Exercícios de

Física Computacional

(Parte 5)

Mestrado em Engenharia Física-Tecnológica (MEFT)

Rui Coelho

Departamento de Física do Instituto Superior Técnico

Ano Lectivo: 2019-20

rui.alves.coelho@tecnico.ulisboa.pt

versão: 1 de Outubro de 2019

5. Sistemas de Equações Lineares

Exercício 41. (*adaptado de Barão 2016*) 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*** (semelhante ao *vector* container) 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:
  ...//many methods to come...
  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.cpp* e às respectivas declarações num ficheiro *Vec.h*, de forma a que a classe possa realizar as operações que se enunciam de seguida:

i. Possuir ***construtores*** que nos permitam a construção dos vectores usando as seguintes formas (a utilizar no programa ***main.cpp***):

```
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 v3(5,a); //array with 5 values given by "a" pointer

Vec v4(v3); //define a vector by using another one
```

ii. Existir um método *SetEntries* de forma a permitir (re)definir um objecto *Vec* de **n**-elementos, com o conteúdo de um array

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

Para testar este método *SetEntries*, inclua no programa ***main.cpp***, a seguir ao teste dos construtores, código onde copie o conteúdo de uma matriz ***CM*** para um array ***cv*** de objectos *Vec*

```
// 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(...);
 }
```

Utilize como teste a seguinte matriz ***CM***

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

ii.1 Desenvolva também uma classe auxiliar *FCTools* com métodos dedicados que permitam ler matrizes a partir de ficheiro ASCII:

```
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); //file name, returns pointer to array of Vec's
};
```

Para facilitar a implementação dos métodos de leitura, pode chamar um dos métodos na implementação dos restantes dois. Recorde os conhecimentos adquiridos na leitura de ficheiros e.g. getline, stringstream, ...
Teste estes métodos no ***main.cpp*** a ler a mesma matriz ***CM*** a partir do ficheiro *matrix.txt*.

ii.2 Usando um dos métodos anteriores e a matriz ***CM*** lida de *matrix.txt*, adicione agora ao código ***main.cpp*** uma secção onde faça o histograma 2D preenchido com as entradas da matriz.

```
// 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));
    }
}
```

iii. Existir overloading de métodos (*com testes dedicados no* ***main.cpp***) para:

- 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]*c)

iv. Existir métodos (*com testes dedicados no* ***main.cpp***):

- *size*: obter a dimensão do vector
- *dot*: fazer o produto interno com outro vector
- *Print()*:imprimir o conteúdo de um vector
- void *swap*(int,int): trocar a ordem de dois elementos do vector

Exercício 42 : (adaptado de *Barão 2016*) Neste problema iremos utilizar a classe Vec para manipular a matriz ***CM*** dada no problema anterior. Escreva um programa ***main.cpp*** onde realize as seguintes acções:

a) Leia a matrix ***CM*** a partir do ficheiro *matrix.txt*.

b) Obtenha um objecto *Vec* que resulte da multiplicação da constante 5 pela 2ª linha de ***CM*** i.e.

```
Vec line=5*cv[1];
```

c) Obtenha uma 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 ***CM***

$$L_2 \leftarrow L_2 - \frac{CM_{21}}{CM_{11}} L_1$$

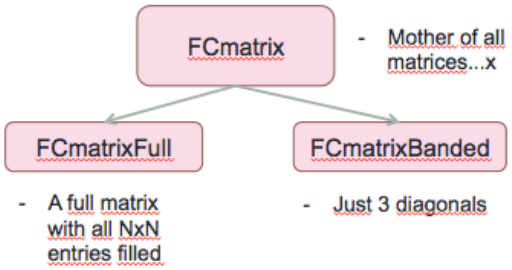
d) Multiplique as duas primeiras linhas da matriz ***CM*** e obtenha um novo objecto *Vec* com o resultado.

e) Após implementar a função

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

que troca o conteúdo de dois vectores *Vec*, utilize esta função para trocar linhas da matriz ***CM***. Por exemplo, troque a 4ª linha com a 5ª linha da matriz.

