Colectânea de Notas/Problemas de Física Computacional

U. Lisboa -IST / MEFT

1º semestre 2020-21

Fernando Barão (fernando.barao@tecnico.ulisboa.pt)

Tópicos adicionais

C++11 revision

A linguagem C++ possui revisões (actualizações) regulares sendo as mais relevantes: C++98 (1998), C++11 (2011) e C++17 (2017).
No curso de Física Computacional usaremos a revisão relevante de 2011 (c++11).
Ver os links que se seguem para mais detalhes:
- cppreference link[1]

ROOT quick documentation

Consultem aqui:

- elementos básicos de ROOT: https://root.cern/manual/basics/
- lista de todas as funções existentes em `TMath`: https://root.cern/doc/master/namespaceTMath.html

C++ compiler options

Lista de opções com compilador de C++ (não exaustiva):

> `-g`: turn on debugging (so GDB gives more friendly output)
> `-Wall`: turns on most warnings
> `-O` or `-O2`: turn on optimizations
> `-o <name>`: name of the output file
> `-c`: output an object file (.o)
> `-I<include path>`: specify an include directory
> `-L<library path>`: specify a library directory
> `-l<library>`: link with library lib.a
> `-pedantic`
> `-std=c++11`: use C++11 revision

Random numbers with ROOT

A classe de base em ROOT para a geração de números aleatórios é `TRandom` cuja documentação pode ser encontrada neste TRandom link[2].

Recordar que a variável global `gRandom` existente em ROOT (para a utilizar incluir "TRandom.h") é um ponteiro para um objecto `TRandom3` instanciado por defeito em ROOT.

```
The following methods are provided to generate random numbers disctributed according to some basic
    distributions:

Exp(tau)
Integer(imax)
Gaus(mean,sigma)
Rndm()
Uniform(x1)
Landau(mpv,sigma)
Poisson(mean)
```

[1] https://en.cppreference.com/w/cpp/11
[2] https://root.cern.ch/doc/master/classTRandom.html#afacc369d9fce4108dd84dfdcd1f52c30

```
Binomial(ntot,prob)
```

Exemplos de utilização:

Geração de números aleatórios inteiros até 100 (de 0 a 99)

```
double x = gRandom->Integer(100);
```

Geração de números aleatórios reais uniformes no intervalo [0., 1.]

```
double x = gRandom->Uniform();
// OU
double x = gRandom->Rndm();
```

Geração de números aleatórios reais uniformes no intervalo [1., 10.]

```
double x = gRandom->Uniform(1, 10);
```

Geração de números aleatórios reais de acordo a distribuição exponencial

```
double decay_constant = -10.;
double x = gRandom->Exp(decay_constant);
```

Utilização das funções TMath::

Determinação do índice do mínimo de um array

```
// generate vector of 100 random numbers between 0 and 1
std::vector v(100);
stl::generate(v.begin(), v.end(), rand);
// find minimum iterator
auto it = TMath::LocMin(v.begin(), v.end());
// relative position of minimum on array
int n = it - v.begin();
```

2ª aula de problemas

pointers: variable pointers, dereferencing pointers

```cpp
// F Barao
// 2020-21 Comp Phys
//
// pointers example
// compilation: g++ -std=c++11 pointers.C -o pointers.exe

#include <cstdio> // printf
#include <typeinfo> // typeid

#include <iostream>

int main() {
  int a = 10; // create an integer and assign a value
  printf("a=%d \n", a);

  int *b = &a; // create pointer to a

  // print variable types
  printf("a:%s b:%s *b:%s \n", typeid(a).name(), typeid(b).name(), typeid(*b).name());

  // print variable values
  printf("a=%d (%s) b=%p (%s) (%s) \n", a, typeid(a).name(), b, typeid(b).name(), typeid(*b).name());

  // dereference variable b (get value pointed by b)
  int c = *b;
  printf("a=%d c=%d \n", a, c);

  // size of types
  std::cout << "int (integer): \t" << sizeof(int) << " bytes" << std::endl;

  // now I'm going to change a value; c changed?
  a = 20;
  printf("a=%d c=%d \n", a, c);

// increment pointer b by 1: what it means? it will add 4 bytes(int size) to b...
  int *b1 = b+1;
  printf("b=%p b1=b+1=%p \n", b, b1);
}
```

arrays and pointers

In this example we are going to create a matrix element with 2 rows and 3 columns, for practicing:

- create matrix
- print matrix
- create pointer to 1st matrix element and rows
- access to matrix elements through pointer and [] operator
- pass a matrix as argument of a function

```cpp
// F Barao (Sep 2020)
// P02_array_pointers.C
#include <cstdio>
#include <iostream>
using namespace std;
```

```

// function prototyping
void func( int x[][3]);

// main program
int main() {
  // make matrix of 2 ROWS * 3 COLS
  int m[2][3] = { {1,2,3},
                  {4,5,6} };

  int a[][3] = { {1,2,3},
                 {4,5,6} };

// print matrix
  for (int i=0; i<2; ++i) {
    cout << "matrix line " << i << ":" << flush;
    for (int i=0; i<2; ++i) {
      cout << m[i][j] << " " << flush;
    }
    cout << endl;
  }

  // pointer to 1st integer
  int *b = &a[0][0];
  printf("%p %p %p\n",b, b+1, b+2);
  printf("%d %d %d\n",*b, *(b+1), *(b+2));

  // pointer to first row (a[0] is a pointer to the row! a[0][0] is a value)
  int *p1 = a[0];
  printf("%p %p %p\n", p1, p1+1, p1+2);
  printf("%d %d %d\n", *p1, *(p1+1), *(p1+2));

  // pointer to second row
  int *p2 = a[1];
  printf("%p %p %p\n",p2, ++p2, ++p2);
  // warning! p2 is now shifted to end
  p2 -=2;
  // using operator [] to access values
  printf("%d %d %d\n",p2[0], p2[1], p2[2]);

  // pass array to function
  func(a);

}

void func( int x[][3]) {
  printf("%p %p\n", x, x+1);
}
```

creating an array of strings on heap memory

```
// F Barao (Sep 2020)
// P02_matrix_new_delete
#include <cstdio>
#include <iostream>
#include <string>
using namespace std;

int main() {

// create array of strings
```

```
int n=100;
string *s = new string[n];

// use now pointer to set strings
s[0] = "Tudo vale a pena quando a alma não é pequena";
s[1] = "Querer não é poder. Quem pôde, quis antes de poder só depois de poder. Quem quer nunca há-de
    poder, porque se perde em querer."

// print strings
for (int i=0; i<n; i++){
     cout << s[i] << endl;
}

// free memory
if (s) {
  delete [] s;
}

}
```

allocating arrays with new operator (heap memory)

```
// F Barao (Sep 2020)
// P02_matrix_new_delete
#include <cstdio>
#include <iostream>
using namespace std;

int main() {

  // make matrix of 10 ROWS * 5 COLS on heap memory
  int **m = new int*[10]; //10 arrays
  for (int i=0; i<10; i++) {
    m[i] = new int[5]; // pointer to 1-dim array of 5 elements
  }

  //  //setting values to the 50 allocated memory positions
  for (int i=0; i<10; i++) {
    for (int j=0; j<5; j++) {
      m[i][j] = i*5 + j;
    }
  }

  // accessing elements
  for (int i=0; i<10; i++) {
    int *p = m[i];
    for (int j=0; j<5; j++) {
      cout << *p << " " << flush;
      p++;
    }
    cout << endl;
  }
  cout << endl;

  for (int i=0; i<10; i++) {
    for (int j=0; j<5; j++) {
      cout << m[i][j] << " " << flush;
    }
    cout << endl;
  }
  cout << endl;
```

```
40
41    // free memory
42    for (int i=0; i<10; i++) {
43       delete[] m[i];
44    }
45    delete[] m;
46
47
48    return 0;
49
50 }
```

3ª aula de problemas

funções: recursividade e passagem de argumentos no programa main

O exemplo da função factorial

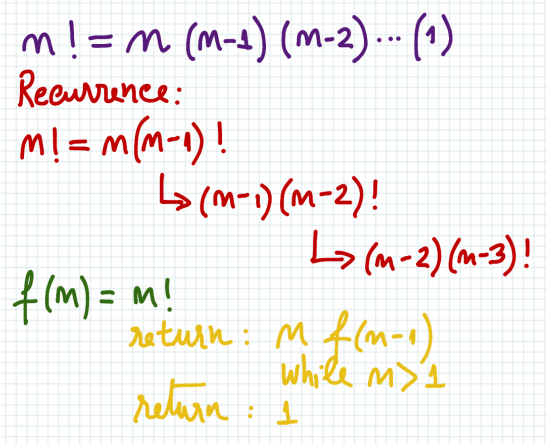

Figura 1: esquema de recursividade para a função factorial

```
#include <iostream>
using namespace std;

int fact(int n);

int main(){
   int num;
   cout<<"Type a number: ";
   cin>>num;
   cout<<"Factorial: "<< fact(num) << endl;
}

int fact(int n){
  if (n <= 1) {
    return 1;
  } else {
    return n*fact(n-1);
  }
}
```

Exercício 6

> Exercício 6 :
> Calcule o quadrado de um número inteiro positivo, x, usando somente as operações:
> a) adição, subtracção, multiplicação (×2)
> b) junte a hipótese de chamar uma função de forma recursiva (recursion)

Resolução:

$$x^2 = (x-1+1)^2 = (x-1)^2 + 1 + 2(x-1)$$

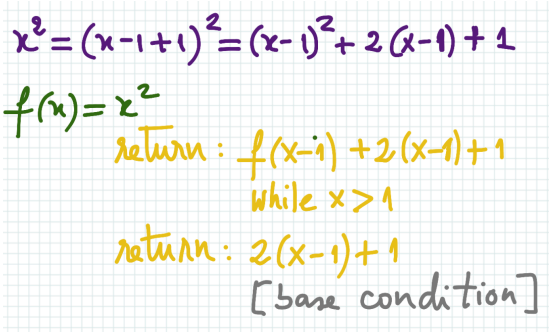

Figura 2: esquema de recursividade para a função factorial

```
1  // F Barao (Oct 2020)
2  // 2020-21 Comp Phys
3  //
4  // Resolução do Exercício 6 da série de problemas
5  // compilation: g++ -std=c++11 E06.C -o E06.exe
6
7  #include <iostream> // cout
8  #include <cstdlib> // atoi
9  #include <cstdio> // printf
10 using namespace std;
11
12 // prototyping
13
14 int fsquare(int x);
15
16 // define main function accepting arguments
17
18 int main(int argc, char* argv[] ) {
19   cout << "main arguments: " << flush;
20   for (int i = 0; i < argc; ++i)
21         cout << argv[i] << " ";
22   cout << endl;
23
24   int N = 0;
25   if (argc==2) {
26     N = atoi(argv[1]);
27     if (N<0) exit(1);
28   } else {
29     exit(1);
30   }
31   printf("computing: %d^2 = %d \n",N, fsquare(N));
32
33
34 }
35
36 // define function
37
38 int fsquare(int x) {
39   if (x==1) {   // base condition (end of recurrence)
40     return 2*(x-1) + 1;
41   } else { // recurrence
42     return fsquare(x-1) + 2*(x-1) + 1;
43   }
44 }
```

funções: mecanismos na passagem de argumentos (Ex.17)

