Analyse asymptotique et complexité

Estimation de l'efficacité d'un algorithme

Matteo Wei

Talens

2 décembre 2023

Plan

Fonctions exponentielles

Proposition (admise)

Soit $a \in \mathbb{R}_+^*$. Alors il existe une unique fonction monotone $e_a : \mathbb{R} \to \mathbb{R}_+^*$ telle que pour tout $q \in \mathbb{Q}$, $e_a(q) = a^q$. Si $x \in \mathbb{R}$, on notera $a^x = e_a(x)$. On a alors $\forall x, y \in \mathbb{R}, a^{x+y} = a^x a^y$.

Remarque

Elle est bijective si $a \neq 1$, strictement croissante si $a > 1$, strictement décroissante si $a < 1$.

Fonctions logarithmes

Définition

Soit $a \in]1, +\infty[$. On note $\log_a : \mathbb{R}_+^* \to \mathbb{R}$ la réciproque de e_a.
Dans ce cours seulement, on note $\log = \log_2$.

Proposition

Soit $a \in]1, +\infty[$. Alors $\log_a(1) = 0$, et $\log_a$ est strictement monotone et vérifie $\forall x, y \in \mathbb{R}_+^*, \log_a(xy) = \log_a(x) + \log_a(y)$.

Proposition

Soit $a, b \in]1, +\infty[$. Alors on a $\log_b = \log_b(a) \log_a$.

Suites

Définition

Soit E un ensemble. On appelle *suite d'éléments de* E une famille d'éléments de E indexée par $\mathbb{N}$. On l'identifie alors à la fonction $\mathbb{N} \to E$ dont c'est le graphe.

Définition

On note $E^{\mathbb{N}}$ l'ensemble de ces suites.

Remarque

D'une manière générale, si E et F sont deux ensembles, on note F^E l'ensemble des fonctions $E \to F$.

Inégalité triangulaire

Proposition, inégalité triangulaire

Soit $x, y \in \mathbb{R}$. Alors $|x + y| \leqslant |x| + |y|$.

Limites

Définition

Soit $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$, et $l \in \mathbb{R}$. On dit que $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ *converge* (ou *tend*) vers l, et on note $u_n \xrightarrow[n\to+\infty]{} l$, si pour tout $\varepsilon \in \mathbb{R}_+^*$, il existe $N \in \mathbb{N}$ tel que pour tout $n \geqslant N$, on a $|u_n - l| \leqslant \varepsilon$.
Avec des quantificateurs,
$\forall \varepsilon \in \mathbb{R}_+^*, \exists N \in \mathbb{N} : \forall n \in \mathbb{N}, n \geqslant N \Rightarrow |u_n - l| \leqslant \varepsilon$
On dit alors que l est une *limite* de la suite $(u_n)_{n\in\mathbb{N}}$.

Remarque

À partir de maintenant, on dira « pour n assez grand », pour dire « il existe $N \in \mathbb{N}$ tel que pour tout $n \geqslant N$ ».

Limites

Exemples

- $\frac{1}{2^n} \xrightarrow[n\to+\infty]{} 0.$
- $2^n \xrightarrow[n\to+\infty]{} +\infty.$
- $\log(n) \xrightarrow[n\to+\infty]{} +\infty.$
- $\frac{n}{n+1} \xrightarrow[n\to+\infty]{} 1.$

Limites

Définition

On dit qu'une suite est *convergente* si elle admet une limite réelle, divergente sinon.

Définition

On dit qu'une suite $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$ *diverge vers* $+\infty$ (resp. $-\infty$) si pour tout $C \in \mathbb{R}$, pour n assez grand on a $u_n \geqslant C$ (resp. $u_n \leqslant C$).

Définition

On pose $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.

Unicité de la limite

Proposition, unicité de la limite

Soit $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$, et $l, m \in \overline{\mathbb{R}}$, tel que $u_n \xrightarrow[n\to+\infty]{} l$ et $u_n \xrightarrow[n\to+\infty]{} m$. Alors $l = m$.

