Vitesses de croissance

Présentation et objectifs

Au programme de Spé maths

En Première : **Contenus** Exemples de modes de génération d'une suite : explicite $u_n=f(n)$, par une relation de récurrence $u_{n+1}=f(u_n)$, par un algorithme, par des motifs géométriques. Notations : $u(n), u_n, (u(n)), (u_n)$. Suites arithmétiques : exemples, définition, calcul du terme général. Suites géométriques : […]	**Capacités attendues** Dans le cadre de l'étude d'une suite, utiliser le registre de la langue naturelle, le registre algébrique, le registre graphique, et passer de l'un à l'autre. […] Calculer des termes d'une suite définie explicitement, par récurrence ou par un algorithme.
En Terminale : La fonction logarithme, sa propriété fondamentale.	

Situation déclenchante

Trois amis italiens, élèves d'un *Liceo*, Elena, Federico et Giancarlo commentaient ainsi, en novembre 2020, les chiffres cumulés des contaminations au CoVid-19 (ci-contre)[1]:

(Federico) – Che calamità ! La contaminazione sta crescendo esponenzialmente.

(Giancarlo) – Come sei stupido ! È solo proporzionale.

(Elena) – Voi ragazzi siete deficienti.

En février 2021, prenant connaissance à nouveau des chiffres (ci-contre, plus bas) … et ils ont le même dialogue.
Elena semble inviter ses amis à plus de raison. Que faut-il en penser ?

Date	Cas
24/10/20	504509
25/10/20	525782
26/10/20	542789
27/10/20	564778
28/10/20	589766
29/10/20	616595
30/10/20	647674
31/10/20	679430
19/01/21	2400598
20/01/21	2414166
21/01/21	2428221
22/01/21	2441854
23/01/21	2455185
24/01/21	2466813
25/01/21	2475372
26/01/21	2485956
27/01/21	2501147
28/01/21	2515507
29/01/21	2529070
30/01/21	2541783
31/01/21	2553032
01/02/21	2560957

Diverses manières de modéliser

Les nombres totaux de personnes contaminées par la CoVid-19 au fil des jours forment une suite qui peut être comparée à des suites de référence, notamment :

- suite *arithmétique* : $u_n=a+n\times r$ pour tout $n\in\mathbb{N}$ ($a\in\mathbb{R}$ est le premier terme, $r>0$ la raison)
- suite *géométrique* : $u_n=a\times q^n$ pour tout $n\in\mathbb{N}$ ($a>0$ est le premier terme, $q>1$ la raison)

On pourra ainsi dire qu'une suite (u_n) a une *croissance arithmétique (ou linéaire)* si la suite $\left(\frac{u_n}{n}\right)$ (définie pour $n\in\mathbb{N}^*$) tend vers une limite $L>0$; c'est le modèle de croissance évoqué par Giancarlo (proportionnalité des accroissements).

1 https://www.coronavirus-statistiques.com/stats-pays/coronavirus-nombre-de-cas-italie

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Concernant les suites géométriques, on remarque que si on prend une suite définie par $u_n = a \times q^n$ pour tout $n \in \mathbb{N}$, alors $\ln(u_n) = \ln(a) + n \times \ln(q)$ forme une suite arithmétique.
On pourra donc dire qu'une suite (u_n) a une *croissance géométrique (ou exponentielle)* si la suite de terme général $v_n = \ln(u_n)$ a une croissance arithmétique. C'est le modèle proposé par Federico.

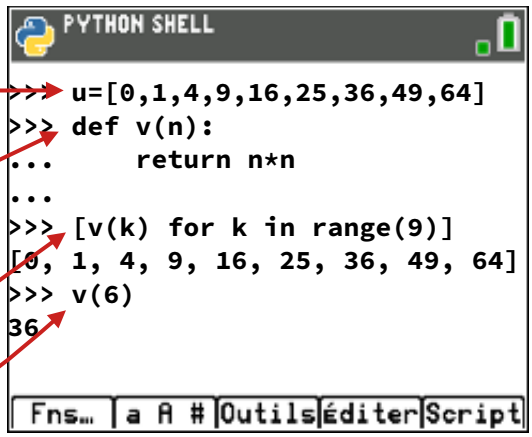
Coder une suite en Python

Une suite peut être simulée en langage Python comme liste ou comme fonction.