> Exercício 17:
> Realize os seguintes códigos em C++:
> a) uma função que inicialize um avariável inteira com um número aleatório e retorne o seu *pointer*

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

> b) uma função que inicialize uma variável inteira com um número aleatório e retorne a sua referência

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

> Verifique que os endereços da variável *int* interna da função e da variável retornada são os mesmos

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

```
// F Barao
// 2020-21 Comp Phys
//
// Resolução do Exercício 17 da série de problemas
// compilation: g++ -std=c++11 E17.C -o E17.exe

#include <iostream> // cout
#include <cmath> // sin, cos,...
#include <cstdlib> //exit, rand, getchar
#include <ctime> // time
#include <cstdio> // printf
using namespace std;

int* func1();
int& func2();

int main() {

  // init
  int n = 1e6;

  // seed random
  time_t T;
  srand(time(&T)); // store time in T variable
  cout << "time: " << T << endl;

  printf("returning by pointer.............\n");

  // return by pointer
  int* ptr = nullptr;
  for(int j = 0; j < n; ++j ){
    ptr = func1();
    if (j<10) {
      printf("main returned random number: %d (address=%p) \n", *ptr, ptr);
    }
  }

  // We have a problem here of memory leakage
  // (1000000 int numbers create in heap memory...and lost!!!)

  // To solve this we are going to create and array of pointers
```

```
  // to keep track of allocated integers

  int **p = new int*[n]; // array of pointers to integers

  printf("returning by reference.....\n");

  // return by reference
  for(int j = 0; j < n; ++j ){
    int& x = func2();
    printf("main returned random number=%d (address=%p)\n", x, &x);
    //      delete &x;
    p[j] = &arr;
  }

  printf("freeing memory.....\n");
  getchar();

  // free memory
  // func2 freeing memory
  for(int j = 0; j < n; ++j ){
    delete p[j]; // delete object, in this case an integer
  }
  delete [] p;

}

////////////////////// functions

int* func1(){
  // allocate integer memory space and initialize it with random
  int *rndnumber = new int(rand());
  printf("random number: %d (address=%p) \n", *rndnumber, rndnumber);
  return rndnumber;
}

int& func2() {
  // allocate integer memory space and initialize it with random
  int* x = new int(rand());
  printf("random number=%d %p\n", *x, x);
  return *x;
}
```

4ª aula de problemas

funções Lambda

> As funções lambda são um meio muito prático de codificarmos uma função ao longo de um programa sem haver necessidade de fazer previamente a sua declaração (revisão C++11).

```
// F Barao
// 2020-21 Comp Phys
//
// lambda functions example

#include <cstdio> // printf
#include <typeinfo> // typeid
#include <iostream> // cout
#include <string> // string class
#include <vector> // STL vector

int main() {

  // [] capture specification: tells the compiler we are creating a lambda function
  // () argument list
  // Note: lambda function requires explicit type on params, auto cannot be used

  // define lambda function f1: no parameters, no return
  auto f1 = []() {
    printf("f1 function: test lambda function \n");
  };
  // call function
  f1();

  // - define lambda function that searchs for a pattern in variable name
  //   that is defined outside function
  // - we need to capture variables outside function (by copy)
  std::string name="teste name";
  auto f2 = [name](std::string patt) {
    return name.find( patt ) != std::string::npos; //  pos is a static var
  };
  std::cout << "teste found? : " << f2("teste") << std::endl;
  std::cout << "xx found ? :" << f2("xx") << std::endl;

  // physical constants
  const double h = 6.62607015E-34; // Planck constant ··−kgm2s1
  const double c = 299792458.; // light speed m/s
  const double eV2J = 1.6022E-19; // 1eV in Joules

  // compute wavelength (nm) given a freq (Hz)
  auto fwl = [c](double freq){
    return c/freq*1E9; // nm
  };

  cout << "fwl(3.E11)=" << fwl(3.E11) << " nm" << endl;

}
```

Input/Output

Reading/Writing into files

Reading files: general rules

> 1: Try to read data (» or getline) 2: Check if you failed 3: Only use the data read from step 1 if you succeeded

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cstdio>
#include <iomanip>      // std::setw
using namespace std;


/* In this example:
   - open a file for writing 3 columns of numbers
     and then open it for reading assuming we knew nothing about the
     number of lines of the file...only the number of columns...
   - read the file.
   - reading of full line can be done through getline
*/

int main() {

//-------------------------------------------------------
// write to file
//-------------------------------------------------------

  // open file to output data
  string filename = "fstream_IO.dat";
  cout << "opening output file..." << filename << endl;
  fstream wFile(filename,ios::out);
  if (!wFile.is_open()) {
    cout << "wFile failed to be opened...!!" << endl;
    exit(1);
  }

  // create 5 lines of 3 numbers
  for (int i=0;i<5;i++) {
    // create array of 3 doubles filled with values
    // the array is local and will be destroyed automatically
    // at the end of loop
    double x[]={(i+1)*10. + 1, (i+1)*10. + 2, (i+1)*10. + 3};

    // write line of 3 numbers with spaces in between
    wFile << fixed << setprecision(9) <<
      x[0] << "  " << *(x+1) << "  " << x[2] << "  " << endl;
  }

  // closing file
  cout << "closing file..." << endl;
  wFile.close();

  //-------------------------------------------------------
  // read file now
  //-------------------------------------------------------

  // create matrix to house numbers
  vector<vector<double> > M2;

  // open file in reading mode
  fstream rFile;
  rFile.open(filename, ios::in);

```

```
  // check that file was correctly opened
  if (!rFile) {
    cout << "Error opening the file" << endl;
    exit(1);
  }
  cout << "file opened for reading...[" << filename << "]" << endl;

  // define line counter
  int count=0;

  // read numbers to y array while file is readable
  double y[3];
  while (rFile >> y[0] >> y[1] >> y[2]) { // returns boolean true if succeeded
    // increment counter
    ++count;
    cout << "reading line: " << count << endl;
    cout << y[0] << " , " << y[1] << " , " << y[2] << endl;

    // store values on vector
    vector<double> aux(y,y+3);
    M2.push_back(aux);

  }
  rFile.close();
  cout << "Nb of lines read from file: " << count << endl;

  //Just to test we assigned the data fine to the vector...
  cout << fixed << setprecision(3);
  for (int i=0; i < (int)M.size(); i++) {
    for (int j=0; j<(int)M[i].size(); j++ ){
      cout << "[" << i << "," << j << "]=" << M2[i][j] << " " << flush;
    }
    cout << endl;
  }


  // read numbers one by one while file is readable
  // with this method we loose the line structure
  rFile.open(filename, ios::in);

  // check that file was correctly opened
  if (!rFile) {
    cout << "Error opening the file" << endl;
    exit(1);
  }
  cout << "file opened for reading...[" << filename << "]" << endl;

  cout << "reading full file number by number..." << endl;
  double d;
  while ( rFile >> d ) {
    cout << d << " " << endl;
  }
  rFile.close();


  // read file now using full line reading and parsing after
  cout << "reading line by line with getline global function..." << endl;
  rFile.open(filename, ios::in);
  // check that file was correctly opened
  if (!rFile) {
    cout << "Error opening the file" << endl;
    exit(1);
  }
  cout << "file opened for reading...[" << filename << "]" << endl;
```

```
124
125   count = 0;
126   string s;
127   while (getline(rFile, s)) { // getline finishes by default at ´\n´
128     cout << "reading line [" << count+1 << "] " << s << endl;
129     // increment counter
130     ++count;
131   }
132   rFile.close();
133   cout << "Nb of lines read from file: " << count << endl;
134   return 0;
135 }
```

The use of stringstream

> A stringstream associates a string object with a stream allowing you to read from the string as if it were a stream (like cin).

```
1  #include <string>       // std::string
2  #include <iostream>     // std::cout
3  #include <sstream>      // std::stringstream
4  #include <fstream>      // std::fstream
5  #include <vector>       // std::vector
6
7  // on purpose I did not place "using namespace std" after includes;
8  // a lot of "std:" has to be inserted in the code...
9
10 int main () {
11
12   // build stringstream
13   std::stringstream ss;
14   ss << 1.23 << ' ' << 2.45;
15   //...and parse it to integer numbers
16   float x,y;
17   ss >> x >> y;
18   // printout
19   std::cout << "x: " << x << '\n';
20   std::cout << "y: " << y << '\n';
21
22   // print stringstream as a string
23   std::cout << ss.str() << std::endl;
24
25   // clear stringstream and reusing it
26   ss.clear();
27   ss.str("");
28
29   // use stringstream to read file
30   // read file now using full line reading and parsing after
31   std::string filename = "fstream_IO.dat";
32   std::fstream rFile;
33   rFile.open(filename, std::ios::in);
34   // check that file was correctly opened
35   if (!rFile) {
36     std::cout << "Error opening the file" << std::endl;
37     exit(1);
38   }
39   std::cout << "file opened for reading...[" << filename << "]" << std::endl;
40
41   int count = 0;
42   std::string line;
43   while (getline(rFile, line)) {
```

```
    std::stringstream ss(line);
    // parsing values to vector
    std::vector<double> v;
    double d;
    // automatic conversion to var type
    // >> operator allways stops reading at whitespace
    while (ss>>d) {
      v.push_back(d);
    }
    // printout vector contents
    for (auto i: v) {
      std::cout << i << " " << std::flush;
    }
    std::cout << std::endl;
  }
  rFile.close();


  return 0;
}
```

STL library

vector container

> This basic example will show:
>
> 1. how to create a vector in many ways
> 2. how to use iterator

```
// Fernando Barao (Oct 2020)
#include <iostream>
#include <vector>
using namespace std;

int main() {

  // create a vector with initialization
  vector<int> vi = {1,2,3,4,5};
  for(auto it = vi.begin(); it != vi.end(); ++it) {
    cout << *it << " " ;
  }
  cout << endl;

  // create a vector from array
  int myInt[] = {1,2,3,4,5};
  vector<int> v(sizeof(myInt)/sizeof(int));
  auto a = copy(myInt, myInt+5, v.begin());
  cout << *(a-1) << endl;
  for(auto it = v.begin(); it != v.end(); ++it) {
    cout << *it << " " ;
  }
  cout << endl;

  // create a vector and fill it with 1,2,3,4,5
  vector<int> v2(5); // vector of 5 elements
  int count = 1;
  for(auto& x : v2) {
    x = count;
    ++count;
  }
```

```
  for(auto it = v2.begin(); it != v2.end(); ++it) {
    cout << *it << " " ;
  }
  cout << endl;