Ainsi, on peut parler de *la* limite de $(u_n)_{n\in\mathbb{N}} \in \mathbb{R}^{\mathbb{N}}$, et on la notera $\lim\limits_{n\to+\infty} u_n$.

Attention !

Une suite n'a pas de raisons d'être convergente ou de diverger vers $\pm\infty$, et peut ne pas avoir de limite.

Les opérations sont continues

Proposition

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}} \in \mathbb{R}^n$ et $l, m \in \overline{\mathbb{R}}$, tels que $u_n \xrightarrow[n\to+\infty]{} l$ et $v_n \xrightarrow[n\to+\infty]{} m$.
Alors $-u_n \xrightarrow[n\to+\infty]{} -l$, et, sauf dans le cas où $l = \pm\infty$ et $m = -l$, on a $u_n + v_n \xrightarrow[n\to+\infty]{} l + m$.
($+$ et $-$ étant les prolongements naturels à $\overline{\mathbb{R}}$ des opérations dans les cas non tordus).

Les opérations sont continues

Proposition

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}} \in \mathbb{R}^n$ et $l, m \in \overline{\mathbb{R}}$, tels que $u_n \xrightarrow[n\to+\infty]{} l$ et $v_n \xrightarrow[n\to+\infty]{} m$.
Alors, sauf dans le cas où $l = \pm\infty$ et $m = 0$ (ou réciproquement), on a $u_n v_n \xrightarrow[n\to+\infty]{} lm$.

Par ailleurs, si $\forall n \in \mathbb{N}, u_n \neq 0$, et $l \neq 0$, alors $\dfrac{1}{u_n} \xrightarrow[n\to+\infty]{} \dfrac{1}{l}$.

Remarque

Dans le cas où la suite ne s'annule jamais et est de signe constant $\varepsilon \in \{-1, 1\}$, et $l = 0$, on a $\dfrac{1}{u_n} \xrightarrow[n\to+\infty]{} \varepsilon\infty$.

Exercices

Exercice

Discuter de la convergence des suites suivantes, et calculer leurs limites si elles existent.

1. $\left(\frac{1}{n}\right)_{n\in\mathbb{N}^*}$.
2. $((-1)^n)_{n\in\mathbb{N}}$.
3. $\left(\frac{n-1}{n+1}\left(2-\frac{1}{n^2}\right)\right)_{n\in\mathbb{N}}$.
4. $(a^n)_{n\in\mathbb{N}}$, avec $a \in \mathbb{R}_+$.

Passage à la limite des inégalités larges

Proposition

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}} \in \mathbb{R}^n$ et $l, m \in \overline{\mathbb{R}}$, tels que $u_n \xrightarrow[n\to+\infty]{} l$, $v_n \xrightarrow[n\to+\infty]{} m$ et $\forall n \in \mathbb{N}, u_n \leqslant v_n$. Alors $l \leqslant m$.

Attention !

Les inégalités strictes ne passent pas forcément à la limite.

Théorème des gendarmes

Théorème

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}}, (w_n)_{n\in\mathbb{N}} \in \mathbb{R}^n$ et $l \in \overline{\mathbb{R}}$, tels $u_n \xrightarrow[n\to+\infty]{} l$, $w_n \xrightarrow[n\to+\infty]{} l$ et $\forall n \in \mathbb{N}, u_n \leqslant v_n \leqslant w_n$. Alors $v_n \xrightarrow[n\to+\infty]{} l$.

O, o, $\sim$: définitions

Définitions

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}} \in {\mathbb{R}_+^*}^{\mathbb{N}}$.

