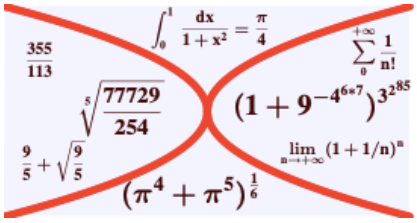
SUITES NUMERIQUES



I- Le raisonnement par récurrence

Soit n_0 un entier naturel. On cherche à démontrer **par récurrence** qu'une propriété P_n est vraie pour tout entier naturel $n \geq n_0$. On procède en **trois étapes :**

Première étape : initialisation : on vérifie que P_n est vraie pour le premier entier n_0

Deuxième étape : on suppose qu'il existe un entier naturel n pour lequel la proposition P_n est vraie, et sous cette hypothèse, on démontre que la proposition P_{n+1} l'est aussi. La propriété P_n est donc **héréditaire**

Troisième étape : **Conclusion** :

Par hérédité, la proposition étant vraie au rang n_0, elle est vraie pour tout entier $n \geq n_0$

Un exemple :

Soit (v_n) la suite définie par $\begin{cases} v_0 = 4 \\ v_{n+1} = 2v_n - 7 \end{cases}$. Démontrer par récurrence que $v_n = 7 - 3 \times 2^n$ pour tout $n \geq 0$

- **Initialisation :** pour n = 0, $7 - 3 \times 2^n = 7 - 3 \times 2^0 = 7 - 3 = 4 = v_0$ donc propriété vraie au rang 0
- **Supposons** qu'il existe un entier n tel que $v_n = 7 - 3 \times 2^n$. **Démontrons que** $v_{n+1} = 7 - 3 \times 2^{n+1}$

  Par hypothèse, $v_n = 7 - 3 \times 2^n$

  donc $2v_n - 7 = 2(7 - 3 \times 2^n) - 7$

  $v_{n+1} = 14 - 3 \times 2^{n+1} - 7$

  $v_{n+1} = 7 - 3 \times 2^{n+1}$

  La propriété P_n est donc héréditaire
- **Conclusion** : La propriété P_n étant vraie au rang 0, par hérédité, elle l'est pour tout $n \in \mathbb{N}$.

II- Comportement global d'une suite

a) Monotonie d'une suite

Soit (U_n) une suite de nombres réels

- La suite (U_n) est dite **croissante** à partir du rang n_0 lorsque pour tout entier $n \geq n_0$, $U_n \leq U_{n+1}$
- La suite (U_n) est dite **décroissante** à partir du rang n_0 lorsque pour tout entier $n \geq n_0$, $U_n \geq U_{n+1}$
- La suite (U_n) est dite **monotone** à partir du rang n_0 si elle est croissante ou décroissante pour $n \geq n_0$
- La suite (U_n) est dite :
  - **stationnaire** lorsque $U_{n+1} = U_n$ pour tout entier $n \geq n_0$
  - **constante** lorsque $U_{n+1} = U_n$ pour tout entier n

Méthode: Trois méthodes permettent d'étudier le sens de variation d'une suite :

- **La méthode algébrique :** elle consiste à comparer u_n et u_{n+1} :

  - Soit en étudiant le signe de la différence $u_{n+1} - u_n$

  - Soit en comparant le quotient $\frac{u_{n+1}}{u_n}$ à 1 si pour tout entier naturel n, $u_n > 0$

- **La méthode fonctionnelle :** elle s'applique aux suites définies de manière explicite de la forme $u_n = f(n)$ (f étant une fonction) et consiste à étudier le sens de variation de f sur $[0;+\infty[$. Le sens de variation de u_n s'en déduit
- **La méthode du raisonnement par récurrence :** Elle s'applique aux suites définies de manière récurrente de la forme $u_{n+1} = f(u_n)$. L'hérédité consiste à démontrer que :
  - si $u_n < u_{n+1}$ alors $u_{n+1} < u_{n+2}$ (suite croissante)
  - si $u_n < u_{n+1}$ alors $u_{n+1} > u_{n+2}$ (suite décroissante)

b) **Suites minorées , majorées , bornées**

Suite minorée, majorée, bornée

Soit une suite $(U_n)_{n\geq n0}$.

- La suite est dite **majorée** s'il existe un réel M tel que : pour tout $n \geq n_0$, $U_n \leq M$
- La suite est dite **minorée** s'il existe un réel m tel que : pour tout $n \geq n_0$, $U_n \geq m$
- La suite est dite **bornée** si elle est minorée et majorée

Remarque :

- Si une suite est minorée alors elle possède une infinité de minorant. On cherche alors souvent le plus grand
- Si une suite est majorée alors elle possède une infinité de majorant . On cherche alors souvent le plus petit

c) **Algorithme**

En langage python

Soit (u_n) la suite définie par $\begin{cases} u_0 = 1 \\ u_{n+1} = 2u_n + 3n \end{cases}$.

Algorithme 1	Algorithme 2	Algorithme 3
def n_termes(n) : u = 1 for i in range(n) : u = 2*u+3*i print u	def nieme_terme(n) : u = 1 for i in range(n) : u = 2*u+3*i print u	def nieme_terme(n) : u = 1 while i < n : u = 2*u+3*i i = i + 1 print u
On affiche **les n premiers** termes de la suite	On affiche **LE n-ième** terme de la suite à l'aide de deux instructions de boucle différentes : le Pour et le Tant Que. La différence avec l'algorithme 1 vient dans la position du " print u "	

Utilisations des listes

Une liste peut être définie de deux façons :

- **avec la commande append (*ajouter*)**

```
L=[ ]
for i in range(10) :
    L.append(i**2)
```

- **en « compréhension » :**

```
L = [ n**2 for n in range(10)]
```

Dans les deux cas ci-dessus, la liste L contient la liste des 10 premiers carrés parfaits .

Un " print (L) " permet d'obtenir : L = [0 , 1 , 4 , 9 , 16 , 25 , 36 , 49 , 64 , 81]