
Exercícios de Física Computacional

Mestrado em Engenharia Física-Tecnológica (MEFT)

Fernando Barao
Departamento de Física do Instituto Superior Técnico
Ano Lectivo: 2018-19

fernando.barao@tecnico.ulisboa.pt

versão: 14 de Novembro de 2018

Conteúdo

I	Elementos de programação com objectos	5
1	Programação em C/C++	7
1.1	Revisão de C/C++	7
1.2	gestão de memória e passagem de parâmetros	9
1.3	biblioteca STL (Standard Template Library)	12
1.4	Programação com classes	16
2	Programação com objectos ROOT	23
3	Representação dos números em computador e arredondamentos	29
II	Resolução numérica de problemas em C++	33
4	Sistemas Lineares	35
5	Raízes de funções	45
6	Interpolação	47
7	Derivação e Integração de funções	49
8	Métodos Monte-Carlo	51
9	Resolução de equações diferenciais	57
III	Problemas de síntese e revisão	61
10	Problemas com objectos C++	63
11	Problemas numéricos	69

Parte I

Elementos de programação com objectos

Programação em C/C++

Revisão de C/C++

Exercício 1 : O programa `addnumbers.C` é suposto adicionar todos os números inteiros entre dois números introduzidos pelo utilizador no programa.

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

Exercício 2 : Considere a fórmula matemática,

$$z(x) = x + f(x)$$

com,

$$f(x) = \sin^2(x)$$

Realize um programa em C/C++ que calcule $z(x)$ para os valores de $x = 0.4, 2.1, 1.5$ e imprima os resultados no monitor do computador.

Nota: a função $f(x)$ deve ser construída autonomamente

Exercício 3 : O espaço em memória ocupado pelas variáveis depende do seu *tipo*. Realize um programa em C/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**

Exercício 4 : Realize um programa em C/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 . Determine a precisão obtida em *float* e *double*.

Exercício 5 : Aspectos relacionados com a implementação do C++ em cada arquitectura podem ser encontrados na *C++ standard library* `<limits>` .

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

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

Exercício 6 : Calcule o quadrado de um número inteiro positivo, x^2 , usando somente as operações:

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

Exercício 7 : Tendo como base o programa **addnumbers.C**, 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 em tipo **int** e em **double**. Compare os resultados para o caso (1, 5000).

Exercício 8 : Realize um programa em C/C++ composto das funções **main()** e **int fact(int)** que determine o factorial do número n introduzido pelo utilizador no programa, $n!$. Obtenha primeiramente o código objecto **.o** e só depois o código executável **.exe**.

Exercício 9 : Realize um programa em C/C++ que calcule:

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

Codifique a soma numa função do tipo,

```
double Sum(int* vi, int*vj); //vi, vj= limites de i e j
```

e realize um programa *mainSum.C* donde chame a função.

Exercício 10 : A função *rand()* declarada em `<cstdlib>` gera um número pseudo-aleatório entre 0 e `RAND_MAX`. Realize um programa em C++ que:

- gere 1000 números aleatórios x entre $x_{min} = 5$ e $x_{max} = 55$.
- determine o valor de $y = \frac{x}{x-10}$ para cada aleatório.
- determine o valor médio de x e o seu desvio padrão.

Exercício 11 : Pretende-se calcular a soma dos seguintes valores,

$$0.1 + 0.2 + \dots + 5.4$$

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.

Exercício 12 : Uma massa é deixada cair de uma altura h , partindo do repouso.

- 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. Despreze a resistência do ar.
- Quanto tempo demora a queda para uma altura de 100 metros?

Exercício 13 : Um satélite orbita circularmente em torno da terra a uma dada altitude h e possuindo um período de tempo T .

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

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

onde R representa a raio da Terra.

- 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.
- 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.

$$G = 6.67 \times 10^{-11} \text{ m}^3 \text{ Kg}^{-1} \text{ s}^{-2}$$

$$M = 5.97 \times 10^{24} \text{ Kg}$$

$$R = 6371 \text{ Km}$$

gestão de memória e passagem de parâmetros

Exercício 14 : No programa que se segue fazem-se *calls* às funções *fintv* e *fdoublev* que retornam ponteiros para arrays de inteiros 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;
}
```

- a) Verifique se o exemplo de código está funcional e em caso negativo, corrija-o e complete com a função que falta.
- b) Com base nas funções anteriores, realize novas funções *fintv* e *fdoublev* que permitam a criação de tensores de até dimensão 3. Coloque funcional o seguinte programa *main*.

```
int main() {
    // retornar uma matriz de inteiros de dimensão 100x50 inicializados a 1
    int ***a = fintv(100,50);
    // retornar um tensor de double de dimensão 100x50x20 inicializados a 5.
    double ***b = fdoublev(100, 50, 20);
    double ***c = fdoublev(100, 50);
}
```

- c) Realize as funções que façam o *printout* para o ecrã dos valores dos tensores

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

- d) Finalmente, no final do programa *main* apague a memória alocada.

Exercício 15 : 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* donde os referencie.

- a) o valor de $f(x)$ é retornado 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);
```

- d) modifique o método anterior de forma a que o valor em graus de x e o *pointer* não sejam modificáveis no interior do método.

*Nota 1: a variável x é passada para a função **func** por cópia, o que pode tornar questionável o uso da declaração **const**. Habitualmente o uso do **const** está ligado ao facto de se querer impedir a modificação de uma variável e que esta modificação se possa reflectir no programa que chama a função (para que isto acontecesse neste caso,*

a variável teria que ser passada por referência). Mas neste caso, uma vez que é feita a cópia da variável, mesmo que se altere esta cópia na função, o exterior não tomaria conhecimento dessa modificação.

Nota 2: algo semelhante se passa com a declaração **const** do ponteiro. Como neste caso é feita uma cópia da variável que guarda o ponteiro, no interior da função, mesmo que esta variável seja alterada não há repercussão sobre o valor para onde aponta.

