matrix fm x fn FCmatrix(double* fM, int fm, int fn); FCmatrix(vector); (…)

```
protected:
 vector<Vec> M;
 string classname;
```

~~~
};
```
~~~

a.2) os métodos puramente virtuais `GetRow` , `GetCol` , `Determinant` que deverão ser implementados nas classes derivadas “‘cpp classe FCmatrix { public: (…) // operators virtual Vec& operator[] (int) = 0; // methods virtual Vec GetRow(int i) = 0; // retrieve row i virtual Vec GetCol(int i) = 0; // retrieve column i virtual double Determinant() = 0; (…)

~~~
 protected:
  vector<Vec> M;
  string classname;
};
```
~~~

a.3) o método `Print` que imprima os elementos armazenados na matriz (e não a matriz reconstruída, porque isso só será possível nos métodos implementados em cada classe derivada)
“‘cpp classe FCmatrix { public: (…) virtual void Print(); (…)

~~~
 protected:
  vector<Vec> M;
  string classname;
};
```
~~~

a.4) Os métodos `GetRowMax` e `GetColMax` :

~~~
- o método *GetRowMax*, retorna o índice da coluna (0,1,...) que possu
- o método *GetColMax*, retorna o índice da linha (0,1,...), a partir
~~~

relativo à escala s (linha a linha)
“‘cpp classe FCmatrix { public: (…) // row max element index
virtual int GetRowMax(int i=0) = 0; // row max element index (scaled by s, from j on)
virtual int GetColMax(int j=0) = 0;

~~~
 protected:
  vector<Vec> M;
  string classname;
};
```
~~~

Nota: usando os métodos *GetRow* e *GetCol* poderíamos definir aqui inteiramente o método Print(), não necessitando de ser definido como virtual.

b. Implemente a classe derivada `FCmatrixFull` , em que todos os elementos da matriz são armazenados, e ainda os métodos que envolvem os diferentes operadores tal como se mostra na declaração seguinte:

```
classe FCmatrixFull : public FCmatrix {
 public:
  // constructors
  FCmatrixFull();
  FCmatrixFull(double** fM, int fm, int fn); //matrix fm x fn
  FCmatrixFull(double* fM, int fm, int fn);
  FCmatrixFull(vector<Vec>);
  // copy constructor
  FCmatrixFull(const FCmatrixFull&);
  // operators
  FCmatrixFull operator+(const FCmatrix&); // add 2 matrices of any
↪   kind
  FCmatrixFull operator-(const FCmatrix&); // sub 2 matrices of any
↪   kind
  FCmatrixFull operator*(const FCmatrix&); // mul 2 matrices of any
↪   kind
  FCmatrixFull operator*(double lambda); // mul matrix by scalar
  Vec          operator*(const Vec&); // multiply matrix by Vec
  // virtual inherited
  // ... retrieve row i (const prevents any change on class elements)
  Vec GetRow(int i) const;
  Vec GetCol(int i) const; // retrieve column i
  double Determinant() const;
  void Print() const;
  void swapRows(int,int);
  ...

 private:
  ... // data members that you find useful to include
};
```

c. Teste as classes desenvolvidas realizando um programa `main` onde manipule as seguintes matrizes:

$$A = \begin{pmatrix} 8 & -2 & 1 & 4 \\ 3 & 1 & -3/2 & 5 \\ 1/2 & 0 & 3 & 3 \end{pmatrix} \qquad B = \begin{pmatrix} 2 & -1 & 3 \\ 1 & 8 & -1/2 \\ 5/2 & 6 & 2 \\ 3 & 4 & 5 \end{pmatrix} \tag{4.6}$$

```
int main() {
  // build matrices
```

```
  A[][4] = {...};
  B[][3] = {...};
  // build objects
  FCmatrixFull MA...
  FCmatrixFull MB...
  // use operators
  double a=2.5;
  FCmatrixFull MC(a*MA); //copy constructor and operator*
  FCmatrixFull MD(MA*MB);
  // print
  MC.Print();
  MD.Print();
  // other methods
  MD.Determinant();
  MC.swapRows(1,2);
  MC.Print();
}
```

