# Master Ingénierie Mathématique (M2)
# Informatique (Projet)

Le but de ce projet est d'appréhender par la pratique l'intérêt de la programmation orientée objet et générique par l'exemple. On s'intéressera à la modélisation et à l'utilisation de classes de fonctions mathématiques afin de permettre certains calculs symboliques comme par exemple la dérivation et l'intégration.

## 1 Objectifs

On souhaite créer un programme permettant la manipulation de fonctions analytiques $f : \mathbb{R} \to \mathbb{R}$ dans un premier temps.

Ces fonctions devront pouvoir être créées et manipulées à l'intérieur même du programme. Il faudra pouvoir afficher leur expression dans un terminal. Il faudra également être capable des les évaluer numériquement. On voudra aussi pouvoir calculer symboliquement leur dérivée, c'est-à-dire créer une nouvelle fonction contenant l'expression de la dérivée de la fonction donnée. On calculera

$$\int_a^b f(x)dx,$$

pour toutes les fonctions considérées. Il faudra mettre en place un mécanisme permettant de savoir si cette intégrale peut être évaluée exactement **de manière simple** ou si on doit avoir recours à une méthode de quadrature.

## 2 Implémentation

1. Écrire la classe d'interface `IFonction` dont hériteront toutes les classes décrivant des fonctions. Cette classe doit contenir l'interface (fonctions membres virtuelles pures) des différentes fonctionnalités requises pour le projet.

2. Écrire une classe instanciable `Polynome` héritant de `IFonction`. Elle utilisera (agrégation) la classe `Rn` vue en cours pour stocker ses coefficients. Cette dernière est donc à créer aussi (version héritage de la classe `template <typename T> class Tableau`).
3. Écrire la classe `FonctionConstante` qui décrit les fonctions constantes.
4. Écrire la classe `CFunction`. Cette classe a pour but de créer un lien entre les fonctions décrites dans la bibliothèque mathématique et notre classe d'interface `IFonction`. Elle sera composée de la manière suivante :
   - un membre de type `std::string` qui contiendra le nom de la fonction,
   - un membre privé `m_cfunction` de type pointeur de fonction défini comme suit

     ```
     double (*m_cfunction)(double x);
     ```

     Ce membre dans le cas de la fonction sinus peut être manipulé de la manière suivante :

     ```
     m_cfunction = std::sin;
     double value = m_cfunction(1.2); // évalue en 1.2
     ```
5. Écrire les classes
   - `FonctionSomme`,
   - `FonctionDifference`,
   - `FonctionProduit`,
   - `FonctionQuotient`,
   - `FonctionPuissance`,
   - `FonctionComposee`.

   Afin de gérer correctement la mémoire (allocations et désallocations) on stockera dans ces classes les fonction opérandes en utilisant non pas des pointeurs mais des objets du type `AutoPtr<IFonction>`. La classe `AutoPtr` vue en cours est donc à programmer.
6. Dans le cas où on ne peut pas calculer analytiquement l'intégrale d'une fonction, on l'évaluera *automatiquement* par une méthode d'intégration numérique. La méthode d'intégration numérique sera celle des trapèzes. Le nombre d'intervalles utilisés devra pouvoir être changé par l'utilisateur dans la fonction `main`.
7. On demandera un tracé des fonction sur un intervalle prescrit à l'aide de `gnuplot`. Le format de fichier à utiliser est le suivant

   ```
   x0 f(x0)
   ```

```
x1 f(x1)
...
xn f(xn)
```

les points où la fonction est évaluée sont donnés par ordre croissant. Ce rôle sera rempli par la classe `GnuplotFunction`.

8. On mettra finalement en place un système de simplification de l'arbre des expressions (de type `AutoPtr<IFonction>`) construit. Par exemple, on remplacera le produit de 2 polynômes dans la description par le polynôme produit. Ces opérations de simplification devront être effectuées en postraitement de l'arbre des expressions initial en en générant un nouveau.