POLYTECH NICE-SOPHIA

# Lab#2: Basic Class Design

Objectives for this lab:

- Using a external library to facilitate the resolution of problem.
- Design and code a complete class
- Using multiple files in a project

# 1 - Interpolation Methods

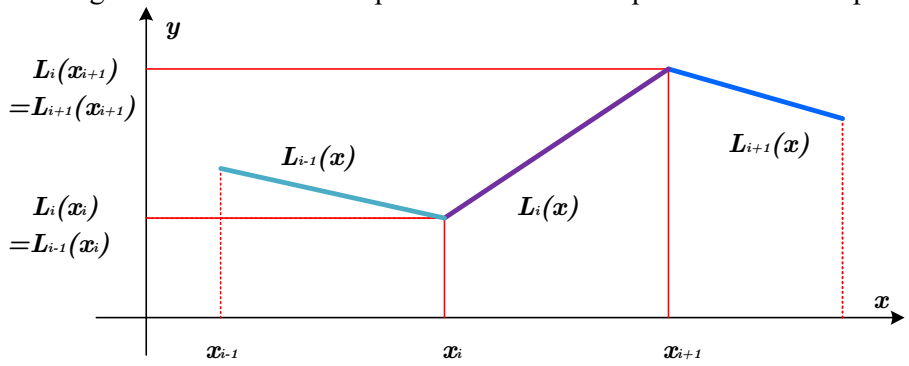
## 1.1 Piece-Wise Linear Interpolation between N Points.

Please read: https://en.wikipedia.org/wiki/Linear_interpolation

Given a set of $N+1$ points $(x_i, y_i), i = 0..N$, with $x_i < x_{i+1}$ , we need to search $N$ segments, $L_i(x)$, such that:
$L_i(x) = a_i x + b_i$, we need to compute $2N$ coefficients $\{a_i, b_i\}$ using the following equalities:

$$L_i(x_i) = y_i$$
$$L_i(x_{i+1}) = y_{i+1}$$

The image below illustrates the piecewise linear interpolation for a few points




Notice that we can find each pair $\{a_i, b_i\}$ independently from the others, so it is easier to write the equations as follows, using a coordinate change:
Let

$$\Delta_i(x) = x - x_i$$

we need to find $\{a_i, b_i\}$, such that

$$D_i(x) = a_i(x - x_i) + b_i = a_i\Delta_i(x) + b_i$$

and:

$$L_i(x) = D_i(x)$$

The coordinate change allows to find:

$$b_i = y_i$$

then

$$a_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$$

### 1.1.1 Simplification and Boundary conditions

To simplify the resolution, we shall assume the following:

1) Only *N* points are given, the last point $(x_N, y_N)$ is such that:

$x_N = 1$
$y_N = y_0$

2) The $x_i$'s are always given in increasing order:

$x_i < x_{i+1}$ .

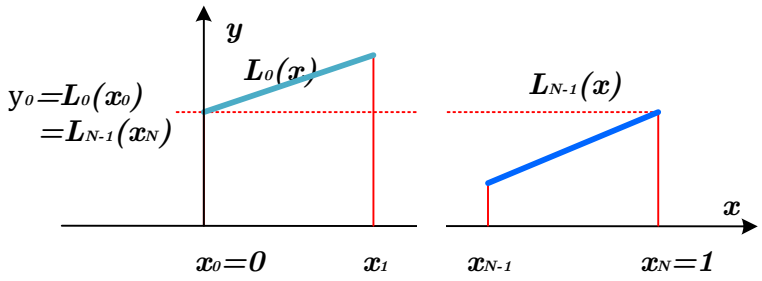
3) For the first point, we always have:

$x_0 = 0$.

4) When evaluating the function for $x > 1$ or $x < 0$:

$L(x) = y_0$ .