Problema 4.1.9

Para a resolução de sistemas de equações é conveniente definirmos a classe `EqSolver` que possua os diferentes métodos de solução.

a. Definamos então a classe `EqSolver` , que implemente os diferentes algoritmos de resolução do sistema.

```
#include "Vec.h"
class EqSolver {
 public:
  EqSolver();
  EqSolver(const FCmatrix&, const Vec&); // matriz M e vector de
↪   constantes B
  // set
  void SetConstants(const Vec&);
  void SetMatrix(const FCmatrix&)
  // Gauss elimination
  Vec GaussEliminationSolver();
  Vec LUdecompositionSolver();

 private:
 // decomposition LU com |L|=1
 void LUdecomposition(FCMatrix&, vector<int>& index);
```

```
// return triangular matrix and changed vector of constants
void GaussElimination(FCmatrix&, Vec&);

FCmatrix *M; //matriz de coeffs
Vec b; //vector de constantes
};
```

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

1)

$$\begin{cases} 4x_1 - 2x_2 + x_3 = 11 \quad (1) \\ -2x_1 + 4x_2 - 2x_3 = -16 \quad (2) \\ x_1 - 2x_2 + 4x_3 = 17 \quad (3) \end{cases} \tag{4.7}$$

2)

$$[\mathbf{A}] = \begin{pmatrix} 2 & -2 & 6 \\ -2 & 4 & 3 \\ -1 & 8 & 4 \end{pmatrix} \qquad [\mathbf{b}] = \begin{pmatrix} 16 \\ 0 \\ -1 \end{pmatrix} \tag{4.8}$$

4.2 Interpolação

Nesta secção de problemas, as classes a desenvolver são: `DataPoints` e `Interpolator` que serão usadas na interpolação de funções.

`Problema 4.2.1`

Para a realização de interpolações, comecemos por definir a classe `DataPoints`, que conterá os dados respeitantes aos pontos bi-dimensionais. Significa isto que teremos que criar os ficheiros `src/DataPoints.h`, que conterá a declaração da classe e `src/DataPoints.C`, que conterá o código C++ dos métodos declarados.

```
class DataPoints {
public:

  // constructors, destructor

  DataPoints() = default;
  DataPoints(int N, double* x, double* y); // build DataPoints from
                                           // C-arrays of x and y
                                           ↪  values
  DataPoints(const std::vector< std::pair<double,double> >&);
  DataPoints(const std::vector< double>& x,
             const std::vector< double>& y);
  virtual ~DataPoints();

  // getters

  const std::vector< std::pair<double,double> >& GetPoints();
  void GetGraph(TGraph&);

  // draw points using ROOT object TGraph

  virtual void Draw();

  // friend functions (optional)

  friend std::ostream& operator<< (std::ostream&, const DataPoints&);

protected:
  std::vector< std::pair<double,double> > P; // points
};
```

E de seguida a classe `Interpolator`, que herda da classe `DataPoints`:

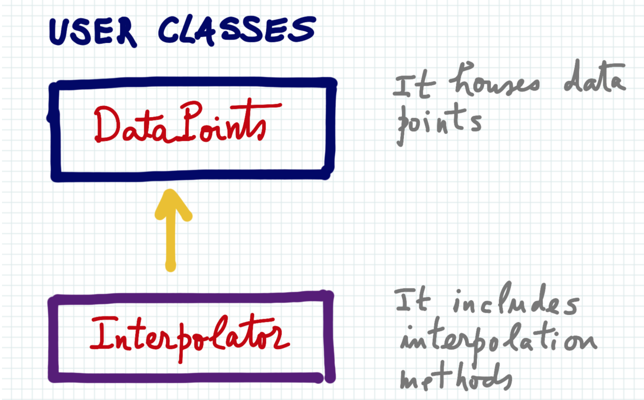


Figura 4.2: Esquema de implementação das classes `DataPoints` e `Interpolator`

Exemplo de declaração para a classe `Interpolator` :

```
class Interpolator : public DataPoints {
   public:

     // constructors, destructor

     Interpolator() = default; //default constructor
     Interpolator(int N, double* x, double* y); // build DataPoints from
                                                // C-arrays of x and y
                                                ↪  values
     Interpolator(const std::vector< std::pair<double,double> >&);
     Interpolator(const std::vector< double>& x, const std::vector<
↪   double>& y);
     ~Interpolator();

     // interpolation methods

     double InterpolateLagrange(double); // Lagrange interpolation
     double InterpolateNewton(double); // Newton interpolation
     double InterpolateSpline3(doible); // spline3

     // draw points and function

     void Draw(std::string s); // s="Lagrange", "Neville", "Spline3"

     // getters

     const TF1& GetFunction(std::string s); // s="Neville", "Spline3"
```

```
  private:
    std::map<std::string,TF1*> MI; // key="Lagrange", "Neville",
     ↪  "Spline3"
};
```

Nota:
O `data member map` da classe `Interpolator` permite guardar as diferentes funções interpoladoras tipo `TF1` no objecto.

`Problema 4.2.2`

Usando as classes construídas anteriormente e dados os seguintes pontos,

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

realize um programa `main` que determine o valor $y(0)$ usando os diferentes métodos:

a. o método de Lagrange
b. o método de Neville
c. o método spline3

`Problema 4.2.3`

Usando as classes construídas anteriormente e dados os seguintes pontos,

coo					
x	1	2	3	4	5
y	0	1	0	1	0

realize um programa `main` que determine e desenhe a função interpoladora dos pontos usando o método spline cúbico.

`Problema 4.2.4`

A realização da experiência do pêndulo simples de massa $m = 500$ gramas e comprimento $L = 4$ metros em laboratório, permitiu obter as seguintes medições para as oscilações:

tempo (sec)	ângulo (rad)	velocidade (rad/sec)
0.71	0.806	-1.594
1.51	-0.702	-1.700
2.31	-1.395	0.065
3.11	-0.608	1.781
3.91	0.889	1.495
4.71	1.371	-0.347
5.51	0.391	-1.918
6.31	-1.051	-1.259
7.11	-1.314	0.626
7.91	-0.162	1.996
8.71	1.183	1.001
9.51	1.225	-0.900

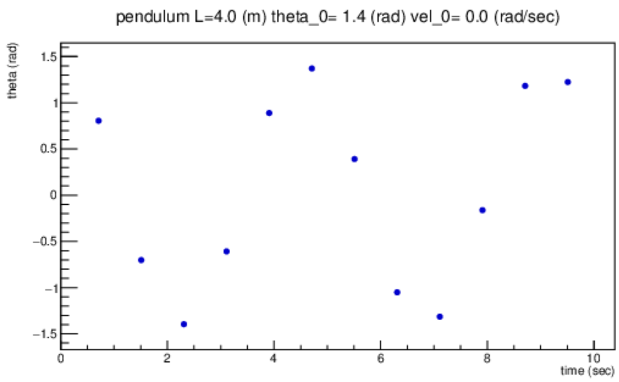


Figura 4.3: Valores das oscilações do pêndulo simples com as condições iniciais: $\theta(t_0) = 80$ graus, $\dot{\theta}(t_0) = 0$

Realize uma função interpoladora da velocidade: $\frac{d\theta}{dt}(t)$

4.3 Derivação e integração numéricas

Nesta secção de problemas, as classes a desenvolver são: `IntegDeriv` e `Functor` que serão usadas na derivação e integração de funções. A classe `Functor` permitirá a definição de funções matemáticas e a classe `IntegDeriv` permitirá a integração e derivação numérica com os diferentes métodos.

`Problema 4.3.1`

Para a definição de funções matemáticas iremos proceder ao desenvolvimento da classe `Functor` , cuja declaração assume a forma:

```
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;
};
```

Esta classe possui um método `Draw()` que deve ser implementado e que permite fazer o *display* das funções e define o `operator()` . A utilização do mecanismo de herança do C++ permite definirmos qualquer função e usar o método `Functor::Draw()` para fazer o seu *display.

