POLYTECH NICE-SOPHIA

TP « Class »

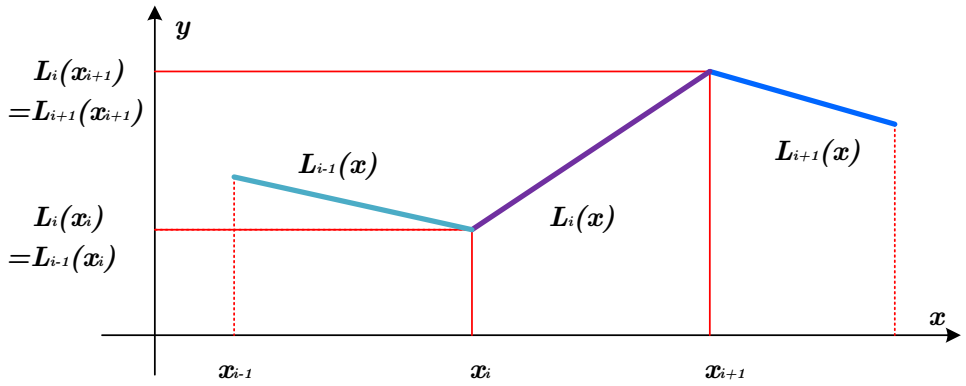
Objectifs de ce TP :

- Utilisation d'une librairie externe pour faciliter la résolution de l'exercice
- Codage d'une classe
- Ajout de modularité dans votre code avec plusieurs fichiers et gestion des noms hiérarchiques.

1 - Les courbes d'interpolation

1.1 Interpolation Linéaire entre N points par segment

A lire : https://en.wikipedia.org/wiki/Linear_interpolation



Pour un ensemble de $N+1$ points $(x_i, y_i), i = 0..N$, avec $x_i < x_{i+1}$, on cherche N droites telles que
$L_i(x) = a_i x + b_i$, et donc $2N$ coefficients $\{a_i, b_i\}$ tels que :

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

On remarque que l'on peut trouver chaque paire $\{a_i, b_i\}$ indépendamment des autres.
Il est plus facile de poser

$$\Delta_i = x - x_i$$

Et de trouver d'autres $\{a_i, b_i\}$, tels que

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

On a toujours :

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

Ce qui permet d'obtenir immédiatement

$$b_i = y_i$$

puis

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

1.2 Interpolation Spline entre N points

Les « splines » permettent de tracer des courbes passant par des points choisis sans rupture de pente.
A lire : https://en.wikipedia.org/wiki/Spline_interpolation
A lire (plus abordable que l'article wikipedia) https://timodenk.com/blog/cubic-spline-interpolation/
A voir : http://jsxgraph.uni-bayreuth.de/wiki/index.php/Cubic_spline_interpolation

On s'intéresse ici à la variante « circulaire » des splines cubiques.
Pour un ensemble de $N+1$ points $(x_i, y_i), i = 0 \,..\, N$, avec les mêmes conditions que la section ci-dessus, on cherche N polynômes du 3ème degré :

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

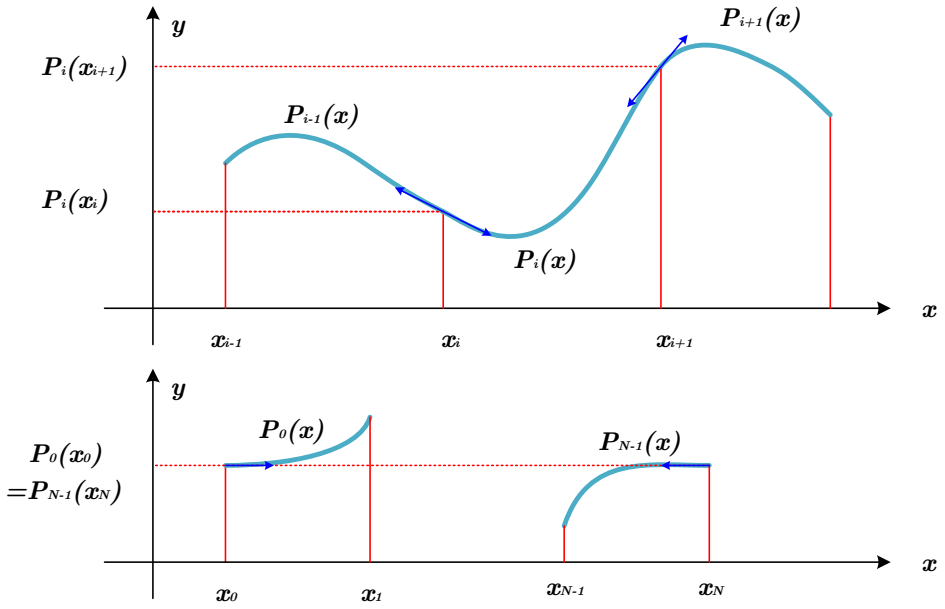
et donc $4N$ coefficients $\{a_i, b_i, c_i, d_i\}$ tels que :