Exercício 16 : Um método/função em C++ desenvolvido para calcular a soma dos elementos contidos num *array*, possuía a seguinte declaração:

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

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

Exercício 17 : Realize o seguinte códigos em C++:

- uma função que inicialize uma variável inteira com um valor aleatório e retorne o seu *pointer* :

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

- uma função que inicialize uma variável inteira com um valor aleatório e retorne o 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.

- 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.

Exercício 18 : Realize um código C++ no qual se definam métodos que realizem as seguintes tarefas:

- calcular e retornar o traço da matriz

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

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

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

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

```
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.

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

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

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

Exercício 19 : Realize um método em C++ e teste-o, 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.

biblioteca STL (Standard Template Library)

Exercício 20 : Realize uma função *rand2vec* e um programa *main()* onde se teste a função, cujo objectivo é proceder à geração de n números aleatórios x com valores compreendidos entre 0 e 360 e que devem ser devovidos ao programa principal usando a estrutura *vector<double>*.

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

Exercício 21 : Realize uma função *array2vec* e o respectivo programa *main()*, cujo objectivo é transferir um *array* de números inteiros para uma estrutura STL *vector*.

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

- a) aplique a função aos *arrays* seguintes: $a = (1, 10, 5, 6, 9, 3)$ e $a = (2, 5, 5, 7, 3)$
- b) elabore numa função *array2vecs*, utilizando a biblioteca `<algorithm>`, a seriação dos valores de cada vector, quer na ordem crescente, quer na ordem decrescente. Retorne o vector ordenado.

```
vector<int> array2vecmaxs(int n, int* a);
```

- 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);
```

- e) realize as modificações necessárias nos métodos desenvolvidos de forma a impedir que os valores n e a seja modificado no interior das funções.
- f) realize a desalocação de memória que tenha utilizado antes do programa terminar.

Exercício 22 : Neste exercício pretende-se explorar os *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 estrutura de dados *ATOM*.

```
int main() {
    // fill with 1st 6 elements
    ATOM hydrogen;
    hydrogen.A = 1;
    hydrogen.Z = 1;
    hydrogen.mass = 938.89; //MeV - natural units
    hydrogen.name = "Hydrogen";

    //allocate and fill vector
    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 estrutura *ATOM* num ficheiro *header atom.h* que contenha os dados de um elemento da tabela periódica
- incluir num vector os primeiros 6 elementos da tabela periódica
- imprimir no ecrã 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 ecrã todas as entradas do mapa
    ...
}
```

Exercício 23 : 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,

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

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".

a) Realize então o seguinte código:

- uma função *GetRandom* que retorne um número aleatório no intervalo $[x_{min}, x_{max}]$

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

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

- uma função *Fobs* que retorne a distribuição acumulada da variável aleatória x num número de intervalos N ,

$$F(x_m) = \sum_{i=1}^m x_i \quad m = 1, 2, \dots, N$$

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.

```
// returns accumulated distribution on x intervals
// NCALLS = number of times generation function is called
// vector<double> x = x boundaries on the N intervals (N+1 boundaries)
// double (*f) (double, double) = pointer to generating function

vector<double> Fobs(int NCALLS, vector<double> x, double (*f) (double, double))
```

Nota: garanta que a distribuição acumulada é normalizada a 1.

- uma função *KolmogorovTest* que retorne a diferença máxima entre a distribuição acumulada e a distribuição de referência

```
// 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)
```

- Realize agora um programa *main* que faça a geração de 10 000 números aleatórios no intervalo $[0, 5]$ (dividido em bins de 0.1) e calcule no final a diferença de kolmogorov.
- Repita o procedimento para 1 000 amostras de 10 000 números aleatórios e faça um *plot* com a distribuição das diferenças de kolmogorov.

Exercício 24 : Pretende-se realizar uma estrutura *map* usando a biblioteca STL do C++ que armazene matrizes de dimensão $n \times m$ qualquer, usando uma chave do tipo *string*, emparelhada com uma estrutura *vector*.

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

- defina uma estrutura STL capaz de armazenar as matrizes abaixo, definindo uma função que devolva a estrutura STL

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

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

- 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 e retorne a matriz.

```
---? Mmapfind(string c);
```

Programação com classes

Exercício 25 : A classe *Box* incompleta é descrita no seguinte ficheiro *header*:

```
Box.h  
class Box {  
public:  
Box(); //cubo de lado 1  
Box(float fx, float fy, floatz);  
...  
private:  
float x;  
float y;  
float z;  
};
```

- a) O seguinte programa *main* 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;  
  
    // somar dois cubos  
    Box B3 = B1 + B2;  
  
    // criar dois paralelepipedos  
    Box B5(1,1,2);  
    Box B6(B5);  
  
    // somar os dois paralelepipedos  
    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 os seus ponteiros:

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

Complete a declaração da classe *Box* e implemente o código C++ necessário.

Exercício 26 : Construi-se uma classe genérica *pessoa* que pretende possuir as 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 *function 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 construa 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* pointers 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.

Exercício 27 : Construa agora uma classe *alunoIST* que derive da classe *pessoa*. A nova classe deverá ter novos *data members* como por exemplo:

- Número do aluno: *int number*
- Curso: *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;
}
```

- Implemente o código da nova classe.
- Construa um *array* de objectos *alunoIST* com conteúdo.
- Construa uma função,

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

que receba um *pointer* genérico para um *array* de *pointers* 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*)?

Exercício 28 : 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 os 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 de pointers de objectos
    int Nalunos;
}

```

- a) Complete a declaração da classe de forma a incluir os seguintes métodos:
- *default constructor*
 - *copy constructor*
 - *copy assignment*
 - métodos *void AddAluno(alunoIST* const)* e *alunoIST* FindAluno(int numero)*
 - método *int GetNalunos()*
- b) Implemente o código da classe e em particular o método *alunoIST* FindAluno(int)* deveria ser implementado da forma mais eficaz usando a procura dicotómica.
- c) Construa um programa *main()* onde possa testar a classe definindo a turma de MEFT T21.