  // create a vector and resize it
  vector<int> v3(5,0); // vector of 5 elements value 0
  v3.resize(10); // resize to 10
  count = 1;
  for(auto& x : v3) { // 1,2,3,4,5,6,7,8,9,10
    x = count;
    ++count;
  }
  for(auto it = v3.begin(); it != v3.end(); ++it) {
    cout << *it << " " ;
  }
  cout << endl;

  // erase elements: from 2nd to 5th
  v3.erase(v3.begin()+1, v3.begin()+5);
  for(auto it = v3.begin(); it != v3.end(); ++it) {
    cout << *it << " " ;
  }
  cout << endl;

  // create a vector with push_back
  vector<int> v4;
  for (auto i : {1,2,3,4,5}) {
    v4.push_back(i);
  }
  for(auto it = v4.begin(); it != v4.end(); ++it) {
    cout << *it << " " ;
  }
  cout << endl;

  v4.clear();

  // check size of a vector and if it is empty
  cout << "v3 size=" << v3.size() << " is empty? " << v3.empty() << endl;

  // build a vector from another
  vector<int> v5(v3);
  for(auto it = v5.begin(); it != v5.end(); ++it) {
    cout << "i=" << it-v5.begin() << " v5=" << *it << " v3=" << v3[it-v5.begin()] << endl;
  }
  cout << endl;

  // are two containers equal? when using with if, you dont need cast...
  cout << "check if containers v5 and v3 are equal: " << bool(v5==v3) << endl;

}
```

5ª aula de problemas

Making libraries

> No esquema de directórios que implementámos:
> `src/`: contém classes ou funções que serão necessários nos programas principais
> `main/`: contém programas que realizam a solução dos problemas com recurso a funções auxiliares ou classes em src/

> Isto torna necessário pensarmos na adição de uma regra de produção de biblioteca no *Makefile*

static libs

```
ar ruv libFC.a  a.o b.o c.o ...
ranlib libFC.a
```

shareable libs

```
On Mac OS X: g++ -dynamiclib -flat_namespace myclass.cc -o myclass.so
On Linux:    g++ -fPIC -shared myclass.cc -o myclass.so
```

adding rule to Makefile

```
lib: $(LIBDIR)/libFC.a

$(LIBDIR)/libFC.a : $(OBJ)
        @echo making lib...
        ar ruv $@  $^
        ranlib $@
```

Makefile complete and modified

```
# Makefile

BINDIR := bin
LIBDIR := lib

CC := g++ -std=c++11 -pedantic

# src/ (declarcaoes de funcoes, de calsses + codigo)
# main/ (programas principais)
# bin/ (temporarios, .o, .exe)
# lib/ (bibliotecas) biblioteca FC

# making library
# - static: .a
# - shareable: .so

VPATH = main:src

SRC := $(wildcard src/*.C)
OBJ := $(patsubst %.C, $(BINDIR)/%.o, $(notdir $(SRC)))
```

```
INC := $(wildcard src/*.h)

lib: $(LIBDIR)/libFC.a

$(LIBDIR)/libFC.a: $(OBJ)
        @echo make lib...
        ar ruv $@ $^
        ranlib $@

%.exe: $(BINDIR)/%.o $(LIBDIR)/libFC.a
        @echo compilink and linking...
        $(CC) $< -o $(BINDIR)/$@ -L lib -l FC

$(BINDIR)/%.o: %.C | $(INC)
        @echo compiling... $<
        $(CC) -I src -c $< -o $@


######### clean

tilde := $(wildcard */*~) $(wildcard *~)
exe := $(wildcard */*.exe) $(wildcard *.exe)
obj := $(wildcard */*.o) $(wildcard *.o)

clean:
        @echo cleaning dir...
        rm -f $(exe) $(obj) $(tilde)
```

Exemplos de classes: point e line

> De seguida, desenvolveremos uma classe **point** 2-dim e um conjunto extenso de métodos que permitam operar os objectos do tipo **point**.

> Comecemos pela declaração da classe e dos métodos associados, que neste caso somente possui membros **private** e **public**

file: point.h, containing class point declaration

```
// F Barao, Oct 2020
// class point


#ifndef __point__
#define __point__

class point {

  float* coo; //pointer

 public:

  point(float x=0., float y=0.); //default constructor
  ~point(); //destructor

  point(const point&); // copy constructor
  const point& operator=(const point&); // copy assignment

  point(point&&); // move constructor
```

```
  const point& GetObject()  { // get my object by reference
    return *this;
  }

  //  ------

  void Dump() const; // method const to prevent change object

};

#endif
```

file: point.C, containing class point code

> Implementação dos métodos associados à classe.

```
#include "point.h"
#include <cstdio>
#include <algorithm> // std::swap
#include <utility> // std::exchange

// ======================= constructors & destructors

// constructor:
// - allocate memory for 2 floats
point::point(float x, float y) : coo(new float[2] {x,y}) {
  printf("[%s] constructor...\n", __PRETTY_FUNCTION__);
}

// copy constructor:
// - allocate memory for 2 floats and copy contents of object copied
point::point(const point& P) : coo(new float[2] {P.coo[0], P.coo[1]}) {
  printf("[%s] copy constructor...\n", __PRETTY_FUNCTION__);
}

// copy assignment
// - object already exists, copy contents of other to it
// - Object.operator=(Other); Object = Other
const point& point::operator=(const point& P) {
  printf("[%s] copy assignment...\n", __PRETTY_FUNCTION__);
  if (this != &P) {
    coo[0]=P.coo[0];
    coo[1]=P.coo[1];
  }
  return *this;
}

// move constructor
// - constructor called when object is built from a temporary object (reference to rvalue)
  point::point(point&& P) : coo(P.coo) {
  printf("[%s] move constructor...\n", __PRETTY_FUNCTION__);
  P.coo = nullptr;
}

// destructor
point::~point() {
  printf("[%s] destructor... (%p)\n", __PRETTY_FUNCTION__, this);
  if (coo) delete [] coo;
}

// ======================= others
```

```
void point::Dump() const {
  printf("[%s] point address = %p | members=[%f, %f] (array address=%p)\n", __PRETTY_FUNCTION__, this,
      coo[0], coo[1], coo);
}
```

programa principal

```
#include "point.h"
#include <cstdio>
#include <vector>

int main() {

  point P1(1,2); // constructor
  P1.Dump(); // dump

  point P2(P1); // copy constructor point P2=P1
  P2.Dump();

  point P3(10,20);
  P2 = P3;
  P2.Dump();

  printf("_____________________ vector\n");
  std::vector<point> V;
  V.push_back(P3);
  V.push_back(point(1.2, 3.374));

  printf("_____________________ reference\n");
  const point& a = P3.GetObject();
  P3.Dump();
  a.Dump();

  printf("_____________________ ending\n");

}
```

6ª aula de problemas

Inclusão das bibliotecas de ROOT e include files

> O ROOT possui um conjunto de bibliotecas que serão utilizadas na linkagem dos programas main
>
> Para vermos o conjunto de bibliotecas disponíveis, fazer:
> `root-config --libs`
>
> Temos então que incluir a linkagem com as bibliotecas de ROOT no Makefile através de duas novas variáveis a definir, `ROOTLIB` e `ROOTINC`.

adding rule to Makefile

> ROOTLIB := $(shell root-config –libs)
>
> ROOTINC := $(shell root-config –incdir)

Makefile complete and modified

```
# Makefile

BINDIR := bin
LIBDIR := lib

CCFLAGS := -pedantic

CC := g++ -std=c++11

# src/ (declarcaoes de funcoes, de calsses + codigo)
# main/ (programas principais)
# bin/ (temporarios, .o, .exe)
# lib/ (bibliotecas) biblioteca FC

# making library
# - static: .a
# - shareable: .so

VPATH = main:src

ROOTLIB := $(shell root-config --libs)
ROOTINC := $(shell root-config --incdir)

SRC := $(wildcard src/*.C)
OBJ := $(patsubst %.C, $(BINDIR)/%.o, $(notdir $(SRC)))
INC := $(wildcard src/*.h)

lib: $(LIBDIR)/libFC.a

$(LIBDIR)/libFC.a: $(OBJ)
    @echo make lib...
    ar ruv $@ $^
    ranlib $@

%.exe: $(BINDIR)/%.o $(LIBDIR)/libFC.a
    @echo compilink and linking...
    $(CC) -I src $< -o $(BINDIR)/$@ -L lib -l FC $(ROOTLIB)

$(BINDIR)/%.o: %.C | $(INC)
```

```
    @echo compiling... $<
    $(CC) -I src -I $(ROOTINC) -c $< -o $@


######### clean

tilde := $(wildcard */*~) $(wildcard *~)
exe := $(wildcard */*.exe) $(wildcard *.exe)
obj := $(wildcard */*.o) $(wildcard *.o)  $(wildcard */*.pcm) $(wildcard */*.d)
mylibs := $(wildcard */*.so) $(wildcard */*.a)

clean:
    @echo cleaning dir...
    rm -f $(exe) $(obj) $(tilde) $(mylibs)
```

Exemplo de classes: point e line (continuação)

Classe point

> Prosseguimos o desenvolvimento da classe point, implementando agora os operadores +, -, +=, /=, !, [] e ainda as funções friend,
>
> ```
> friend ostream& operator<<(ostream& s, const point& );
> friend point operator*(float, const point&);
> ```
>
> Alterámos também o qualificativo dos data members da classe de private para protected de forma a assegurar a sua visibilidade nas classes que herdem.

file: point.h, containing class point declaration

```
// F Barao, Oct 2020
// class: point

#ifndef __point__
#define __point__

#include <iostream> // ostream
using namespace std;

// inheritance

class point {

protected:

  float* coo; // pointer

 public:

  point(float x=0., float y=0.); //default constructor
  virtual ~point(); //destructor

  point(const point&); // copy constructor
  const point& operator=(const point&); // copy assignment

  point(point&&); // move constructor

  const point& GetObject()  {
    return *this;
  }
```

```
  // ------- operators

  point operator+(const point&) const;
  point operator-(const point&) const;
  const point& operator+=(const point&);
  const point& operator/=(float);

  float operator!(); // norma sqrt(x^2 + y^2)

  // -------- access operator

  float& operator[](int n); // point P, P[0]=10;
  const float& operator[](int n) const;

  // -------- friends

  friend ostream& operator<<(ostream& s, const point& );
  friend point operator*(float, const point&); // point P2 = 4*P1

  // ------- other
  void Dump() const;

  virtual void PrintObject();
  virtual void Print();

};

#endif
```

point.C

```
#include "point.h"
#include <cstdio>

// constructor
point::point(float x, float y) : coo(new float[2] {x, y}) {
  // allocate memory + init values
  printf("constructor called (%p) \n", this);
  //  coo = new float[2];
  //  coo[0] = x;
  //  coo[1] = y;
}

// destructor
point::~point() {
  // deallocate memory
  printf("destructor called (%p) \n", this);
  delete [] coo;
}