$$P_i(x_i) = y_i$$
$$P_i(x_{i+1}) = y_{i+1}$$
$$P'_i(x_i) = P'_{i-1}(x_i), \quad \text{pour } 0 < i \leq N-1$$
$$P''_i(x_i) = P''_{i-1}(x_i), \text{ pour } 0 < i \leq N-1$$

On impose les conditions limites suivantes (variante circulaire)

$$P_{N-1}(x_N) = y_0$$
$$P'_0(x_0) = 0$$
$$P'_{N-1}(x_N) = 0$$

Ce qui se traduit graphiquement :



L'ensemble de ces équations permet de construire un système linéaire de $4N$ équations à $4N$ inconnues, qu'il faut résoudre pour trouver les inconnues : $\{a_i, b_i, c_i, d_i\}$.

Pour résoudre ce système linéaire, vous devez tout d'abord construire une matrice, $\boldsymbol{M}$, de dimension $4N \times 4N$, et un vecteur colonne, $\boldsymbol{v}$, de dimension $4N$ tel que

$$\boldsymbol{Mu} = \boldsymbol{v}$$

Avec le vecteur $\boldsymbol{u}$ tel que
$\boldsymbol{u} = (a_0, b_0, c_0, d_0, a_1, b_1, \dots, a_{N-1}, b_{N-1}, c_{N-1}, d_{N-1})^{\mathrm{T}}$

Une librairie externe permettra facilement la résolution numérique de $\boldsymbol{u}$.

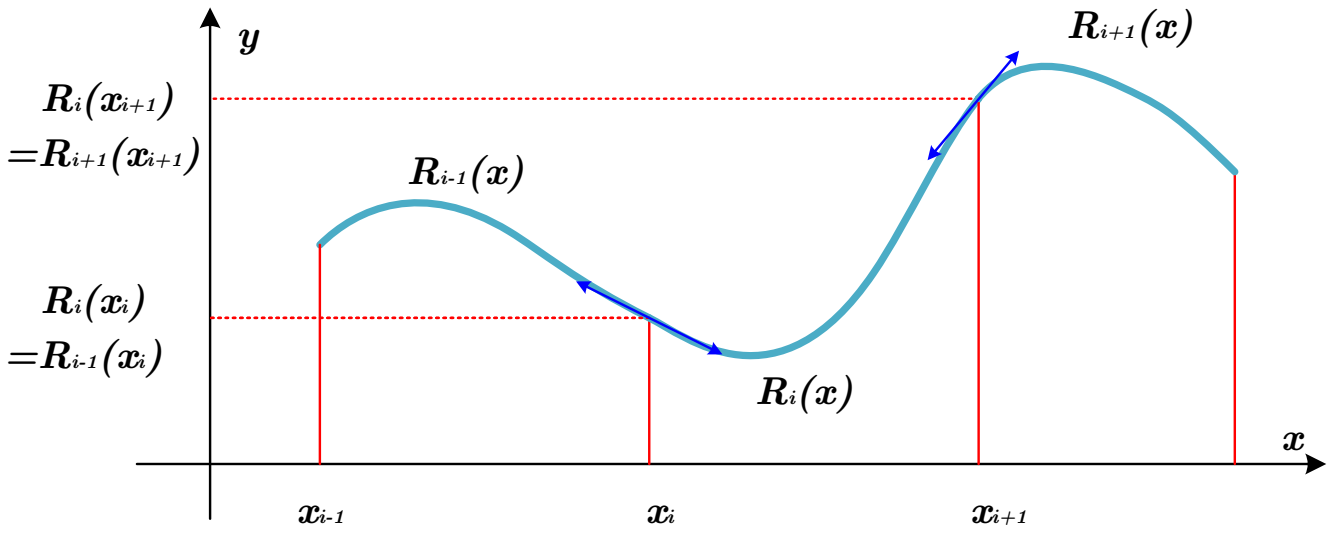
Les hypothèses suivantes sont applicables pour la suite de l'exercice :

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

Pour améliorer la qualité du résultat numérique, on utilisera la variante suivante :

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

Q Cette variante permet de réduire la taille de la matrice à $3N \times 3N$. Pourquoi ?
Vous utiliserez cette variante dans votre programme.