Exercício 29 : O movimento de um corpo a uma dimensão pode ser descrito pela classe **Motion1D**, que se apresenta de seguida. Nesta classe registam-se as N posições do corpo e os tempos.

Motion1D.h

```

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**.

Uniform1D.h

```
class Uniform1D : public Motion1D {
public:
    Uniform1D(int fN=0, float ti=0., float xi=0., float dt=0., float vel=0.);
    ~Uniform1D();

    void Print();
private:
    float ti; // initial time
    float dt; // time duration
    float xi; // initial position
    float vel; // velocity (m/s)
};
```

Uniform1D.C

```
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 **Runiform1D.C** onde realize as seguintes acções:

- a) Instancie um objecto **Uniform1D** na memória *heap* com 100 pontos discretos, durante 1000 segundos de tempo 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 1 m/s.

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

- e) Remova os objectos criados da memória.

Programação com objectos ROOT

Exercício 30 : Lance o root em sessão interactiva e utilize o interpretador de ROOT para correr código C++ que realize as seguintes tarefas:

- crie um *array* de 2 histogramas *TH1F* utilizando o default constructor.
- crie um *array* de 2 histogramas *TH1F* com as seguintes características numa só linha de comandos: 10 canais e limites inferior e superior respectivamente, 0.5 e 10.5
- crie um *array* de 2 histogramas *TH1F* com 5 canais de largura variável dada por: **0.5, 1.5, 4.5, 2.0, 1.0**
- crie agora o *array* de 2 histogramas *TH1F* utilizando o default constructor e inicializando-os de seguida com as características da alínea b)
- crie agora um *array* de 2 ponteiros que aponte para os histogramas com características da alínea b)
- construa uma macro **mHisto.C** onde reuna o conjunto de operações da alínea d) e execute-a.

Exercício 31 : Lance o root em sessão interactiva e utilize o interpretador de ROOT para correr código C++ que realize as seguintes tarefas:

- faça um array de dois inteiros sem inicializar os valores e verifique os valores existentes em cada posição do array.
- liste os objectos existente em memória do ROOT.
Nota: utilize a classe TROOT, consultando a sua documentação em root.cern.ch, e em particular o ponteiro já existente para um objecto TROOT instanciado
- construa um *array* de três objectos histograma que armazene floats (TH1F) entre os valores -10. e 10, com canais de largura 0.2

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

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

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

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

- e) Liste agora os objectos existentes em memória de ROOT e verifique que os três histogramas que construiu se encontram lá.
- f) Defina agora um *array* de duas funções uni-dimensionais

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

onde implemente as seguintes funções:

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

- g) Liste de novo os objectos existentes em memória de ROOT e verifique que os três histogramas que construiu e as duas funções se encontram lá.
- h) Procure o ponteiro para o segundo histograma usando a classe *TROOT* (ou melhor, o ponteiro disponível para o objecto *TROOT* instanciado).
- i) Tendo o ponteiro para o objecto histograma, desenhe-o no ecrã, usando o método da class *TH1*, *Draw()*.
- j) Obtenha agora o número de canais (bins) do histograma, usando o método da class *TH1*, *GetNbinsX()*. Teve sucesso com esta operação?
- k) Desenhemos agora cada um dos outros histogramas e cada uma das funções.
- l) Antes de abandonar a sessão de ROOT armazene os objectos construídos num ficheiro ROOT.

Exercício 32 : Reúna agora todos os comandos C++ que introduziu linha a linha no exercício anterior, numa macro de nome **mRoot1.C**. Corra a macro de forma interpretada, usando quer os métodos da classe *TROOT*:

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

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

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

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

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

c) quer os comandos

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

Exercício 33 : No exercício anterior o código C++ existente na macro **mRoot1.C** foi interpretado. Pretende-se agora compilar este mesmo código usando o compilador ACLIC do ROOT. Para tal execute na linha de comandos ROOT,

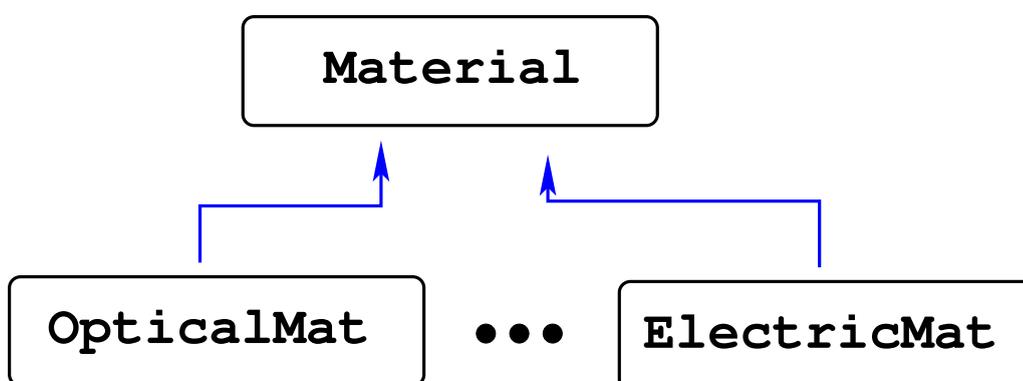
```
root> .L mRoot1.C+ //compile and load macro (but not execute it)
```

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

Exercício 34 : O índice de refração 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.



```

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 vamos colocar as características ópticas do material como sejam o índice de refração. A classe *OpticalMat* deve possuir métodos que permitam:

- definir o índice de refração
- ajustar por uma lei o índice de refração em função do comprimento de onda
- desenhar o índice de refração (usando a classe *cFCgraphics* que será disponibilizada)

```

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 characteristic 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 refração (n) com o comprimento de onda (λ) é conhecida como lei de dispersão do material e pode ser ajustada com a fórmula de Sellmeir.

