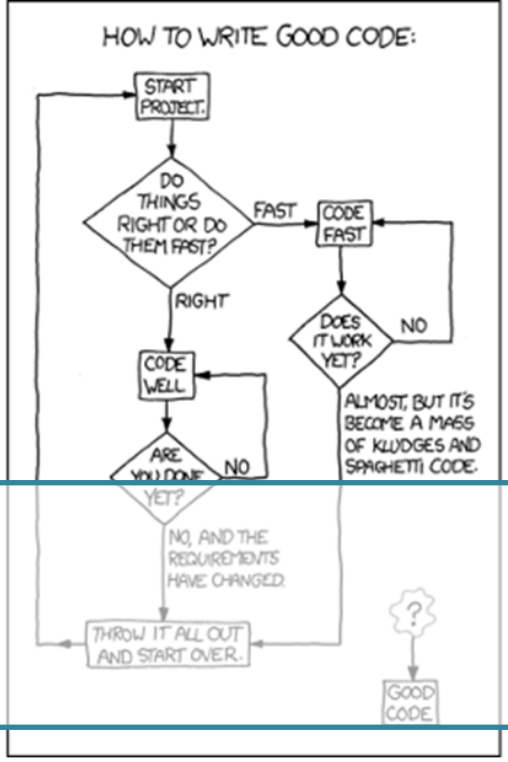
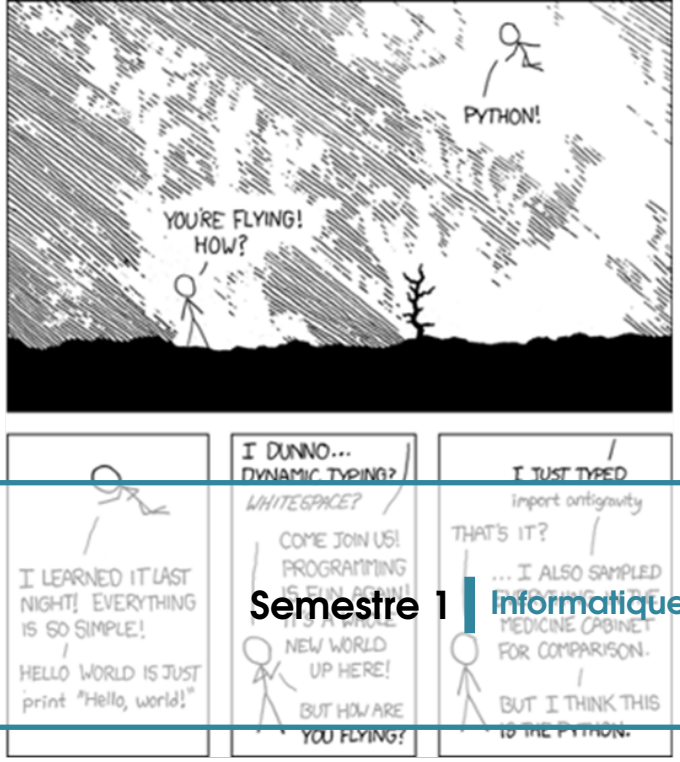




Thèmes d'étude

Recherche séquentielle

Recherche séquentielle

Exercice 1 – Exercices d'échauffement

Objectif Rechercher séquentiellement un élément dans un tableau unidimensionnel ou dans un dictionnaire.

Recherche d'un nombre dans une liste

Nous allons commencer par rechercher si un nombre est dans un tableau.
Commençons par définir la liste des entiers pairs compris entre 0 et `nb` exclus.

```
def generate_pair_01(nb: int) -> list :
    """
        Génération d'une liste de nb entiers compris entre 0 (exclus) et nb (exclus).
    """
    res = []
    for i in range(1,nb//2):
        res.append(2*i)
    return res
```

Recopier la fonction dans un terminal.

Question 1 *Vérifier que la fonction* `generate_pair_01` *fonctionne pour* `nb=0`, `nb=9`, `nb=10`.

Question 2 *Écrire une fonction de signature* `generate_pair_02(nb: int) -> list` *en utilisant une boucle* `while`.

Question 3 *Écrire une fonction de signature* `recherche_nb_01(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On utilisera une boucle* `for`.

