Algorithmique et pseudo-langage

Caroline Japhet

Version du 28 septembre 2023

Table des matières

1 Notations **1**

2 Algorithmes : scripts et fonctions **3**
2.1 Généralités 3
2.2 Boucle POUR 3
2.3 Boucle TANTQUE 5
2.4 Instruction SI 6
2.5 Intérêt des fonctions 8
2.6 Fonctions prédéfinies 8

Références :

[1] F. Cuvelier, Analyse numérique élémentaire, Notes de cours Ingénieurs MACS 1ère année, 2023
https://www.math.univ-paris13.fr/∼cuvelier/

Ce document est une introduction à l'algorithmique, dont une partie est extraite de la référence [1].
Pour tester les algorithmes, on utilisera le logiciel GNU Octave (`http://www.octave.org`).

1 Notations

On utilisera le vocabulaire de base suivant :

- *donnée* : introduite par l'utilisateur
- *résultat* : ce que retourne l'algorithme (script ou fonction)
- *constante* : objet non modifiable
- *variable* : un objet dont la valeur est modifiable, qui possède un nom et un type (entier, caractère, réel, complexe, tableau, matrice, vecteur...).

L'opérateur d'affectation d'une valeur "b " (d'un type donné) à une variable "a" s'écrit :

Nom	Symbole	Exemple
affectation	$\longleftarrow$	$a \longleftarrow b$

Exemple : on veut affecter la valeur 3 à la variable a, on écrit : $a \longleftarrow 3$.

Les opérateurs arithmétiques sont notés par :

Nom	Symbole	Exemple
addition	$+$	$a + b$
soustraction	$-$	$a - b$
multiplication	$*$	$a * b$
division	$/$	a/b
puissance	$\hat{}$	$a \hat{} b$

Les opérateurs relationnels sont notés par :

Nom	Symbole	Exemple	Commentaire
identique	==	$a == b$	vrai si a et b ont même valeur, faux sinon
différent	~=	$a \sim= b$	faux si a et b ont même valeur, vrai sinon
inférieur	<	$a < b$	vrai si a est plus petit que b, faux sinon
supérieur	>	$a > b$	vrai si a est plus grand que b, faux sinon
inférieur ou égal	<=	$a <= b$	vrai si a est plus petit ou égal à b, faux sinon
supérieur ou égal	>=	$a >= b$	vrai si a est plus grand ou égal à b, faux sinon

Les opérateurs logiques sont notés par :

Nom	Symbole	Exemple	Commentaire
négation	~	$\sim a$	vrai si a est faux (ou nul), faux sinon
ou	|	$a \mid b$	vrai si a ou b est vrai (non nul), faux sinon
et	&	$a \,\&\, b$	vrai si a et b sont vrais (non nul), faux sinon

Vecteurs et tableaux 1D

Un vecteur $\boldsymbol{v} \in \mathbb{R}^n$ (ou tableau 1D) est un ensemble de n scalaires $v_i \in \mathbb{R}$, $i \in \{1, \dots, n\}$.

Il peut être représenté en ligne : $\boldsymbol{v} = (v_1, v_2, \dots, v_n)$, ou en colonne : $\boldsymbol{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}$.

La i-ème composante du vecteur $\boldsymbol{v}$ sera notée :
- ◇ v_i pour la notation mathématique.
- ◇ $v(i)$ pour la notation algorithmique.

Le premier indice est $i = 1$.

Matrices ou tableaux 2D

Une matrice $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$ (ou tableau 2D), est un ensemble de $m \times n$ scalaires $A_{i,j} \in \mathbb{R}$:

$$\mathbb{A} = \begin{pmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{pmatrix}.$$

On utilise l'abréviation $\mathbb{A} = (A_{i,j})_{1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n}$, i est l'indice de ligne, j est l'indice de colonne.
La matrice est carrée si $m = n$ (on note $\mathbb{A} \in \mathcal{M}_n(\mathbb{R})$ dans ce cas).

La i-ième ligne de $\mathbb{A}$ est $(A_{i,1},\ A_{i,2},\ \dots,\ A_{i,n})$, et la j-ième colonne de $\mathbb{A}$ est $\begin{pmatrix} A_{1,j} \\ A_{2,j} \\ \vdots \\ A_{m,j} \end{pmatrix}$

L'élément situé sur la i-ième ligne et la j-ième colonne sera noté :
- ◇ $A_{i,j}$ pour la notation mathématique.
- ◇ $A(i,j)$ pour la notation algorithmique.