Exercício 43 : (adaptado de *Barão 2016*) 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 matrizes que necessitam de um armazenamento integral (***FCmatrixFull***) de todos os elementos e outras matrizes que possuam estruturas em banda como por exemplo as matrizes tridiagonais (***FCmatrixBanded***).
[5]



a) Requisitos/descrição da classe base ***FCmatrix***:

```
class FCmatrix {

public:

//constructors
FCmatrix();
FCmatrix(double** fM, int fm, int fn); //matrix fm x fn given from pointer of pointers
FCmatrix(double* fM, int fm, int fn); //matrix fm x fn given as single pointer (what length ?!)
FCmatrix(vector<Vec>); //matrix fm x fn given as vector of Vec

// operators
virtual Vec& operator[] (int) = 0; //get a row by giving índex inside []

// methods
virtual int Get_nRows() const = 0; //number of rows of M
virtual int Get_nCols() const = 0; //number of columns of M
virtual Vec GetRow(int i) const = 0; // retrieve row i
virtual Vec GetCol(int i) const = 0; // retrieve column i
```

```
virtual double Determinant() const = 0;

// in row-i, return column-index of max element (in absolute value)
virtual int GetRowMax(int i=0) const = 0;
// in column-j, return row-index (>=j) for which relative amplitude of M_ij on the row is highest.
virtual int GetColMax(int j=0) const = 0; //

virtual void Print() const; //print e.g. row by row (GetRow to the rescue...)

protected:

vector<Vec> M; //the matrix is a vector of Vec objects...
string classname; //give the class a name...
};
```

b) Requisitos/descrição da classe derivada ***FCmatrixFull*** (incompletos):

```
class FCmatrixFull : public FCmatrix {

public:

// constructors
FCmatrixFull();
FCmatrixFull(double** fM, int fm, int fn);
FCmatrixFull(double* fM, int fm, int fn);
FCmatrixFull(vector<Vec>);

// copy constructor
FCmatrixFull(const FCmatrixFull&);

// operators
FCmatrixFull operator=(const FCmatrix &); // equal 2 matrices of any kind
FCmatrixFull operator+(const FCmatrix &) const; // add 2 matrices of any kind
FCmatrixFull operator-(const FCmatrix &) const; // sub 2 matrices of any kind
FCmatrixFull operator*(const FCmatrix &) const; // mul 2 matrices of any kind
FCmatrixFull operator*(double lambda) const; // mul matrix of any kind by scalar
Vec operator*(const Vec &) const; // mul matrix by Vec
...
// virtual inherited
...
Vec GetRow(int i) const; // retrieve row i
Vec GetCol(int i) const; // retrieve column i
...
// swap 2 given rows in the matrix...
void swapRows(int,int);


};
```

c) Teste as classes desenvolvidas realizando um programa ***main.cpp*** que manipule as seguintes matrizes:

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

```
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* (constant x Matrix)
```

```
    FCmatrixFull MD(MA*MB);
    // print
    MC.Print();
    MD.Print();
    // other methods
    MD.Determinant();
    MC.swapRows(1,2);
    MC.Print();
}
```

Exercício 44 : (*adaptado de Barão 2016*) 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.

```
class EqSolver {
public:
EqSolver();
EqSolver(const FCmatrixFull&, const Vec&); // matriz M e vector de constantes
…
// set
void SetConstants(const Vec&);
void SetMatrix(const FCmatrixFull&);
…
//eliminação de Gauss:
//resolução do sistema pelo método de eliminação de Gauss
Vec GaussEliminationSolver();
//Resolucao pelo metodo Doolittle LU (A=LU)
Vec LUdecompositionSolver_Doolittle();
//Tridiagonal system by Thomas algorithm
Vec TridiagonalSolver();

private:
/* return triangular matrix and changed vector of constants */
void GaussElimination(FCmatrixFull&, Vec&);
//decomposição Doolittle LU com |L|=1
FCmatrixFull * LUdecomposition_Doolittle(FCmatrixFull&);
Vec TridiagonalThomas(FCmatrixBanded &, Vec &);
FCmatrixFull M; //matriz de coeffs
…
Vec b; //vector de constantes
… //some bookkeeping members maybe ?!
};
```

b) Resolva os seguintes sistemas de equações lineares pelos vários métodos:

i. $\begin{bmatrix} 4 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 11 \\ -16 \\ 17 \end{bmatrix}$

iii. $\left[\begin{array}{ccccc|c} -2 & 1 & & & & 1 \\ 1 & -2 & 1 & & & 2 \\ & 1 & -2 & 1 & & 3 \\ & & 1 & -2 & 1 & 4 \\ & & & 1 & -2 & 5 \end{array}\right]$

ii. $\left[\begin{array}{ccc|c} 2 & -2 & 6 & 16 \\ -2 & 4 & 3 & 0 \\ -1 & 8 & 4 & -1 \end{array}\right]$