Programmation fonctionnelle 2023-2024

Feuille 2bis : Exercices complémentaires

Exercice 2.1 1. Écrire la fonction `multiplicateur` qui prend en paramètre un entier `n` et retourne la fonction de type `int -> int` qui à un entier `i` associe `2 * i`.

$$\begin{array}{rccc} multiplicateur: & \mathbb{N} & \to & \mathbb{N} \mapsto \mathbb{N} \\ & n & \mapsto & i \mapsto n * i \end{array}$$

Exemples :

```
utop[2]> multiplicateur;;
- : int -> int -> int = <fun>
utop[3]> multiplicateur 10;;
- : int -> int = <fun>
utop[4]> multiplicateur 10 2;;
- : int = 20
utop[5]> multiplicateur 100 3;;
- : int = 300
```

Exercice 2.2 On considère la suite récurrente d'ordre 2 suivante :

$$\begin{cases} u_0 & = & 0 \\ u_1 & = & 3 \\ u_n & = & -u_{n-1} + 2u_{n-2}, \forall n \geqslant 2 \end{cases}$$

1. Écrire une fonction récursive `seq_aux` de type `int -> int * int` qui à tout entier naturel n associe le couple (u_n, u_{n+1}).
2. En déduire une fonction `seq` qui à tout entier naturel n associe u_n.
3. Quel est le type de `seq` ?
4. Quelle est la complexité de `seq` en fonction de son paramètre n ?
5. Rechercher sur internet (ou dans le cours d'algo des arbres) comment on peut obtenir une complexité logarithmique pour ce format de suite (à la Fibonacci) en utilisant des matrices et la multiplication "à la Grecque" pour multiplier les matrices.

Exemples :

```
# seq_aux;;
- : int -> int * int = <fun>
# seq_aux 0;;
- : int * int = (0, 3)
# seq_aux 1;;
- : int * int = (3, -3)
# seq_aux 2;;
- : int * int = (-3, 9)
# seq 3;;
- : int = 9
```

Exercice 2.3 Soit f une fonction de $\mathbb{R}$ dans $\mathbb{R}$.

Si f est dérivable, la fonction dérivée de f, f' peut être définie par

$$\begin{array}{rccl} f' : & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & \lim_{h\to 0} \tau(f,h,x) \end{array}$$

où

$$\tau(f,h,x) = \frac{f(x+h) - f(x-h)}{2h}$$

En prenant un h petit (proche de 0), on obtient la fonction f'_h, dérivée approchée de f, définie par

$$\begin{array}{rccl} f'_h : & \mathbb{R} & \to & \mathbb{R} \\ & x & \mapsto & \tau(f,h,x) \end{array}$$

Écrire la fonction `derivee` qui prend en paramètre f, h et retourne f'_h. Exemples :

```
# epsilon;;
- : float = 1e-06
# derivee;;
- : (float -> float) -> float -> float -> float = <fun>
# derivee (fun x -> 4. *. x +. 3.) epsilon 10.;;
- : float = 3.99999999700639819
# derivee (fun x -> x *. x *. x +. 5.) epsilon 2.;;
- : float = 11.9999999999009788
```

On s'intéresse maintenant à la dérivée $n$ème $f^{(n)}$ d'une fonction f qui peut être définie par

$$\begin{cases} f^{(0)} = f \\ f^{(n)} = (f')^{(n-1)} \end{cases}$$

Écrire la fonction `derivee_n` qui prend en paramètre un entier n, f et h et retourne la fonction dérivée $n$ème (approchée) de f.

Exemples :

```
# derivee_n;;
- : int -> (float -> float) -> float -> float -> float = <fun>
# derivee_n 2 (fun x -> x *. x *. x +. 5.) epsilon 2.;;
- : float = 12.0015108961979422
```

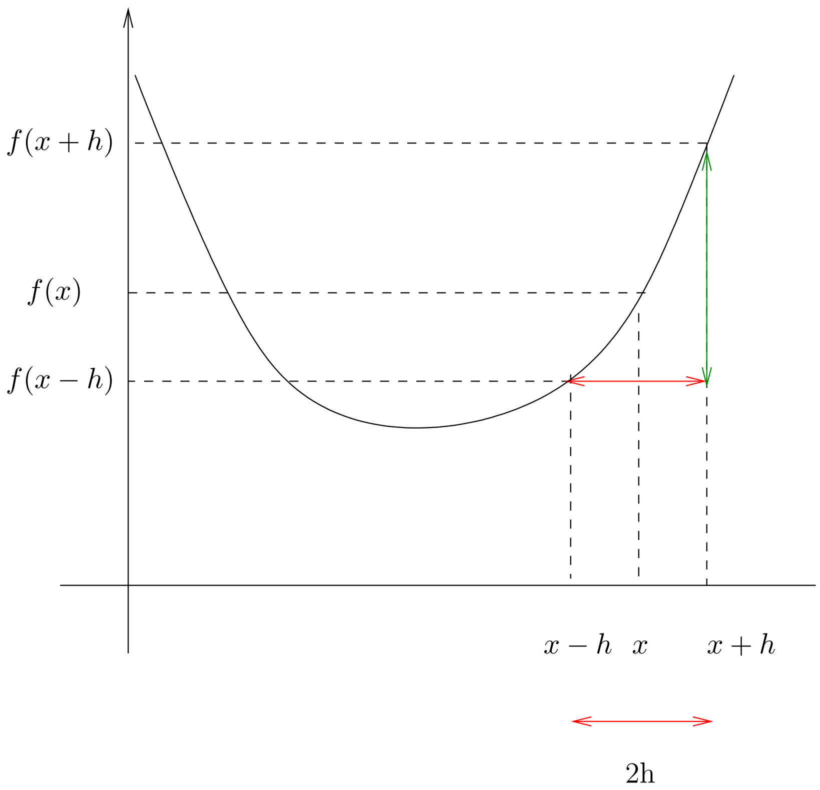

Fig. 1 : Taux d'accroissement