Les indices commencent à $i = 1$ et $j = 1$.

Quelques rappels sur les vecteurs et les matrices

- ◇ Le produit scalaire euclidien de deux vecteurs $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ est défini par :

$$\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \sum_{i=1}^{n} u_i v_i.$$

◇ Le produit matrice-vecteur de $\mathbb{A} \in \mathcal{M}_n(\mathbb{R})$ par $\boldsymbol{u} \in \mathbb{R}^n$ est le vecteur $\boldsymbol{v} = \mathbb{A}\boldsymbol{u} \in \mathbb{R}^n$ avec

$$v_i = \sum_{j=1}^{n} A_{i,j} u_j \ , \quad i \in \{1, \ldots, n\}.$$

◇ Le produit de deux matrices $\mathbb{A}, \mathbb{B} \in \mathcal{M}_n(\mathbb{R})$ est la matrice $\mathbb{C} = \mathbb{A}\mathbb{B} \in \mathcal{M}_n(\mathbb{R})$ avec

$$C_{i,j} = \sum_{k=1}^{n} A_{i,k} B_{k,j} \ , \quad i, j \in \{1, \ldots, n\}.$$

2 Algorithmes : scripts et fonctions

2.1 Généralités

Avant d'écrire un algorithme il est important de :
- *clarifier* l'énoncé du problème
- Rechercher une *stratégie* de construction de l'algorithme, en décomposant le problème en *sous problèmes* partiels plus *simples* et que l'on peut tester *séparément* de l'algorithme global

Ensuite, l'algorithme doit être écrit de sorte que :
- le type des *données* et des *résultats* doivent être précisés
- l'algorithme doit fournir au moins un résultat
- l'algorithme doit être exécuté en un nombre *fini* d'opérations
- l'algorithme doit être écrit de façon claire et concise

Un algorithme peut-être écrit sous forme :
- d'un *script* : ensemble d'instructions qui joue le rôle de programme principal
- d'une *fonction* : ensemble d'instructions mis dans une fonction pour l'utilisateur, qui sera (en général) ensuite appelée dans le script principal

Dans la suite nous allons voir des exemples de scripts et de fonctions avec les boucles "Pour", "Tant que", et l'instruction "Si".

2.2 Boucle Pour

Ce type de boucle est utilisé lorsque la valeur d'arrêt de la boucle est **connue**. On l'utilise par exemple pour le calcul d'une somme ou d'un produit.
Remarque : on pourrait utiliser à la place une boucle Tantque, mais la boucle Pour est plus simple, plus facile à débuguer, et plus rapide.

Exemple 1 : calcul d'une somme

Soit $n \in \mathbb{N}^*$ donné. On cherche à calculer

$$s = \sum_{k=1}^{n} k^2 = 1 + 4 + 9 + \ldots + n^2$$

1. Écrire une script permettant de calculer s pour n fixé (par exemple $n = 10$).

Algorithm 2.1 Script de calcul de $\sum_{k=1}^{n} k^2$

Donnée : n : entier ($n \geqslant 1$)
Résultat : s : réel, égal à $\sum_{k=1}^{n} k^2$

```
1: n ← 10                                   ▷ valeur de la donnée n
2: s ← 0                                    ▷ initialisation de s
3: Pour k ← 1 à n faire
4:     s ← s + k * k
5: fin Pour
```

2. Écrire une fonction SOMME, prenant en paramètre d'entrée n et retournant s.

Algorithm 2.2 Fonction SOMME : calcule $\sum_{k=1}^{n} k^2$

Donnée : n : entier ($n \geqslant 1$)
Résultat : s : réel, égal à $\sum_{k=1}^{n} k^2$

```
1: Fonction s ← Somme(n)
2:     s ← 0                                   ▷ initialisation de s
3:     Pour k ← 1 à n faire
4:         s ← s + k * k
5:     fin Pour
6: fin Fonction
```

3. Réécrire le script du point 1) en utilisant la fonction SOMME.

Algorithm 2.3 Script de calcul de $\sum_{k=1}^{n} k^2$, en utilisant la fonction SOMME

Donnée : n : entier ($n \geqslant 1$)
Résultat : s : réel, égal à $\sum_{k=1}^{n} k^2$

```
1: n ← 10                                    ▷ valeur de la donnée n
2: s ← Somme(n)
```

4. Comment vérifier que le script est correct ?
   - ◇ commencer par tester avec n "petit" pour lesquel on peut calculer s à la main.
   - ◇ Puis tester pour n "quelconque" : comparer à la valeur de s si on la connait (ici : $s = \frac{n(n+1)(2n+1)}{6}$) sinon, trouver d'autres tests.