// copy constructor
point::point(const point& other) : coo(new float[2] {other.coo[0], other.coo[1]}) {
  printf("copy constructor called (obj=%p, other=%p) \n", this, &other);
}

// copy assignment
const point& point::operator=(const point& other) {
  printf("copy assignment (operator=) called (obj=%p, other=%p) \n", this, &other);
  if (this != &other) {
    coo[0] = other.coo[0];
    coo[1] = other.coo[1];
  }
  return *this;
}
```

```
// move
point::point(point&& other) : coo(other.coo) {
    printf("move constructor called (obj=%p, other=%p) \n", this, &other);
    other.coo = nullptr;
}

// ============== operators

point point::operator+(const point& p) const {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  return point(coo[0]+p.coo[0], coo[1]+p.coo[1]);
}

point point::operator-(const point& p) const {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  return point(coo[0]-p.coo[0], coo[1]-p.coo[1]);
}

const point& point::operator+=(const point& p) {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  coo[0] += p.coo[0];
  coo[1] += p.coo[1];
  return *this;
}

const point& point::operator/=(float a) {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  coo[0] /= a;
  coo[1] /= a;
  return *this;
}

float point::operator!() {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  return sqrt(pow(coo[0],2.) + pow(coo[1], 2.));
}

float& point::operator[](int n) {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  return coo[n];
}

const float& point::operator[](int n) const {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  return coo[n];
}


// ============== friends

ostream& operator<<( ostream& s, const point& p) {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  s << "[" << p.coo[0] << " , " << p.coo[1] << "]";
  return s;
}

point operator*(float k, const point& p) {
  float a = k*p.coo[0];
  float b = k*p.coo[1];
  return point(a, b);
}

// ==============
```

```
void point::PrintObject() {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  this->Print();
}

void point::Print() {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  cout << *this;
}



void point::Dump() const {
  printf("Dump(): this=%p [%f , %f] (%p)\n", this, coo[0], coo[1], coo);
}
```

Classe line

> A classe line herda da classe point e permite e definição de uma recta.

line.h

```
#ifndef __line__
#define __line__

/*

point
  ^
  |
line

*/

#include "point.h"
#include <cstdio>

class line: public point {

 protected:

  point uvector; // versor

 public:

  line();
  line(const point& , const point&);
  ~line() {
    printf("[%s] \n", __PRETTY_FUNCTION__);
  };

  friend ostream& operator<<(ostream&, const line&);

};

#endif
```

line.C

```
#include "line.h"
#include <cstdio>
```

```
line::line() : uvector(point(0,0)) {
  printf("[%s] \n", __PRETTY_FUNCTION__);
}

line::line(const point& P1, const point& P2) : point(P1), uvector(P2) {
  printf("[%s] \n", __PRETTY_FUNCTION__);
  if ( fabs(!uvector - 1.) > 0.001 )
    uvector /= !uvector;
}

// ============= operators

ostream& operator<<(ostream& s, const line& L) {
  s << "point=[" << point(L.coo[0], L.coo[1]) << "] " << "uvector=[" << L.uvector;
  //  s << "point=[" << *(point*)&L << "] " << "uvector=[" << L.uvector;
  /*
    s << "point=[" << L.coo[0] << " , " << L.coo[1] << "] "
    << "uvector=[" << L.uvector[0] << " , " << L.uvector[1] << "]";
  */
  return s;
}
```

main program using line and point classes

```
#include "line.h"

int main() {

  point P1(1,2);
  point P2(10,20);
  point P3 = P1+P2;
  cout << P3 << endl;

  point P4(P2);
  P4 /= 2.;
  cout << P4 << endl;

  line L(P1,P2);
  cout << L << endl;

}
```

7ª aula de problemas

Introdução a ROOT

Directory tree structure extended to include ROOT specific analysis

> Iremos necessitar de ROOT para fazer display dos resultados que obtivermos.
> A forma mais simples de o fazer é:
> - integrarmos os códigos desenvolvidos em `src/` nas bibliotecas de ROOT e assim podermos usar directamente as classes desenvolvidas.
> Veremos de seguida como fazê-lo.
> - desenvolvermos uma macro em C++ que faça o equivalente ao que faz habitualmente um programa main() e fazermos aí a interação com ROOT, ou seja, os displays.

> Proponho desde já que criemos um novo directório com o nome rootANA/.
> Teremos assim:
> src/
> main/
> rootANA/
> bin/
> lib/
>
> Todas as macros de análise em C++ serão desenvolvidas neste directório rootANA/.

How to integrate USER code in ROOT?

```
Vamos também adicionar a esse directório uma macro com o nome rootlogon.C, que tem a particularidade de
    ser imediatamente corrida no início de ROOT.
Essa macro rootlogon.C vai conter as instruções que permitem adicionar as classes ou funções (USER code)
    a ROOT.
Eis a minha macro rootlogon.C que permite adicionar as classes point e line:
```

```
void rootlogon() {
  printf("running my rootlogon...\n");
  gInterpreter->AddIncludePath("../src/"); // permite a ROOT saber onde estão os meus include files
  gSystem->CompileMacro("../src/point.C", "k O"); // compila a classe point
  gSystem->CompileMacro("../src/line.C", "k O"); // compila a classe line
}
```

```
Portanto, dirijo-me para o directório rootANA/ e lanço ROOT (root -l). Aí vejo imediatamento rootlogon.C
    a ser processado automaticamente.

Desenvolvo ainda uma macro de análise, mline.C,  que usa as classes point e line:
```

```
#include "line.h"
#include "TH1F.h"

void mline() {
  line L(point(0.11, 0.22), point(1.,2.));
  cout << L << endl;
}
```

```
Sequência:

root -l
> .L mline.C++
```

```
> mline()
```

Example: histogram with gaussian distribution from random number generation

```
 - Edit file mhist.C and write C++ code
 - run macro using interpreter
 - run macro using automatic compiler (ACLIC)
 - save ROOT objects into a file
 - open a ROOT session, open objects file and Draw them
```

```
void mhist() {
  ////////////////////////////////////////////////////////////////////////
  // drawing a simple histogram
  ////////////////////////////////////////////////////////////////////////

  // setting up the histogram
  auto h1 = new TH1D("histo1", "My First Histogram", 100 /*bins*/, -5. /*min*/, 5. /*max*/);

  // filling the histogram
  for (int i = 0; i < 10000; i++) {
    // root has a random class TRandom we can use
    double x = gRandom->Gaus(0. /*mean*/, 1. /*sigma*/);

    h1->Fill(x);
  }

  // store my histogram into ROOT file
  TFile F("fstorage.root", "recreate");
  h1->Write();
  F.Close();
```

Example: Input/Output

```
 - open ROOT file
 - list objects
 - Draw object
```

```
root -l
root> TFile F("fstorage.root")
root> F.ls() // list objects (h1 shall be visible)
root> h1->Draw() // Draw histogram
```

Example: make TGraph from data read from file

```
 - edit macro with name: mgraph_data.C
 - make plots as TGraph
 - save plots and canvas to ROOT file
```

data file to read (gdata.txt)

```
  # data
  # comments
  # more comments
  1,    2,  1
  2,    4,  4
  3,    6,  9
  4,    8,  16
```

```
  5,   10,  25
  6,   12,  36
  7,   14,  49
  8,   16,  64
  9,   18,  81
  10,  20,  100
```

```
// doc:
// - graph plotting options: https://root.cern.ch/doc/master/classTGraphPainter.html#GP01

void mgraph_data() {
  TCanvas *c1 = new TCanvas("c1","gdata",20,10,500,700);
  c1->SetFillColor(kBlue-8);
  c1->SetGrid();

  // draw a frame to define the range
  TH1F *hr = c1->DrawFrame(0,0,11,21);
  hr->SetXTitle("X data");
  hr->SetYTitle("Y data");
  c1->GetFrame()->SetFillColor(21);
  c1->GetFrame()->SetBorderSize(12);

  // create graph from data file: reading 1st and 2nd columns
  auto gr = new TGraph("gdata.txt","%lg %lg %*lg", ",");
  gr->SetMarkerColor(kBlue);
  gr->SetMarkerStyle(21);
  gr->Draw("LP");
  c1->SaveAs("FIG_mgraph.pdf"); // can save any extension (.png, .eps, .jpg, ...)

  // good way of stopping execution in ROOT such that we can have a look to the graph
  // double-click out of the plot frame with the mouse or just type any key to move on
  c1->WaitPrimitive();

  // create second graph: reading 1st and 3rd column values
  auto gr2 = new TGraph("gdata.txt","%lg %*lg %lg", ",");
  gr2->SetLineWidth(4);
  gr2->SetLineColor(kRed+2);
  gr2->SetTitle("reading data file - example 2");
  gr2->GetXaxis()->SetTitle("X axis title");
  gr2->GetYaxis()->SetTitle("Y axis title");
  gr2->Draw("AC"); // smooth curve
  c1->Update(); // Frame is drawn after Update()
  c1->GetFrame()->SetFillColor(21);
  c1->GetFrame()->SetBorderSize(12);
  c1->Modified();

  // save graph
  TFile F("file.root", "recreate");
  gr->Write("g1");
  gr2->Write("g2");
  F.Close();
}
```

Example: use random number generator and make TGraph and Histogram

```
 - edit macro with name: mgraph_correl.C
 - study quality of random number generators: rand() and TRandom3
 - use STL to study eventual autocorrelation on generated numbers (previous output has implications on
     next output)
```

```
#include <ctime>
```

```
#include <vector>
#include <algorithm>
#include <iostream>
#include "TRandom3.h"
#include "TGraph.h"
#include "TCanvas.h"

void mgraph_correl() {

  // random generator
  // https://root.cern.ch/doc/v606/classTRandom.html
  // TRandom3 is the best: period of about 10**6000

  // gRandom: is a pointer to the current random number generator.
  // By default, it points to a TRandom3 object.
  gRandom->SetSeed(0); // random seed

  // vectors with 1000 elements
  int N = 10000;
  vector<Double_t> vrand1(10000);

  // generate values
  std::generate(vrand1.begin(), vrand1.end(), []() ->Double_t { return gRandom->Uniform(); });

  // output
  std::copy(vrand1.begin(), vrand1.begin()+100, std::ostream_iterator<double>(std::cout, " "));
  cout << '\n';

  // ROOT text
  TText *t = new TText();
  t->SetTextAlign(22);
  t->SetTextColor(kRed+2);
  t->SetTextFont(43);
  t->SetTextSize(40);
  t->SetTextAngle(0);

  // look to correlations between elements
  // i, i+1
  TGraph *g = nullptr;
  TCanvas *c = new TCanvas("c", "random correlation", 10, 10, 800, 600);
  for (int k=0; k<10; ++k) { // element lag
    if (g) delete g;
    g = new TGraph(N-k, vrand1.data(), vrand1.data()+k);
    g->SetTitle("");
    g->SetMarkerStyle(20);
    g->SetMarkerSize(0.2);
    g->Draw("AP");
    t->DrawTextNDC(0.5, 0.95, Form("lag = %d", k));
    c->Update();
    c->WaitPrimitive();
  }

