Programmation fonctionnelle
Projet 3
Bibliothèque de calcul symbolique

La plupart des calculs numériques en informatique sont effectués sur des nombres flottants, car les ordinateurs sont équipés d'unités matérielles qui permettent d'effectuer de tels calculs rapidement. Cette représentation introduit toutefois des erreurs (à titre d'exemple, ouvrez un interpréteur et évaluez l'expression `0.1 + 0.2`). Pour faire des calculs mathématiques exacts, il est possible d'utiliser des nombres rationels et une représentation symbolique des fonctions. Le but de ce projet est de développer une librairie qui offre ces outils.

Structures de données Vous devrez définir

- un type de données `Rational` pour représenter les nombres rationnels, ainsi qu'une classe annexe `RationalIsFractional` qui implémentera le trait `Scala.math.Fractional[Rational]`, afin de fournir des opérations numérique sur le type
- un type de données `SymbolicFunction` pour représenter des fonctions arbitraires (à une variable, de type `Rational`) par leur arbre syntaxique
- un type de données `Polynomial` pour représenter des polynômes (à une variable, de type `Rational`)

Fonctionnalités à implémenter

- fonctions pour convertir une fonction arbitraire en polynôme (lorsque c'est possible) et vice versa
- fonctions pour évaluer un fonction en un point, et de même pour les polynômes (la valeur retournée sera elle aussi de type `Rational`)
- fonctions pour calculer la dérivé d'une fonction ou d'un polynôme
- fonction pour trouver les racines d'un polynôme (de degrée 1 ou 2, voir 3 si vous y parvenez)
- fonction pour calculer la limite d'une fonction en un point (à droite ou à gauche). Pour cette fonction, il faudra définir un type qui étende les rationnels avec deux valeurs $+\infty$ et $-\infty$.

Pour aller plus loin Cette partie est optionnelle : il s'agira de généraliser les types des fonctions et des polynômes. Plutôt que de définir ces types uniquement pour une variable de type `Rational`, ces types devront être polymorphiques (avec un paramètre de type `T`). Les fonctions d'évaluation seront elles aussi polymorphes, et elle devront prendre un argument additionel : un élément de type `Fractional[T]`, qui fournira les opérations numériques nécessaires sur le type `T`.