Algorithm 2.4 Script de validation de la fonction SOMME

```
1: n ← 4                                     ▷ test n petit
2: s_e ← 30                                  ▷ valeur de s calculée à la main
3: s ← Somme(n)
4: afficher (s − s_e)/s_e                    ▷ erreur relative sur s
5:
6: n ← 1000                                  ▷ test n quelconque
7: s_e ← n * (n + 1) * (2 * n + 1)/6         ▷ valeur exacte de s
8: s ← Somme(n)
9: afficher (s − s_e)/s_e                    ▷ erreur relative sur s
```

⚠ les tests permettent de trouver d'éventuelles erreurs, ils ne montrent pas que l'algorithme est correct.

Exemple 2 : calcul d'un produit

Soit $m \in \mathbb{N}^*$ donné. On cherche à calculer

$$p = \prod_{i=1}^{m} \cos(3i) = \cos(3) \times \cos(6) \times \ldots \times \cos(3m)$$

1. Écrire une script permettant de calculer p, pour m fixé (par ex. $m = 5$)

Algorithm 2.5 Script de calcul de $\prod_{i=1}^{m} \cos(3i)$

Données : m : entier ($m \geqslant 1$)
Résultat : p : réel, égal à $\prod_{i=1}^{m} \cos(3i)$

```
1: m ← 5                                     ▷ valeur de la donnée m
2: p ← 1                                     ▷ initialisation de p
3: Pour i ← 1 à m faire
4:     p ← p * cos(3 * i)
5: fin Pour
```

2. Écrire une fonction PRODUIT, prenant en paramètre d'entrée m et retournant p.

Algorithm 2.6 Fonction PRODUIT : calcule $= \prod_{i=1}^{m} \cos(3i)$

Données : m : entier $(m \geqslant 1)$
Résultat : p : réel, égal à $\prod_{i=1}^{m} \cos(3i)$

```
1: Fonction p ← PRODUIT(m)
2:     p ← 1                                   ▷ initialisation de p
3:     Pour i ← 1 à m faire
4:         p ← p * cos(3 * i)
5:     fin Pour
6: fin Fonction
```

3. Réécrire le script du point 1) en utilisant la fonction PRODUIT.

Algorithm 2.7 Script de calcul de $\prod_{i=1}^{m} \cos(3i)$, en utilisant la fonction PRODUIT

Données : m : entier $(m \geqslant 1)$
Résultat : p : réel, égal à $\prod_{i=1}^{m} \cos(3i)$

```
1: m ← 5                                     ▷ valeur de la donnée m
2: p ← PRODUIT(m)
```

4. Comment vérifier que le script est correct ?
   - ◇ commencer par tester avec m "petit" pour lesquel on peut calculer p à la main.
   - ◇ Puis tester pour m "quelconque" : ici on ne connait pas la valeur de p, on pourrait la comparer à celle que donne la fonction PROD du logiciel OCTAVE

Algorithm 2.8 Script de validation de la fonction PRODUIT

```
1: m ← 3                                     ▷ test m petit
2: p_e ← cos(3) * cos(6) * cos(9)            ▷ valeur de p calculée à la main
3: p ← PRODUIT(m)
4: afficher (p − p_e)/p_e                    ▷ erreur relative sur p
5:
6: m ← 300                                   ▷ test m quelconque
7: p_e ← PROD(m)                             ▷ valeur de p avec Octave
8: p ← PRODUIT(m)
9: afficher (p − p_e)/p_e                    ▷ erreur relative sur p
```

⚠ les tests permettent de trouver d'éventuelles erreurs, ils ne montrent pas que l'algorithme est correct

2.3 Boucle TANTQUE

Elle est en général utilisée lorsque la valeur d'arrêt de la boucle **n'est pas connue**. C'est par exemple le cas du calcul d'une suite numérique, avec un critère d'arrêt donné.