- On note $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$ s'il existe $C \in \mathbb{R}_+^*$ tel que pour n assez grand, $u_n \leqslant Cv_n$.
- On note $\dfrac{u_n}{v_n} \xrightarrow[n\to+\infty]{} 0$.
- On dit que $(u_n)_{n\in\mathbb{N}}$ et $(v_n)_{n\in\mathbb{N}}$ sont *équivalentes*, et on note $u_n \underset{n\to+\infty}{\sim} v_n$, si $\dfrac{u_n}{v_n} \xrightarrow[n\to+\infty]{} 1$.

O, o, $\sim$: définitions

Proposition

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}} \in {\mathbb{R}_+^*}^{\mathbb{N}}$. Alors $u_n \underset{n\to+\infty}{\sim} v_n$ si, et seulement si, $u_n = v_n + \underset{n\to+\infty}{\mathrm{o}}(v_n)$.

Remarque

Soit $(u_n)_{n\in\mathbb{N}} \in {\mathbb{R}_+^*}^{\mathbb{N}}$ et $l \in \mathbb{R}_+$. Alors $(u_n)_{n\in\mathbb{N}}$ est bornée ssi $u_n = \underset{n\to+\infty}{\mathrm{O}}(1)$, et il est équivalent de demander que $u_n \xrightarrow[n\to+\infty]{} l$, que $u_n = l + \underset{n\to+\infty}{\mathrm{o}}(1)$, ou que $u_n \underset{n\to+\infty}{\sim} l$ si $l > 0$.

O, o, $\sim$: propriétés de base

Proposition

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}}, (w_n)_{n\in\mathbb{N}} \in {\mathbb{R}_+^*}^{\mathbb{N}}$. Alors :

- Si $u_n \underset{n\to+\infty}{\sim} v_n$ et $v_n \underset{n\to+\infty}{\sim} w_n$, alors $u_n \underset{n\to+\infty}{\sim} w_n$ (transitivité, aussi vrai pour O et o).
- Si $u_n \underset{n\to+\infty}{\sim} v_n$, alors $v_n \underset{n\to+\infty}{\sim} u_n$ (symétrie).
- $u_n \underset{n\to+\infty}{\sim} u_n$ (réflexivité, aussi vrai pour o).
- Si $u_n \underset{n\to+\infty}{\sim} v_n$, alors $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$.
- Si $u_n = \underset{n\to+\infty}{\mathrm{o}}(v_n)$, alors $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$.

Opérations

Proposition

Soit $(u_n)_{n\in\mathbb{N}}, (v_n)_{n\in\mathbb{N}}, (w_n)_{n\in\mathbb{N}}, (x_n)_{n\in\mathbb{N}} \in {\mathbb{R}_+^*}^{\mathbb{N}}$. Alors :

- Si $u_n \underset{n\to+\infty}{\sim} v_n$ alors $u_n w_n \underset{n\to+\infty}{\sim} v_n w_n$.
- Si $u_n \underset{n\to+\infty}{\sim} v_n$ alors $\dfrac{1}{u_n} \underset{n\to+\infty}{\sim} \dfrac{1}{v_n}$. Pour o et O, on renverse.
- Si $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$, et $w_n = \underset{n\to+\infty}{\mathrm{O}}(x_n)$ alors $u_n w_n = \underset{n\to+\infty}{\mathrm{O}}(v_n x_n)$.

Opérations

Proposition

- Si $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$, et $w_n = \underset{n\to+\infty}{\mathrm{o}}(x_n)$ alors $u_n w_n = \underset{n\to+\infty}{\mathrm{o}}(v_n x_n)$.
- Si $u_n \underset{n\to+\infty}{\sim} v_n$, alors $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$.
- Si $u_n = \underset{n\to+\infty}{\mathrm{o}}(v_n)$, alors $u_n = \underset{n\to+\infty}{\mathrm{O}}(v_n)$.
- Si $u_n \underset{n\to+\infty}{\sim} v_n$ et $w_n \underset{n\to+\infty}{\sim} x_n$, alors $u_n + w_n \underset{n\to+\infty}{\sim} v_n + x_n$ (aussi vrai avec o et O, l'hypothèse de positivité est importante).