Construa um programa teste, que faz uso da classe `Functor` e onde defina a função $f(x) = x^4\,log(x + sqrt(x^2 + 1))$.

```
#include "Functor.h"
class MyFunction: public Functor {
public:
```

```
 MyFunction() : Functor("MyFunction") {};
 ~MyFunction() = default;

  double operator()(double x) {
    return pow(x,4)*log(x+sqrt(x*x+1));
  }

};

int main() {

  // create MyFunction object
  MyFunction F1;

  // create x vector with values: 0.0, 0.1, 0.2, ...
  vector<double> x(21,0); // create vector of 10 values filled with
↪   zero's
  double xi=0., step=0.1;
  transform(x.begin()+1, x.end(), x.begin()+1,
       [xi, step](double a) {
       static double buffer=xi; // buffer will keep its value between
        ↪   calls
       return buffer+=step;
       }); // 0.1 is the increment

  // calculate funtion value at x
  for (auto a: x) {
     cout << F1(a) << " " << flush;
  }

  // Draw function
  F1.Draw(0,2,1000); // using 1000 equally spaced values

}
```

Problema 4.3.2

Para a integração e derivação de funções vamos definir a classe `IntegDeriv` onde se definirão os métodos de derivação e ainda de integração trapezoidal, Simpson e de monte-carlo. Implemente os algoritmos e estruture a classe:

```
class IntegDeriv {

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

  void Dump(double, double);

  // derivative methods

  double D1backward(double h, double x);
  double D1forward(double h, double x);

  double D2backward(double h, double x);
  double D2forward(double h, double x);

  double D4backward(double h, double x);
  double D4forward(double h, double x);

  (...)

  // integration methods

  void trapezoidalRule(double xi, double xf,
                       double& Integral, double& Relative_Error);
  void simpsonRule(double xi, double xf,
                   double& Integral, double& Relative_Error);
  void integrateMC(double xi, double xf,
                   double& Integral, double& Relative_Error, int& Ngen);
  (...)

private:
  Functor& F;
};
```

Determine o integral $\int_0^{\frac{\pi}{2}} cos(x)\, dx$.

Problema 4.3.3

O oscilador harmónico, cujo potencial é dado por $V(x) = kx^2$, possui um período de tempo $T^{-1} \propto \sqrt{k}$; isto é, o período de tempo não depende da amplitude da oscilação. Um potencial diferente deste com $V(x) = V(-x)$ dará origem a um oscilador anarmónico, onde o período dependerá da amplitude da oscilação.

a. A equação do movimento do oscilador pode ser obtida partindo da conservação de energia:

$$E = \frac{1}{2} m \left(\frac{\mathrm{d}x}{\mathrm{d}t} \right)^2 + V(x) \tag{4.9}$$

Sabendo que em $t = 0$, a massa m se encontra em repouso em $x = a$, mostre que o período do movimento pode ser calculado como sendo:

$$T = \sqrt{8m} \int_0^a \frac{\mathrm{d}x}{\sqrt{V(a) - V(x)}} \tag{4.10}$$

b. Verifique se expressão anterior é dimensionalmente correcta.

c. Suponha que o termo do potencial é dado por $V(x) = x^4$ e a massa do corpo é $m = 1$ Kg. Escreva um programa em C++ que calcule o período do oscilador para amplitudes de $a = 1.$ até $a = 3.$ com um passo de 0.1.

Problema 4.3.4

Neste problema pretende-se calcular a eficiência de uma lâmpada de tungsténio para uma radiação na região do visível ($\lambda_1 - \lambda_2\,\mathrm{nm}$). A potência emitida pela lâmpada por unidade de comprimento de onda λ obedece à lei de radiação de Planck,

$$I(\lambda) \propto \frac{\lambda^{-5}}{e^{\frac{hc}{\lambda k_B T}} - 1} \tag{4.11}$$

onde T é a temperatura em Kelvin, h é a constante de Planck e k_B a constante de Boltzman. $k_B = 8.617 \times 10^{-5}\,\mathrm{eV/K}$
$h = 4.136 \times 10^{-15}\,\mathrm{eV\,s}$

a. Determine a eficiência da lâmpada para o intervalo de comprimentos de onda $[\lambda_1, \lambda_2]$.

b. Faça a transformação de variável

$$x = \frac{hc}{\lambda k_B T}$$

e mostre que a eficiência se pode escrever como,

$$\eta = \frac{15}{\pi^4} \int_{x_2}^{x_1} \frac{x^3}{e^x - 1} dx \tag{4.12}$$

c. Escreva um programa em C++ que determine a eficiência da lâmpada de tungsténio à temperatura de 3000 K, na região de comprimentos de onda $[400, 700]$ nm.

d. Realize um gráfico com a evolução da eficiência com a temperatura do filamento, no intervalo de T=300 K até 10 000 K.