Question 4 *Écrire une fonction de signature* `recherche_nb_02(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On utilisera une boucle* `while`.

Question 5 *Écrire une fonction de signature* `recherche_nb_03(nb: int, L: list) -> bool` *qui renvoie* `True` *si* `nb` *est dans* `L`, `False` *sinon. On n'utilisera pas explicitement de boucles* `for` *ou* `while`.

Question 6 *Écrire une fonction de signature* `recherche_first_index_nb_01(nb: int, L: list) -> int` *qui renvoie l'index de la première appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `for`.

Question 7 *Écrire une fonction de signature* `recherche_first_index_nb_02(nb: int, L: list) -> int` *qui renvoie l'index de la première appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `while`.

Question 8 *Écrire une fonction de signature* `recherche_last_index_nb_01(nb: int, L: list) -> int` *qui renvoie l'index de la dernière appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `for`.

Question 9 *Écrire une fonction de signature* `recherche_last_index_nb_02(nb: int, L: list) -> int` *qui renvoie l'index de la dernière appartion du nombre* `nb` *dans la liste* `L`. *La fonction renverra* `-1` *si* `nb` *n'est pas dans la liste. On utilisera une boucle* `while`.

Question 10 *Écrire une fonction de signature* `recherche_index_nb_01(nb: int, L: list) -> list` *qui renvoie la liste des index du nombre* `nb` *dans la liste* `L`. *La fonction renverra une liste vide si* `nb` *n'est pas dans la liste.*

Recherche d'un caractère dans une chaîne (de caractères)

Question 11 *Écrire une fonction de signature* `is_char_in_str_01(lettre: str, mot: str) -> int` *qui renvoie* `True` *si* `lettre` *est dans* `mot`, `False` *sinon. On utilisera une boucle* `for` *ou* `while`.

Question 12 *Écrire une fonction de signature* `is_char_in_str_02(lettre: str, mot: str) -> int` *qui renvoie* `True` *si* `lettre` *est dans* `mot`, `False` *sinon. On n'utilisera ni boucle* `for` *ni* `while` *explicite.*

Question 13 *Écrire une fonction de signature* `compte_lettre_01(lettre: str, mot: str) -> int` *qui renvoie le nombre d'occurrences de* `lettre` *dans le* `mot`.

Les instructions suivantes permettent de charger l'ensemble des mots du dictionnaire dans la variable `dictionnaire`. `dictionnaire` est une liste de mots. Chacune des lettres de l'alphabet sont stockées dans la variable `alphabet`.

```
alphabet = 'abcdefghijklmnopqrstuvwxyz'
def load_fichier(file):
    fid = open(file,'r')
    mots = fid.readlines()
    fid.close()
    return mots
dictionnaire = load_file('liste_francais.txt')
```

Question 14 *Écrire une fonction de signature* `compte_lettre_02(lettre: str, mots: list) -> int` *qui renvoie le nombre d'occurrences de* `lettre` *dans une liste de mots* `mots`.

Question 15 *Quelle consonne apparaît le plus souvent ? Quelle consonne apparaît le moins souvent ? Indiquer le nombre d'occurrences dans chacun des mots*

Question 16 *Écrire une fonction de signature* `mots_plus_long( mots: list) -> str` *qui renvoie le mot le plus long.*

Question 17 *Écrire une fonction de signature* `cherche_mot_in_chaine_01(mot: str, chaine: str) -> int` *qui renvoie* `True` *si* `mot` *est dans* `chaine`, `False` *sinon. On utilisera des boucles* `for` *ou* `while`.

Question 18 *Écrire une fonction de signature* `cherche_mot_in_chaine_02(mot: str, chaine: str) -> int` *qui renvoie* `True` *si* `mot` *est dans* `chaine`, `False` *sinon. On n'utilisera ni boucle* `for` *ni* `while`.