La suite des carrés nous servira d'exemple : $u_n = n^2$ pour $n \in \mathbb{N}$.

Première approche : on donne une liste des premiers termes. La mémoire de la machine peut contenir beaucoup de termes, mais évidemment pas une infinité de termes !

Seconde approche : on code la suite comme une fonction Python `v` d'un paramètre `n` ayant des valeurs entières ; ainsi la valeur `v(n)` désignera le terme v_n de rang n de la suite.
Pour revenir vers la première approche, on peut ensuite créer une « liste en compréhension » (ou bien étendre une liste par ajouts successifs).



Attention : selon le contexte, on écrira `v(k)` (si **v** est une fonction) ou `u[k]` (si `u` est une liste).

▶ **Objectif 1 :** écrire une fonction créant les 9 premiers termes d'une suite arithmétique de premier terme et de raison donnés, et une autre similaire pour une suite géométrique.

Indication : pour ajouter un élément **x** à la fin d'une liste **L**, on code : `L.append(x)`.

Observer les données

On peut représenter une suite (u_n) de nombres en portant n en abscisses et la valeur u_n en ordonnées.
En représentant les suites à comparer sur le même graphique, on a un bon moyen pour conjecturer (visuellement) un comportement particulier : au vu de la liste des premiers termes, cette suite semble-t-elle avoir une croissance arithmétique, ou faut-il plutôt penser à une suite géométrique ?

▶ **Objectif 2 :** Créer des listes `a`, `b` représentant les termes décrivant les nombres totaux de cas de CoVid-19 en Italie aux deux périodes considérées. Utiliser l'approche graphique pour répondre à la question précédente.

Modéliser les données

▶ **Objectif 3 :** écrire une fonction Python permettant de vérifier si une liste est le début d'une suite arithmétique, ou d'une suite géométrique ; si oui, renvoyer la raison et 0 sinon.

▶ **Objectif 4 :** à l'aide de suites auxiliaires et de la fonction logarithme, juger si la croissance des listes `a`, `b` (vues comme le début de deux suites $a = (a_n)$ et $b = (b_n)$) est plutôt arithmétique ou géométrique.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

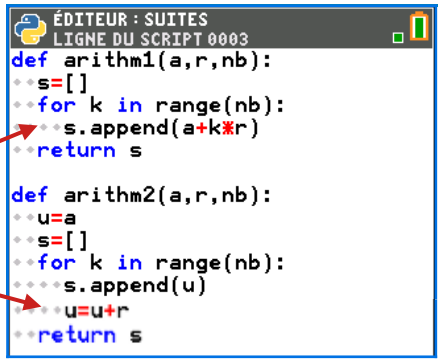
Fiche méthode

Objectif 1 : coder une suite en Python

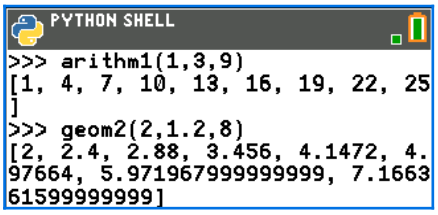
Il y a deux méthodes pour créer une fonction donnant les 9 premiers termes d'une suite arithmétique de premier terme et de raison donnés ; chacune des deux a ses mérites !

- Soit on s'appuie sur une formule explicite : $u_n = u_0 + n \times r$.
- Soit on calcule les termes en utilisant la relation de récurrence $u_{n+1} = u_n + r$.

Le codage Python suit ces idées.



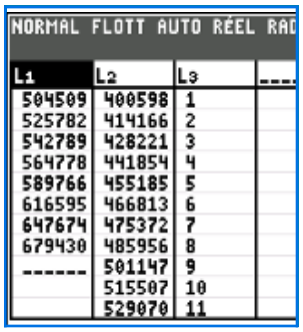
Et on procède de même pour les suites géométriques : soit à partir de $s_n = a \times q^n$ soit de $s_{n+1} = s_n \times q$ (le code est tout à fait similaire).



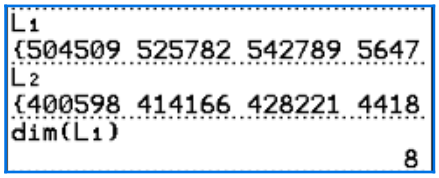
Objectif 2 : représenter des suites

TI-83 On dispose, dans cette machine, de possibilités de représentations graphiques des suites qui permettent de se faire une idée de leur comportement.