4.4 Raízes de funções

Problema 4.4.1

O ponto de Lagrange é o local entre a Terra e a Lua onde um satélite aí colocado, possuirá uma órbita em sincronia total com a Lua. Do ponto de vista da dinâmica, nesse ponto o balanço das forças gravíticas da Terra e da Lua produzem uma força resultante que assegura a manutenção do satélite na órbita.

a. Assumindo órbitas circulares, mostre que a distância radial a partir do centro da Terra a que se encontra o satélite, obedece à seguinte equação:

$$\frac{G\,M_E}{r^2} - \frac{G\,M_L}{(R-r)^2} = \omega^2 r \tag{4.13}$$

onde ω é a veclocidade angular do satélite e da Lua.

b. Construa um programa em C++, utilizando vários métodos de determinação de raízes, para calcular numericamente o raio orbital r.

Problema 4.4.2

Um canhão encontra-se colocado a uma altura h e pode ser disparado com um ângulo θ, medido com a horizontal, variável. Desprezando as forças de atrito,

a. Escreva as equações do movimento da bala.

b. Mostre que a distância horizontal percorrida pela bala obedece à seguinte equação:

$$x = \frac{v_o\,\cos\theta}{g}\left(v_0\,\sin\theta + \sqrt{(v_0\,\sin\theta)^2 + 2hg}\right) \tag{4.14}$$

c. Construa um programa em C++ que receba a partir do terminal a velocidade inicial da bala (v_0) e a distância horizontal que pretende alcançar (x) e calcule o ângulo θ com que o disparo deve ser feito.

4.5 Métodos de monte-carlo

Problema 4.5.1

Os geradores de números aleatórios usam relações do tipo: \

$$I_{i+1} = (aI_i + c)\%m \tag{4.15}$$

a. Construamos uma classe em C++ `FCrand` que implemente o método das congruências lineares para geração de números aleatórios. Implemente o *default constructor* que tenha como argumento um parâmetro semente (*seed*) usando a função `time` (unix time in seconds).

```
class FCrand {
 public:
   FCrand(int seed); // constructor (incomplete declaration)
   // 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:

a.1) distribuições de cada um dos números aleatórios para um milhão de amostragens. Determine o valor médio e o desvio padrão da amostra e compare com os valores esperados. a.2) Divida a distribuição em 10 intervalos e coloque num gráfico os valores médios de cada intervalo bem como o erro do valor médio. a.3) Um teste à independência dos números aleatórios produzidos por um gerador é o chamado teste de auto-correlação,

$$C_k = \frac{< x_{i+k}\, x_i > - < x_i >^2}{< x_i^2 > - < x_i >^2} \tag{4.16}$$

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 $k = 1, 2, ..., 1000$. a.4) distribuição de um número aleatório .vs. outro (escolha quais) a.5) distribuição de um número aleatório .vs. outro .vs. outro (escolha quais)

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

Problema 4.5.2

A área de um círculo é dada por $A = \pi r^2$. Assim, de forma indirecta, pode-se calcular o valor de π realizando o cálculo da área de um círculo de raio 1.

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 obtenha o valor calculado bem como o seu erro, em função do número de amostragens $N = (10000, 100000, 1000000)$.

Problema 4.5.3

A classe `IntegDeriv` pode ser expandida de forma a incluir os métodos de integração de Monte-Carlo, simples, *importance sampling* ou de *aceitação-rejeição*. No método de *importance sampling* é utilizada uma função auxiliar $p(x)$ para geração dos números aleatórios, de forma a minimizar-se o erro da integração. A geração dos aleatórios é feita, recorrendo à função acumulada $y(x) = \int_a^x p(x)dx$,
fazendo a sua inversão de forma a obter-se $x(y)$. As funções $p(x)$ e $x(y)$ devem ser passadas como parâmetros do método `ImportanceSampling` da classe.