  // look now to differences and make histogram
  /*y0 = x0
    y1 = x1 - x0
    y2 = x2 - x1
    y3 = x3 - x2
    y4 = x4 - x3
  */
  vector<double> vdif(N);
  std::adjacent_difference(vrand1.begin(), vrand1.end(), vdif.begin());
  auto h = new TH1D("h", "", 200, -1., 1.);
  std::for_each(vdif.begin()+1, vdif.end(), [h](double x){h->Fill(x);});
  h->SetFillColor(kGreen+2);
```

```
  h->SetMarkerStyle(20);
  h->SetLineWidth(3);
  h->SetMarkerColor(kBlue+2);
  t->DrawTextNDC(0.3, 0.95, Form("histo entries = %7.1f", h->GetEntries()));
  h->DrawCopy("E");
  c1->Modified();
  c1->Update();
  c->SaveAs("FIG_diff.png");

}
```

TF1 functor example: classe OpticalMat

> A classe **OpticalMat** possui como membro privado uma função que modeliza o índice de refração de um material $n(\lambda)$, `TF1 *f`.

> Apesar da TF1 poder ser instanciada com recurso a uma **lambda function**, neste exercício pretende-se que utilizem a técnica **functor** para passar a função (ver manual da TF1, nomeadamente o método E.).
> De seguida exemplifico:

header: OpticalMat.h

location: src/OpticalMat.h

```
#ifndef __OpticalMat__
#define __OpticalMat__

#include "TROOT.h"
#include "TF1.h"

#include <string>
#include <vector>
#include <utility>
using namespace std;

#include "Material.h"

class OpticalMat : public Material {
 public:

  OpticalMat (string fname="", Double_t fdens=0., TF1* ff=nullptr) : Material(fname, fdens), f(ff) {};
  OpticalMat (vector<pair<float,float>>, string fname="", Double_t fdens=0., TF1* ff=nullptr);
  ~OpticalMat();


private:
  double fRefIndex(double *x, double *par); // functor
  vector<pair<float,float>> RefIndex;
  TF1 *f;

};


#endif
```

source: OpticalMat.C

location: src/OpticalMat.C

```
#include "OpticalMat.h"

// construtor

OpticalMat::OpticalMat( vector<pair<float,float>> v, string fname, Double_t fdens, TF1* ff) :
Material(fname, fdens), f(ff) {
  RefIndex = v;
  if (!f) {
    f = new TF1("findex", this, &OpticalMat::fRefIndex, 0.1, 0.8, 5); // 5 params
  }
}

// destructor
OpticalMat::~OpticalMat() {
  delete f;
}

// ref index functor
double fRefIndex(double *x, double *par) {
  // parameters: A = par[0]
  //             B = par[1]
  //             C = par[2]
  //             D = par[3]
  //             E = par[4]
  // variable: x[0] = lambda (mu m)
  return par[0] + par[1]/(x[0]*x[0]-0.028) + par[2]/(x[0]*x[0]-0.028)/(x[0]*x[0]-0.028) +
      par[3]*x[0]*x[0] + par[4]*x[0]*x[0]*x[0]*x[0] ;
}
```

8ª aula de problemas

Sistemas lineares

classe Vec

> Na manipulação de matrizes pede-se que desenvolvam uma classe `Vec` que manipule arrays e que possua vários métodos (vejam exercício 48 da série de problemas) que tornarão fáceis as manipulações a efectuar nas matrizes para a resolução de sistemas lineares.

> Nesta aula iniciei o desenvolvimento desta classe `Vec`; no entanto, falta ainda a implementação de vários operadores e métodos genéricos que se encontram pedidos no exercício 48.

header: Vec.h
location: src/Vec.h

```
#ifndef __Vec__
#define __Vec__

#include <iostream> // ostream

class Vec {

 public:

  // constructors
  Vec(int i=0, double x=0); // Vec v;
  Vec(int, const double*);
  Vec(const Vec&);

  // destructor
  ~Vec();

  // operators
  double  operator[](int i) const;
  double& operator[](int i);

  void operator=(const Vec&); // A=B
  const Vec& operator+=(const Vec&);

  //(...)
  /*
  the overloading of then * operator allows multiply a vector by a
  scalar:
  Vec * scalar
  */
  Vec operator*(double) const; //Vec.operator*(double) <=> A*5.


  // friend methods
  friend std::ostream& operator<<(std::ostream&, const Vec&);
  /*
  in order to multiply a scalar to a vector (scalar*Vec) we need
  to implement a friend method:
  friend Vec operator*(double, const Vec&);
  */

 private:
  int N; // number of elements
  double *entries; // array
```

```
45
46 };
47 #endif
```

source: Vec.C

> location: src/Vec.C

> I introduced in the code, a pre-compiler variable DEBUG that once defined (#define DEBUG) allow us to include useful prints to the screen. In case we don't need anymore such a tool, just undefine the variable (#undef DEBUG)

> Notice also the definition of the friend method operator« that allow us to output easily the class contents

```
1  #include "Vec.h"
2  #include <cstdio>
3  #include <algorithm>
4  #include <stdexcept>
5  #include <iomanip>
6
7  #include "TROOT.h"
8
9  #define DEBUG
10
11 ////////////////////// constructors
12
13 Vec::Vec(int i, double x) : N(i) {
14 #ifdef DEBUG
15   printf("[%s]\n", __PRETTY_FUNCTION__);
16 #endif
17   if (N<0) throw std::invalid_argument(Form("[%s] received negative number of elements...!\n",
       __PRETTY_FUNCTION__));
18   entries = new double[N];
19   std::fill_n(entries, N, x);
20 }
21
22 Vec::Vec(int i, const double* x) : Vec(i) { //c++11 on...
23 #ifdef DEBUG
24   printf("[%s]\n", __PRETTY_FUNCTION__);
25 #endif
26   if (x)
27     std::copy(x, x+i, entries);
28   else
29     throw std::invalid_argument(Form("[%s] null pointer to array...!\n", __PRETTY_FUNCTION__));
30 }
31
32 Vec::Vec(const Vec& v) : Vec(v.N, v.entries) { //c++11 on...
33 #ifdef DEBUG
34   printf("[%s]\n", __PRETTY_FUNCTION__);
35 #endif
36 }
37
38 ////////////////////// destructor
39 Vec::~Vec() {
40 #ifdef DEBUG
41   printf("[%s]\n", __PRETTY_FUNCTION__);
42 #endif
43   delete [] entries;
44 }
```

```
45
46  ////////////////////// operators
47  double Vec::operator[](int i) const {
48  #ifdef DEBUG
49    printf("[%s]\n", __PRETTY_FUNCTION__);
50  #endif
51    if (i>=N)
52      throw std::invalid_argument(Form("[%s] index out of bounds...(i=%d N=%d)!\n", __PRETTY_FUNCTION__,
          i, N));
53    return entries[i];
54  }
55
56  ////////////////////// operators
57  double& Vec::operator[](int i) {
58  #ifdef DEBUG
59    printf("[%s]\n", __PRETTY_FUNCTION__);
60  #endif
61    if (i>=N)
62      throw std::invalid_argument(Form("[%s] index out of bounds...(i=%d N=%d)!\n", __PRETTY_FUNCTION__,
          i, N));
63    return entries[i];
64  }
65
66  void Vec::operator=(const Vec& v) {
67  #ifdef DEBUG
68    printf("[%s]\n", __PRETTY_FUNCTION__);
69  #endif
70    if (this != &v) {
71      if (v.N != N) {
72        N = v.N;
73        delete [] entries;
74        entries = new double[N];
75      }
76      std::copy(v.entries, v.entries+N, entries);
77    }
78  }
79
80  const Vec& Vec::operator+= (const Vec& v) {
81  #ifdef DEBUG
82    printf("[%s]\n", __PRETTY_FUNCTION__ );
83  #endif
84    if (v.N != N) {
85      throw std::invalid_argument(Form("[%s] objects with different size...(N=%d v.N=%d)!\n",
          __PRETTY_FUNCTION__, N, v.N));
86    }
87    for (int i=0; i<N; ++i) {
88      entries[i] += v[i];
89    }
90    return *this;
91  }
92
93  Vec Vec::operator*(double x) const {
94  #ifdef DEBUG
95    printf("[%s]\n", __PRETTY_FUNCTION__ );
96  #endif
97    if (abs(x-1.)<1E-9)
98      return *this;
99    double a[N];
100   for (int i=0; i<N; ++i) {
101     a[i] = entries[i] * x;
102   }
103   return Vec(N, a);
104 }
105
```

```
106
107 ////////////////////// friend methods
108
109 std::ostream& operator<<(std::ostream& s, const Vec& v) {
110   s << "[";
111   for (int i=0; i<v.N; ++i) {
112     s << std::fixed << std::setprecision(6) << v.entries[i];
113     if (i<v.N-1) s << ", ";
114   }
115   s << "]";
116   return s;
117 }
```

test program: tVec.C

location: main/tVec.C

```
1  #include "Vec.h"
2
3  #include <iostream>
4
5  int main() {
6    std::cout << "============ throw" << std::endl;
7    //  Vec v0(-1); // uncomment if you want to see throw
8
9    std::cout << "============ constructors" << std::endl;
10   double a[] = {1,2,3,4,5,6,7,8,9};
11   Vec v1(9,1);
12   std::cout << v1 << std::endl;
13   Vec v2(9,a);
14   std::cout << v2 << std::endl;
15   Vec v3(v2);
16   std::cout << v3 << std::endl;
17
18   std::cout << "============ operator[]" << std::endl;
19   std::cout << v2[3] << std::endl;
20   v2[3] = 33.;
21   std::cout << v2[3] << std::endl;
22
23   std::cout << "============ operator*" << std::endl;
24   Vec v4 = v2*10;
25   std::cout << v4 << std::endl;
26 }
```

9ª aula de problemas

Sistemas lineares (continuação)

Na 9ª aula continuámos a desenvolver a classe **Vec** nomeadamente implementando novos métodos.

classe Vec

> Introduzimos uma protecção no caso do objecto `Vec` ser construído com `N=0` elementos, inicializando o `pointer entries` a `nullptr`

> O `delete` do array `entries` deve ser feito após testar se o ponteiro não é nulo

> Implementámos os seguintes métodos da classe `Vec`:
> - produto de dois vectores necessário na multiplicação de matrizes: `double dot(const Vec&)`
> - troca de dois elementos do array: `double swap(int,int)`
> - dimensão de Vec: `int size()const`
> - soma dos valores absolutos de Vec (necessários para o cálculo da norma da matriz): `double sumAbs()`

classe FCmatrixT

> Iniciámos a construção da classe de teste **FCmatrixT** na qual se podem inspirar para implementação das classes **FCmatrix** e **FCmatrixFull**. Procedemos à implementação dos seguintes métodos:
> - default constructor: `FCmatrixT()`
> - constructor: `FCmatrixT(double**, int, int)`
> - constructor: `FCmatrixT(double*, int, int)`
> - constructor: `FCmatrixT(const vector<Vec>&)`
> - copy constructor: `FCmatrixT(const FCmatrixT&)`
> - friend operador«: `friend ostream& operator<<(ostream&, const FCmatrixT&)`

> Os programas de teste das duas classes **Vec** e **FCmatrixT** apresentam-se abaixo.

classe Vec

header: Vec.h
location: src/Vec.h