2 - Utilisation de Eigen

Eigen est une librairie dédiée au calcul matricielle que l'on va utiliser pour résoudre notre système linéaire de $3N$ équations à $3N$ inconnues.

2.1 Installation

Vous devez installer Eigen version 3.3.7 sur votre ordinateur. La manip à faire est la suivante :

1) Ouvrir votre shell, puis tapez les commandes suivantes :

```
shell> mkdir -p /usr/local/include
shell> cd /usr/local/include
shell> git clone https://gitlab.com/libeigen/eigen.git
shell> cd eigen
shell> git checkout 3.3.7
shell> ls -d $PWD/Eigen
/usr/local/include/eigen/Eigen
```

Bien faire attention au nom en bleu : c'est le vrai nom du fichier qui est différent de l'URL.

En fonction de votre installation, il faudra mettre sudo devant les commandes suivantes

```
shell> sudo mkdir -p /usr/local/include
shell> sudo git clone ...
shell> sudo git checkout ...
```

J'ai ajouté la couleur rouge pour bien faire ressortir le chemin d'accès à la librairie Eigen sur ma machine. Sur votre machine ce chemin peut-être différent.

2) Prendre note de ce chemin.

2.2 Vérification de l'installation

Lire le tutos avec le lien ci-dessous pour les informations de base :
http://eigen.tuxfamily.org/dox/GettingStarted.html
Bien lire la section « *Compiling and running your first program* »
Vous allez maintenant compiler et exécuter le programme de test.
Faire un copier/coller du programme donné ci-dessous :

```
#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;
}
```

C'est un exemple de résolution d'un système linéaire, pour plus de détail voir le lien suivant :
http://eigen.tuxfamily.org/dox/group__TutorialLinearAlgebra.html

Attention : Dans la librairie Eigen, certaines lignes peuvent générer des « *warnings* ».
En fonction de votre compilateur, il faut ajouter une ou plusieurs options dans la ligne de compilation. Les options possibles sont : `-Wno-deprecated-declarations -Wno-ignored-attributes`

```
shell> g++ -std=c++14 -O3 -Wno-deprecated-declarations ... test.cpp
```

C'est à vous de trouver ce qui doit remplacer les …
Si ça ne compile pas, vous n'avez sans doute pas bien lu la page « *GettingStarted* ».

3 - Conception d'une classe

A vous de concevoir une classe qui devra permettre à minima de résoudre le système linéaire pour un ensemble de N points (x_i, y_i).
calculer $y = f(x)$

3.1 Modèle de Classe

Vous pouvez vous inspirer du modèle ci-dessous.

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

  double get_value(const double x) const {
  …
  }

};
```

Expliquez le rôle du mot clé `const` mis en rouge dans le code ci-dessus ?
Quelles sont vos prérequis sur les tableaux `xs` et `ys` ?

3.2 Validation de votre classe

Cette classe sera utilisée dans le programme ci-dessous (que vous devrez compléter).

```
int main(int argc, char *argv[]) {
  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 = 0.0 ; x <= 1.0 ; x += 0.1) {
    std::cout << "Value for " << x << " is " << spline1.get_value(x) << std::endl;
  }
}
```

Pour donner le résultat suivant :

```
shell> test-spline
Value for 0.0 is  100.000
Value for 0.1 is  140.427
Value for 0.2 is  208.977
Value for 0.3 is  251.007
Value for 0.4 is  244.055
Value for 0.5 is  174.349
Value for 0.6 is   76.420
Value for 0.7 is    5.035
Value for 0.8 is   -8.350
Value for 0.9 is   46.165
Value for 1.0 is  100.000
```