Question 19 *Écrire une fonction de signature* `cherche_mot_in_dico(nb: int, dico: lst) -> str` *qui permet de trouver le mot de* `nb` *lettres qui est le plus contenu dans d'autres mots.*

Recherche dans un dictionnaire

Il est possible de définir la fonction `generate_tab_alea` différemment.

```
def generate_tab_alea_02(deb: int, fin: int,nb: int) -> list :
    """
        Génération d'une liste de nb entiers compris entre deb (inclus) et fin (exclus).
    """
    return [rd.randrange(deb,fin) for i in range(nb)]
```

Pour cela commençons par générer une liste d'entiers aléatoires.

```
import random as rd # Permet de charger une bibliothèque permettant de générer des nombres alé
    atoires.

def generate_tab_alea_01(deb: int,fin: int,nb: int) -> list :
    """
        Génération d'une liste de nb entiers compris entre deb (inclus) et fin (exclus).
    """
    res = []
    for i in range(nb):
        res.append(rd.randrange(deb,fin))
    return res
```

Il est possible de définir la fonction `generate_tab_alea` différemment.

```
def generate_tab_alea_02(deb: int,fin: int,nb: int) -> list :
    """
        Génération d'une liste de nb entiers compris entre deb (inclus) et fin (exclus).
    """
    return [rd.randrange(deb,fin) for i in range(nb)]
```

Recherche d'un élément Recherche du maximum Recherche du second maximum

Exercice 2 – La conjecture de Syracuse

D'après Jean-Pierre Becirspahic, `https://info-llg.fr/`

Objectif

- ...

On doit cette conjecture au mathématicien allemand Lothar Collatz qui, en 1937, proposa à la communauté mathématique le problème suivant : « on part d'un nombre entier strictement positif; s'il est pair on le divise par 2, s'il est impair on le multiplie par 3 et on ajoute 1. On réitère ensuite cette opération.»

Par exemple, à partir de 14 on construit la suite de nombres :

$$14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2...$$

Après que le nombre 1 ait été atteint, la suite des valeurs (1, 4, 2, 1, 4, 2 ...) se répète indéfiniment en un cycle de longueur 3.

La conjecture de Syracuse [1] est l'hypothèse mathématique selon laquelle n'importe quel entier de départ conduit à la valeur 1 au bout d'un certain temps.

Nous allons expérimenter cette conjecture en programmant l'évolution de la suite $(u_n)_{n\in\mathbb{N}}$ définie par les relations :

$$u_0 = c \text{ et } \forall n \geq 1, u_{n+1} = \begin{cases} \dfrac{u_n}{2} & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}.$$

Temps de vol et altitude maximale

Le temps de vol d'un entier c et le plus petit entier n (en admettant qu'il existe) pour lequel $u_n = 1$. Par exemple, le temps de vol pour $c = 14$ est égal à 17.

Question 1 *Définir une fonction nommée* `tempsdevol` *prenant un paramètre entier* `c` *et retournant le plus petit entier n pour lequel $u_n = 1$.*

De manière tout aussi imagée, on appelle altitude maximale de c la valeur maximale de la suite $(u_n)_{n\in\mathbb{N}}$. Par exemple, l'altitude maximale de $c = 14$ est égale à 52.

Question 2 *Modifier votre algorithme pour définir une fonction nommée* `altitude` *qui calcule cette fois-ci l'altitude maximale pour un entier c donné en paramètre.*

Vérification expérimentale de la conjecture

On désire désormais vérifier la validité de la conjecture pour toute valeur $c \leq 1\,000\,000$. Une première solution consisterait à calculer le temps de vol pour toutes ces valeurs, mais ce calcul est long et il y a mieux à faire en observant que si la conjecture a déjà été vérifiée pour toute valeur $c' < c$, il suffit qu'il existe un rang n pour lequel $u_n < c$ pour être certain que la conjecture sera aussi vérifiée au rang c.

On appelle temps d'arrêt (ou encore temps de vol en altitude) le premier entier n (s'il existe) pour lequel $u_n < c$.

Question 3 *Écrire une fonction* `tempsdarret` *prenant un paramètre entier c et retournant le temps d'arrêt de la suite de Syracuse correspondante.*

Nous souhaitons maintenant mesurer le temps nécessaire pour vérifier la conjecture jusqu'à un paramètre entier m. Pour cela, nous allons utiliser la fonction `time` du module time du même nom, sans argument, qui retourne le temps en secondes depuis une date de référence (qui dépend du système).