Calcul d'une suite numérique avec critère d'arrêt

Soit $q \in]-1, 1[$ donné. Considérons la suite géométrique :

$$\begin{cases} u_0 = 1 \\ u_{n+1} = q u_n, \quad \forall n \geqslant 0 \end{cases}$$

Le problème que l'on se pose est le suivant : trouver le premier entier n tel que $|u_n| < tol$, où $tol > 0$ est un réel donné (appelé "tolérance"). L'algorithme devra s'arrêter en au plus $nmax$ itérations.

1. Écrire une script permettant de calculer n, pour q, tol et $nmax$ fixés (par exemple $q = \frac{2}{3}$, $tol = 10^{-4}$, $nmax = 100$).

Algorithm 2.9 Script de calcul de n tel que $|u_n| < tol$

Données : q : réel ($q \in]-1, 1[$)
tol : réel ($tol > 0$)
$nmax$: entier ($nmax \geqslant 1$)
Résultat : n : entier, $n \in \{0, \ldots, nmax\}$, tel que $|u_n| < tol$ ou $n = nmax$

```
q ← 2/3                                          ▷ valeur de la raison q
tol ← 1e − 4                                     ▷ valeur de la donnée tol
nmax ← 100                                       ▷ valeur de la donnée nmax
n ← 0                                            ▷ initialisation de n
u ← 1                                            ▷ initialisation u = u_0
Tantque abs(u) ⩾ tol & n ⩽ nmax faire
    u ← q ∗ u
    n ← n + 1
fin Tantque
```

2. Écrire une fonction SuiteG, prenant en paramètre d'entrée q, tol, $nmax$, et retournant n.

Algorithm 2.10 Fonction SuiteG : calcule n tel que $|u_n| < tol$

Données : q : réel ($q \in]-1, 1[$)
tol : réel ($tol > 0$)
$nmax$: entier ($nmax \geqslant 1$)
Résultat : n : entier, $n \in \{0, \ldots, nmax\}$, tel que $|u_n| < tol$ ou $n = nmax$

```
Fonction n ← SuiteG(q, tol, nmax)
    n ← 0                                        ▷ initialisation de n
    u ← 1                                        ▷ initialisation u = u_0
    Tantque abs(u) ⩾ tol & n ⩽ nmax faire
        u ← q ∗ u
        n ← n + 1
    fin Tantque
fin Fonction
```

3. Réécrire le script du point 1) en utilisant la fonction SuiteG.

Algorithm 2.11 Script de calcul de n tel que $|u_n| < tol$, en utilisant la fonction SuiteG

Données : q : réel ($q \in]-1, 1[$)
tol : réel ($tol > 0$)
$nmax$: entier ($nmax \geqslant 1$)
Résultat : n : entier, $n \in \{0, \ldots, nmax\}$, tel que $|u_n| < tol$ ou $n = nmax$

```
q ← 2/3                                          ▷ valeur de la raison q
tol ← 1e − 4                                     ▷ valeur de la donnée tol
nmax ← 100                                       ▷ valeur de la donnée nmax
n ← SuiteG(q, tol, nmax)
```

⚠ Si l'algorithme s'arrête lorsque $n = nmax$, alors il n'y a aucune garantie que $|u_n| < tol$

2.4 Instruction Si

L'instruction conditionnelle Si est une instruction qui n'est exécutée que si une condition est validée. Il est possible d'ajouter une instruction alternative avec Sinon.
Exemples : calcul de la valeur absolue d'un réel, tester si un réel est dans $\mathbb{N}$, ...

Exemple 1 : calcul de la valeur absolue d'un réel

Soit $x \in \mathbb{R}$. On cherche à calculer

$$|x| = \begin{cases} x & \text{si } x \geqslant 0 \\ -x & \text{si } x \leqslant 0 \end{cases}$$

1. Écrire une script permettant de calculer $|x|$, pour x fixé (par exemple $x = -0.3$).

Algorithm 2.12 Script de calcul de la valeur absolue d'un réel

Donnée : x : réel
Résultat : a : réel positif, tel que $a = |x|$

```
1: x ← −0.3
2: Si x ⩾ 0 alors
3:     a ← x
4: Sinon
5:     a ← −x
6: fin Si
```

2. Écrire une fonction VABS, prenant en paramètre d'entrée x et retournant $|x|$.