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

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

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

com λ em μm

Exercício 35 : 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 píxeis distribuídos aleatoriamente numa região de $2 \times 2 \text{ cm}^2$. 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 píxeis 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 píxeis 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 píxeis.

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

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

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

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

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

Representação dos números em computador e arredondamentos

Exercício 36 : Considere o número real de precisão simples e 32 bits,

senal	expoente	mantissa					
0	0000 1110	1010	0000	0000	0000	0000	000

- Determine o valor do expoente verdadeiro.
- Mostre que a mantissa vale **1.625**
- Determine o valor do número real.

Exercício 37 :

- Escreva uma função em C++ que determine os limites *underflow* e *overflow* do seu computador e linguagem de programação, dentro de um factor **2**.
- Obtenha os valores limite de *underflow* e *overflow* para números reais de precisão simples.
- Obtenha os valores limite de *underflow* e *overflow* para números reais de precisão dupla.

Exercício 38 : 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**.

- para números reais de precisão simples.
- para números reais de precisão dupla.

Exercício 39 : Habitualmente considera-se que os erros de arredondamento são de natureza aleatória. Para verificarmos essa hipótese podemos desenvolver um código em C++ que calcule os erros de arredondamento associados a uma dada operação de cálculo em precisão *float* e usando como referência a representação *double* do resultado. Defina uma classe em C++ de nome **FCtools** onde implemente os seguintes métodos estáticos:

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

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

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

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

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

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

Exercício 40 : Resolva a equação quadrática $x^2 - 2bx + c = 0$, $b^2 > c$ pode ser feito com recurso à formula resolvente dando lugar à seguinte solução:

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

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

$$x_{1,2} = b \pm \sqrt{b^2 - c}$$
$$x_1 \times x_2 = c$$

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

(a) $x_1 = b + \sqrt{b^2 - c}$
 $x_2 = b - \sqrt{b^2 - c}$

(b) *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.

Exercício 41 : 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:

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

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:

$$\cos(\alpha) - \cos(\beta) = -2 \sin\left(\frac{\alpha + \beta}{2}\right) \sin\left(\frac{\alpha - \beta}{2}\right)$$

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

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

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

Parte II

Resolução numérica de problemas em C++

Sistemas Lineares

Exercício 42 : 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 **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-se então à implementação dos métodos da classe num ficheiro *Vec.C* e às respectivas declarações 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 nos 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(...);
    }
}
```

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

- 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
```

Construa uma classe auxiliar *FCtools* que possua os seguintes métodos para ler o ficheiro:

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

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 bidimensional e mostrá-lo (*Draw()*) no ecrã, usando para a tal a classe *cFCgraphics*.
 Nota: ara 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));
    }
  }
  // graphics class
  cFCgraphics G;
  (...)
}

```

- 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 ($a[i] = b[i]*\lambda$)
- 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.

Exercício 43 : 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 troca duas linhas da matriz. Construa com o auxílio desta função uma matriz **E** onde a 4ª linha seja trocada com a 5ª linha da matriz **C**.

Exercício 44 : Considere a seguinte matriz $M_{3 \times 3}$ preenchida com os seguintes números:

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

- a) Defina a classe em C++ *FCmatrixTeste* que manipule esta matriz e que armazene o seu conteúdo nas diferentes formas expressas na seguinte definição da classe:

```
classe FCmatrixTeste {
public:
    FCmatrixTeste();
    FCmatrixTeste(double** fM, int fm, int fn); //matrix fm x fn
    FCmatrixTeste(double* fM, int fm, int fn);
    FCmatrixTeste(vector<Vec>);
private:
    double** M1;
    double* M2;
    vector<Vec> M3;
    int m; //nb rows
    int n; //nb cols
    int flag; // integer with a definition of which constructor was used
};
```

- 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.

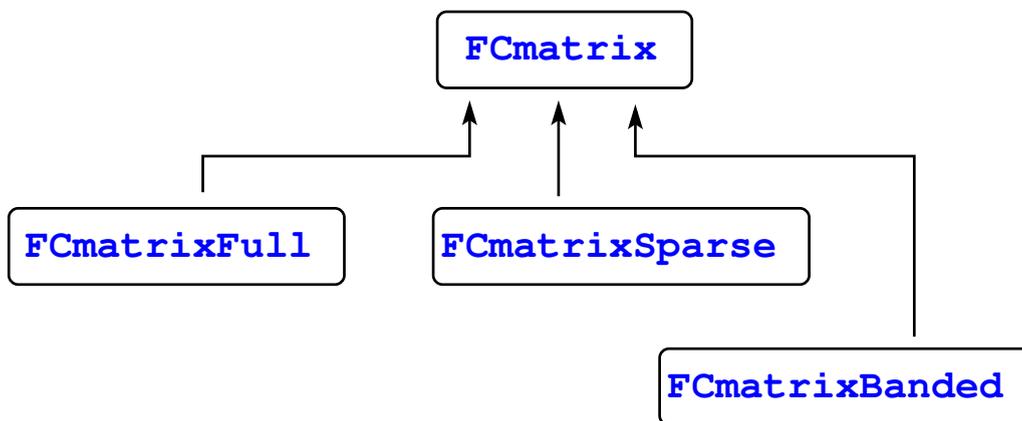
```

int main() {
    ...
    FCmatrixTeste A ...;
    cout << A[1][4] << endl;
    A[1][4] = 3.;
    cout << A[1][4] << endl;
}

```

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

Exercício 45 : 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 (*FCmatrixFull*) de todos os elementos, outras matrizes que possuem muitos zeros entre os seus elementos (*FCmatrixSparse*) e ainda matrizes que possuam 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:

- os construtores que permitam armazenar os elementos necessários e suficientes para a reconstrução da matriz, na classe

```

classe FCmatrix {
public:
    FCmatrix();
    FCmatrix(double** fM, int fm, int fn); //matrix fm x fn
    FCmatrix(double* fM, int fm, int fn);
    FCmatrix(vector<Vec>);
    (...)
protected:
    vector<Vec> M;
    string classname;
};

```

- os métodos puramente virtuais *GetRow*, *GetCol*, *Determinant* que deverão ser implementados nas classes derivadas