Échelle asymptotique : $n \mapsto n^\alpha$

Proposition

Soit $\alpha, \beta \in \mathbb{R}_+$ tels que $\alpha < \beta$. Alors $n^\alpha = \underset{n \to +\infty}{o}(n^\beta)$.

Corollaire

Soit $a_0, a_1, \ldots, a_k \in \mathbb{R}$, avec $a_k > 0$.
Alors $a_k n^k + \cdots + a_1 n + a_0 \underset{n \to +\infty}{\sim} a_k n^k$.

Exercices

Exercices

1. Trouver un équivalent simple de
$$\left(\frac{27n^{42}+3n^9+n^4+22}{2n^8+5n^8-1}\left(4-\frac{1}{3n+2}\right)\right)_{n\in\mathbb{N}}$$
2. Soit $n \in \mathbb{N}$, on note $S_n = 1 + \cdots + n$. Calculer S_n, en donner un équivalent simple.
3. Plus généralement, si $\alpha \in \mathbb{R}_+$ et $n \in \mathbb{N}$, on note $S_{\alpha,n} = 1 + \cdots + n^\alpha$. Montrer que $S_{\alpha,n} = \underset{n\to+\infty}{\mathrm{O}}(n^{\alpha+1})$ et $n^{\alpha+1} = \underset{n\to+\infty}{\mathrm{O}}(S_{\alpha,n})$.

Échelle asymptotique : exponentielles

Proposition

Soit $\alpha \in \mathbb{R}_+^*$. Alors $\log(n) = \underset{n\to+\infty}{\mathrm{o}}(n^\alpha)$.

Proposition

Soit $a \in]1, +\infty[$ et $\alpha \in \mathbb{R}_+^*$. Alors $n^\alpha = \underset{n\to+\infty}{\mathrm{o}}(a^n)$.

Échelle asymptotique : logarithmes

Proposition

Soit $\alpha \in \mathbb{R}_+^*$. Alors $\log(n) = \underset{n\to+\infty}{\mathrm{o}}(n^\alpha)$.

Exercice

Soit $n \in \mathbb{N}$. On note $H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$. Minorer et majorer, si $n \in \mathbb{N}$, $H_{2n} - H_n$. En déduire que $H_n = \underset{n\to+\infty}{\mathrm{O}}(\log(n))$ et $\log(n) = \underset{n\to+\infty}{\mathrm{O}}(H_n)$.

Échelle asymptotique : logarithmes

Remarque

Avec des méthodes plus avancées, on peut obtenir de meilleurs *développements asymptotiques* de $(H_n)_{n\in\mathbb{N}}$. Par exemple, il existe deux constantes e et γ telles que

$$H_n = \log_e(n) + \gamma + \frac{1}{2n} + \underset{n\to+\infty}{\mathrm{O}}\left(\frac{1}{n^2}\right).$$

e apparaît dans d'autres équivalents très variés. On a par exemple $n! = 1 \times \cdots \times n \underset{n\to+\infty}{\sim} \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$, ou encore, si π est la fonction qui à n associe le nombre de nombres premiers $\leqslant n$, $\pi(n) \underset{n\to+\infty}{\sim} \frac{n}{\log_e(n)}$ (théorème des nombres premiers).

Évaluation d'un algorithme

Problématique

Si on considère un algorithme, comment évaluer si c'est une bonne solution à un problème donné ?

Éléments à prendre en compte

- Est-t-il correct ? Les résultats renvoyés sont-ils toujours bons ?
- Termine-t-il toujours ? Est-ce qu'il existe des entrées sur lesquelles il tourne sans fin ?
- Dans le cas précédent, que coûte-t-il en temps et en mémoire ?