These boundary conditions are illustrated in the figure below:



## 1.2 Interpolation with Spline

Splines allow to draw smooth curves going through N points. Smoothness is defined by continuous first and second derivatives.
To read: https://en.wikipedia.org/wiki/Spline_interpolation
https://timodenk.com/blog/cubic-spline-interpolation

To see: http://jsxgraph.uni-bayreuth.de/wiki/index.php/Cubic_spline_interpolation

For this lab, we focus on a variant of the splines named "circular" splines.
Given a set of $N+1$ points $(x_i, y_i), i = 0..N$, with identical pre-conditions defined in Section 1.1.
We need to find $N$ 3rd degree polynomials:

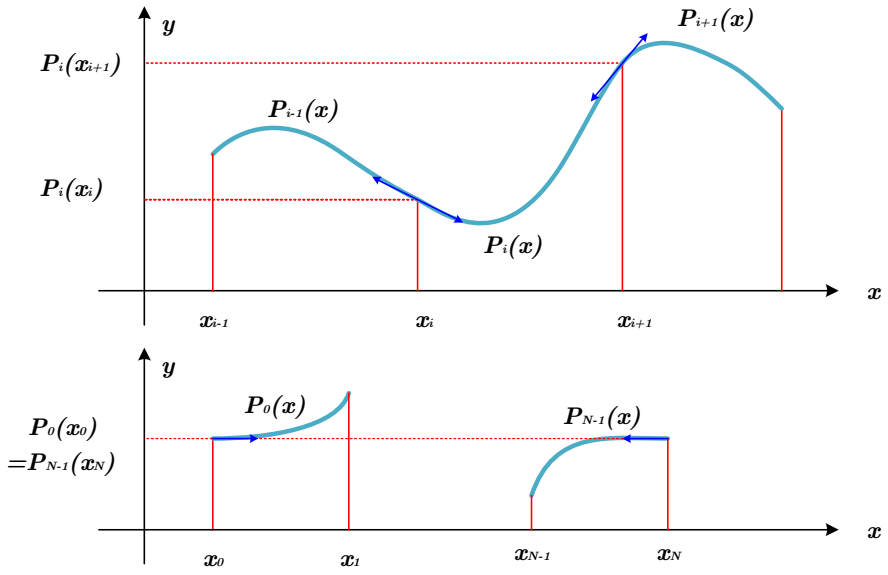
$$P_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i,$$

We need to compute $4N$ coefficients $\{a_i, b_i, c_i, d_i\}$ such that:

$P_i(x_i) = y_i$
$P_i(x_{i+1}) = y_{i+1}$
$P'_i(x_i) = P'_{i-1}(x_i)$, with $0 < i \leq N-1$, condition on the first derivative.
$P''_i(x_i) = P''_{i-1}(x_i)$, with $0 < i \leq N-1$, condition on the second derivative.

The following conditions must also be satisfied (circular splines variant)
$P_{N-1}(x_N) = y_0$
$P'_0(x_0) = 0$
$P'_{N-1}(x_N) = 0$

On the first graph, we can see continuous derivative at $x_i$. The second graph shows derivative at the boundary points $x_0$ and $x_N$.



Overall, we have $4N$ equations with $4N$ unknowns which can be solved easily with linear algebra. First, you have to initialize a matrix, $\boldsymbol{M}$, of size $4N \times 4N$, and a column vector, $\boldsymbol{v}$, of size $4N$ such that

$$\boldsymbol{M}\boldsymbol{u} = \boldsymbol{v}$$

The vector $\boldsymbol{u}$ is composed of all the unknown variables:
$\boldsymbol{u} = (a_0, b_0, c_0, d_0, a_1, b_1, \dots, a_{N-1}, b_{N-1}, c_{N-1}, d_{N-1})^{\mathrm{T}}$

With simple linear algebra, we can compute $\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{v}$.