Question 4 *À l'aide de cette fonction écrire une fonction* `verification` *qui prend en argument un paramètre entier m et retourne le temps nécessaire pour vérifier que toutes les valeurs $c \in [\![2, m]\!]$ ont bien un temps d'arrêt fini. Quelle durée d'exécution obtient-t-on pour $m = 1\,000\,000$?*

Question 5 *Quel est le temps d'arrêt d'un entier pair ? et d'un entier de la forme $c = 4n + 1$? En déduire qu'on peut restreindre la recherche aux entiers de la forme $4n + 3$, et modifier en conséquence la fonction précédente. Combien de temps gagne-t-on par rapport à la version précédente pour $m = 1\,000\,000$? Vérifier ensuite la conjecture pour $m = 10\,000\,000$.*

1. Du nom de l'université américaine qui a popularisé ce problème.

Records

Question 6 *Déterminer l'altitude maximale que l'on peut atteindre lorsque $c \in [\![1, 1\,000\,000]\!]$, ainsi que la valeur minimale de c permettant d'obtenir cette altitude.*

Question 7 *Déterminer le temps de vol en altitude (autrement dit le temps d'arrêt) de durée maximale lorsque $c \in [\![1, 1\,000\,000]\!]$ ainsi que la valeur de c correspondante.*

Question 8 *On appelle vol en altitude de durée record un vol dont tous les temps d'arrêt de rangs inférieurs sont plus courts. Par exemple, le vol réalisé pour $c = 7$ est un vol en altitude de durée record (égale à 11) car tous les vols débutant par $c = 1, 2, 3, 4, 5, 6$ ont des temps d'arrêt de durées inférieures à 11. Déterminez tous les vols en altitude de durée record pour $c \leq 1\,000\,000$.*

Affichage du vol

Pour obtenir des graphes, on utilise la fonction plot qui appartient à un module appelé `matplotlib.pyplot` et dédié au tracé de graphes. Vous allez donc commencer par importer celui-ci à l'aide de la commande : `import matplotlib.pyplot as plt` Désormais toutes les fonctions de ce module vous sont accessibles à condition de les préfixer par `plt`. Nous découvrirons progressivement les nombreuses possibilités qu'offre ce module, mais aujourd'hui nous n'aurons besoin que de deux fonctions : `plt.plot` et `plt.show`. Sous sa forme la plus simple, la fonction `plt.plot` n'exige qu'une liste en paramètre : `plt.plot([a0, a1, ..., an])` crée un graphe constitué d'une ligne brisée reliant les points de coordonnées (k, a_k) pour $k \in [\![0, n]\!]$. En Python, une liste est encadrée par des crochets et ses éléments séparés par une virgule. Nous étudierons les listes plus tard dans le cours ; pour l'instant nous n'aurons besoin que du résultat suivant : si `lst` est une liste, on ajoute un élément `x` à celle-ci à l'aide de la commande : `lst.append(x)`. Une fois votre graphe créé par la fonction `plt.plot`, il reste à le faire apparaître dans une fenêtre annexe à l'aide de l'instruction `plt.show()`.

Question 9 *Définir une fonction nommée graphique qui prend un entier c en paramètre et qui construit le graphe de la suite $(u_n)_{n\in\mathbb{N}}$ durant son temps de vol.*

Structures imbriquées

Utilisation Modules

Exercice 3 – Surfing Porquerolles

D'après Concours Mines Ponts 2018.