```

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;
};

```

- 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)

```

classe FCmatrix {
public:
  (...)
  virtual void Print();
  (...)
protected:
  vector<Vec> M;
  string classname;
};

```

- Os métodos *GetRowMax* e *GetColMax*:
 O método **GetRowMax**, retorna o índice da coluna (0,1,...) que possui na linha i, o elemento máximo absoluto (módulo)
 O método **GetColMax**, retorna o índice da linha (0,1,...), a partir da linha j, que possui o elemento máximo (módulo) relativo à escala s (linha a linha)

```

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 of any kind by scalar
    Vec operator*(const Vec&); // multiply matrix by Vec
    // virtual inherited
    Vec GetRow(int i); // retrieve row i
    Vec GetCol(int i); // retrieve column i
    double Determinant();
    void Print();
    void swapRows(int,int);
    ...
private:
    int rowindices[fm]; // row indices (0,1,2,...)
};

```

- c) Teste as classes desenvolvidas realizando um programa *main* onde manipule as seguintes matrizes:

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

```

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();
}

```

Exercício 46 : 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&)
    //eliminação de Gauss:
    //resolução do sistema pelo método de eliminação de Gauss
    Vec GaussEliminationSolver();
    Vec LUdecompositionSolver();
private:
    //decomposição 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)

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

2)

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

Raízes de funções

Exercício 47 : 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:

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

onde ω é a velocidade 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 .

Exercício 48 : 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,

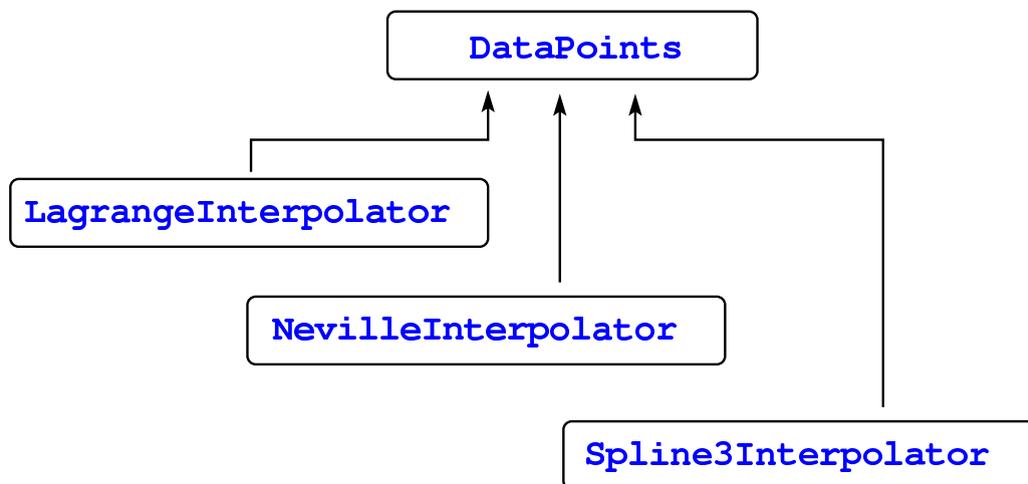
- a) Escreva as equações do movimento da bala.
- b) Mostre que a distância horizontal percorrida pela bala obedece à seguinte equação:

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

- 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.

Interpolação

Exercício 49 : Para a realização de interpolações, pode-se definir uma classe **DataPoints**, que conterá os dados respeitantes aos pontos e classes que herdem desta onde se implementarão os diferentes métodos de interpolação. Teremos assim as classes de interpolação **NevilleInterpolator**, **NewtonInterpolator** e **Spline3Interpolator**.



A classe **DataPoints** deve ser usada para o armazenamento dos pontos e possui ainda o objecto **cFCgraphics**, para a representação gráfica.

```
class DataPoints {
public:
    DataPoints();
    DataPoints(int, double*, double*);
    virtual ~DataPoints();

    virtual double Interpolate(double x) {return 0.;}
    virtual void Draw();
    virtual void Print(string FILE="");
protected:
    int N; // number of data points
    double *x, *y; // arrays
};
```

```
cFCgraphics G;
};
```

Aqui mostra-se um exemplo da classe **NevilleInterpolator** onde se implementa o método de interpolação de Neville sobre o conjunto de pontos transmitidos no construtor da classe.

```
class NevilleInterpolator : public DataPoints {
public:
    NevilleInterpolator(int N=0, double *x=NULL, double *y=NULL, TF1* fF0=NULL);
    ~NevilleInterpolator() {};

    double Interpolate(double x);
    void Draw(); //draw everything (points and interpolation function)
    TF1* GetInterpolationFunction();

    void Print(string FILE=""); // print results

private:
    double fInterpolator(double *fx, double *par) {
        return Interpolate(fx[0]);
    }
    TF1* FInterpolator; //interpolation function
    TF1* F0; //eventual underlying function
};
```

Proceda à implementação das classes.

Exercício 50 : Usando as classes construídas anteriormente e dados os seguintes pontos,

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:

- o método de Neville
- o método de Lagrange
- o método de Newton
- o método do spline cúbico

Exercício 51 : Usando as classes construídas anteriormente e dados os seguintes pontos,

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

realize um programa *main* que utilize a classe *Spline3Interpolator* e que determine e desenhe a função interpoladora dos pontos.

Derivação e Integração de funções

Exercício 52 : Para a integração de funções vamos definir a classe **Integrator** onde se definirão os métodos de integração trapezoidal e de Simpson. Implemente os algoritmos e estruture a classe:

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

private:
    TF1 *func;
};
```

Exercício 53 : Determine o integral $\int_0^{\frac{\pi}{2}} \cos(x) dx$.

