7

Algorithmes et programmes importants

7.1 Recherche dans une liste

⇒ **Activité 7.35**

L'algorithme suivant recherche un élément el dans une liste t.

```
 1: VARIABLES
 2: el
 3: i : int
 4: t[1..n] : tableau
 5: ENTRÉES
 6: LIRE el.
 7: SORTIES
 8: AFFICHER si l'élément est présent ou non.
 9: DEBUT_ALGORITHME
10:     POUR i ALLANT_DE 1 A taille(t)
11:         DEBUT_POUR
12:         SI (el = t[i]) ALORS
13:             DEBUT_SI
14:             AFFICHER ("L'élément recherché a été découvert")
15:             FIN_SI
16:         SINON
17:             DEBUT_SINON
18:             AFFICHER ("L'élément recherché n'est pas présent")
19:             FIN_SINON
20:         FIN_POUR
21: FIN_ALGORITHME
```

Algorithme 21 : Recherche dans un tableau

Écrivez en *python* la fonction correspondante.

On obtient par exemple le programme *recherche-liste.py* :

7.2 Maximum et minimum d'une liste de valeurs

⇒ **Activité 7.36**

Complétez l'algorithme suivant qui recherche le maximum et le minimum d'une liste t puis écrire le programme *python* correspondant. **On créera des fonctions** en *python*.

```
 1: VARIABLES
 2: min, max : float
 3: i : int
 4: t[1..n] : tableau de nombres
 5: ENTRÉES
 6: LIRE t.
 7: SORTIES
 8: AFFICHER le maximum et le minimum
 9: DEBUT_ALGORITHME
10:   INITIALISATION
11:   .........
12:   .........
13:   POUR i ALLANT_DE ... A ...
14:     DEBUT_POUR
15:     SI (t[i] < min) ALORS
16:       DEBUT_SI
17:       .........
18:       FIN_SI
19:     SINON
20:       DEBUT_SINON
21:       SI (t[i] > max) ALORS
22:         DEBUT_SI
23:         .........
24:         FIN_SI
25:       FIN_SINON
26:     FIN_POUR
27:   AFFICHER ("Le min est: ", min)
28:   AFFICHER ("Le max est: ", max)
29: FIN_ALGORITHME
```

Algorithme 22 : Mini et maxi d'une liste

On peut aussi créer une fonction qui retourne 2 arguments (minimum et maximum) sous forme de tuple (programme *minetmax.py*) :

```
1  # -*- coding: utf-8 -*-
2
3  tab=[12,24,98,2,-4,78.567,100]
4
5  def min_et_max(t):
6      mini,maxi = t[0],t[0]
7      for i in range(len(t)):
8          if maxi < t[i]:
9              maxi = t[i]
10         elif mini > t[i]:
11             mini = t[i]
```

```python
    return(mini,maxi)

print(min_et_max(tab))
# c'est un tuple
```


7.3 Variance d'une liste de valeurs

⇒ **Activité 7.37**

Soit une liste $t = [t[1],...,t[n]]$. La variance de t est définie par la relation :

$$var = \frac{1}{n}\sum_{i=1}^{n}(t[i] - t_m)^2$$

avec t_m moyenne de t.

L'algorithme suivant recherche la moyenne et la variance d'une liste t de nombres.

```
VARIABLES
moy, var, somme, sommecar : float
i : int
t[1..n] : tableau de nombres
ENTRÉES
LIRE t.
SORTIES
AFFICHER la moyenne et la variance
DEBUT_ALGORITHME
    INITIALISATION
    somme = 0
    taille = 0
    sommecar = 0
    POUR i ALLANT_DE 1 A n
        DEBUT_POUR
        somme ← somme + t[i]
        taille ← taille + 1
        FIN_POUR
    moy ← somme/taille
    AFFICHER ("La moyenne est: ", moy)
    POUR i ALLANT_DE 1 A n
        DEBUT_POUR
        sommecar ← sommecar + (t[i] − moy)^2
        taille ← taille + 1
        FIN_POUR
    var ← sommecar/taille
    AFFICHER ("La variance est: ", var)
FIN_ALGORITHME
```

Algorithme 23 : Moyenne et variance d'une liste

Écrivez les fonctions correspondantes en *python*.
On obtient par exemple le programme *moy-var.py* :

