Algorithmique et programmation – Terminale spécialité

Chapitre 5 – Approcher une limite ou une constante

Pr. Elakili · Algorithme · Python · TI-82 · Octobre 2026

Site web : missionmaths360.com

Méthode

Si (u_n) converge vers ℓ, on dit que u_n est une **valeur approchée de ℓ à ε près** lorsque $|u_n - \ell| < \varepsilon$. Pour trouver le premier rang où c'est le cas, on fait une **recherche de seuil** (chapitre 4) sur l'**écart** : on répète **tant que l'écart est** $\geqslant \varepsilon$. On prend en général $\varepsilon = 10^{-p}$ (« p décimales »).

Pourquoi la boucle s'arrête : comme $\lim u_n = \ell$, l'écart $|u_n - \ell|$ finit par être plus petit que tout nombre $\varepsilon > 0$.

Pack 1 – limite connue

$u_0 = 2$, $u_{n+1} = 0{,}5\,u_n + 3$; on a montré que $u_n = 6 - 4 \times 0{,}5^n$, donc (u_n) croît vers 6

Algorithme

Saisir p
$n \leftarrow 0$; $u \leftarrow 2$
Tant que $6 - u \geqslant 10^{-p}$
  $u \leftarrow 0{,}5\,u + 3$
  $n \leftarrow n + 1$
Fin Tant que
Afficher n

Python

```
def rang(p):
    n = 0
    u = 2
    while 6 - u >= 10**(-p):
        u = 0.5*u + 3
        n = n + 1
    return n
```

TI-82

```
:Prompt P
:0→N
:2→U
:TantQue 6-U≥10^(-P)
:0.5U+3→U
:N+1→N
:Fin
:Disp N
```

p	1	2	3	6
`rang(p)`	6	9	12	22

Contrôle pour $p = 1$: $6 - u_6 = 4 \times 0{,}5^6 = 0{,}0625 < 0{,}1$, mais $6 - u_5 = 0{,}125 \geqslant 0{,}1$. Ici la suite est croissante, donc $|u_n - 6| = 6 - u_n$: pas besoin de valeur absolue.

Pack 2 – la racine carrée de 2 par la méthode de Héron (ou de Babylone)

On pose $u_0 = 2$ et $u_{n+1} = \dfrac{1}{2}\left(u_n + \dfrac{2}{u_n}\right)$. On admet que pour tout n : $\dfrac{2}{u_n} \leqslant \sqrt{2} \leqslant u_n$ et que (u_n) converge vers $\sqrt{2}$. Ainsi $\sqrt{2}$ est **encadré** par $\frac{2}{u_n}$ et u_n : dès que l'amplitude $u_n - \frac{2}{u_n}$ est inférieure à ε, on a une valeur approchée de $\sqrt{2}$ à ε près, **sans connaître** $\sqrt{2}$!

Algorithme

Saisir ε
$u \leftarrow 2$
Tant que $u - \dfrac{2}{u} \geqslant \varepsilon$
  $u \leftarrow \dfrac{1}{2}\left(u + \dfrac{2}{u}\right)$
Fin Tant que
Afficher $\dfrac{2}{u}$ et u

Python

```
def heron(eps):
    u = 2
    while u - 2/u >= eps:
        u = (u + 2/u) / 2
    return 2/u, u
```

TI-82

```
:Prompt E
:2→U
:TantQue U-2/U≥E
:(U+2/U)/2→U
:Fin
:Disp 2/U,U
```

n	$2/u_n$	u_n
0	1	2
1	1,333 333	1,5
2	1,411 764 7	1,416 666 7
3	1,414 211 4	1,414 215 7
4	1,414 213 562 37	1,414 213 562 37

`heron(10**-6)` s'arrête après **4 tours seulement** et renvoie 1,414 213 56…

Le nombre de décimales exactes **double à peu près** à chaque étape : c'est une méthode extrêmement rapide (comparer avec la dichotomie, chapitre 6).

Pack 3 – limite inconnue : deux suites qui l'encadrent

(suites du chapitre 2, pack 4)

$u_0 = 1$, $v_0 = 12$, $u_{n+1} = \dfrac{u_n + 2v_n}{3}$, $v_{n+1} = \dfrac{u_n + 3v_n}{4}$. On admet que $u_n \leqslant \ell \leqslant v_n$ pour tout n, où ℓ est la limite commune. On s'arrête dès que l'amplitude $v_n - u_n$ est inférieure à 10^{-p}.