Objectif

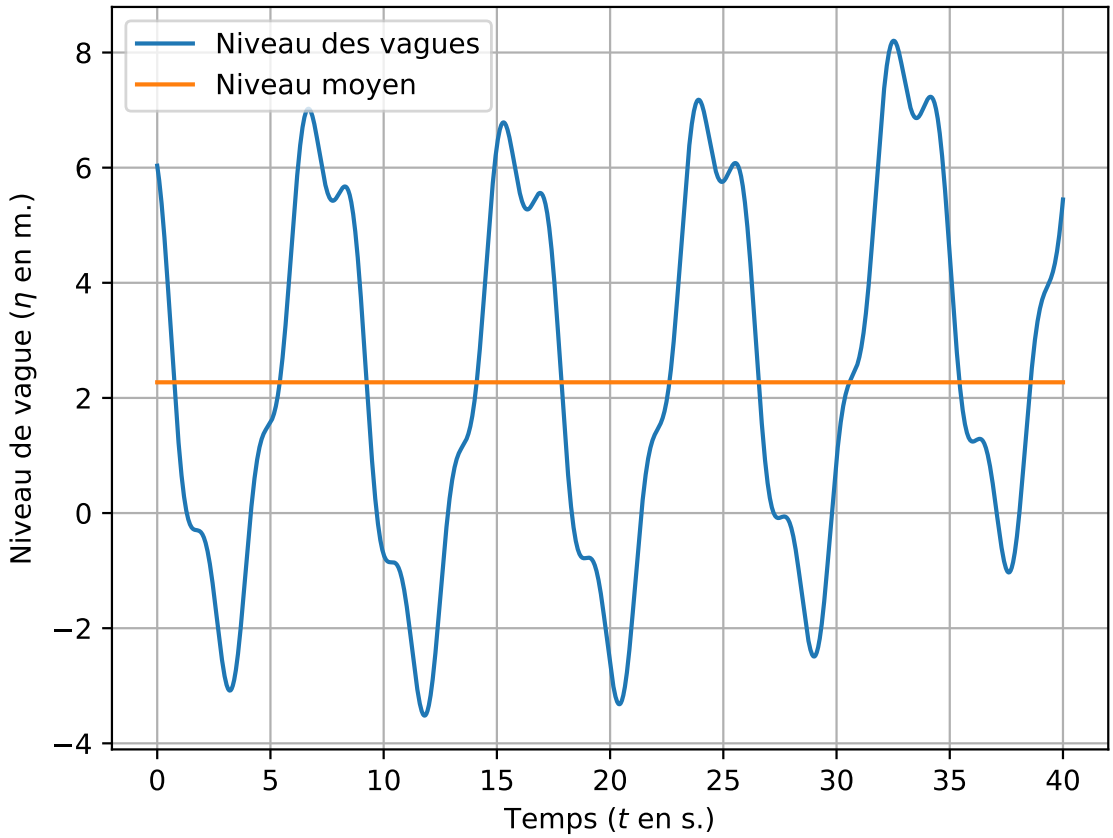
- Lire un fichier texte.
- Analyser les données d'un fichiers.

Analyse vague par vague

On considère ici que la mesure de houle est représentée par un signal $\eta(t) \in \mathbb{R}$, $t \in [0, T]$, avec η une fonction C^1. On appelle niveau moyen m la moyenne de $\eta(t)$ sur $[0, T]$. On définit Z_1, Z_2, ..., Z_n l'ensemble (supposé fini) des Passages par le Niveau moyen en Descente (PND) (voir Figure suivante). À chaque PND, le signal traverse la valeur m en descente. On suppose $\eta(0) > m$ et $\dfrac{\mathrm{d}\eta}{\mathrm{d}t}(0) > 0$. On en déduit que $\eta(t) - m \geq 0$ sur $[0, Z_1]$. Les hauteurs de vagues H_i sont définies par les différences :

$$\begin{cases} H_1 = \max_{t\in[0,Z_1]} \eta(t) - \min_{t\in[Z_1,Z_2]} \eta(t) \\ H_i = \max_{t\in[Z_{i-1},Z_i]} \eta(t) - \min_{t\in[Z_{i-1},Z_{i+1}]} \eta(t) \\ \qquad \text{pour } 2 \leq i < n \end{cases}$$

On définit les périodes de vagues par $T_i = Z_{i+1} - Z_i$.



Le fichier `vagues.txt` contient un relevé des niveaux d'eau mesurés par une bouée au large de Porquerolles. (Pour ne pas se mentir, on a plutôt généré un profil qui pourrait vaguement ressembler à un tel relevé.)

Il est constitué de deux colonnes, séparées par une virgule, la première colonne correspondant à une mesure de temps (en secondes), la seconde colonne correspondant à une mesure de niveau de hauteur d'eau (en mètres).

Question 1 *Écrire une fonction* `lire_fichier(file: str) -> list,list` *prenant comme argument le nom d'un fichier et renvoyant la liste des temps que l'on notera* `les_t` *et la liste des niveaux de vagues que l'on notera* `liste_niveaux`.