7.4 Recherche par dichotomie dans un tableau trié

⇒ **Activité 7.38**

Complétez l'algorithme suivant puis écrivez le programme *python* correspondant permettant de trouver un élément dans un tableau trié.

```
VARIABLES
a, b, c, elem, i : int
tab[1..n] : tableau trié de nombres
ENTRÉES
LIRE elem et tab.
SORTIES
AFFICHER si elem est dans tab
DEBUT_ALGORITHME
  INITIALISATION
  a ← 1
  b ← taille(tab)
  TANT_QUE ............. FAIRE
    DEBUT_TANT_QUE
    .............
    SI ............. ALORS
      DEBUT_SI
      AFFICHER (elem, 'est trouvé')
      FIN_SI
    SINON
      DEBUT_SINON
      SI ............. ALORS
        DEBUT_SI
        .............
        FIN_SI
      FIN_SINON
    SINON
      DEBUT_SINON
      .............
      FIN_SINON
    FIN_TANT_QUE
  AFFICHER (elem, "non trouvé")
FIN_ALGORITHME
```

Algorithme 24 : Recherche dans un tableau

En *python*, on obtient par exemple le programme *recherche-tab.py* :

7.5 Recherche par dichotomie du zéro d'une fonction

⇒ **Activité 7. 39**

Soit une fonction f continue et monotone, $[a, b]$ un intervalle de recherche, p la précision, telle que par exemple, $p = 10^{-prec}$.
On souhaite rechercher le zéro de cette fonction : la recherche s'arrêtera lorsque $b - a < p$.

L'algorithme peut s'écrire par exemple comme ceci :

```
1:  VARIABLES
2:  prec : int
3:  p, a, b, c : float
4:  DEBUT_ALGORITHME
5:  | LIRE p
6:  | p ← 10^-prec
7:  | LIRE a
8:  | LIRE b
9:  | SI (f(a) * f(b) > 0) ALORS
10: | | DEBUT_SI
11: | | AFFICHER ("Pas de solution")
12: | | FIN_SI
13: | SINON
14: | | DEBUT_SINON
15: | | TANT_QUE (b − a > p) FAIRE
16: | | | DEBUT_TANT_QUE
17: | | | c ← (a + b)/2
18: | | | SI (f(a) * f(c) > 0) ALORS
19: | | | | DEBUT_SI
20: | | | | a ← c
21: | | | | FIN_SI
22: | | | SINON
23: | | | | DEBUT_SINON
24: | | | | b ← c
25: | | | | FIN_SINON
26: | | | FIN_TANT_QUE
27: | | FIN_SINON
28: | AFFICHER a
29: FIN_ALGORITHME
```

Algorithme 25 : Zéro d'une fonction par dichotomie

Écrivez le programme *python* correspondant permettant de trouver le zéro de cette fonction dans l'intervalle considéré.
Vous créerez une fonction et obtiendrez par exemple le programme *zero-dicho.py* :

7.6 Intégrales : Méthode des rectangles et des trapèzes

L'intégration numérique est un outil indispensable en physique numérique.
Les méthodes numériques d'intégration d'une fonction sont nombreuses et les techniques très diverses. Des très simples, comme la méthode des rectangles aux très complexes comme certaines variétés de la méthode de Monte-Carlo. Nous n'aborderons ici que des méthodes simples voire simplistes.

7.6.1 Méthode des rectangles

Considérons une fonction continue $f(x)$ sur un intervalle $[a, b]$. Pour un physicien, intégrer une fonction signifie calculer l'aire ou la surface sous la courbe de la fonction $f(x)$ entre a et b.
La première méthode qui vient à l'esprit, c'est de découper l'aire entre la courbe $f(x)$, l'axe des x et les droites $x = a$ et $x = b$, en une multitude de petits rectangles de largeur faible h, et de hauteur $f(h)$. L'aire sous la courbe est obtenue en sommant tous ces petits rectangles.

Nous avons le choix entre trois techniques :

- faire coïncider le sommet haut gauche du rectangle avec la courbe : c'est la méthode des rectangles à gauche,

- faire coïncider le sommet haut droit du rectangle avec la courbe : c'est la méthode des rectangles à droite,
- faire coïncider le milieu du coté haut du rectangle avec la courbe : c'est la méthode du point milieu.

Nous choisirons la dernière, la plus précise.

Posons $h = \dfrac{b-a}{n}$, où n est le nombre de rectangles avec lesquels nous allons paver l'aire à calculer. Évidemment, plus n sera grand, et plus la précision du calcul sera grande (du moins en première approche). En voici une illustration :