```
1  #ifndef __Vec__
2  #define __Vec__
3
4  #include <iostream> // ostream
5
6  class Vec {
7
8   public:
9
10   (...)
11
12   // additional methods
13   double dot(const Vec&); // double result = a.dot(b)
14   void swap(int, int);
15   int size() const;
16   double sumAbs();
```

```
17
18    // friend methods
19    friend std::ostream& operator<<(std::ostream&, const Vec&);
20    //  friend Vec operator*(double, const Vec&); // 5*A
21
22   private:
23    int N; // number of elements
24    double *entries; // array
25
26  };
27  #endif
```

source: Vec.C

location: src/Vec.C

```
1   Vec::Vec(int i, double x) : N(i) { // default constructor
2   #ifdef DEBUG
3     printf("[%s]\n", __PRETTY_FUNCTION__);
4   #endif
5     if (N<0) throw std::invalid_argument(Form("[%s] received negative number of elements...!\n",
          __PRETTY_FUNCTION__));
6     if (N==0) {
7       entries = nullptr;
8     } else {
9       entries = new double[N];
10      std::fill_n(entries, N, x);
11    }
12  }
13
14  Vec::~Vec() {
15  #ifdef DEBUG
16    printf("[%s]\n", __PRETTY_FUNCTION__);
17  #endif
18    if (entries) delete [] entries;
19  }
20
21  ////////////////////// additional methods
22
23  /*
24  Atenção:
25  O tipo de valor inicial no método inner_product é importante ser definido inequivocamente.
26  Neste caso, pretendemos trabalhar com `double`, daí colocar `0.`. Poderíamos ainda fazer `double(0)`
27  */
28  double Vec::dot(const Vec& v) {
29    if (v.N != N)
30      throw std::invalid_argument(Form("[%s] objects with different size...(N=%d v.N=%d)!\n",
          __PRETTY_FUNCTION__, N, v.N));
31    return std::inner_product(entries, entries+N, v.entries, double(0.);
32  }
33
34  void Vec::swap(int i, int j) {
35    if (std::max(i,j)>=N)
36      throw std::invalid_argument(Form("[%s] indices out of range...(N=%d max index=%d)!\n",
          __PRETTY_FUNCTION__, N, std::max(i,j)));
37    if (i!=j) std::swap(entries[i], entries[j]);
38  }
39
40  double Vec::sumAbs() {
41    // summ of all absolute values
42    return std::accumulate(entries, entries+N, 0, [](double accum, double x){return accum+fabs(x);});
43  }
44
45  int Vec::size() const {
```

```
  return N;
}
```

testing Vec: tVec.C

location: main/tVec.C

```
  (..)

  std::cout << "============ dot()" << std::endl;
  auto result = v2.dot(v4);
  std::cout << "result=" << result << std::endl;

  std::cout << "============ swap(int, int)" << std::endl;
  cout << v2 << endl;
  v2.swap(1,7);
  cout << v2 << endl;

  std::cout << "============ sumAbs()" << std::endl;
  cout << v2.sumAbs() << endl;
```

classe FCmatrixT

header: FCmatrix.h

location: src/FCmatrix.h

```
#ifndef __FCmatrixT__
#define __FCmatrixT__

#include "Vec.h"
#include <vector>
#include <iostream>
using namespace std;


class FCmatrixT {

 public:

  // constructors
  FCmatrixT();
  FCmatrixT(double**, int, int); // rows, columns
  FCmatrixT(double*, int, int); // rows, columns
  FCmatrixT(const vector<Vec>&);
  FCmatrixT(const FCmatrixT&);

  // methods
  int GetRowN() const; // nb of rows
  int GetColN() const; // nb of columns

  // friend methods
  friend   ostream& operator<< (ostream& , const FCmatrixT& );


 private:
  // nb of rows = vector.size()
  // nb of columns = Vec.size()
  vector<Vec> M;

};

#endif
```

source: FCmatrixT.C

location: src/FCmatrixT.C

```
#include "FCmatrixT.h"
#include <stdexcept>
using namespace std;

#include <cstdio>

#include "TROOT.h"

FCmatrixT::FCmatrixT() {
  printf("[%s] \n", __PRETTY_FUNCTION__);
}

FCmatrixT::FCmatrixT(double** a , int m, int n) {
  for (int i=0; i<m; ++i) {
    M.emplace_back(n, a[i]);
  }
}

FCmatrixT::FCmatrixT(double* a , int m, int n) {
  if (!a)
     throw std::invalid_argument(Form("[%s] null pointer...!\n", __PRETTY_FUNCTION__));
  for (int i=0; i<m; ++i) {
    M.emplace_back(n, &a[i*n]);
  }
}

FCmatrixT::FCmatrixT(const vector<Vec>& v) {
  for (int i=0; i<v.size(); ++i) {
    M.emplace_back(v[i]);
  }
}

FCmatrixT::FCmatrixT(const FCmatrixT& matrix) {
  for (int i=0; i<matrix.GetRowN(); ++i) {
    M.emplace_back(matrix.M[i]);
  }
}

///////////////// methods

int FCmatrixT::GetRowN() const {
  return M.size();
}

int FCmatrixT::GetColN() const {
  return M[0].size();
}

// friend methods

ostream& operator<< (ostream& s, const FCmatrixT& matrix) {
 s << "matrix: [\n";
  for (int i=0; i<matrix.GetRowN(); ++i) {
    s << "           " << matrix.M[i] << "\n";
  }
 s << "        ]";
 return s;
}
```

testing FCmatrixT: tFCmatrixT.C

location: main/tFCmatrixT.C

```
#include "FCmatrixT.h"
#include <iostream>
using namespace std;

int main() {

  cout << "============ default constructor" << endl;
  FCmatrixT Ma;
  cout << Ma << endl;

  cout << "============ constructor double** " << endl;
  int m=5; // nb of rows
  int n=4;
  double **a = new double*[m];
  a[0] = new double[n] {1,2,3,4};
  a[1] = new double[n] {11,12,13,14};
  a[2] = new double[n] {21,22,23,24};
  a[3] = new double[n] {31,32,33,34};
  a[4] = new double[n] {41,42,43,44};
  FCmatrixT Mb(a, m, n);
  cout << Mb << endl;

  cout << "============ constructor double* " << endl;
  m=3; // nb of rows
  n=3;
  double *c = new double[m*n] {1,2,3,4,5,6,7,8,9};
  FCmatrixT Mc(c, m, n);
  cout << Mc << endl;

  cout << "============ copy constructor " << endl;
  FCmatrixT Md(Mc);
  cout << Md << endl;

}
```

10ª aula de problemas

Sistemas lineares (continuação)

> Na 10ª aula prosseguimos com o desenvolvimento das classes necessárias à solução dos sistemas lineares, nomeadamente:

classe Vec

> Implementámos os seguintes métodos da classe **Vec**:
> - `double maxAbs()` : determinação do valor máximo absoluto de um objecto `Vec` que será útil para determinar o factor de escala linha a linha

classe FCmatrixT

> Procedemos à implementação dos seguintes métodos:
> - `void operator=(const FCmatrix&)` : permite igualar duas matrizes ainda que de tamanho diferente
> - `Vec& operator[](int)` : permite aceder à linha da matriz

classe FCmatrixAlgorithm

> Nesta classe implementamos métodos de redução de matrizes que permitam a resolução dos sistemas lineares.
> Definimos estes métodos como estáticos, isto é, são métodos que pertencem à classe e não a objectos da classe.
> Poderíamos em alternativa ter definido estes métodos utilizando `namespace` .
> - `static void GaussElimination(FCmatrixT&)` : método de eliminação de Gauss
> - `static void GaussEliminationPivot(FCmatrixT&)` : método de eliminação de Gauss com recuros a `pivoting`
> - `static void LUdecomposition(FCmatrixT&)` : método de decomposição LU

classe EqSolver

> A classe `EqSolver` irá ser utlizada para a resolução dos sistemas lineares.
> Poderá ter os seguintes métodos:
> - `Vec GaussEliminationSolver()`
> - `Vec GaussEliminationPivotSolver()`
> - `Vec LUdecompositionSolver()`
> - `Vec JacobiIterativeSolver`
> - `Vec GaussSeidelIterativeSolver`

classe Vec

header: Vec.h
location: src/Vec.h

```
1  double maxAbs(); // maximal abs(value) of Vec
```

source: Vec.C
location: src/Vec.C

> O método que se segue implementa a procura do elemento de `Vec` máximo usando os valores absolutos. Usaremos para isso o método `max_element` pertencente a `<algorithm>` da `STL library`.
> Este método retorna o iterador para elemento máximo (sendo um iterador, temos que desreferenciá-lo para ter acesso ao valor do elemento... daí o uso do *). Como se pretende usar o valor absoluto, há necessidade de proceder à implementação de uma função (neste caso lambda) dita `binary predicate` (binária porque possui duplo argumento e `predicate` porque retorna um booleano) que indica se o primeiro argumento é menor que o segundo (neste caso os valores absolutos).

```
double Vec::maxAbs() {
  // max of elements (absolute value)
  return *std::max_element(entries, entries+N, [](double x1, double x2){ return fabs(x1) < fabs(x2);});
}
```

classe FCmatrixT

header: FCmatrixT.h
location: src/FCmatrixT.h

```
 void operator=(const FCmatrixT&);
  Vec& operator[](int);
```

source: FCmatrixT.C
location: src/FCmatrixT.C

> A implementação do método `operator=` permite adaptar (redinir ou antes `morphing`) a matriz **M** (membro da classe FCmatrixT) a a uma outra qualquer matriz que se iguale. Possuímos assim uma ferramenta fundamental para transformarmos um objecto `FCmatrixT` noutro.

> A definição do `operator[]` com retorno de uma referência para `Vec`, permite acedermos directamente à linha (row) da matriz para lê-la (como `Vec`) ou para modificá-la. Torna-se também possível aceder directamente a um qualquer elemento da matriz usando os parentesis rectos `M[i][j]`.