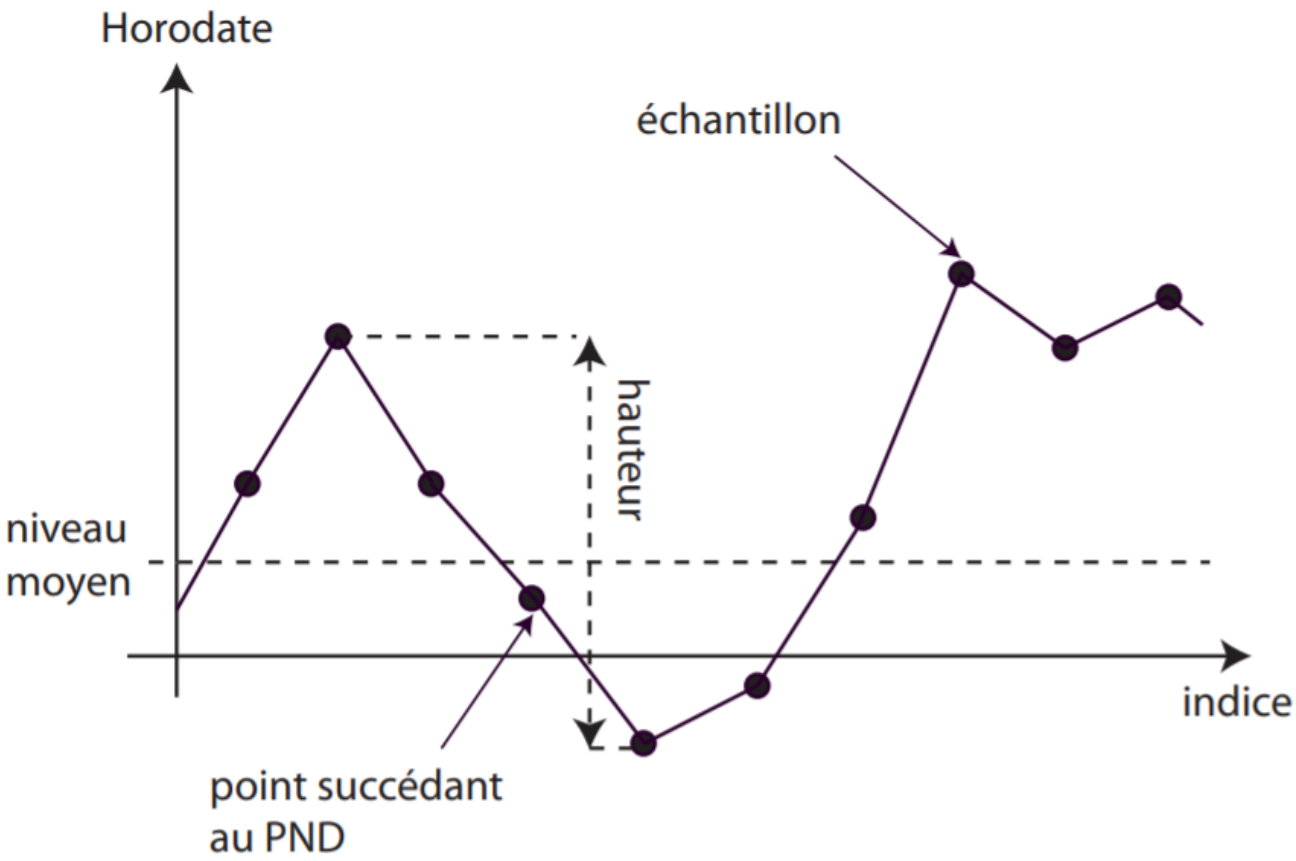
Question 2 *Écrire une fonction* `trace_vagues(file: str) -> None` *prenant comme argument le nom d'un fichier affichant le profil des vagues en fonction du temps.*

Question 3 *Écrire une fonction* `moyenne(liste_niveaux: list) -> float` *prenant comme argument une liste non vide* `liste_niveaux`*, et retournant sa valeur moyenne.*

Question 4 *Écrire une fonction* `ind_premier_pzd(liste_niveaux: list) -> int` *retournant, s'il existe, l'indice du premier élément de la liste tel que cet élément soit supérieur à la moyenne et l'élément suivant soit inférieur à la moyenne. Cette fonction devra retourner* -1 *si aucun élément vérifiant cette condition n'existe.*

Question 5 *Écrire une fonction* `ind_dernier_pzd(liste_niveaux: list) -> int` *retournant l'indice* i *du dernier élément de la liste tel que cet élément soit supérieur à la moyenne et l'élément suivant soit inférieur à la moyenne. Cette fonction devra retourner -2 si aucun élément vérifiant cette condition n'existe.*

On souhaite stocker dans une liste successeurs, les indices des points succédant (strictement) aux PND (voir figure suivante).