```
shell> test-linear
Value for 0.0 is  100.000
Value for 0.1 is  151.875
Value for 0.2 is  191.000
Value for 0.3 is  211.000
Value for 0.4 is  231.000
Value for 0.5 is  164.888
Value for 0.6 is   77.247
Value for 0.7 is   33.314
Value for 0.8 is   21.686
Value for 0.9 is   40.351
Value for 1.0 is..100.000
```

Si vous n'obtenez pas ces valeurs, vous avez une erreur dans votre code.

Avant de passer la version spline, vous pouvez si vous le souhaitez coder le projet avec la version des droites, puis si tout va bien changer votre code pour la version spline.

Quelques fonctions Eigen intéressantes :

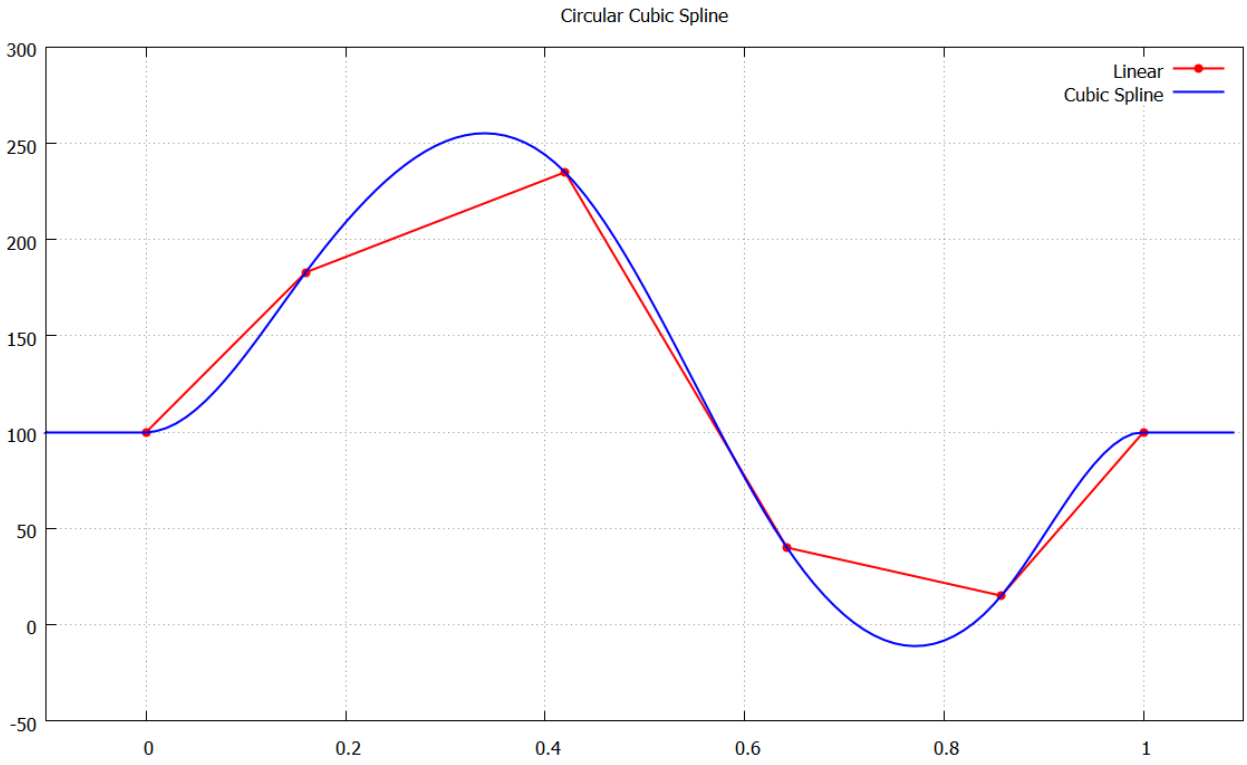
```
MatrixXd ma(4,4);  // constructor for a 4x4 matrix
ma(r,c) = v;       // set the matrix element at row 'r' and col 'c' to value 'v'
ma.setZero()       // clear the matrix ma;
```

Vous avez moins de 100 lignes de code à écrire !

C'est assez facile si on a bien compris l'organisation de la matrice $\boldsymbol{M}$, qui est composée majoritairement de zéro. Il faut donc n'écrire que les valeurs non nulles dans cette matrice.

3.3 Addendum (en option)

J'ai ajouté une méthode dans la class Spline, qui permet de générer un fichier au format gnuplot (un traceur de courbe). Le plot vous permet de mieux comprendre la différence entre l'interpolation linéaire et l'interpolation par spline.
On voit bien l'effet « circulaire » pour $x = 1$: la tangente est nulle et la valeur du point vaut bien y_0. Certaines variantes de spline permettent de limiter les « *overshoots* » ou « *undershoots* » que l'on voit clairement sur cette figure.



4 - Vers un Code Pro

4.1 Modularité

On souhaite réutiliser la class `Spline` dans des projets futurs, il faut donc bien s'assurer que cette classe est déclarée et définie dans des fichiers spécifiques.
Un fichier `#include`, que l'on nommera `elec4_util.h`, décrira entièrement la class `Spline`. Il ne faut pas créer de fichier `.cpp`.
Modifier votre code source en conséquence.

4.1.1 Structure des fichiers « `#include` »

Vous remarquerez que les fichiers `.h` ont toujours la même structure :

```
#ifndef ELEC4_UTIL_H_
#define ELEC4_UTIL_H_

...