Pour ce faire, on peut commencer par saisir les listes a et b dans l'environnement natif de la calculatrice[2], disons dans les listes prédéfinies L_1 et L_2. Cela se fait commodément en appuyant sur la touche [stats], choix 1 Modifier (voir ci-contre). On ajoute en troisième colonne la suite des entiers (en prévision des représentations graphiques), ici associée à la liste L_3.

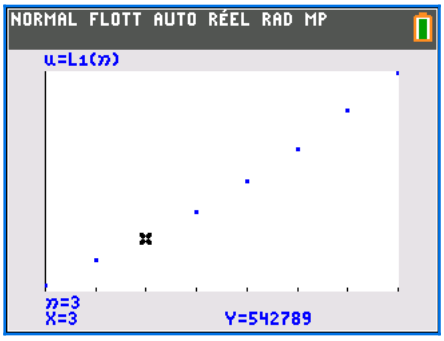


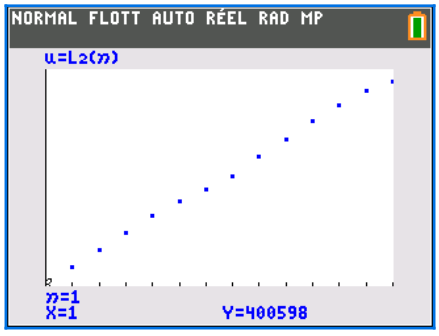
Pour afficher les suites, on règle le mode graphique (touche [mode]) en « suite » et « point épais », puis on choisit la suite à tracer (touche [f(x)]), on définit la fenêtre (touche [fenêtre]) et on trace (touche [trace]).



Le mode de tracé est interactif et permet de « suivre » les suites en bougeant le curseur.

Les points associés à la liste b (rangée dans L_2) sont à peu près alignés, ce qui suggère une croissance arithmétique. La liste a (rangée dans L_1) s'affiche avec un caractère plus « incurvé », qui nous dirigera plutôt vers un modèle de suite géométrique.





2 Pour alléger les affichages, nous avons saisi la suite b en lui soustrayant d'emblée 2 000 000.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

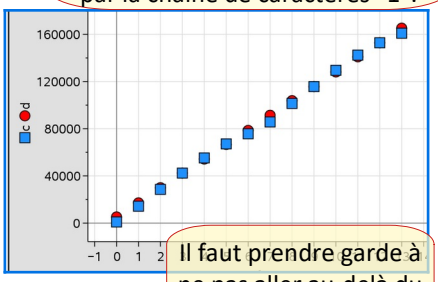
Il ne nous reste plus qu'à transférer ces listes dans l'environnement Python, ce qui se fait en important la bibliothèque `ti_system` et en appelant la fonction `recall_list`.

On profite de l'occasion pour simplifier un peu la liste `b` en lui retirant 400 000 (pour une suite à croissance arithmétique, cela ne change rien mais nous permet de travailler avec de plus petits nombres) ; cela nous donne une liste `c` (premiers termes d'une suite $c=(c_n)$).

Nspire CX Les choses sont plus simples : on copie les listes à étudier dans l'environnement « natif » de la machine avec une commande du type `store_list("a",a)`. et de même pour les autres listes. Il suffit alors de représenter les listes dans une page supplémentaire du classeur, choisie avec l'environnement Données et listes.

```
ÉDITEUR : LISTS
LIGNE DU SCRIPT 0008
from ti_system import *
a=recall_list("1")
b=recall_list("2")
c=[]
for i in range(14):
  c.append(b[i]-400000)
```

L'interface de la fonction `recall_list` est particulière : la liste système L_1 est ici désignée par la chaîne de caractères "1".



Il faut prendre garde à ne pas aller au-delà du « bout » de la liste !

Objectif 3 : analyser une liste comme une suite

Pour décider qu'une liste est en progression arithmétique, on peut tester la différence des termes consécutifs. S'il y a discordance, on renvoie 0 et autrement on renvoie la raison qui est le nombre étudié.