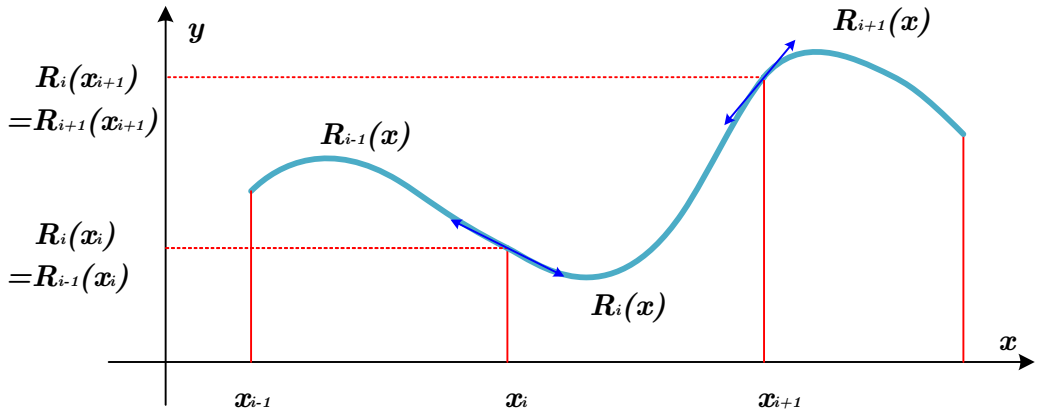
To simplify the problem, we assume the same boundary conditions of Section 1.1.1, i.e.

$x_0 = 0, x_N = 1$
$x_{i-1} < x_i$
and $P_{N-1}(x_N) = y_0$

Solving a large $4N \times 4N$ system can be difficult and may lead to numerical instability. A simple trick can be used to reduce the problem to a $3N \times 3N$ system.

$$R_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

Explain briefly how you can apply the above equation to reduce the size of the matrix to $3N \times 3N$.



# 2 - Using an external Library: Eigen

`Eigen` is an open source C++ library dedicated to linear algebra which we shall use to solve our system of $3N$ equations with $3N$ unknowns.
More information https://eigen.tuxfamily.org/index.php?title=Main_Page

## 2.1 Package Install

You shall install the latest stable release of Eigen (version ≥ 3.3.9). Please follow these instructions at the shell prompt:

```
shell> sudo mkdir -p /usr/local/include
shell> cd /usr/local/include
shell> sudo git clone https://gitlab.com/libeigen/eigen.git
shell> cd eigen
shell> sudo git checkout 3.3.9
shell> ls -d
/usr/local/include/eigen
```

The path in blue is the suggested install path, you can change it if you want to.
Please note the path in red, you will need it in the following sections.

## 2.2 Verification Step

This section will help you verify that the installation of Eigen is correctly done.

Please read the following link:
http://eigen.tuxfamily.org/dox/GettingStarted.html
In particular, the section « *Compiling and running your first program* »

Then, copy/paste the example below into your favorite code editor, compile it with g++ and run it.

```
#include <iostream>
#include <Eigen/Dense>
using namespace std;
using namespace Eigen;

int main() {
  MatrixXd ma(3,3);
  VectorXd b(3);
  ma << 1, 2, 3,  4, 5, 6,  7, 8, 10 ;
  b << 3,  3, 4;
  cout << "Here is the matrix A:\n" << ma << endl;
  cout << "Here is the vector b:\n" << b << endl;
  VectorXd x = ma.colPivHouseholderQr().solve(b);
  cout << "The solution is:\n" << x << endl;
}
```

More information on the resolution of linear systems, please check this link:
http://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html

When compiling the test program, some students see the following error message:

```
wsl shell > g++ -std=c++20 test.cpp
test.cpp:2:10: fatal error: Eigen/Dense: No such file or directory
    2 | #include <Eigen/Dense>
      |          ^~~~~~~~~~~~~
compilation terminated.
```

What must added to the compile statement to avoid such error message?

*Hint*: The red path from Section 2.1 is useful.

## 2.3 Expected Results

Once you have managed to compile and execute the test program, verify that the output matches the screen shown below.

```
wsl shell > ./a.out
Here is the matrix A:
 1  2  3
 4  5  6
 7  8 10
Here is the vector b:
3
3
4
The solution is:
-2
 1
 1
```

POLYTECH NICE-SOPHIA

# 3 - Class Design

You are now ready for the class design step. Your class must have all the necessary methods to initialize, solve a linear system and compute for a given $x$ the value $y = f(x)$.

If you are interested in Class Design, you will find large numbers of books, articles, tutorials, video on the web. A good start would be to focus on the "SOLID" principles.

