DM2 - Racines d'un polynôme

On s'intéresse aux nombres rationnels qui permettent - en théorie - de palier aux problèmes d'arrondis avec les calculs sur les flottants. **En aucun cas vous ne devez donc repasser par les flottants pour réaliser vos fonctions**, sinon le travail réalisé en amont ne sert à rien...
Partie obligatoire : 1.1 ; 1.2 ; 1.3 ; 1.4 ; 1.5 ; 2.1 ; 2.2 ; 2.3 ; 2.4 ; 2.5

Exercice 1 [Les nombres rationnels et OCaml]
Le type "nombre rationnel" n'existe pas en OCaml. Heureusement, on peut créer de nouveaux types :

```
#type frac = {num : int; den : int};;
```

On peut alors définir des variables :

```
# let a = {num = 2 ; den = 3};;
val a : frac = {num = 2; den = 3}
# let b = {num = -1 ; den = 1};;
val b : frac = {num = -1; den = 1}
# let c = {num = 6 ; den = 10};;
val c : frac = {num = 6; den = 10}
```

On peut accéder en lecture (données non mutables) aux différents **champs** :

```
# a.num;;
- : int = 2
```

ou encore créer des fonctions avec ces nouveaux types :

```
# let double x = {num = 2*x.num ; den = x.den};;
val double : frac -> frac = <fun>
# double a;;
- : frac = {num = 4; den = 3}
```

1. Ecrire une fonction `pgcd : int -> int -> int` qui reçoit deux entiers non simultanément nuls et retourne leur PGCD :

```
#pgcd 4 15;;
- : int = 1
#pgcd (-12) 15;;
- : int = 3
```

2. Écrire une fonction `simplifie : frac -> frac` qui reçoit une fraction et retourne la fraction sous forme irréductible.

```
# simplifie c;;
- : frac = {num = 3; den = 5}
```

3. Écrire une fonction d'impression `print_frac : frac -> unit` qui reçoit une fraction et affiche "proprement" la celle-ci.

```
#print_frac a;;
2/3- : unit = ()
#print_frac b;;
-1- : unit = ()
```

4. Écrire une fonction `sommeQ : frac -> frac -> frac` qui, recevant 2 fractions, retourne leur somme sous forme irréductible. On peut alors transformer cette fonction en **opérateur infixe** avec l'instruction `let (++) = sommeQ;;`. Ainsi :

```
#a ++ b;;
- : frac = {num = -1; den = 3}
```

```
#a ++ c;;
- : frac = {num = 19; den = 15}
```

5. Définir de la même idée les opérateurs infixes `**`, `--` et `//` de type `frac -> frac -> frac`.
6. Ecrire une fonction `frac_of_string : string -> frac` qui parse une chaine de caractères pour la transformer en fraction. Pour plus de souplesse, on renommera cette fonction `f` :

```
#f"1";;
- : frac = {num = 1; den = 1}
```

```
#f"-2/3";;
- : frac = {num = -2; den = 3}
```

7. Écrire une fonction `eval : frac -> int -> unit` qui reçoit une fraction x et un entier p et affiche le développement décimal de x à p chiffres derrière la virgule.

```
# eval (f"2020/5") 0;;
404- : unit = ()
```

```
# eval (f"-111/19") 20;;
-5,84210526315789473684- : unit = ()
```

Exercice 2 On travaille sur les polynômes de $\mathbb{Q}[X]$, que l'on choisit de représenter par des listes de nombres de type `frac` précédemment définis en prenant comme convention :

- Le polynôme nul est représenté par la liste vide `[]`.
- Un polynôme de la forme $a_0 + a_1X + a_2X^2 + \cdots a_NX^N$ où les $(a_i)_{0 \leqslant i \leqslant N}$ sont des réels et $a_N \neq 0$ est représenté par la liste `[a0;a1;...aN]`.
- Aucune liste n'a donc 0 pour dernier élément.

Par exemple :

```
let p1 = [];; (* Polynôme nul *)
let p2 = [f"2" ; f"1" ; f"4"];; (* P2(X) = 4X^2+X+2 *)
```

1. Ecrire une fonction `degre` qui renvoie le degré d'un polynôme entré sous forme d'une liste et -1 pour le polynôme nul.

```
# degre p1;;
- : int = -1
```

```
# degre p2;;
- : int = 2
```

2. Ecrire une fonction `affiche` qui affiche "proprement" un polynôme `p`. Ne perdez pas trop de temps sur cette fonction pour tous les cas particuliers, c'est surtout pour visualiser les résultats.

```
# affiche p2;;
2+1X^1+4X^2- : unit = ()
```

3. Ecrire une fonction `image` qui reçoit un polynôme P donné sous forme de liste et une fraction a et renvoie la valeur de $P(a)$. On pourra appuyer la programmation de la fonction sur le fait que : $a_0 + a_1x + \cdots + a_nx^n = a_0 + x\left(a_1 + a_2x + \cdots a_nx^{n-1}\right)$

```
# image (f"3/2") p2;;
- : frac = {num = 25; den = 2}
```

```
# image (f"5") p1;;
- : frac = {num = 0; den = 1}
```

4. Écrire une fonction `produitXn : frac list -> int -> frac list` qui reçoit un polynôme P donné sous forme de liste et un entier n et retourne le polynôme $X^nP(X)$

```
# produitXn p1 3;;
- : frac list = []
```

```
# affiche (produitXn p2 2);;
0+0X^1+2X^2+1X^3+4X^4- : unit = ()
```

5. Ecrire une fonction `somme` qui renvoie la somme de deux polynômes.

```
# affiche (somme p2 [f"-2";f"1";f"-4"]);;
0+2X^1- : unit = ()
```

```
# affiche (somme p2 [f"1";f"2"]);;
3+3X^1+4X^2- : unit = ()
```