```
#include <functional> //std::function

class IntegDeriv  {
  public:

   (...)

   void integrateMC(double xi, double xf,
                    double& Integral, double& Relative_Error, int&
                     ↪  Ngen);

   // importance sampling

   void integrateMC_IS(double xi, double xf,
                       std::function<double(double)> px, // p(x)
                       std::function<double(double)> xy, //x(y) with
                        ↪  y[0,1]
                       double& Integral, double& Relative_Error, int&
                        ↪  Ngen);
```

```
    private:
     (...)
  };
```

Problema 4.5.4

Determine o integral,

$$\int_0^1 \frac{dx}{1+x^2} \tag{4.17}$$

usando:

a. o método trapezoidal utilizando um passo $h = 0.2$
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.

Problema 4.5.5

Determine o integral da função gaussiana $g(x),

$$\int_{x_1}^{x_2} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}\, dx \tag{4.18}$$

usando:

a. 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.
b. o método de monte-carlo com importance sampling, fazendo uso da função auxiliar de Cauchy,

$$p(x) = \frac{1}{1+x^2} \tag{4.19}$$

Problema 4.5.6

Determine o integral,

$$\int_0^1 \int_0^1 \frac{x^2 - y^2}{(x^2+y^2)^2}\, dy\, dx \tag{4.20}$$

usando o método de monte-carlo, usando 100 e 1000 amostragens.

Problema 4.5.6

No problema da agulha de Buffon, colocado pelo naturalista francês Buffon em 1733, pretende-se saber qual é a probabilidade de uma agulha de comprimento ℓ lançada aleatoriamente num solo com riscas paralelas espaçadas de d, cair sobre uma linha. Este problema teve uma solução cerca de quatro décadas depois de ter sido colocado e é um problema típico que pode ser resolvido pelo método de simulação monte-carlo.

Consideremos uma agulha de comprimento $\ell = 2cm$ e riscas de espessura desprezável espaçadas de $d = 10cm$.

a. Mostre que a probabilidade de uma agulha cruzar uma linha/risca do solo é dada por

$$P = \frac{2\,\ell}{\pi\,d}$$

Consideremos agora um quadrado de dimensão $100 \times 100cm$ onde existem onze riscas horizontais (paralelas ao eixo x) espaçadas de $10cm$. Um acontecimento na simulação de monte-carlo consiste em colocar o centro de uma agulha de forma aleatória no interior do quadrado e ainda gerar de forma aleatória a sua direcção θ.

b. Construa as classes `Point2D` e `line2D` que permitam construir os elementos riscas e agulha.

Point2D.h

```
class Point2D {
  public:
   Point2D(double fx=0., double fy=0.);
   (..)

  private:
   double x; // x coo
   double y; // y coo
};
```

Point2D.C

```
class line2D {
 public:
  // constructors
  // ... provide extreme points
  line2D(Point2D P1=Point2D(), Point2D P2=Point2D());
  // ... provide line center, line length and line theta (angle with
   ↪  X-axis)
  line2D(Point2D P1=Point2D(), double length, double ftheta);

  // set line: line center, line length and line theta (angle with
   ↪  X-axis)
  void SetLine(Point2D, double, double);

  // check if the two lines are crossing
  bool Crossing(const line2D&); // true if two lines crossing

 private:
  vector<Point2D> vP;
```

```
  Point2D center;
  double theta;
};
```

c. Construa um programa *main* onde comece por definir, com o auxílio das classes acima definidas, o quadrado (frame) de $100 \times 100cm$ com onze riscas e de seguida gere um número $N = 100$ agulhas aleatórias.