Algorithme

Saisir p
$u \leftarrow 1$; $v \leftarrow 12$; $n \leftarrow 0$
Tant que $v - u \geqslant 10^{-p}$
  $w \leftarrow u$
  $u \leftarrow (u + 2v)/3$
  $v \leftarrow (w + 3v)/4$
  $n \leftarrow n + 1$
Fin Tant que
Afficher n, u, v

Python

```
def encadre(p):
    u, v, n = 1, 12, 0
    while v - u >= 10**(-p):
        u, v = ((u + 2*v)/3,
                (u + 3*v)/4)
        n = n + 1
    return n, u, v
```

TI-82

```
:Prompt P
:1→U
:12→V
:0→N
:TantQue V-U≥10^(-P)
:U→W
:(U+2V)/3→U
:(W+3V)/4→V
:N+1→N
:Fin
:Disp N,U,V
```

Résultats : pour $p = 3$, le programme renvoie $n = 4$ et $8{,}999\,6 \leqslant \ell \leqslant 9{,}000\,2$; pour $p = 6$, $n = 7$. On conjecture $\ell = 9$ (ce que l'on démontre en étudiant la suite $3u_n + 8v_n$, constante égale à 99).

Pack 4 – comparer deux vitesses de convergence vers e $w_n = \left(1 + \frac{1}{n}\right)^n$ (on admet que $w_n < \mathrm{e}$ et $\lim w_n = \mathrm{e}$)

Algorithme

Saisir ε
$n \leftarrow 1$
Tant que $\mathrm{e} - \left(1 + \frac{1}{n}\right)^n \geqslant \varepsilon$
  $n \leftarrow n + 1$
Fin Tant que
Afficher n

Python

```
from math import e

def rang_e(eps):
    n = 1
    while e-(1+1/n)**n >= eps:
        n = n + 1
    return n
```

TI-82

```
:Prompt E
:1→N
:TantQue e-(1+1/N)^N≥E
:N+1→N
:Fin
:Disp N
```

(E est une variable ; la constante e : 2nde ÷)

Ici la suite est **explicite** : on ne garde pas de terme en mémoire, on recalcule w_n à chaque tour.
Résultats : `rang_e(0.01)` renvoie 135 et `rang_e(0.001)` renvoie 1 359 : pour gagner une décimale, il faut environ 10 fois plus de termes. Avec la somme $\sum \frac{1}{k!}$ du chapitre 3, $n = 9$ suffit déjà pour 10^{-6} : **toutes les suites ne convergent pas à la même vitesse**.

À vous de jouer (algorithme + Python + TI-82)

1. Adapter la méthode de Héron pour $\sqrt{3}$: $u_0 = 2$, $u_{n+1} = \frac{1}{2}\left(u_n + \frac{3}{u_n}\right)$. Combien d'étapes pour $\varepsilon = 10^{-6}$?

2. Avec la fonction `approx_e(n)` du chapitre 3, écrire un programme qui renvoie le plus petit n tel que $\mathrm{e} - \sum_{k=0}^{n} \frac{1}{k!} < 10^{-6}$ (en Python, `from math import e` ; sur la TI, e est obtenu par 2nde ln puis 1).

3. Dans le pack 1, déterminer **par le calcul** le plus petit n tel que $4 \times 0{,}5^n < 10^{-20}$ (à la calculatrice), puis comparer avec la valeur renvoyée par `rang(20)` en Python. Expliquer.

Réponses : 1. 3 étapes : $1{,}732\,050\,805 \leqslant \sqrt{3} \leqslant 1{,}732\,050\,810$. **2.** $n = 9$. **3.** Le calcul exact donne $n = 69$, mais Python renvoie 53 : au rang 53, l'écart $4 \times 0{,}5^{53} \approx 4{,}4 \times 10^{-16}$ est trop petit pour la machine (environ 16 chiffres significatifs), u_{53} est arrondi à exactement 6 et l'écart calculé vaut 0. On ne demande jamais une précision supérieure à celle de la machine.