## 3.1 Starter

This example can help you to get started.

```
class Spline {
 private:
  …
 public:
  …
  Spline(…) {
  …
  }

  double get_value(const double x) const {
  …
  }

};
```

## 3.2 Step by Step

### 3.2.1 Class Members

Questions to help you decide on the members for your class:

- Which values must be kept in the class? $\boldsymbol{M}, \boldsymbol{v}, \boldsymbol{u}, x_i, y_i, a_i, b_i, c_i, d_i$? Other values?
- What shall be the types for these values: Eigen Matrix, vector, map, double?
- What protection must be given to the members, if not private, why?

### 3.2.2 Constructors

You define what the constructor will do and the input parameters for the constructor.
Do you need multiple constructors? What are the types of the parameters?

### 3.2.3 Class Methods

Which methods do you need? Private or Public
Don't write accessor and mutator for each class member, it is generally a bad practice.

I have provided the `get_value()` method, you may need to add more

Q Explain what is the meaning of the red `const` key-word in my example above?

## 3.3 Validation of your Class

Please use the values below to validate your class.

```
int main(int argc, char *argv[]) {
  //
  // Important only N points are given, the point N+1 is implicitly defined
  // see section 1.1.1 on boundary condition
  //
  vector<double> xs{  0.,   0.16,   0.42,   0.6425,  0.8575};
  vector<double> ys{100., 183.  , 235.  ,  40.    , 15.    };

  Spline spline1...

  for (double x = -0.125 ; x <= 1.125 ; x += 0.125) {
    std::cout << "Value for " << x << " is " << spline1.get_value(x) << std::endl;
  }
}
```

Q What are the pre-requisites for the vectors `xs` and `ys`, if any?

Expected results:

SPLINE

```
shell> test-spline
Value for  -0.1250 is  100.000
Value for  -0.0625 is  100.000
Value for   0.0000 is  100.000
Value for   0.0625 is  117.746
Value for   0.1250 is  157.956
Value for   0.1875 is  201.291
Value for   0.2500 is  234.845
Value for   0.3125 is  253.233
Value for   0.3750 is  251.496
Value for   0.4375 is  224.744
Value for   0.5000 is  174.349
Value for   0.5625 is  112.890
Value for   0.6250 is   54.108
Value for   0.6875 is   11.065
Value for   0.7500 is   -9.706
Value for   0.8125 is   -5.325
Value for   0.8750 is   26.890
Value for   0.9375 is   74.843
Value for   1.0000 is  100.000
Value for   1.0625 is  100.000
Value for   1.1250 is  100.000
```

LINEAR

```
shell> test-linear
Value for  -0.1250 is  100.000
Value for  -0.0625 is  100.000
Value for   0.0000 is  100.000
Value for   0.0625 is  132.422
Value for   0.1250 is  164.844
Value for   0.1875 is  188.500
Value for   0.2500 is  201.000
Value for   0.3125 is  213.500
Value for   0.3750 is  226.000
Value for   0.4375 is  219.663
Value for   0.5000 is  164.888
Value for   0.5625 is  110.112
Value for   0.6250 is   55.337
Value for   0.6875 is   34.767
Value for   0.7500 is   27.500
Value for   0.8125 is   20.233
Value for   0.8750 is   25.439
Value for   0.9375 is   62.719
Value for   1.0000 is  100.000
Value for   1.0625 is  100.000
Value for   1.1250 is  100.000
```

If you have different values, your code does not work!

## 3.4 Code Development

It is highly recommended to code and validate the piece-wise linear interpolation first. Once you have obtained the correct results you can move on to the spline version.

Overall, for the piecewise interpolation, your class can be written with less than 50 lines of code. For the spline, you need less than 100 lines of code.

The code for the cubic spline cannot be written in just 10 minutes, you really need to think...

I have found these Eigen methods useful:

```
MatrixXd ma(4,4);  // constructor for a 4x4 matrix
ma(r,c) = v;       // set the matrix element at row 'r' and col 'c' to value 'v'
ma.setZero()       // clear the matrix ma;
```

*Hint*: For the spline, the matrix $\boldsymbol{M}$, is composed mainly of zeros, so you just write the non-zero elements of this matrix.