6. Ecrire une fonction `dérive` qui renvoie le polynôme dérivé d'un polynôme donné sous forme d'une liste. On pourra s'inspirer de la forme présentée dans le calcul de `image` ou programmer une solution plus optimisée.

```
# affiche (derive p2);;
1+8X^1- : unit = ()
```

```
# derive p1;;
- : frac list = []
```

7. Écrire une fonction `produit_nb : frac list -> frac -> frac list` `produit_nb p a` renvoie le polynôme $aP(X)$

```
# affiche (produit_nb p2 (f"2/3"));;
4/3+2/3X^1+8/3X^2- : unit = ()
```

```
# produit_nb p1 (f"3");;
- : frac list = []
```

8. En déduire une fonction `produit` qui renvoie le produit de deux polynômes.

```
# affiche (produit p2 p2);;
4+4X^1+17X^2+8X^3+16X^4- : unit = ()
```

```
# produit p2 p1;;
- : frac list = []
```

9. Écrire une fonction `coeff_dom` qui renvoie le coefficient d'un polynôme non nul.
10. Ecrire une fonction `reste` qui reçoit deux polynômes `p1` et `p2` et renvoie le reste dans la division du polynôme `p1` par `p2`

```
# reste p2 [f"-1"; f"2"];;
- : frac list = [{num = 7; den = 2}]
```

Exercice 3 Théorème de Sturm pour isoler les racines d'un polynôme.

Soit P un polynôme, on appelle **chaîne de Strum** à partir du polynôme P est une suite finie de polynômes $P_0, P_1, ..., P_m$ construite par récurrence :

- $P_0 = P$
- $P_1 = P'$ (polynôme dérivé)
- Pour $i \geqslant 2$, P_i est l'opposée du reste de la division de P_{i-2} par P_{i-1}.
- La construction s'arrête au dernier polynôme non nul.

1. Écrire (à la main) la suite de Sturm à partir du polynôme $P(X) = 2 + 4X + 3X^3 + X^4$
2. Écrire une fonction `sturm` qui reçoit un polynôme P et retourne la chaîne de Sturm sous forme d'une liste de polynômes.

```
# sturm [f"-1" ; f"-1" ; f"0" ; f"1" ; f"1"];;
- : frac list list =
[[{num = -3; den = 16}]; [{num = -64; den = 1}; {num = -32; den = 1}];
 [{num = 15; den = 16}; {num = 3; den = 4}; {num = 3; den = 16}];
 [{num = -1; den = 1}; {num = 0; den = 1}; {num = 3; den = 1};
  {num = 4; den = 1}];
 [{num = -1; den = 1}; {num = -1; den = 1}; {num = 0; den = 1};
  {num = 1; den = 1}; {num = 1; den = 1}]]
# List.iter affiche (sturm [f"-1" ; f"-1" ; f"0" ; f"1" ; f"1"]);;
-3/16
-64-32X^1
15/16+3/4X^1+3/16X^2
-1+0X^1+3X^2+4X^3
-1-1X^1+0X^2+1X^3+1X^4
```

Soit P un polynôme, et $P_0, P_1, ..., P_m$ sa chaîne de Sturm. Pour $a \in \mathbb{R}$, on note $\sigma(a)$ le nombre de changement de signes de la suite $P_0(a), P_1(a), ..., P_m(a)$. (Les 0 sont ignorés)

3. Ecrire une fonction `signes` qui reçoit une liste de fractions et retourne le nombre de changements de signes de celle-ci.

```
# signes [f"1" ; f"-1" ; f"2" ; f"-4"];;
- : int = 3
# signes [f"1" ; f"-1" ; f"2" ; f"0" ; f"-4"];;
- : int = 3
# signes [f"1" ; f"-1" ; f"-2" ; f"0" ; f"-4"];;
- : int = 1
```

On admet le théorème suivant :

Si P n'a **que des racines simples**,
alors le nombre de racines de P dans l'intervalle $[a, b]$ est $\sigma(a) - \sigma(b)$.

4. Ecrire une fonction `nbRacines` telle que `nbRacines p a b` donne le nombre de racines dans l'intervalle $[a, b]$ du polynôme p (n'ayant que des racines simples)

```
# nb_racines [f"-4";f"1";f"4"] (f"-2") (f"2");;
- : int = 2
# nb_racines [f"-4";f"1";f"4"] (f"-2") (f"-1");;
- : int = 1
```

On admet le théorème suivant (**Borne de Cauchy**). Soit $P(X) = X^n + a_{n-1} + ... + a_1 X + a_0$ un polynôme de degré $n \geqslant 0$ à coefficients réels. Toute racine α de P vérifie $|\alpha| \leqslant 1 + \max\limits_{0 \leqslant i \leqslant n} |a_i|$

5. Écrire une fonction `borne` qui reçoit un polynôme non nul et renvoie le majorant $1 + \max\limits_{0 \leqslant i \leqslant n} |a_i|$ du théorème précédent.
6. En déduire une fonction `isole` qui retourne une liste $[x_0, x_1, ...x_p]$ encadrant les racines $\alpha_1, \alpha_2, ...\alpha_p$ du polynôme P. C'est à dire $x_0 < \alpha_1 < x_1 < \alpha_2 < x_2 ... < \alpha_p < x_p$.
7. Proposer enfin une fonction `racines` telle que `racines p` ε retourne la liste des racines de `p` approchées à ε.

Exercice 4 [Un peu de maths !] Si le polynôme P a des racines multiples, le théorème n'est plus valable... En vous appuyant sur votre cours de mathématiques, donner une technique permettant de se ramener à un autre polynôme $\tilde{P}$ ayant les mêmes racines P mais toutes simples. Programmez alors votre idée.