```
void FCmatrixT::operator= (const FCmatrixT& matrix) {
  if (this != &matrix) {
    M.clear();
    for (int i=0; i< matrix.M.size(); ++i) {
      M.push_back(matrix.M[i]);
    }
  }
}

Vec& FCmatrixT::operator[] (int i) {
  return M[i];
}
```

classe FCmatrixAlgorithm

> A classe `FCmatrixAlgorithm` é uma classe repositório de métodos que podem eventualmente ser usados directamente pela classe `FCmatrixT` ou `EqSolver`.
> **Nota:** a classe `FCmatrixT` recordo foi aqui usada de forma experimental como exemplo para a implementação que farão das classes `FCmatrix`, `FCmatrixFull`, ...

header: FCmatrixAlgorithm.h

location: src/FCmatrixAlgorithm.h

```
  /*
    Gauss Elimination
    - it can receive a simple matrix
    - linear system: augmented matrix (A | b)
  */

  static void GaussElimination(FCmatrixT&);

  /*
    Gauss Elimination with pivoting
    - it can receive a simple matrix
    - linear system: augmented matrix (A | b)
    - return: (A | b | Index)
  */

  static void GaussEliminationPivot(FCmatrixT&);

  static void LUdecomposition(FCmatrixT&);
```

source: FCmatrixAlgorithm.C

location: src/FCmatrixAlgorithm.C

> O exemplo de implementação do método de eliminação de Gauss que aqui se mostra visa antes de mais a demonstração de forma simples e pedagógica (mas correcta) que o uso dos operadores `operator*`, `operator+=` definidos na classe `Vec` torna muito simples o algoritmo de redução da matriz. No entanto, em termos de eficácia (somente notável se tivermos matrizes muito grandes), na redução de cada linha da matriz através da soma com a linha pivot (i), faz-se uma subtração de elementos de $[i][J = 0, ..., i]$ que seria dispensável uma vez que estes tornam-se nulos (na fase de debug do algoritmo é fundamental verificar que se tornam nulos!)
> **Nota:** No algoritmo de redução simples das matrizes há que verificar se o elemento pivot é nulo, o que daria nesse caso uma divisão por zero e a geração de uma excepção (crash!).

```
void FCmatrixAlgorithm::GaussElimination(FCmatrixT& MR) {
  printf("[%s]\n", __PRETTY_FUNCTION__);
  for (int i=0; i<MR.GetRowN()-1; ++i) {
    if (MR[i][i] == 0) {
      // swap lines: think about...
    }
    // not the most effective
    for (int j=i+1; j<MR.GetRowN(); ++j) {
      double m = MR[j][i]/MR[i][i];
      MR[j] += MR[i]*(-m);
    }
  }
}
```

programa teste: tFCmatrixAlgorithm.C

location: main/tFCmatrixAlgorithm.C

```
#include "FCmatrixT.h"
#include "FCmatrixAlgorithm.h"

int main() {

  // create matrix 3x3
  const int n = 3;
```

```
  double *a[n] {new double[n] {12, -2, 3} ,  new double[n] { -2, 15., 6} , new double[n] {1, 6, 20}};
  FCmatrixT M1(a, n, n);
  cout << M1 << endl; // print matrix

  // make matrix reduction using Gauss elimination
  FCmatrixAlgorithm::GaussElimination(M1);
  cout << M1 << endl; // print reduced matrix
}
```

classe EqSolver

header: EqSolver.h

location: src/EqSolver.h

```
#ifndef __EqSolver__
#define __EqSolver__

#include "FCmatrixT.h"
#include "Vec.h"

class EqSolver {

 public:
  EqSolver();
  EqSolver(const FCmatrixT&, const Vec&);

  // solvers
  Vec GaussEliminationSolver();

 private:
  FCmatrixT Mcoeff;
  Vec Vconst;
};
```

header: EqSolver.C

location: src/EqSolver.C

```
include "EqSolver.h"
#include <cstdio>

EqSolver::EqSolver() {
    printf("[%s]\n", __PRETTY_FUNCTION__);
}

EqSolver::EqSolver(const FCmatrixT& matrix, const Vec& v) {
    printf("[%s]\n", __PRETTY_FUNCTION__);
    Mcoeff = matrix;
    Vconst = v;
}

Vec EqSolver::GaussEliminationSolver() {

  // get reduced matrix (FCmatrixAlgorithm)

  // back substitution to get solution

  return Vec();
}
```

11ª aula de problemas

Interpolação

Na 11ª aula iremos desenvolver as classes associadas à interpolação `DataPoints`` e`LagrangeInterpolator‘ :

DataPoints Esta classe funciona como a classe base que armazena a informação dos pontos a interpolar

LagrangeInterpolator Esta classe herda da classe `DataPoints` e é onde se realiza a interpolação de Lagrange

classe DataPoints

header: DataPoints.h
location: src/DataPoints.h

```
#include "TGraph.h"

#include <iostream>

class DataPoints {

 public:
  // constructors and destructor
  DataPoints();
  DataPoints(unsigned int, const double*, const double*);
  virtual ~DataPoints();

  // graphics
  virtual void Draw() const;
  const TGraph& GetGraph() const;

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

 protected:
  int N; // number of points
  double *x, *y; // data arrays

  TGraph *gPoints;

  double xmin, xmax;
  double ymin, ymax;

  void SetMinMaxX();
  void SetMinMaxY();

};
```

source: DataPoints.C
location: src/DataPoints.C

```
#include "DataPoints.h"

#include <stdexcept>
#include <algorithm>
#include <iomanip> // setprecision()

#include "TROOT.h"
#include "TAxis.h"
```

```
9
10 ////////////////////////////////////////////////////////////////////////
11
12 DataPoints::DataPoints() :
13 N(0), // nb of elements = 0
14 x(nullptr), y(nullptr), // set null pointers
15   gPoints(nullptr) {};
16
17 DataPoints::DataPoints(unsigned int fN, const double* fx, const double* fy) : N(fN), x(new double[N]),
       y(new double[N]) {
18   // manage wrong args
19   if (!fx || !fy)
20     throw std::invalid_argument(Form("[%s] null arrays!!!", __PRETTY_FUNCTION__));
21
22   // copy arrays
23   std::copy(fx, fx+N, x);
24   std::copy(fy, fy+N, y);
25
26   //retrieve min and max values
27   SetMinMaxX();
28   SetMinMaxY();
29
30   // create graph
31   gPoints = new TGraph(N, x, y);
32   gPoints->SetMarkerStyle(20);
33   gPoints->SetMarkerColor(kRed+2);
34   gPoints->SetMarkerSize(1.5);
35
36   gPoints->GetXaxis()->SetRangeUser(0.9*xmin, 1.1*xmax);
37   gPoints->GetYaxis()->SetRangeUser(0.9*ymin, 1.1*ymax);
38 }
39
40 DataPoints::~DataPoints() {
41   if (x) delete [] x;
42   if (y) delete [] y;
43   delete gPoints;
44 }
45
46 ////////////////////////////////////////// graphics
47
48 void DataPoints::Draw() const {
49   gPoints->Draw("AP");
50 }
51
52 const TGraph& DataPoints::GetGraph() const {
53   return *gPoints;
54 }
55
56 void DataPoints::SetMinMaxX() {
57   auto it = std::minmax_element(x, x+N);
58   xmin = *it.first;
59   xmax = *it.second;
60 }
61
62 void DataPoints::SetMinMaxY() {
63   auto it = std::minmax_element(y, y+N);
64   ymin = *it.first;
65   ymax = *it.second;
66 }
67
68 ////////////////////////////////////////// output
69
70 std::ostream& operator<< (std::ostream& s, const DataPoints& D) {
71   s << "Nb points stored: " << D.N << std::endl;
```

```
  for (int i=0; i<D.N; ++i) {
    s << std::fixed << std::setprecision(3)
      << "(" << D.x[i] << "," << D.y[i] << ")" ;
  }
  return s;
}
```

classe LagrangeInterpolator

header: LagrangeInterpolator.h

location: src/LagrangeInterpolator.h

```
//////////////////////////////////////////////////////////

class LagrangeInterpolator : public DataPoints {

 public:
  // constructors and destructor
  LagrangeInterpolator(unsigned int=0, const double* x=nullptr, const double* y=nullptr, const TF1*
      fF0=nullptr);
  ~LagrangeInterpolator();

  // copy constructor
  LagrangeInterpolator(const LagrangeInterpolator&);

  // interpolation methods
  double Interpolate(double) const;
  const TF1& GetInterpolationFunction() const { return *FInterpolator; }
  void SetResolution(int n=200) const { FInterpolator->SetNpx(n); }
  void Draw() const;
  const TCanvas& GetCanvas();

  // underlying function
  void SetFunction(const TF1*);

  // output
  friend std::ostream& operator<< (std::ostream&, const LagrangeInterpolator&);

 protected:
  TF1* F0; // underlying function
  TF1* FInterpolator; // interpolator function
  TCanvas* cInterpolator;

  double fInterpolator(double *fx, double *par)  {
    return Interpolate(fx[0]);
  }

};
```

source: LagrangeInterpolator.C

location: src/LagrangeInterpolator.C

```
LagrangeInterpolator::LagrangeInterpolator(unsigned int fN, const double* fx, const double* fy, const
    TF1* fF0) :
DataPoints(fN, fx, fy),
F0(nullptr),
cInterpolator(nullptr) {
  if (fF0) F0 = new TF1(*fF0);
  FInterpolator = new TF1("FInterpolator", this, &LagrangeInterpolator::fInterpolator, xmin, xmax, 0);
}

```

```
LagrangeInterpolator::~LagrangeInterpolator() {
  if (FInterpolator) delete FInterpolator;
  if (F0) delete F0;
  if (cInterpolator) delete cInterpolator;
}

LagrangeInterpolator::LagrangeInterpolator(const LagrangeInterpolator& LI) :
LagrangeInterpolator(LI.N, LI.x, LI.y, LI.F0) {;}

////////////////////////////////////////// interpolator methods

double LagrangeInterpolator::Interpolate(double xval) const {
  double result = 0.;
  for (int i=0; i<N; ++i) {
    double lx = 1.;
    for (int k=0; k<N; ++k)
      if (i!=k) lx *= (xval - x[k])/(x[i] - x[k]);
    result += y[i]*lx;
  }
  return result;
}

void LagrangeInterpolator::Draw() const {
  DataPoints::Draw();
  FInterpolator->SetLineColor(38);
  FInterpolator->SetLineWidth(4);
  FInterpolator->Draw("same");
}

const TCanvas& LagrangeInterpolator::GetCanvas()  {
  cInterpolator = new TCanvas("cInterpolator","", 0,0,800,600);
  DataPoints::Draw();
  FInterpolator->SetLineColor(38);
  FInterpolator->SetLineWidth(4);
  FInterpolator->Draw("same");
  return *cInterpolator;
}
/////////////////////////////////////// underlying func

void LagrangeInterpolator::SetFunction(const TF1* fF0) {
  if (fF0) F0 = new TF1(*fF0);
}

/////////////////////////////////////// output

std::ostream& operator<< (std::ostream& s, const LagrangeInterpolator& LI) {
  s << "Lagrange Interpolator " << "x:[" << LI.xmin << "," << LI.xmax << "]" << std::endl;
  for (int i=0; i<LI.N; ++i) {
    s << std::fixed << std::setprecision(3)
      << "(" << LI.x[i] << "," << LI.y[i] << ") f(x)=" << LI.Interpolate(LI.x[i]) << " " ;
  }
  s << "\n";
  return s;
}
```

programa teste do LagrangeInterpolator

> Neste programa testa-se a interpolação de um conjunto de 10 pontos usando o método de Lagrange

> Faz-se uso também do objecto `TApplication` de ROOT que permite a visualização de gráficos directemente no programa principal

location: main/tLagrangeInterpolator.C