# 4 - Improvements

## 4.1 Reusable Code

At this stage, it is very likely that you have used only one file for all your source code.
To allow multiple programs to reuse the spline class, you shall split your source code in two parts: a "*dot h*" with your spline class and a "*dot cpp*" with the main function and other related non-class functions.
As an example, after the split, you now have two files:

- `main.cpp`
- `elec4_util.h`

### 4.1.1 Common Practice for "dot h" files

All the "dot h" have a common structure, please adopt the new style if your project uses CPP14 or later standard.

| Old Style | New Style |
|---|---|
| `#ifndef ELEC4_UTIL_H_`<br>`#define ELEC4_UTIL_H_`<br><br>`// your code here`<br><br>`#endif  // ELEC4_UTIL_H_` | `#pragma once`<br><br><br>`// your code here` |

Explain why these preprocessor statements must be added in each include file.

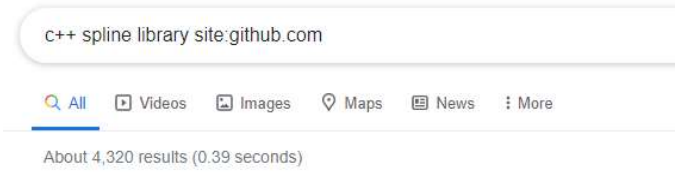
When the include file is completed, you must ensure that it compiles without error.

```
g++ -o tmp.o elec4_util.h  -other_compile_options_here_if_needed
```

It's a good sanity check, very important when your code base has lots of include files.

### 4.1.2 Hierarchical Name

A quick search with Gooble or Bing for C++ spline libraries returns thousands for candidates.




In large projects, multiple external libraries are used, so it is essential to avoid name collision. We have already seen a file modifier variable: `std::ios::in` last week. The goal now is to use the namespace feature of C++ to do the same with your `Spline` class, to obtain a hierarchical name, like `ELEC4::Spline` which is unique and would not collide with other Spline classes.

In your code, you create a `Spline` object with a statement very similar to the one given below:

```
...
Spline spline_r(xs, yr);
...
```

You shall modify the file `elec4_util.h` to create a Spline object with a hierarchical name as shown below:

```
...
ELEC4::Spline spline_r(xs, yr);
…
```

## 4.2 Using or not Using?

To avoid hierarchical names, you may be tempted to add a using statement similar to the example below.

```
using namespace std;
using namespace Eigen;
```

Global using statements are not a recommended practice.
Let's assume that the next release of Eigen introduces a class `Eigen::vector<>`, but you code is already using vector.

```
...
//
  vector<double> xs{  0.,   0.16,   0.42,   0.6425,  0.8575};
  vector<double> ys{100., 183.  , 235.  ,  40.    , 15.    };
```

The compiler will issue an error stating that vector is ambiguous.

```
snipets.cpp:38:3: error: reference to 'vector' is ambiguous
   38 |   vector<double> xs{   0.,   0.16,   0.42,   0.6425,  0.8575};
      |   ^~~~~~
snipets.cpp:28:9: note: candidates are: 'template<class T> class Eigen::vector'
   28 |   class vector {
      |         ^~~~~~
In file included from /usr/lib/gcc/x86_64-pc-cygwin/10/include/c++/vector:67,
                 from snipets.cpp:18:
/usr/lib/gcc/x86_64-pc-cygwin/10/include/c++/bits/stl_vector.h:389:11: note:
ctor'
  389 |     class vector : protected _Vector_base<_Tp, _Alloc>
      |           ^~~~~~
```

### 4.2.1 Recommendations

1) In your “dot h” file, you must use explicit hierarchical names
2) In your “dot ccp” file, prefer solution1, then solution 2, avoid solution 3.

```
// solution #1
// full hierarchical name
// no using statement
...
Eigen::MatrixXd ma(4,4);


// solution #2
// specific using statement
// only import what you need
using Eigen::MatrixXd;
...
MatrixXd ma(4,4);


// solution #3 (bad)
// global using statement
// import all objects in Eigen

using namespace Eigen;
...
MatrixXd ma(4,4);
```

Make the necessary changes in your code, compile and execute.