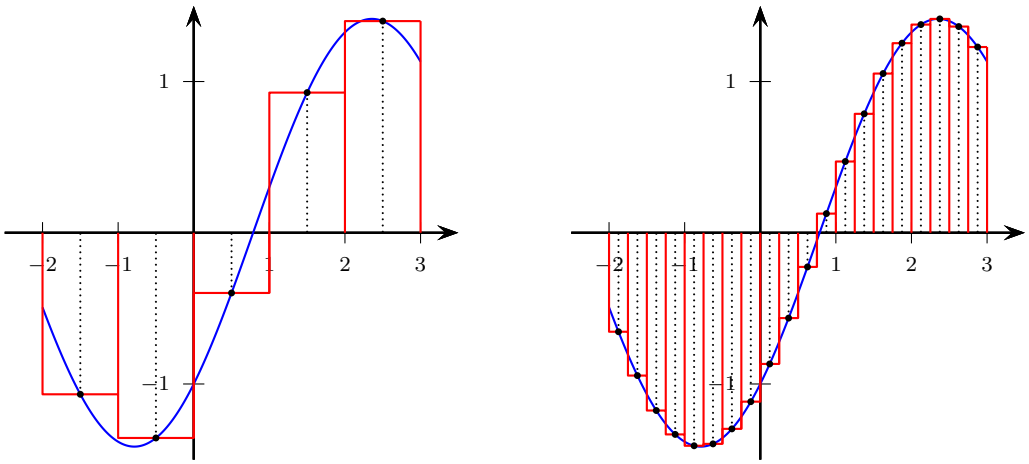

FIGURE 7.1 – Méthode des rectangles

Un rapide calcul nous montre que dans ce cas, l'intégrale, notée I, vaut :

$$I = \frac{b-a}{n} \sum_{i=0}^{n-1} f\left(\frac{x_i + x_{i+1}}{2}\right)$$

⇒ **Activité 7.40**

Soit l'algorithme suivant qui permet d'approcher une intégrale par la méthode des rectangles.
Écrivez le programme *python* correspondant à l'aide d'une fonction.

```
1:  DEBUT_ALGORITHME
2:      VARIABLES
3:      a, b, integrale : float
4:      i, n : int
5:      LIRE a
6:      LIRE b
7:      LIRE n
8:      INITIALISATION
9:      integrale ← 0
10:     pas ← (b − a)/n
11:     POUR i ALLANT_DE 0 A n − 1
12:         DEBUT_POUR
13:         integrale ← integrale + pas ∗ f(a + (i + 1/2) ∗ pas)
14:         FIN_POUR
15:     AFFICHER integrale
16: FIN_ALGORITHME
```

Algorithme 26 : Intégrale, Méthodes des rectangles

En *python*, on obtient par exemple le programme suivant *rect1.py* :

7.6.2 Méthode des trapèzes

La méthode des trapèzes est voisine de celle des rectangles. On utilise non plus des rectangles pour paver l'aire, mais des trapèzes. Ainsi, la partie du pavé qui jouxte la courbe est plus proche.

Comme plus haut, l'intervalle $[a, b]$ est partagée en n petits trapèzes de largeur $h = \dfrac{b-a}{n} = pas$.

L'aire de chaque petit trapèze est :

$$\begin{aligned} A_i &= \frac{h}{2}\,[f(x_i) + f(x_{i+1})] \\ &= \frac{h}{2}\,[f(a + i\,h) + f(a + (i+1)\,h)] \end{aligned}$$

On obtient l'aire recherchée en sommant l'aire de tous les trapèzes entre a et b, ce qui donne :

$$I = \frac{b-a}{2n}\left[f(a) + f(b) + 2\sum_{i=0}^{n-1} f(x_i)\right]$$

7.7 Recherche de la position d'un mot dans une chaîne

⇒ **Activité 7.41**

L'algorithme suivant permet de trouver la position d'un mot dans une chaîne de caractères.
Traduisez cet algorithme en langage *python*.

```
1: VARIABLES
2: taillemot, taillechaine, i, k : int
3: mot, chaine : string
4: LIRE mot
5: LIRE chaine
6: DEBUT_ALGORITHME
7:     INITIALISATION
8:     taillemot ← taille(mot)
9:     taillechaine = taille(chaine)
10:    POUR k ALLANT_DE 1 A taillechaine − taillemot + 1
11:        DEBUT_POUR
12:        i ← 0
13:        TANT_QUE (i < taillemot) FAIRE
14:            DEBUT_TANT_QUE
15:            SI (mot[i]! = chaine[k + i]) ALORS
16:                DEBUT_SI
17:                i ← taillemot
18:                FIN_SI
19:            SINON
20:                DEBUT_SINON
21:                SI (i == taillemot − 1) ALORS
22:                    DEBUT_SI
23:                    AFFICHER (k)
24:                    FIN_SI
25:                SINON
26:                    DEBUT_SINON
27:                    i ← i + 1
28:                    FIN_SINON
29:                FIN_SINON
30:            FIN_TANT_QUE
31:        FIN_POUR
32:    AFFICHER ("Le mot n'est pas présent")
33: FIN_ALGORITHME
```

Algorithme 27 : Mot dans une chaîne

En *python*, on peut écrire le programme *recherche-mot.py* :