Question 6 *Écrire une fonction* `construction_successeurs(liste_niveaux: list) -> list` *retournant la liste successeurs.*

Question 7 *Écrire une fonction* `decompose_vagues(liste_niveaux: list) -> list` *qui permet de décomposer une liste de niveaux en liste de vagues. On omettra les données précédant le premier PND et celles succédant au dernier PND. Ainsi* `decompose_vagues([1, -1, -2, 2, -2, -1, 6, 4, -2, -5])` *(noter que cette liste est de moyenne nulle) retournera* `[[-1, -2, 2], [-2, -1, 6, 4]]`.

On désire maintenant caractériser les vagues. Ainsi, on cherche à concevoir une fonction `proprietes(liste_niveaux: list) -> list` retournant une liste de listes à deux éléments `[Hi,Ti]` permettant de caractériser chacune des vagues i par ses attributs :

- `Hi`, sa hauteur en mètres (m) ;
- `Ti`, sa période en secondes (s).

Question 8 *Écrire une fonction* `proprietes(liste_niveaux: list, dt:float) -> list` *réalisant cet objectif.* `dt` *représente la durée entre deux échantillons de la liste des temps obtenu à la question 2. On pourra utiliser les fonctions de Python* `max(L)` *et* `min(L)` *qui retournent le maximum et le minimum d'une liste* `L`*, respectivement.*

Contrôle des données

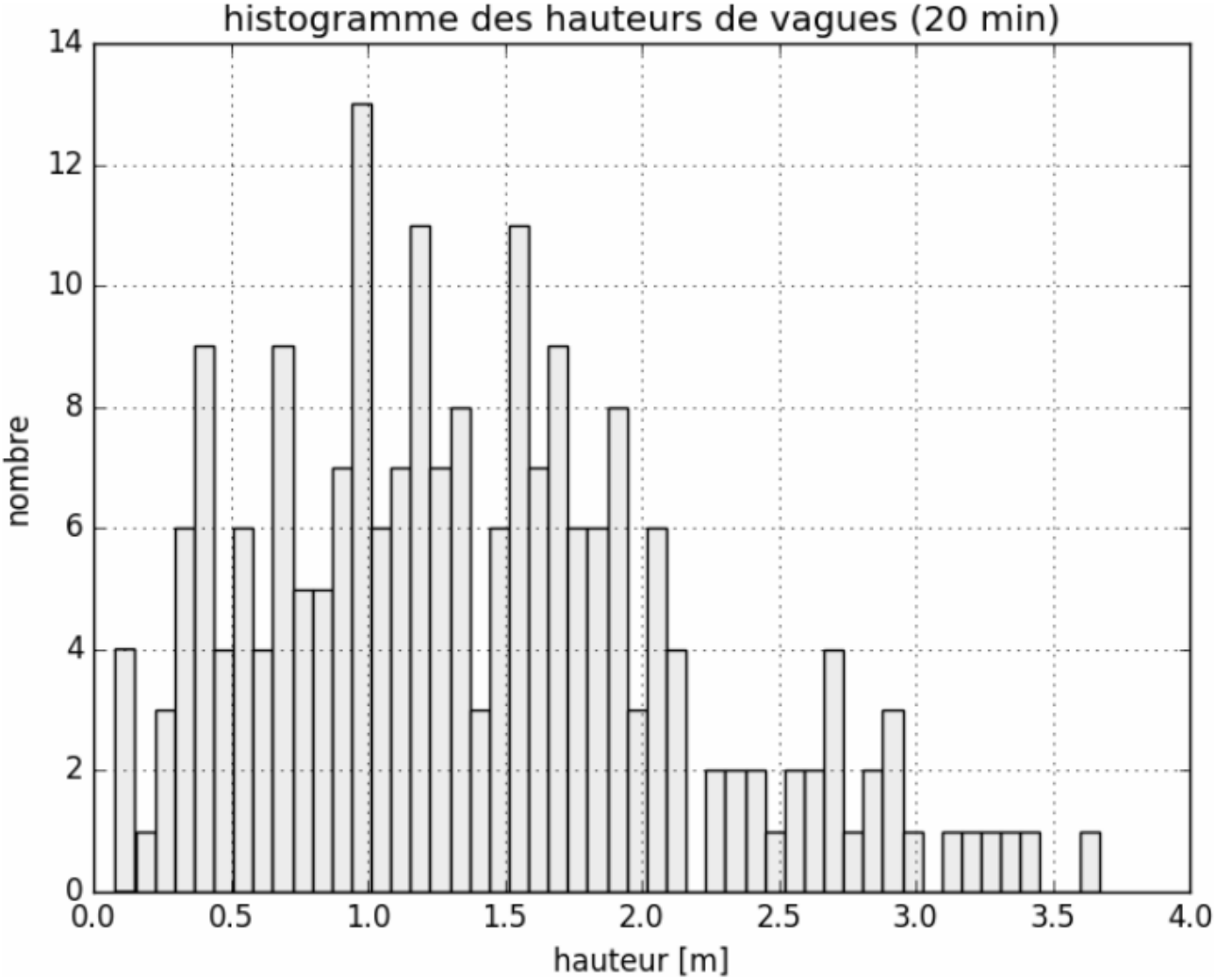
Plusieurs indicateurs sont couramment considérés pour définir l'état de la mer. Parmi eux, on note :