Pour une suite géométrique, une démarche similaire (en testant le quotient de termes successifs) ne conviendra pas car la division produit des erreurs d'arrondi pouvant invalider le test d'égalité. En formulant le test comme égalité à un <u>produit</u> (et non égalité à un quotient), le fonctionnement devient plus acceptable[3].

```
ÉDITEUR : SUITES2
LIGNE DU SCRIPT 0037
def estgeo(L):
  r=L[1]/L[0] # raison
  ok=True # contrôle
  i=1 # index
  while i<len(L)-1 and ok :
    if L[i+1]*r==L[i] :
      ok=False
      r=0
    i=i+1
  return r
```

Objectif 4 : comparer deux suites

La croissance arithmétique

Nous souhaitons examiner si la suite $c=(c_n)$, définie par la liste de ses premiers termes, a une « vitesse de croissance » arithmétique.

On crée la liste des premiers termes de la suite des quotients $\frac{c_n}{n}$ (pour $n\geq 1$) et on affiche le résultat qui ressemble à une suite convergente. Prenons par exemple 12 300 comme limite potentielle, cela revient à modéliser la suite (c_n) par une suite arithmétique de raison 12 300.

```
ÉDITEUR : SUITES3
LIGNE DU SCRIPT 0046
def cari(L):
  n=len(L)
  s=[]
  for i in range(1,n):
    s.append(round(L[i]/i,2))
  return s
```

```
PYTHON SHELL
>>> cari(c)
[14166.0, 14110.5, 13951.33, 137
96.25, 13362.6, 12562.0, 12279.4
3, 12643.38, 12834.11, 12907.0,
12889.36, 12752.67, 12381.31]
```

Pour en avoir le cœur net, on compare donc la suite (c_n) avec une suite arithmétique (d_n) de raison 12 300 et de premier terme 5000 (choix obtenu par tâtonnement), calculée comme ci-contre (on pourrait aussi faire usage de la fonction `arithm1` proposée ci-dessus).

```
ÉDITEUR : SUITES2
LIGNE DU SCRIPT 0070
a=recall_list("1")
b=recall_list("2")
c=[]
d=[]
for i in range(len(b)):
  c.append(b[i]-400000)
  d.append(12300*i+5000)
trc(c,d)
store_list("4",c)
store_list("5",d)
```

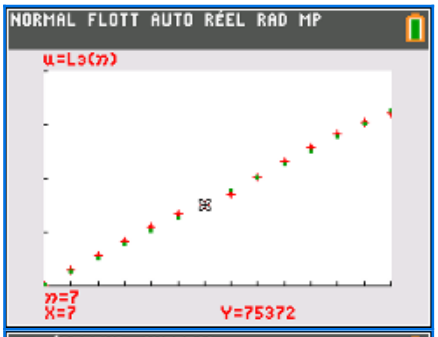
L'appel trc(c,d) réalise un tracé graphique ; cette fonction est présentée à la fin de la fiche.

3 La définition de la variable `r` comme quotient n'est pas idéale : il vaudrait mieux formuler le test d'égalité avec deux produits faisant intervenir `L(0)` et `L[1]` de chaque côté du signe `==`.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

La commande `store_list` copie ces listes dans l'environnement natif de la calculatrice, et il n'y a plus qu'à afficher (utiliser le menu [graph stats] pour définir deux graphes X-Y se superposant) : la coïncidence est très bonne ! *Giancarlo aurait-il ici raison ?*



La croissance géométrique

Pour analyser les suites géométriques, il est très commode de se ramener au cas précédent au moyen de la fonction logarithme (qui a été inventée pour cela).

Attention : la fonction logarithme fait partie de la bibliothèque `math` et est notée `log`.

Suivant l'observation de Federico, nous imaginons que la suite $a=(a_n)$ pourrait posséder une croissance géométrique.