Exercício 54 : 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:

$$E = \frac{1}{2m} \left(\frac{dx}{dt} \right)^2 + V(x)$$

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:

$$T = \sqrt{8m} \int_0^a \frac{dx}{V(a) - V(x)}$$

b) 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**.

Exercício 55 : Neste problema pretende-se calcular a eficiência de uma lâmpada de tungsténio para uma radiação na região do visível ($\lambda_1 - \lambda_2$ nm). A potência emitida pela lâmpada por unidade de comprimento de onda λ obedece à lei de radiação de Planck,

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

onde T é a temperatura em Kelvin, h é a constante de Planck e k_B a constante de Boltzman.

$$k_B = 8.617 \times 10^{-5} \text{ eV/K}$$

$$h = 4.136 \times 10^{-15} \text{ eV s}$$

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,

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

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.

Métodos Monte-Carlo

Exercício 56 : Os geradores de números aleatórios usam relações do tipo:

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

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

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

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

- b) distribuições de cada um dos números aleatórios para 1000000 de amostragens. Determine o valor médio e o desvio padrão da amostra e compare com os valores esperados.
- c) Divida a distribuição em 10 intervalos e coloque num gráfico os valores médios de cada intervalo bem como o erro do valor médio.

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

$$C_k = \frac{\langle x_{i+k} x_i \rangle - \langle x_i \rangle^2}{\langle x_i^2 \rangle - \langle x_i \rangle^2}$$

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

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

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

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

Exercício 57 : Determinemos a superfície de um de círculo de raio 1, isto é, o valor de π . Consideremos um grande número N de pares de números aleatórios (r_1, r_2) , tirados a partir de uma distribuição aleatória entre 0 e 1. Construa um algoritmo que determine o valor de π e obtenha o valor calculado bem como o seu erro, em função do número de amostragens N (10000, 100000, 1000000).

Exercício 58 : A classe `Integrator` pode ser estendida de forma a incluir os métodos de integração de Monte-Carlo, simples, *importance sampling* e de *aceitação-rejeição*. No método de *importance sampling* é usada 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 $(y(x) = \int_a^x p(x) dx)$ e 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.

```
class IntegratorMC : public Integrator {
public:
    IntegratorMC(TF1*, int N=1); //integrand function (1-dim by default)
    IntegratorMC(TFormula); //integrand function
    ...
    // the following methods need the number of samplings (N) and shall return
    // the integral value and error
    void UniformRandom(...);
    void ImportanceSampling(...); // auxiliar function and inverse
                                   // accumulated
    void AcceptanceRejection(...);

private:
    ...
};
```

Exercício 59 : Determine os seguintes integrais,

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

usando:

- o método trapezoidal utilizando um passo $h = 0.2$
- o método de Simpson usando o mesmo passo
- o método de monte-carlo com variável aleatória uniforme, usando 100, 1000 e 10000 amostragens. Determine o erro associado ao cálculo do integral em cada um dos casos.

Exercício 60 : O problema da agulha de Buffon, colocado pelo naturalista francês Buffon em 1733, pretende 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 $\ell = 2\text{cm}$ e riscas de espessura desprezável espaçadas de $d = 10\text{cm}$.

- 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 $100 \times 100\text{cm}$ 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 .

- Construa as classes **Point2D** e **line2D** que permitam construir os elementos riscas e agulha.

```
class Point2D {
public:
Point2D(double fx=0., double fy=0.);
(..)
private:
double x; // x coo
double y; // y coo
};
```

```
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;
};

```

4. Construa um programa **main** onde comece por definir, com o auxílio das classes acima definidas, o quadrado (frame) de $100 \times 100\text{cm}$ 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;  
}
```

5. Represente graficamente o valor da probabilidade em função do número de lançamentos aleatórios da agulha.
6. 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.

Resolução de equações diferenciais

Exercício 61 : 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 estrutura **ODEpoint** (*struct* ou *class*) que armazene em cada iteração os valores das variáveis.

a) Implemente a estrutura **ODEpoint**.

```
...
double t; // time
vector<double> var; //dependent variables
...
```

b) Implemente a classe **ODEsolver** que vise a solução de um sistema de equações diferenciais de primeira-ordem, $\frac{dy_i}{dt} = f_i(t, y_i, y'_i)$
Implemente pelo menos, os construtores e métodos que estão indicados abaixo. Nesta implementação o construtor recebe as funções através de objectos de ROOT *TFormula*, que permitem definir funções com tantas variáveis quanto se queira.