Complexité en temps, en espace

Définition

Considérons un algorithme (ou une fonction au sens informatique). On définit ;

- Sa *complexité temporelle* comme le nombre d'opérations élémentaires nécessaires à son exécution, en fonction des arguments.
- Sa *complexité spatiale* comme la mémoire maximale occupée en plus des arguments lors de l'exécution, en fonction des arguments.

Complexité en temps, en espace

Remarque

Ces complexités sont presque toujours données sous la forme $\mathrm{O}(f(n))$, pour une certaine fonction f des arguments. En effet, on ne s'amuse en général pas à aller calculer le nombre précis d'opérations que prennent les calculs.

Définition

En particulier, on parlera de complexité :

- *Constante* pour $\mathrm{O}(1)$
- *Logarithmique* pour $\mathrm{O}(\log(n))$

Complexité en temps, en espace

Définition

En particulier, on parlera de complexité :

- *Linéaire* pour $\mathrm{O}(n)$
- *Quasilinéaire* pour $\mathrm{O}(n \log(n))$
- *Quadratique* pour $\mathrm{O}(n^2)$
- *Polynomiale* si elle est $\mathrm{O}(n^k)$ pour un certain $k \in \mathbb{N}^*$
- *Exponentielle* si elle est $\mathrm{O}(a^n)$ pour un certain $a \in]1, +\infty[$.

Ordres de grandeur

n	$\log(n)$	n	$n\log(n)$	n^2	2^n
10	$3 \cdot 10^{-10}s$	$10^{-9}s$	$3 \cdot 10^{-9}s$	$10^{-8}s$	$10^{-7}s$
$5 \cdot 10^1$	$6 \cdot 10^{-10}s$	$5 \cdot 10^{-9}s$	$3 \cdot 10^{-8}s$	$3 \cdot 10^{-7}s$	10^5s
10^2	$7 \cdot 10^{-10}s$	$10^{-8}s$	$7 \cdot 10^{-8}s$	$10^{-6}s$	$10^{20}s$
10^3	$1 \cdot 10^{-9}s$	$10^{-7}s$	$10^{-6}s$	$10^{-4}s$	-
10^6	$2 \cdot 10^{-9}s$	$10^{-4}s$	$2 \cdot 10^{-2}s$	10^2s	-
10^9	$3 \cdot 10^{-9}s$	$10^{-1}s$	$3s$	10^8s	-

Remarque

Le Big Bang a eu lieu il y a $4 \times 10^{17}s$.

NoneType

Description

Consiste en le seul et unique objet `None`. On l'utilise pour donner une valeur défaut aux choses.

Booléens : `Bool`

Description

Deux valeurs possibles : `True` et `False`. Les opérations `and`, `or`, `not`, et les comparaisons `==`, `!=`, `<`, `<=`, `>`, `>=` sont en $O(1)$.

Entiers : `Int`

Description

Les entiers python sont typiquement stockés en base 2 sur 32 ou 64 bits. Cependant Python alloue plus de place (logarithmiquement en la taille de l'entier) si un entier dépasse les bornes. Si ce n'est pas le cas, `+`, `-`, `*`, `//`, `%` sont en $O(1)$. Le calcul de puissances, avec `x ** n`, est en $O(\log(n))$.
Un entier est vu, au besoin, comme un flottant.

Flottants : `Float`

Description

Les flottants sont stockés en notation scientifique en base 2. Les opérations sont les mêmes que pour les entiers, plus `/`, et sont toutes en $O(1)$.

Attention !

À cause de leur représentation, les flottants sont susceptibles aux erreurs d'arrondi, qui rendent les comparaisons problématiques. Par exemple, Python évalue `0.1 + 0.2 == 0.3` à `False`. Ainsi, il vaut mieux toujours privilégier l'utilisation d'entiers, si possible.