Algorithm 2.13 Fonction VABS : calcule la valeur absolue d'un réel

Donnée : x : réel
Résultat : a : réel positif, tel que $a = |x|$

```
1: Fonction a ← VABS(x)
2:     Si x ⩾ 0 alors
3:         a ← x
4:     Sinon
5:         a ← −x
6:     fin Si
7: fin Fonction
```

3. Réécrire le script du point 1) en utilisant la fonction VABS.

Algorithm 2.14 Script de calcul de la valeur absolue d'un réel, en utilisant la fonction VABS

Donnée : x : réel
Résultat : a : réel positif, tel que $a = |x|$

```
1: x ← −0.3
2: a ← VABS(x)
```

Exemple 2 : tester si un réel est dans $\mathbb{N}$

Soit $x \in \mathbb{R}$. Écrire un script retournant la valeur x, si $x \in \mathbb{N}$, et 0 sinon.
On pourra utiliser la fonction FLOOR qui retourne la partie entière q d'un réel x ($q \leftarrow$ FLOOR(x)).

Algorithm 2.15 Script pour tester si un réel est dans $\mathbb{N}$

Donnée : x : réel
Résultat : n : entier naturel, tel que $n = x$ si $x \in \mathbb{N}$, et $n = 0$ sinon

```
1: Si (x >= 0) & (FLOOR(x) == x) alors
2:     n ← x
3: Sinon
4:     n ← 0
5: fin Si
```

2.5 Intérêt des fonctions

Les *fonctions* permettent
- d'automatiser certaines tâches répétitives au sein d'un même algorithme
- d'ajouter de la clarté à l'algorithme
- d'utiliser une portion de code dans un autre algorithme
- de valider plus efficacement un algorithme (en testant d'abord séparément chaque fonction, puis l'algorithme complet).

Un exemple

On cherche à calculer

$$a = \left| \prod_{i=1}^{m} \cos(3i) \right|$$

1. Écrire une script (sans utiliser de fonction) permettant de calculer a, pour m fixé (par exemple $m = 5$).

Algorithm 2.16 Script de calcul de $\left|\prod_{i=1}^{m} \cos(3i)\right|$, sans utiliser de fonction

Données : m : entier ($m \geqslant 1$)
Résultat : a : réel positif, égal à $\left|\prod_{i=1}^{m} \cos(3i)\right|$

```
1: m ← 5                                   ▷ valeur de la donnée m
2: p ← 1                                   ▷ initialisation de p
3: Pour i ← 1 à m faire
4:     p ← p * cos(3 * i)
5: fin Pour
6: Si p ⩾ 0 alors
7:     a ← p
8: Sinon
9:     a ← −p
10: fin Si
```

⚠ pour des problèmes plus complexes, l'algorithme deviendra très vite illisible et difficile à valider.

2. Réécrire le script du point 1) en utilisant les fonctions PRODUIT et VABS.

Algorithm 2.17 Script de calcul de $\left|\prod_{i=1}^{m} \cos(3i)\right|$, en utilisant les fonctions PRODUIT et VABS

Données : m : entier ($m \geqslant 1$)
Résultat : a : réel positif, égal à $\left|\prod_{i=1}^{m} \cos(3i)\right|$

```
1: m ← 5                                   ▷ valeur de la donnée m
2: p ← PRODUIT(m)
3: a ← VABS(p)
```

Cet algorithme est clair, concis, avec des fonctions PRODUIT et VABS qui peuvent-être validées séparément.

2.6 Fonctions prédéfinies

Les *fonctions prédéfinies* sont :

- les fonctions mathématiques : SIN, COS, EXP, LOG, SQRT, ...
- la fonction LENGTH : retourne la longueur n d'un vecteur $\boldsymbol{u} \in \mathbb{R}^n$
  $n \leftarrow$ `length`$(\boldsymbol{u})$
- la fonction SIZE : retourne la dimension (m, n) d'une matrice $\mathbb{A} \in \mathcal{M}_{m,n}(\mathbb{R})$
  $[m, n] \leftarrow$ `size`$(\mathbb{A})$
- la fonction RAND : retourne un vecteur colonne $\boldsymbol{v}$ de $\mathbb{R}^n$, ou une matrice $\mathbb{A}$ carrée de taille n, contenant des valeurs aléatoires dans $[0, 1]$
  $\boldsymbol{v} \leftarrow$ `rand`$(n, 1)$
  $\mathbb{A} \leftarrow$ `rand`(n)
- la fonction FLOOR : retourne la partie entière q d'un réel x
  $q \leftarrow$ FLOOR(x)