```
class ODEsolver {
public:
    ODEsolver(vector<TFormula>);
    SetODEfunc(vector<TFormula>);
    vector<ODEpoint> Euler(ODEpoint i, double step, double T);
    vector<ODEpoint> PredictorCorrector(ODEpoint i, double step, double T);
    vector<ODEpoint> RK2(ODEpoint i, double step, double T);
    vector<ODEpoint> RK4(ODEpoint i, double step, double T);
    ...
private:
    vector<TFormula> F;
```

};

Exercício 62 : Considere a equação diferencial $\frac{dy}{dx} = 3x - y + 8$ cujo valor inicial é $y(0) = 3$. Determine a solução da equação $y(x)$ no intervalo $x \in [0, 2.0]$ e usando um passo de 0.1 , com os diferentes métodos numéricos.

Exercício 63 : Considere o sistema de duas equações diferenciais de primeira-ordem,

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

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

Exercício 64 : 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:
 $z(0) = 2 \text{ Km}$
 $\dot{z}(0) = 0 \text{ m/s}$
 $m = 80 \text{ Kg}$
 $k = 0.3 \text{ Kg/s}$
 $g = 9.81 \text{ m/s}^2$

- 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?

Exercício 65 : 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:

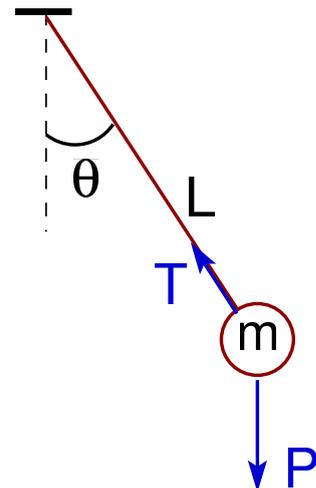
$$\ddot{x} + \omega^2 x = 0$$

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

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

Exercício 66 : Considere uma massa $m = 1 \text{ Kg}$ suspensa por um fio de comprimento $\ell = 1 \text{ m}$ e sujeita a uma força de atrito $\vec{F}_a = -k\vec{v}$, com $k = 0.3 \text{ kg/s}$.

- escreva a equação do movimento
- escreva o sistema de equações diferenciais de primeira ordem e identifique as variáveis dependentes
- 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. Qual a amplitude de oscilação ao fim de 10 minutos? Faça o plot da amplitude de oscilação ao longo do tempo.



Exercício 67 : Uma barra cilíndrica de diâmetro $D = 101 \text{ cm}$ e comprimento de $L = 100 \text{ cm}$ está em contacto com uma fonte de calor à temperatura de $T_s = 40$ graus Celsius.

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

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

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 $s = 10 \text{ cm}$ e $s = 2 \text{ cm}$. Produza um plot.

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

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

onde $T_a = 23$ graus Celsius é a temperatura do ar e os coeficientes $h = 17 \text{ W.m}^{-2}.\text{K}^{-1}$ e $k = 206 \text{ W.m}^{-1}.\text{K}^{-1}$ 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.

Parte III

Problemas de síntese e revisão

Problemas com objectos C++

Problema de Exame (2015-16)

Neste problema manipulam-se objectos de duas classes distintas: a classe **Vector** e a classe **Point**. A classe **Point** representa pontos bi-dimensionais e a classe **Vector** representa vectores no mesmo espaço.

Implementa as classes **Point** e **Vector** de forma que os dois programas principais que se seguem possam ser executados sem quaisquer modificações.

Programa: **rPoint.C**

```
1 int main() {
2
3     Point *P12;
4     {
5         Point P1(1.5, 3.);
6         Point P2;
7         P2.SetX(0.2);
8         P2.SetY(-0.4);
9
10        P1.Print();
11        P2.Print();
12
13        P12 = new Point(P1);
14    }
15
16    Point P3 = *P12;
17    Point P4;
18    P3.Print();
19 }
```

Programa: **rVector.C**

```
1 int main() {
2     Point P1(1, 3.);
3     Point P2(5, 2.);
4     P1.Print();
5     P2.Print();
6     Vector V(P1,P2); //V=vector de P1 para P2
7
8     V = P1;
9     V.Print();
10
11    Point* A = new Vector(P1);
```

```
12 delete A;
13 }
```

Realize um Makefile que permita obter os executáveis dos programas, respectivamente, **rPoint.exe** e **rVector.exe**, através dos seguintes comandos:

- make Point
- make Vector

Problema de Exame (2014-15)

```
1 #include <iostream>
2
3 class vector {
4     public:
5     vector() {
6         size = 5;
7         vec = new double[size];
8     }
9     ~vector() {}
10    double* GetVector() {
11        double tmp[size];
12        for (int i = 0; i < size; i++) tmp[i] = vec[i];
13        double *tmp2 = tmp;
14        return tmp2;
15    }
16    int GetSize() {return size;}
17    private:
18    double *vec;
19    int size;
20 };
21
22 int main() {
23     vector V;
24     //print vector
25     for (int i=0; i<V.GetSize(); i++) {
26         cout << V[i] << endl;
27     }
28     //get array defined inside vector and delete it
29     double *f = V.GetVector();
30     delete *f;
31 }
```

No código em cima encontra-se implementada uma classe *vector* e um pequeno programa *main()* onde se realizam três operações:

- impressão dos elementos do vector
- acesso ao ponteiro do *array* existente na classe *vector*
- *delete* do *array* existente na classe *vector*

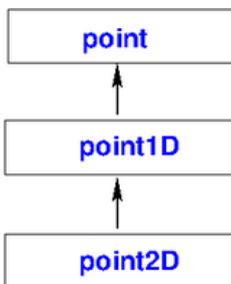
Corrija o código da classe e do programa, mantendo as ações do programa, de forma a que este compile e corra sem erros.

Problema de Exame (2014-15)

Consideremos as classes *point*, *point1D* e *point2D* em que *point1D* é usada para representar pontos uni-dimensionais e *point2D* é usada para representação de pontos bi-dimensionais.

Parte 1

As classes possuem o esquema de herança definido pela figura que se segue.



```
1  int main() {
2      point1D P(4.,4.);
3      P.Print();
4      double a = P->Norma();
5  }
```

Com este programa pretende-se realizar os seguintes passos:

- instanciar um objecto *point1D*
- imprimir os detalhes do ponto
- calcular a norma do ponto (módulo)

Complete o programa e corrija se necessário as três linhas de código existentes, sem adicionar mais nenhuma ao programa, de forma a que este possa compilar e correr e ainda produza o seguinte output:

```
1  [point1D :: Print()]
2  [point1D :: Norma()]
```

Para resolver o problema deve partir das implementações das classes que se apresentam a seguir sem modificar os **data members**. Se achar necessário corrigir ou completar o resto da classe, pode fazê-lo.