Listes : `List`

Description

Une liste est une suite finie ordonnée d'objets, notés dans l'ordre entre crochets. On peut étant donné une liste `l` de longueur n :

- Accéder à n avec `len(l)`, en $O(1)$
- Accéder à son i-ème élément (l'indexation commence à 0), avec `l[i]`, en $O(1)$.
- Accéder au $n - i$-ème élément avec avec `l[-i]`, en $O(1)$.
- Tester si `x` est un élément de `l`, avec `x in l`, en $O(n)$.
- Ajouter `x` en fin de liste avec `l.append(x)`, en $O(1)$.

Listes : `List`

Description

- Supprimer le dernier élément, et le renvoyer, avec `l.pop()`, en $O(1)$.
- Insérer `x` en position i avec `l.insert(i)`, en $O(n-i)$.
- Supprimer son i-ème élément et le renvoyer avec `l.pop(i)`, en $O(n-i)$.
- Remplacer son i-ème élément par `x` en faisant `f[i] = x`, en $O(1)$.

Chaînes de caractères : `String`

Description

Suites de caractères, entre ' ou ". Supportent à peu près toutes les opérations des listes qui ne modifient pas leurs arguments. Peuvent aussi être comparées pour l'ordre lexicographique, en O du minimum des longueurs des deux chaînes comparées.

Syntaxe : boucles

Boucle `while`

```
while condition:
    ...
```

Boucle `for`

```
for i in iterable:
    ...
```

Fonction `range`

En particulier, `for i in range(m, n):` permet d'itérer sur $m, m+1, \ldots, n-1$.

Syntaxe : tests

Tests : if, elif, else

```
if condition0:
    ...
elif condition1:
    ...
...
elif conditionk:
    ...
else:
    ...
```

Syntaxe : fonctions

Déclaration de fonctions : def

```
def f(x1, ..., xn):
    ...
    return y
```

Remarque

`f` peut très bien s'appeler elle-même dans sa définition. On parle alors de fonction *récursive*.

Remarque

S'il n'y a pas de `return`, `None` est renvoyé en pratique.

Calcul de puissances

Algorithme non récursif

```
def pow(a, n):
  x = 1
  for i in range(n):
    x = a * x
  return x
```

Calcul de puissances

Algorithme récursif

```
def pow(a, n):
  if n == 0:
    return 1
  return a * pow(a, n - 1)
```

Calcul de puissances

Algorithme récursif (meilleur)

```
def pow(a, n):
  if n == 0:
    return 1
  if n % 2 == 0:
    return pow(a, n // 2) ** 2
  return a * pow(a, n // 2) ** 2
```

Appartenance d'un élément à une liste

Algorithme

```
def is_in_list(x, l):
  for y in l:
    if x == y:
      return True
  return False
```

Appartenance d'un élément à une liste triée

Algorithme (recherche dichotomique)

```
def is_in_sorted_list(x, l):
  i, j = 0, len(l) - 1
  while j > i:
    k = (i + j) // 2
    if l[k] == x:
      return True
    elif l[k] < x:
      j = k - 1
    else:
      i = k + 1
  return False
```

Insertion d'un élément dans une liste

Algorithme

```
def insert(x, l, i):
  n = len(l)
  l.append(None)
  for k in range(1, n - i):
    l[-k] = l[-(k + 1)]
  l[i] = x
```

Suite de Fibonacci

Définition

On définit la *suite de Fibonacci* $(F_n)_n$ par récurrence comme l'unique suite telle que $F_0 = 0$, $F_1 = 1$, et pour tout $n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.

Remarque

On peut montrer que $F_n = \frac{1}{\sqrt{5}} \left(\frac{1+\sqrt{5}}{2} \right)^n + \underset{n \to +\infty}{\mathrm{o}}(1)$.

Suite de Fibonacci

Algorithme

```
def fibo(n):
  if n == 0 or n == 1:
    return n
  return fibo(n - 1) + fibo(n - 2)
```

Suite de Fibonacci

Algorithme (meilleur)

```
def fibo(n):
  x, y = 0, 1
  for i in range(n):
    x, y = y, x + y
  return x
```