- $H_{\max}$: la hauteur de la plus grande vague observée sur l'intervalle d'enregistrement $[0, T]$;
- $H_{1/3}$: la valeur moyenne des hauteurs du tiers supérieur des plus grandes vagues observées sur $[0, T]$;
- $T_{H1/3}$: la valeur moyenne des périodes du tiers supérieur des plus grandes vagues observées sur $[0, T]$.

Question 9 *Écrire une fonction* `H_max(liste_niveaux: list, dt: float) -> float` *prenant en argument la liste liste_niveaux de la question renvoyant* H_{max}.

Afin de déterminer $H_{1/3}$ et $T_{H1/3}$, il est nécessaire de trier la liste des propriétés des vagues.

La distribution des hauteurs de vague (voir figure suivante) lors de l'analyse vague par vague est réputée être gaussienne. On peut contrôler ceci par des tests de skewness (variable désignée par S) et de kurtosis (variable désignée par K) définis ci-après. Ces deux tests permettent de quantifier respectivement l'asymétrie et l'aplatissement de la distribution.



On appelle $\overline{H}$ et σ^2 les estimateurs non biaisés de l'espérance et de la variance, n le nombre d'éléments H_1, H_2, ..., H_n.

On définit alors :

$$S = \frac{n}{(n-1)(n-2)} \times \left(\frac{1}{\sigma^3}\right) \times \sum_{i=1}^{n} \left(H_i - \overline{H}\right)^3$$

$$K = \frac{n}{(n-1)(n-2)(n-3)} \times \left(\frac{1}{\sigma^4}\right) \times \sum_{i=1}^{n} \left(H_i - \overline{H}\right)^4 - \frac{3(n-1)^2}{(n-2)(n-3)}.$$

Le test suivant est appliqué :

- si la valeur absolue de S est supérieure à 0,3 alors l'horodate est déclaré non valide ;
- si la valeur de K est supérieure à 5 alors l'horodate est déclaré non valide.

On utilise la fonction `moyenne` pour estimer la valeur de $\overline{H}$. Même s'il serait aisé de coder la fonction écart type, on utilisera la fonction `pstdev` de la bibliothèque `statistics` qui permet de retourner la valeur de l'écart type non biaisé σ.

Question 10 *Écrire une fonction* `skewness(liste_niveaux) -> float` *permettant de déterminer S.*

Algorithmes dichotomiques

Fonctions récursives

Algorithmes gloutons

Traitement d'images

Tris