```
int main() {
  // define frame
  line2D FRAME[11];
  for (int i=0; i<11; i++) {
      FRAME[i].SetLine(Point2D(50., i*10.), 100., 0.);
  }
  // generate random needles
  const int N = 100;
  line2D NEEDLES[N];
  int Nc = 0; //counter of needles crossing
  for (int i=0; i<N; i++) {
     // generate needle center position
     (...)
     // generate needle direction
     (...)
     // seet needle data
     NEEDLES[i].SetLine(...);
     // check if needle is crossing a line
     for (int j=0; j<11; j++) {
        if (NEEDLES[i].Crossing(FRAME[j])) {
          Nc++;
          cout << "needle nb " << i << flush;
          cout << " is crossing frame line nb " << j << endl;
        }
     }
  }
  // compute probability
  double Prob = Nc/N;
}
```

d. Represente graficamente o valor da probabilidade em função do número de lançamentos aleatórios da agulha.

e. Represente graficamente a o quadro das riscas (a negro) conjuntamente com as agulhas lançadas aleatoriamente. As agulhas que atravessam riscas deverão ser representadas a vermelho e as outras a verde.

4.6 Resolução numérica de equaçoes diferenciais ordinárias

Problema 4.6.1

A resolução de equações diferenciais ordinárias (ODEs) por via numérica, exige a implementação dos diferentes métodos iterativos (Euler, Runge-Kutta, etc.) de forma a obter-se a solução das equações. Dado que o número de variáveis dependentes do sistema, correspondentes ao número de equações diferenciais de primeira ordem a resolver, podem ser diferentes em cada problema, será útil implementar uma classe em C++ suficientemente flexível e capaz de lidar com os diferentes números de variáveis. Além do mais, na resolução do sistema de equações diferenciais, a variável independente (habitualmente o tempo) e as variáveis dependentes (os graus de liberdade necessários à resolução do problema bem como as suas primeiras derivadas) são iteradas e os seus valores registados. Torna-se por isso conveniente, definir uma classe `ODEpoint` que armazene em cada iteração os valores das variáveis. Esta classe é realizada sobre uma classe de base `Xvar` que armazena as variáveis dependentes do problema.

a. Implemente as classes *ODEpoint* e *Xvar*, de acordo com a declaração que se segue.

ODEpoint.h

```
// class with dependent variables

class Xvar {

 public:

  Xvar() = default;
  Xvar(int); // number of dependent variables
  Xvar(std::vector<double>); // passing vector
  // using initializer list to build object: Xvar({1,2})
  Xvar(const std::initializer_list<double>& v);
  ~Xvar();

  Xvar(const Xvar&); // copy constructor
  Xvar& operator=(const Xvar&); // assignment operator
  Xvar operator+(const Xvar&); // operator+
  double& operator[](int);  // Xvar[i]

  friend Xvar operator*(double, const Xvar&); // scalar*Xvar
  friend std::ostream& operator<< (std::ostream&, const Xvar&);

  std::vector<double>& X(); // accessor to vector of vars

 protected:
```

```
  std::vector<double> x;

};


class ODEpoint : public Xvar {
 private:
  double t; // time

public:
  ODEpoint();
  ODEpoint(double t_, Xvar a_);
  ODEpoint(double t_, const std::vector<double>& v);
  ODEpoint(double t_, const std::initializer_list<double>& v);

  void SetODEpoint(double t_, Xvar& p);
  void SetODEpoint(double t_, const std::initializer_list<double>&
   ↪  v);
  void SetODEpoint(double t_, std::vector<double> v);

  double& T(); // accessor to time

  friend std::ostream& operator<< (std::ostream& s, const ODEpoint&
   ↪  p);

