Sorbonne Université **M1 Mathématiques**
2018–2019 **4M056 : Programmation en C++**

TP noté 1 : Polynômes d'interpolation de Lagrange

L'objectif de ce TP est de montrer comment modéliser les polynômes d'interpolation de Lagrange sous forme de classe, et d'étudier leur application dans l'approximation de fonctions. On veillera à inclure dans chaque fichier toutes les bibliothèques nécessaires et les options de compilation nécessaires. **Il est également impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant de chacun des membres du binôme.**

1 Définition des polynômes de Lagrange

Soit $((x_0, y_0), ..., (x_N, y_N))$ $N+1$ couples de nombres réels tels que les x_i soient deux à deux distincts. On définit pour tout $0 \leq j \leq N$ le polynôme

$$P_j(X) = \prod_{\substack{i=0 \\ i \neq j}}^{N} \frac{X - x_i}{x_j - x_i} = \frac{(X-x_0)(X-x_1)...(X-x_{j-1})(X-x_{j+1})...(X-x_N)}{(x_j-x_0)(x_j-x_1)...(x_j-x_{j-1})(x_j-x_{j+1})...(x_j-x_N)}.$$

On peut montrer que P_j est de degré N pour tout j, et que $P_j(x_i) = \delta_{ij}$ pour tout $0 \leq i, j \leq N$. Par conséquent, le polynôme

$$P(X) = \sum_{j=0}^{N} y_j P_j(X) \tag{1}$$

est un polynôme de degré au plus N qui vérifie $P(x_i) = y_i$ pour tout i, et on peut vérifier que c'est le seul qui vérifie ces deux propriétés. Ce polynôme permet d'avoir une interpolation polynomiale d'une fonction $f : \mathbb{R} \to \mathbb{R}$ suffisamment régulière [1] sur l'intervalle $[x_0, x_N]$ en posant $y_i = f(x_i)$ pour tout $0 \leq i \leq N$.

On peut définir les polynômes d'interpolation de Lagrange de la façon suivante :

```
class LagrangeInterpolation{
protected:
    std::vector<std::pair<double,double>> points;
    int nb_points;

public:
    LagrangeInterpolation();
    void add_point(const std::pair<double,double> &);
    double operator()(const double &);
};
```

Voici à quoi servent les différents champs privés et méthodes :

— `points` est un vecteur contenant les couples (x_i, y_i) ;

— `nb_points` correspond au nombre de points utilisés pour l'interpolation (en particulier on doit avoir `nb_points = points.size()` pour des soucis de cohérence !) ;

— `LagrangeInterpolation()` est un constructeur par défaut qui initialise `points` en un vecteur de taille 0 et `nb_points` à 0 également ;

1. La régularité de f contrôle la précision de l'approximation de l'interpolation.

— `add_point` est une méthode qui permet d'ajouter un point (défini sous la forme d'un couple `std::pair<double,double>`) au vecteur `points` ;

— `operator()` est un opérateur qui calcule le polynôme d'interpolation de Lagrange à partir de `points` et l'applique à l'argument entre parenthèses. Autrement dit, si l'on définit `LagrangeInterpolation P1` , que l'on lui ajoute un certain nombre de points $(x_0, y_0), ..., (x_N, y_N)$, l'appel de `P1(x)` pour `double x` renvoie le résultat de $P(x)$ calculé par l'équation (1).

1. Dans un fichier `"interpolation.hpp"` , recopier le code ci-dessus en complétant la ligne du constructeur pour initialiser les champs comme indiqué précédemment.

2. Dans un fichier `"interpolation.cpp"` , écrire le code de la méthode `add_point` . Ne pas oublier de tester si la première coordonnée du point que l'on ajoute n'est pas déjà dans le vecteur `points` ! On pourra éventuellement utiliser `std::make_pair` en procédant comme suit :

```
double x,y;
x=...;
y=...;
std::pair<double,double> p = std::make_pair(x,y);
```

ou utiliser des parenthèses :

```
std::pair<double,double> p(x,y);
```

et on rappelle également que l'on accède au premier (resp. second) élément d'une paire `p` par `p.first` (resp. `p.second`).

3. Toujours dans `"interpolation.cpp"` , écrire le code de la méthode `operator()` .

On souhaite à présent pouvoir lire et écrire les champs de `LagrangeInterpolation` dans des fichiers. On introduit pour cela les opérateurs suivants :

```
std::ostream & operator<<(std::ostream &, const LagrangeInterpolation &);
std::istream & operator>>(std::istream &, LagrangeInterpolation &);
```

4. Écrire le code de `operator<<` de sorte que l'affichage de `LagrangeInterpolation P` dans un flux de sortie se fasse sous la forme

```
x_0  y_0
x_1  y_1
...
x_N  Y_N
```

5. Écrire le code de `operator>>` permettant de lire des flux d'entrée sous la même forme que ci-dessus.

6. Test : créer un fichier `"test_interpolation.cpp"` qui fait appel à `"interpolation.hpp"`, et dans le code `main`, lire dans un objet `LagrangeInterpolation P1` le contenu du fichier `"sample.dat"`, puis le réécrire dans un fichier `"sample2.dat"`. **Ne pas continuer le TP tant que les contenus des deux fichiers `.dat` ne sont pas identiques !!!**

2 Application : approximation de fonctions

2.1 Une "bonne" fonction

Soit la fonction

$$f_0 : \left\{ \begin{array}{ccc} \mathbb{R} & \to & \mathbb{R} \\ x & \mapsto & \sin(x) + e^{\cos(x)}. \end{array} \right.$$

On veut étudier son approximation sur $[0, 10]$ par une interpolation de Lagrange.

