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

TP noté 2 : recherche de minimum par descente de gradient

À l'issue du TP, chaque binôme envoie par email ses fichiers `.hpp` et `.cpp`. **Il est impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant de chacun des membres du binôme.**

Description Soit $f : \mathbb{R}^d \to \mathbb{R}$ une fonction de classe C^1. Le gradient de f en $x = (x_1, \dots, x_d)$ est le vecteur :

$$\mathrm{grad}(f)(x) = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_d}, \right)$$

qui indique la pente ascendante de f en x. Pour trouver un minimum de f, un idée naïve consiste à suivre l'opposé du gradient (d'où le nom de *descente de gradient*).

Soit $\lambda > 0$ et soit $x \in \mathbb{R}^d$. On définit alors la suite $(x_n)_{n\in\mathbb{N}}$ par $x_0 = x$ et

$$x_{n+1} = x_n - \lambda \,\mathrm{grad}(f)(x)$$

Si λ est suffisamment petit et si x est suffisamment près d'un minimum de f alors la suite précédente converge vers ce minimum. Nous allons mettre en œuvre cet algorithme sous la forme d'un *template* de fonction. Si α est un zéro de f, alors il existe un voisinage de α tel que, pour tout x dans ce voisinage, la suite $(x_n)_{n\in\mathbb{N}}$ définie précédemment converge vers α.

1. Dans un fichier `gradient.hpp`, écrire un *template* de fonction

```
template <class Func, class V>
std::tuple<V,double,V> gradient_descent(const Func & f, V x0,
                unsigned max_iter, double lambda);
```

qui itère la suite (x_n) initialisée à `x0` jusqu'à ce que le nombre d'itérations dépasse `max_iter`. Nous supposerons que les types `V` et `Func` représentent respectivement un espace vectoriel et l'ensemble des fonctions de `V` dans $\mathbb{R}$ et vérifient les propriétés suivantes :

- les objets de type `V` sont munis de `+` et `-` et la multiplication par un scalaire.
- les objets `f` de type quelconque `Func` possède un opérateur `()` tel que `f(x)` soit de type `double` pour `x` de type `V`, ainsi qu'une méthode `grad` telle que `f.grad(x)` donne la dérivée (de type `V` également) au point `x`.
- le `tuple` renvoyée (défini dans la bibliothèque `<tuple>` et créé éventuellement par `std::make_tuple(a,b,c)`) a pour premier élément l'estimation du point où le minimum de `f` est atteint, en deuxième argument le minimum estimé et en troisième argument le gradient de `f` calculé au même point.

Étant donné un élément `c` de type `std::tuple<V,double,V>`, on accède en lecture ou en écriture au k-ème élément par l'instruction `std::get<k>(t)`.

2. Faire ce qu'il faut pour que, si l'argument `lambda` n'est pas précisé, il est choisi égal à `0.01`.

3. Test sur une fonction simple. Soit $m > 0$. La fonction $q(x) = x^4 - 2m^2x^2$ possède deux minima en $\pm m$ avec valeur $f(\pm m) = -m^4$. Nous proposons une classe très simple dans un fichier `quartic.hpp`

```
class Quartic {
        protected:
                double m;
        public:
                Quartic(double m0);
                ... operator() ....
                ... grad(...) ...
};
```

La compléter comme il le faut pour qu'elle puisse être utilisée comme premier argument de `gradient_descent`. En dimension un comme ici, le gradient est tout simplement la dérivée.

4. Dans le même fichier `quartic.cpp`, utilisez `gradient_descent` avec les valeurs par défaut pour vérifier que vous retrouvez au moins un minimum de `q(x)` en moins de 1000 itérations pour $m = 1$ et $m = 2$ en partant de $x = 0.5$.

5. Implémentation de $\mathbb{R}^d$. On introduit dans un fichier `vect.hpp` le *template* de classe suivant pour décrire des points de $\mathbb{R}^d$:

```
template <unsigned d>
class Vect {
        protected:
        double * coord;//tableau dynamique avec d cases
        public:
        Vect();
        double operator[](unsigned i) const; //accesseur à la i-ème  coordonnée
                ...
};
```

Compléter le constructeur par défaut qui alloue le tableau et met toutes les coordonnées à zéro, l'accesseur aux coordonnées et le mutateur associé.

6. Surcharger l'opérateur `<<` pour afficher un point de `Vect<d>` sous la forme :

```
(x1,x2,...,xd)
```

7. (*rule of three*) Écrire le constructeur par copie, le destructeur et l'opérateur d'affectation.

8. Ajouter une fonction de multiplication par un nombre réel :

```
template <int d>
Vect<d> operator*(double x,const Vect<d> & u);
```

ainsi qu'une fonction d'ajout de deux vecteurs avec `operator+`. *Cela permet ainsi aux classes* `Vect<d>` *d'être utilisées comme classe* `V` *dans* `gradient_descent`.

9. Soit la fonction $G : \mathbb{R}^2 \to \mathbb{R}^2$ définie par

$$G(x, y) = x^4 + y^4 - 2x^2 - 2y^2 + x + y$$

Écrire dans un programme `test2d.cpp` un programme complet qui trouve un minimum de G en partant du point initial $(0, 0)$. Le programme devra bien évidemment utiliser `gradient_descent` ainsi que la classe `Vect<2>` qui sera utilisée pour les arguments de G et son gradient. On affichera les valeurs du point où le minimum est trouvé, la valeur du minimum et la valeur du gradient (pour s'assurer que cette dernière est proche de 0).

Remarque : le minimum attendu est en $(-1.10716, -1.10716)$ *approximativement. Vous ajusterez le nombre d'itérations et* λ *pour assurer la convergence du programme.*