};
```

Problema 4.6.2

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 $x \in [0, 2.0]$, usando um passo de 0.1:

a. usando método de Euler
b. usando método trapezoidal
c. usando método RK2
d. usando método RK4

Problema 4.6.3

Considere o sistema de duas equações diferenciais de primeira-ordem,

$$\begin{cases} \frac{dz}{dx} & = & sin(x) + y \\ \frac{dy}{dx} & = & cos(x) - z \end{cases} \tag{4.21}$$

cujos valores iniciais são: $z(0) = y(0) = 0$. Determine a solução das equações $y(x)$ e $z(x)$ no intervalo $x \in [0, 2.0]$ usando um passo de 0.1, com o método Runge-Kutta de ordem-4 (RK4).

Problema 4.6.4

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?

Problema 4.6.5

Os problemas físicos oscilatórios como sejam o de uma massa m ligada a uma mola de constante de restituição k ou o do pêndulo gravítico de comprimento ℓ no limite das pequenas oscilações, implicam a resolução da equação diferencial:

$$\ddot{x} + \omega^2\, x = 0 \tag{4.22}$$

Determine numericamente a solução da equação tendo em conta as condições iniciais, $x(0) = 1$ cm e $x(\dot{0}) = 0$:

a. utilizando o método de Taylor de 2a ordem (Stormer-Verlet)
b. utilizando o método de Runge-Kutta de 4a ordem

Problema 4.6.6

Considere uma massa $m = 1\ Kg$ suspensa por um fio de comprimento $\ell = 1\ m$ e sujeita a uma força de atrito $\vec{F}_a = -k\vec{v}$, com $k = 0.3\ kg/s$.

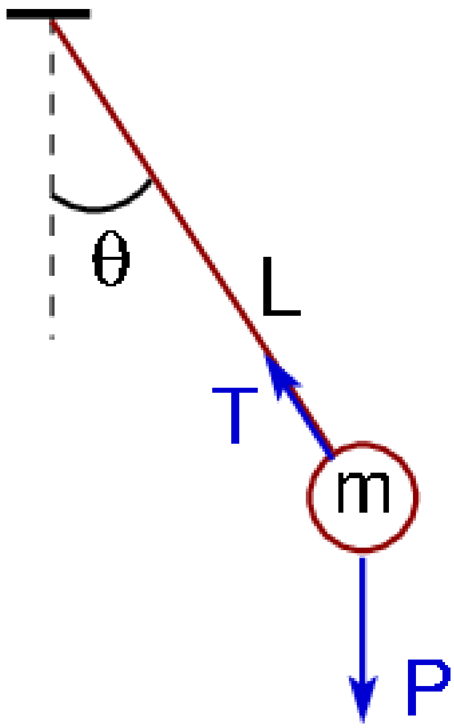


Figura 4.4: pêndulo gravítico

a. Escreva a equação do movimento
b. Escreva o sistema de equações diferenciais de primeira ordem e identifique as variáveis dependentes
c. Tendo em conta que a massa é deixada oscilar a partir do ângulo inicial $\theta_i = 70$ graus, partindo do repouso, determine numericamente a solução das equações utilizando o método de Runge-Kutta de 4a ordem (RK4).
   Qual a amplitude de oscilação ao fim de 10 minutos?
   Faça o plot da amplitude de oscilação ao longo do tempo.

4.7 Resolução numérica de equaçoes diferenciais parciais (PDE's)

`Problema 4.7.1`

Uma barra cilíndrica de diâmetro $D = 10\ cm$ e comprimento de $L = 100\ cm$ está em contacto com uma fonte de calor à temperatura de $T_s = 40$ graus Celsius.

a. Admitindo que a barra está isolada, conduzindo calor entre as duas extremidades, a temperatura da barra obedece à seguinte equação:

$$\frac{\partial^2 T}{\partial x^2} = 0 \tag{4.23}$$

resolva numericamente esta equação sabendo que as temperaturas nas extremidades da barra são $T(x = 0) = 40$ e $T(x = L) = 10$ graus Celsius. Utilize como passo $s=10; cm $ e $s = 2\ cm$. Produza um plot.

b. Admitindo que é retirado o isolamento da barra e que esta conduz calor por convecção também, a temperatura da barra obedece à seguinte equação:

$$\frac{d^2T}{dx^2} = \frac{4h}{kD}(T - T_a) \tag{4.24}$$

onde $T_a = 23$ graus Celsius é a temperatura do ar e os coeficientes $h = 17\ W.m^{-2}.K^{-1}$ e $k = 206\ W.m^{-1}.K^{-1}$ Resolva numericamente esta equação sabendo que a temperatura na extremidade da barra é $T(x = 0) = 40$ graus Celsius e que o gradiente de temperatura na outra extremidade é nulo ($\frac{dT}{dx} = 0$). Produza um plot.