#endif  // ELEC4_UTIL_H_
```

Expliquez le rôle de ces instructions du préprocesseur. Pour info avec C++14, on peut aussi utiliser un « *pragma* » : `#pragma once`.

Une règle essentielle pour les fichiers `#include` est la compilation unitaire directe : c'est-à-dire que tous les `#include` utilisés par votre .h doivent être spécifiés.
Dans mon cas, je compile avec la commande suivante :

```
g++ -std=c++14 -I/usr/local/include/eigen3 -o tmp.o elec4_util.h
```

Celle commande doit créer un fichier `tmp.o` sans erreur et warning. Assurez-vous que c'est bien le cas pour tous vos fichiers `.h`

4.1.2 Protection des noms

Faire une recherche rapide sous Google ou Bing pour savoir combien de librairies de « spline » sont disponibles.

Dans des projets importants (dizaine de milliers de lignes à plusieurs millions), il est critique de ne pas avoir de collision de nom entre les classes.

On pourrait tout à fait avoir un projet avec plusieurs types de spline. Si vous créez un objet comme ci-dessous :

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

Il faut être bien certain du type de classe appelée.
C++ a une notion de nom hiérarchique similaire au système de nom de fichier, vous connaissez déjà le token séparateur de hiérarchies, c'est :: (le double 2-points).

On peut créer une nouvelle hiérarchie qui par défaut sera à la racine avec la structure suivante dans le fichier `elec4_util.h`.

```
namespace ELEC4 {

class Spline {

};

} // namespace ELEC4
```

Attention : pas de « ; » après l'accolade de fermeture et on n'indente pas le code. Le commentaire de l'accolade de fermeture est absolument nécessaire pour faciliter la relecture.

Ce nom de hiérarchie s'utilise ainsi :

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

Modifiez votre code pour introduire le namespace `ELEC4` dans le fichier `elec4_util.h`.

4.2 Eviter les conflits de symbole

Lorsque vous écrivez une ligne du type :

```
using namespace Eigen;
```

le compilateur ajoute l'ensemble des symboles `Eigen` (nom des classes, nom des variables globales) dans sa liste de recherche.

Pour éviter les conflits et des temps de compilation trop long, il est préférable d'utiliser directement le nom hiérarchique complet de la classe ou bien une directive `using` spécifique.

```
// solution #1 [preferred] (full hierarchical name)
...
Eigen::MatrixXd ma(4,4) ;


// solution #2 [acceptable] (specific using statement)
using Eigen::MatrixXd;
...
MatrixXd ma(4,4) ;
```

Faire les changements dans votre code.
Vous ne devez plus utiliser le mot clé `using`.
Vérifiez que votre programme compile et s'exécute correctement.

5 - Livrables pour la section 3 & 4

1) Une archive avec l'ensemble de votre projet qui doit compiler sans erreur dans l'environnement MSYS2 et donne le résultat attendu.
L'archive doit avoir le bon nom (tp2_NOM1_NOM2.tar.gz) et doit contenir au moins les fichiers suivants
`Makefile`
`elec4_util.h`
`test_spline.cpp`
`rapport.pdf`

2) Un document `.pdf` décrivant les points clé suivants
Construction de la matrice $\boldsymbol{M}$ et $\boldsymbol{u}$
La classe `Spline` : constructeur, destructeur, membres et méthodes.
Les réponses aux questions posées.

6 - Pour aller plus loin (en option)

6.1 Amélioration d'algorithme

Si le nombre N devient important, vous devez analyser la complexité de votre algorithme en temps et en besoin mémoire.
On remarque que la résolution du système linéaire n'est fait qu'une fois mais que la méthode `get_value()` peut être appelée de nombreuse fois. On s'intéresse donc à la complexité de cette méthode. Pour une valeur x donnée, `get_value()` doit trouver l'index i qui vérifie :

$$x_i \leq x < x_{i+1}$$

Vous avez sans doute utilisé une boucle qui parcourt chaque intervalle jusqu'à trouver celui qui vérifie l'inéquation ci-dessus.
La complexité temporelle de cette boucle est $O(N)$.
Il est possible de faire beaucoup mieux avec une recherche binaire qui a une complexité temporelle en $O(\log(N))$.
On peut s'appuyer sur le fait que le tableau des x_i est en ordre croissant pour faire une recherche dichotomique. Mais cela demande à écrire un peu de code.
Une meilleure solution consiste à trouver dans la STL le « *container* » approprié.
Quel « *container* » allez-vous utiliser ?
Vous pouvez coder une méthode `get_value_fast()` et comparer les 2.
Quelles conclusions faites-vous ?