```
#include "LagrangeInterpolator.h"
#include "Vec.h"

#include <cmath>
#include "TMath.h"
#include "TApplication.h"
#include "TCanvas.h"

#include <iostream>
using namespace std;

int main() {

  auto f = [](double x) { return sin(TMath::TwoPi()*x) + exp(x) ; };

  // sampling 10 points
  int N = 10;
  Vec vx(N), vy(N);
  double step = 1./N;
  for (int i=0; i<10; ++i) {
    vx[i] = i*step;
    vy[i] = f(vx[i]);
  }
  cout << vx << endl;

  // lagrange interpolator
  LagrangeInterpolator L((unsigned int)vx.size(), vx.data(), vy.data(), nullptr);
  cout << L << endl;

  // graph
  TApplication tapp("app", 0, 0);
  TCanvas *cc = new TCanvas("cc", "", 0,0,1000,800);
  L.Draw();
  cc->Modified();
  cc->Update();
  tapp.Run();

}
```

12ª aula de problemas

Integração numérica

Na 12ª aula iremos desenvolver as classes associadas à integração numérica `Func1D` e `Integrator`:

Func1D Esta classe funciona como a classe base que guarda a função (usando um objecto TF1 internamente)

Integrator Esta classe herda da classe `Func1D` e implementa os métodos integradores (trapezoidal, simpson, adaptativos, Romberg)

classe Func1D

header: Func1D.h

location: src/Func1D.h

```
class Func1D {

 public:
  // constructor, destructor
  Func1D(const TF1* fp=nullptr);
  Func1D(const TF1&);
  virtual ~Func1D();

  // drawing
  void Draw() const;

  // evaluate
  double Eval(double) const;

 protected:

  void Settings();

  TF1* f;

};
```

source: Func1D.C

location: src/Func1D.C

```
////////////////// constructors, destructor

Func1D::Func1D(const TF1* fp) : f(nullptr) {
  if (fp) {
    f = new TF1(*fp);
    Settings();
  }
}

Func1D::Func1D(const TF1& fp) : Func1D(&fp) {};

Func1D::~Func1D() {
  if (f) delete f;
}


////////////////// methods

void Func1D::Settings() {
```

```
  f->SetNpx(1000);
  f->SetLineColor(38);
  f->SetLineWidth(4);
}

double Func1D::Eval(double xval) const {
  return f->Eval(xval);
}

void Func1D::Draw() const {
  TApplication A("A",0,0);
  TCanvas c("c", "Func1D canvas", 0, 0, 1000, 800);
  f->Draw();
  c.Update();
  A.Run();
}
```

classe Integrator

header: Integrator.h

location: src/Integrator.h

```
class Integrator: public Func1D {

 public:

  // construtors and destructor
  Integrator(double fx0=0., double fx1=1., const TF1 *fp=nullptr) : x0(fx0), x1(fx1), Func1D(fp) {};
  Integrator(double fx0, double fx1, const TF1& fp) : Integrator(fx0, fx1, &fp) {};
  ~Integrator() = default;

  // integrator methods
  /*
    n ............. number of slices (input)
    Integral ...... integral value by reference (input/output)
    error ......... error value by reference (input/output)
   */
  void Trapezoidal(int n, double& Integral, double& Error);
  void TrapezoidalAdaptive(double& Integral, double& Error);
  void Simpson(int n, double& Integral, double& Error);
  void Romberg(...);

protected:
  double x0; // function range for integration
  double x1;
};
```

programa teste da classe Func1D

source: tFunc1D.C

location: main/tFunc1D.C

```
int main() {

  // instantiate object Func1D
  auto f = [](double *x, double *par=nullptr)
    {
      return sin(x[0])/x[0] + 0.5*cos(0.5*x[0]);
    };

  Func1D F1(new TF1("F1",f, 0.1, 10., 0));
```

```
  Func1D F2(TF1("F2",f, 0.1, 10., 0));

  // output
  for (double a=0.1; a<1; a+=0.1) {
    cout << a << " " << f(&a) << " " << F1.Eval(a) << " " << F2.Eval(a) << endl;
  }

  // drawing
  F2.Draw();

}
```

13ª aula de problemas

Integração por método de monte-carlo

Na 13ª aula e última aula de problemas iremos desenvolver a classe associada à integração por monte-carlo `IntegratorMC`.

IntegratorMC Neste exemplo optei por fazer a classe `IntegratorMC` herdar da classe `Integrator` que por sua vez herda de `Func1D`.

classe IntegratorMC

header: IntegratorMC.h

location: src/IntegratorMC.h

```
#ifndef __IntegratorMC__
#define __IntegratorMC__

#include "Integrator.h"
#include "TF1.h"

class IntegratorMC : public Integrator {

 public:

  // function to be integrated: f
  IntegratorMC() = default;
  IntegratorMC(const TF1& f) : Integrator(f.GetXmin(), f.GetXmax(), f) {};
  ~IntegratorMC() = default;

  // integration methods
  void ImportanceSampling(int& N, double& value, double& error, const TF1& px, const TF1& xy);

  // additional methods
  /*
  It will be useful to have an additional method for having access to the distribution
  of randoms generated according to a given TF1:
  the idea is to be able to check what distribution is being generated
  note: just to be used for testing because ROOT display will stop program run
  */

  static void  RandomGen(TF1& px, TF1& xy);

};
#endif
```

header: IntegratorMC.C Implementation of the importance sampling method. Our method will receive the number of random (**N**) to generate, and eventually the result precision (**error**) and the integral will be returned on **value**.
If **error** is passed to method as 0., it means that the integral error will be dictated by the number of randoms passed to the method and pdf(x) shape. Otherwise, the number of randoms will be defined by the error passed to the method.

location: src/IntegratorMC.C

```
#include "IntegratorMC.h"

#include "TRandom.h"
```

```
4  #include "TF1.h"
5  #include "TH1.h"
6  #include "TApplication.h"
7  #include "TCanvas.h"
8
9  #include <cstdio>
10 #include <cmath>
11
12 void IntegratorMC::ImportanceSampling(int& N, double& value, double& error, const TF1& px, const TF1&
       xy) {
13
14   // check if N is valid or error
15   bool bN = true;
16   if (error >0.) {
17     N=0;
18     bN = false;
19   }
20
21   // integral
22   double Fsum = 0.;
23   double Fsum2 = 0.;
24
25   int count = 0;
26   double error_t = 1.;
27   gRandom->SetSeed(0);
28   while ( (bN && count < N) || ( !bN && error_t < error ) ) {
29
30     // check pdf normalization
31     if (count ==0) {
32       TF1 ftmp(px);
33       printf("integral pdf [%f, %f]= %f \n", x0, x1, ftmp.Integral(x0, x1));
34     }
35
36     // generate uniform random y
37     double y = gRandom->Uniform(); // [0,1]
38     double x = xy.Eval(y);
39
40     // compute function ratio
41     double ratiof = Eval(x)/px.Eval(x);
42
43     // integral
44     Fsum += ratiof;
45     Fsum2 += ratiof*ratiof;
46
47     // count
48     count++;
49
50     // error variance: variance = <f^2> - <f>^2
51     double Fmean = Fsum/count;
52     double Fmean2 = Fsum2/count;
53     double variance = Fmean2 - Fmean*Fmean;
54     error_t = sqrt(variance/count);
55
56   }
57   N = count;
58   error = error_t;
59   value = Fsum/N;
60
61   printf("N=%d value=%f error=%f \n", N, value, error);
62 }
63
64
65 void IntegratorMC::RandomGen(TF1& px, TF1& xy) {
66   // draw x random
```

```
  TH1F H("H", "random generated from xy", 120, -60, 60);
  TApplication A("A",0,0);
  gRandom->SetSeed(0);
  for (int i=0; i<1000; ++i) {
    double r = gRandom->Uniform(); // [0,1]
    H.Fill( xy.Eval(r) );
  }

  // scale histogram to integral=1
  H.Scale(1./H.Integral("width"));

  // display histogram and pdf
  TCanvas c("c","", 0, 0, 600,600);
  H.SetLineColor(38);
  H.SetLineWidth(4);
  H.Draw("HISTO");
  px.SetLineColor(20);
  px.SetLineWidth(4);
  px.Draw("same");
  c.Update();
  A.Run();

}
```

programa teste da classe IntegratorMC

```
location: main/tIntegratorMC.C
```

```
#include "IntegratorMC.h"

#include "TMath.h"
#include "TF1.h"
#include "TApplication.h"
#include "TH1.h"
#include "TCanvas.h"
#include "TRandom.h"

#include <cmath>

#include <iostream>

int main() {

  // integration limits
  double xlow = -50.;
  double xup = 50.;

  // integrand
  auto g = [](double* x, double* par) {
    return 1./sqrt(2*TMath::Pi())*exp(-0.5*x[0]*x[0]);
  };
  TF1 G("G", g, xlow, xup, 0); // npar=0, ndim=1 (default)
  G.SetNpx(1000);

  // auxiliary functions

  // ... uniform pdf (normalized): $\int_{x_1}^{x_2} k dx = 1 \Rightarrow k = 1/(x_2 - x_1)$

  auto p = [xlow, xup](double* x, double* par) {
    return 1./(xup-xlow);
  };
```

```
34    TF1 P("P", p, xlow, xup, 0); // npar=0, ndim=1 (default)
35    P.SetNpx(1000);
36
37    // ... pdf cumulative and inverted: $y = \int_{x_1}^{x} p(x) dx  = \frac{x - x_1}{x_2 - x_1}
          \Rightarrow x = x_1 + y * [x_2 - x_1]$
38
39    auto pi = [xlow, xup](double* x, double* par) {
40      return xlow + x[0]*(xup-xlow);
41    };
42    TF1 PI("PI", pi, xlow, xup, 0); // npar=0, ndim=1 (default)
43    PI.SetNpx(1000);
44
45    // check randoms from pdf (note: ROOT will stop here after this call; just check distribution and take
          it out)
46
47    IntegratorMC::RandomGen(P,PI);
48
49    // integrator
50
51    TH1F HI("HI", "uniform integral", 100, 0, 2);
52    IntegratorMC I(G);
53    for (int i=0; i<2000; ++i) {
54      int N=1000;
55      double value =0.;
56      double error =0.;
57      I.ImportanceSampling(N, value, error, P, PI);
58      HI.Fill(value);
59    }
60
61    TApplication A("A",0,0);
62    TCanvas c("c","", 0, 0, 600,600);
63    HI.SetLineColor(38);
64    HI.SetLineWidth(4);
65    HI.Draw("HISTO");
66    c.Update();
67    A.Run();
68  }
```