# 5 - Deliverables for this Lab

An archive created with `tar` linux utility containing the following files:

- `Makefile`
- `elec4_util.h`
- `main.cpp`
- `report.pdf`

The archive name must be `lab2_NAME1_NAME2_NAME3.tar.gz`
Name1, Name2 and Name3 are the names of the people in the group.
To create the archive:
`tar czf lab2_NAME1_NAME2_NAME3.tar.gz Makefile elec4_util.h main.cpp report.pdf`

The `report.pdf` document must

- Show how you have built your matrix $\boldsymbol{M}$ and vector $\boldsymbol{u}$
- Demonstrate execution of your program (with screen capture)
- Provides detail on your class `Spline` : constructor, destructor, members and methods.
- Answer all the questions found in this lab document

I will un-archive the deliverable and compile your files using the `make` command with gcc version 10.2

No credit will be given if the archive is not created with tar
No credit will be given if the report is not a pdf file
No credit will be given if your files cannot be compiled.

# 6 - Extra Credits (i.e. Optional)

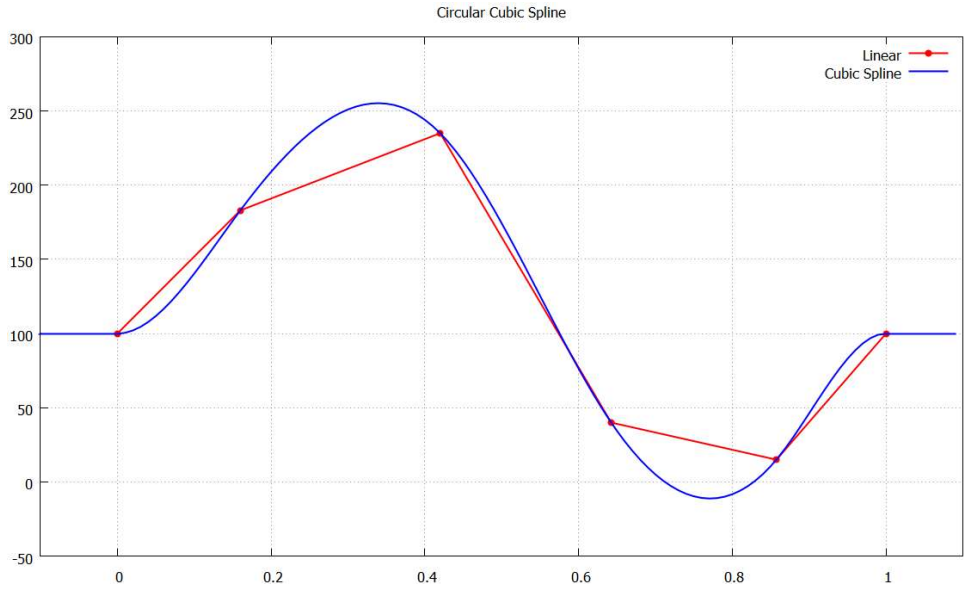
## 6.1 Math Optimization (easy)

The method `get_value()` must evaluate a 3rd degree polynomial.
If you use any of following statements, please update your code

```
...
// evaluation of a 3rd degree polynomial
// method#1, explicit => do not code like this

double y = a*x*x*x + b*x*x + c*x + d;

// method#2, explicit variant => do not code like this

double y = a*std::pow(x,3) + b*std::pow(x,2) + c*x + d;
```

## 6.2 Optional Code

If you want to invest some extra effort, you can add a method to your spline class to allow visual display of you curves, as shown below:




In my example, I created a file in a format understood by gunplot an open source chart and curve viewer. Other alternatives are possible (plotly for example) or it could be as simple as a csv format to import into excel and use the chart functionality of excel.

## 6.3 Algorithm Improvement

It is always important to analyze the behavior of your program when the size of the input parameters increases. In our case, that would be a large number of $x_i$, let's call this number *N*.

On the matrix solving part, there is not much you can do as we are using an external library. You need focus on the `get_value()` method and analyze its complexity as a function of *N*.

Is the time complexity linear, i.e. $O(N)$?
Can you think of a way change the time complexity to $O(N \log N)$

How can you code this update with minimal effort?

Suggestion: You can introduce a new method `get_value_fast()` and compare the two implementations.