```

1  class point {
2
3      public:
4          point() {};
5          virtual double Norma()=0;
6          virtual void Print()=0;
7          virtual void SetPoint()=0;
8          void Setname(string);
9
10         protected:
11             string name;
12     };

```

```

1  class point1D : public point {
2      public:
3          point1D();
4          float Norma();
5          void Print();
6      protected:
7          double x;
8  }

```

Parte 2

No programa que se segue pretende-se construir um array de pontos uni e bi-dimensionais, usando o elemento *vector* da biblioteca STL. O programa que deve completar e corrigir de forma a que corra, implementa as seguintes etapas:

- preenchimento do array com os pontos (não modifique a ordem de preenchimento que se encontra no programa)
- imprimir, usando o método `Print()`, os pontos do array
- listar/imprimir os pontos do array no sentido ascendente do seu módulo

```

1  int main() {
2
3      v.push_back(new point1D(10.));
4      v.push_back(new point2D(4.,4.));
5      v.push_back(new point1D(1.));
6      v.push_back(new point2D(3.,3.));
7      v.push_back(new point2D(2.,3.));
8      v.push_back(new point2D(5.,3.));
9      v.push_back(new point2D(1.,3.));
10     v.push_back(new point1D(5.));
11     v.push_back(new point1D(4.));
12     v.push_back(new point2D(10.,5));
13
14     //imprimir a lista de pontos tal qual foi introduzida
15     for (int i=0; i<v.size(); i++) {
16         v[i]->Print();
17     }
18
19     //calcular a norma de cada ponto, seriar e imprimir os pontos no sentido
20     //crescente da norma
21     ...
22     double a = v[i]->Norma();
23     ...
24 }

```

Note que terá que completar a classe `point2D` para completar o exercício. Em baixo poderá ver um começo da implementação da mesma.

```
1  class point2D : ... {  
2      public :  
3          ...  
4      protected :  
5          double y;  
6  };
```


Problemas numéricos

Problema de Exame (2015-16)

Pretende-se integrar a seguinte função:

$$I = \int_0^{\pi} f(x) dx = \int_0^{\pi} \frac{1}{x^2 + \cos^2(x)} dx$$

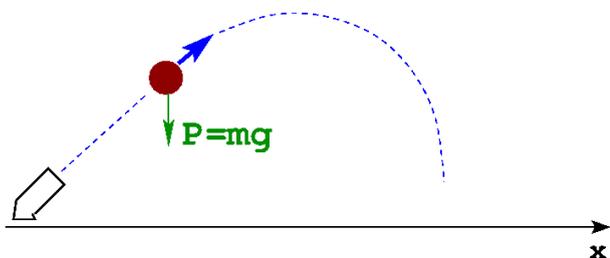
Realize um programa main `rIntegralMC.C` que pode recorrer a código já desenvolvido na disciplina, onde seja determinado o valor do integral utilizando o método de Monte-Carlo e 1000 pontos aleatórios:

- determine o integral pelo método MC simples e o seu erro
- determine o integral pelo método de **importance sampling** utilizando a função auxiliar $p(x) = Ce^{-ax}$
 - o valor do integral e o seu erro para $a = 0$
 - o valor de a que minimiza o erro do integral

Complete o Makefile de forma a que o comando `make IntegralMC` produza o executável `rIntegralMC.exe`

Problema de Exame (2015-16)

Um corpo de massa $m = 100$ Kg é disparado por um canhão que faz um ângulo α com a horizontal. A velocidade inicial é de $v = 200$ Km/h.



Determine:

- as equações do movimento
- o sistema de equações necessárias à resolução do problema pelo método RK4
- Utilizando o método RK4 determine:
 - A que distância x o corpo aterra, se fôr disparado com um ângulo de 30 graus?
 - Qual o ângulo do disparo, se o alvo se encontrar em $x = 200$ metros?

Realize um programa principal **rCanhao.C** onde procede à resolução deste problema (podendo obviamente recorrer a código já desenvolvido).

Complete o Makefile de forma a que o comando `make Canhao` produza o executável `rCanhao.exe`

Problema de Exame (2014-15)

Calcule utilizando o método de Monte-carlo e somente 100 números aleatórios gerados com o **TRandom3** de ROOT, o seguinte integral e o erro associado,

$$\int_0^1 e^{-x^2} dx$$

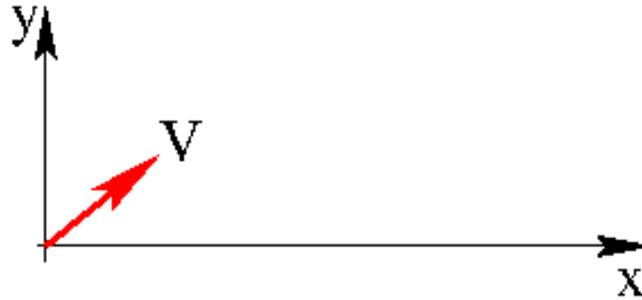
Utilize como base de partida o programa

```
1 int main() {
2   // loop para calculo do integral
3   for (int i=0; i<100; i++) {
4   }
5   // imprimir resultados
6 }
```

Problema de Exame (2014-15)

Considere um corpo de massa m no campo gravítico terrestre e próximo da superfície da Terra de forma a que possa considerar a aceleração gravítica como constante. Pretende-se estudar a trajectória do corpo quando lançado com uma velocidade inicial $\mathbf{v} = v_x \mathbf{e}_x + v_y \mathbf{e}_y$, usando o método Runge-Kutta de 4ª ordem. Considere que o corpo sente uma força de atrito proporcional à velocidade e dada por: $\mathbf{F} = -mk\mathbf{v}$.

- Determine as equações diferenciais de 1ª ordem
- Desenvolva um programa **Rprojectile.C** onde se obtenha:
 - uma figura (**Prob4_trajectorias.pdf**) com as trajectórias y .vs. x sobrepostas, para os seguintes coeficientes:



- * $k=0$ (linha preta)
 - * $k=0.2$ (vermelho)
 - * $k=1$ (verde)
 - * $k=5$ (azul)
 - * $k=10$ (azul, traço interrompido)
- o tempo gasto e o espaço percorrido, que o corpo demora a atingir o solo nas diferentes situações de atrito
- uma figura (**Prob4_energia.pdf**) com a energia mecânica em função do tempo, para as diferentes situações de atrito (conserva o mesmo código de cores)

Dados:

- condições iniciais: $\mathbf{r}(t = 0) = \mathbf{0}$
- $m = 100\text{Kg}$
- $g = 10\text{m/s}^2$