7. Toujours dans `"test_interpolation.cpp"`, déclarer un objet `LagrangeInterpolation P2` et lui ajouter les points $(i, f_0(i))$ pour $1 \leq i \leq 10$. Faire de même avec `LagrangeInterpolation P3` mais en prenant $(i/2, f_0(i/2))$ pour $1 \leq i \leq 20$ (attention aux types utilisés dans la division !).

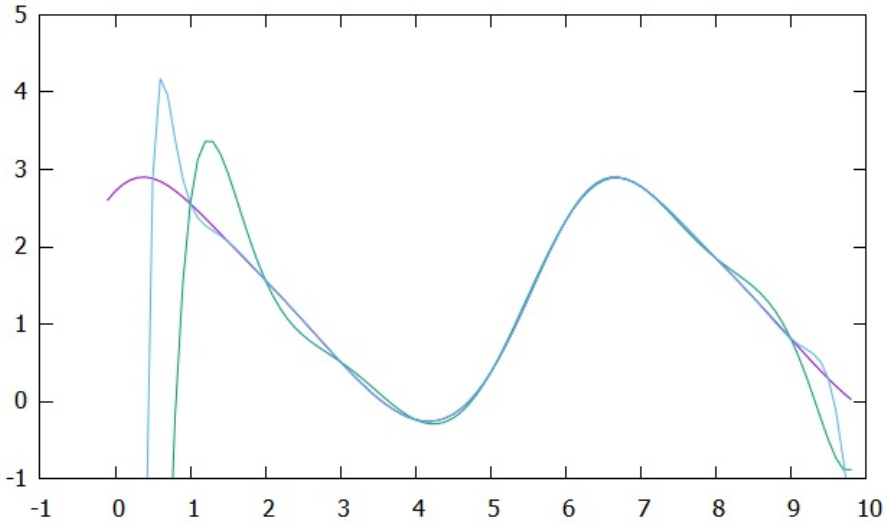
8. Dans un fichier `"comparaison1.dat"`, écrire pour x allant de 0.1 à 10 par pas de 0.1 les valeurs de $f_0(x)$, $P_2(x)$ et $P_3(x)$. Le résultat doit être mis en forme comme suit :

```
0   f(0)   P2(0)   P3(0)
0.1 f(0.1) P2(0.1) P3(0.1)
0.2 f(0.2) P2(0.2) P3(0.2)
...
9.9 f(9.9) P2(9.9) P3(9.9)
10  f(10)  P2(10)  P3(10)
```

9. Ouvrir `gnuplot`, écrire la commande `set yrange [-1:5]` pour fixer l'échelle des ordonnées et afficher le contenu de `"comparaison1.dat"` en utilisant la commande suivante :

```
plot for [i=2:4] 'comparaison1.dat' using 1:i with lines
```

Cela devrait ressembler plus ou moins à ça :



Comparer ensuite les deux interpolations. Laquelle semble la plus précise ?

2.2 Une "moins bonne" fonction

On souhaite procéder de la même façon que pour f_0 avec la fonction

$$f_1 : \left\{ \begin{array}{ccc} \mathbb{R}^* & \to & \mathbb{R} \\ x & \mapsto & \sin\left(\frac{1}{x}\right). \end{array} \right.$$

La réunion du graphe de cette fonction et du point $(0,0)$, appelée *courbe sinus du topologue*, est un contre-exemple très intéressant en topologie : c'est un espace topologique connexe mais ni localement connexe, ni connexe par arcs ; c'est également un exemple d'espace non localement compact qui est l'image continue d'un espace localement compact. Ces propriétés découlent du fait que la fonction oscille de plus en plus vite à mesure que l'on s'approche de 0. On va étudier son approximation près de 0.

10. Déclarer un objet `LagrangeInterpolation P4` et lui ajouter les points $(i/100, f_0(i/100))$ pour $1 \leq i \leq 10$. Faire de même avec `LagrangeInterpolation P5` mais en prenant $(i/200, f_0(i/200))$ pour $1 \leq i \leq 20$.

11. Dans un fichier `"comparaison2.dat"` , écrire pour x allant de 0.001 à 0.1 par pas de 0.001 les valeurs de $f_1(x)$, $P_4(x)$ et $P_5(x)$.

12. Dans `gnuplot` , écrire la commande `set yrange [-5:5]` puis afficher le contenu du fichier `"comparaison2.dat"` . Comparer les deux interpolations, entre elles d'une part, et vis-à-vis de celles de la question 9 d'autre part.

13. Reprendre les questions 8 et 11, et calculer une version discrétisée de la norme $L^2_{[a,b]}$ des différences $f_i - P_j$, i.e.

$$\sqrt{\frac{b-a}{N}\sum_{k=1}^{N}(f_i(x_k) - P_j(x_k))^2}$$

pour $N = 100$, et $[a,b] = [1,10]$ pour f_1 et $[0.01, 0.1]$ pour f_2. Comparer ces normes : que peut-on en dire ?

14. (Bonus) Refaire les questions 9 et 12 sans passer par les commandes `set yrange [...]` . Que constate-t-on ? Quelle approximation semble alors la plus précise pour chaque fonction ? Proposer une explication heuristique. Il est envisageable de refaire la question 13 en prenant des intervalles $[a,b]$ plus grands pour confirmer ou infirmer cette intuition (par exemple $[0.1, 10]$ pour f_1 et $[0.001, 0.1]$ pour f_2).