On crée la liste des logarithmes des termes de L.

```
ÉDITEUR : SUITES3
LIGNE DU SCRIPT 0053
from math import *
def loga(L):
    s=[]
    for i in range(len(L)):
        s.append(log(L[i]))
    return s
```

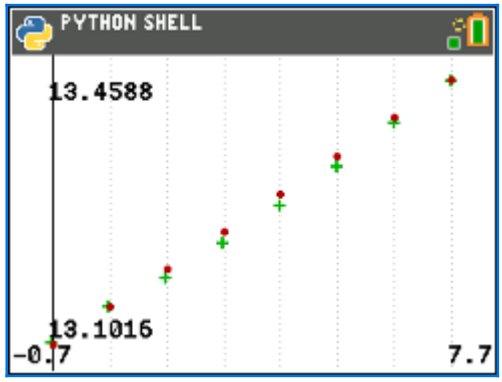
Pour en juger, on forme avec la fonction `loga` la liste des logarithmes des premiers termes de la suite `a`, puis on enchaîne avec la fonction `cari`. On obtient une liste suggérant une suite positive décroissante… peut-être convergente ! Il n'est en fait pas possible de conclure par cette approche, car le nombre de valeurs est très insuffisant pour avoir une bonne approximation de la raison.

```
PYTHON SHELL
>>> L=loga(a)
>>> L
[13.13134095806167, 13.172641957
14306, 13.20447594144643, 13.244
18801256117, 13.2874811270418, 1
3.33196768541655, 13.38114276232
423, 13.42900949034252]
>>> cari(L)
[13.17, 5.6, 4.41, 3.32, 2.67, 2
.23, 1.92]
```

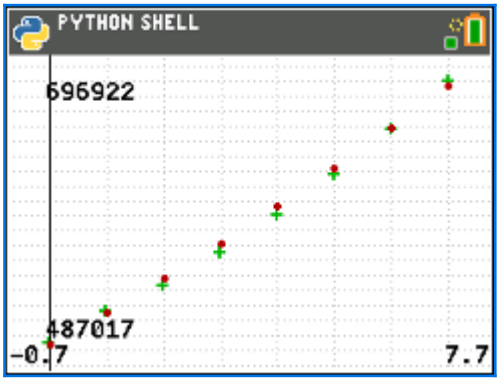
Nous allons donc procéder autrement. Considérons la suite $L_n=\log(a_n)$ et calculons les accroissements $L_{n+1}-L_n$: au-delà des premières valeurs, ils évoluent peu. Prenons par exemple comme valeur « cible » 0,043. On peut donc tenter de modéliser la suite (L_n) par $M_n=13{,}1+0{,}043n$, conduisant à approcher a_n par $\exp(M_n)=\exp(13{,}1)\times\exp(0{,}043)^n\approx 503833\times 1{,}043^n$.

```
PYTHON SHELL
>>> M=[]
>>> for i in range(7):
...     M.append(L[i+1]-L[i])
...
>>> M
[0.0413009990813844, 0.031833984
30336482, 0.03971207111474406, 0
.04329311448062789, 0.0444865583
747589, 0.04917507690767309, 0.0
4786672801829184]
```

L'adéquation est assez bonne comme le montrent les figures ci-dessous.



(L_n) *en vert,* (M_n) *en rouge*



(a_n) *en vert,* $(\exp(M_n))$ *en rouge*

Ainsi, Federico n'a pas tort non plus !

Ce document est mis à disposition sous licence Creative Commons

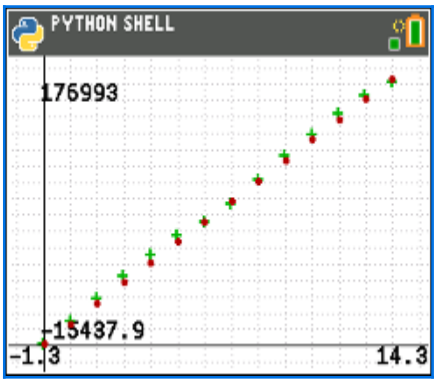
© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

Pour aller plus loin

Représenter graphiquement en Python

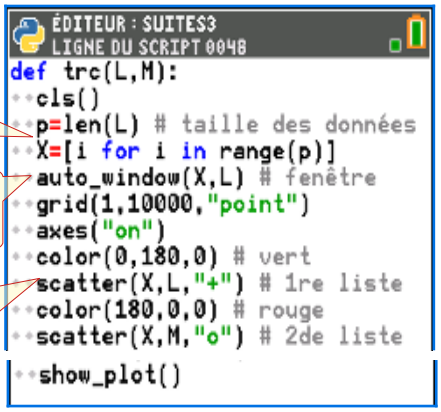
L'environnement Python permet tout à fait de représenter les suites en recourant à la bibliothèque `ti_plotlib`. Il y a un certain nombre de commandes à utiliser pour produire le dessin voulu, mais elles sont assez simples ; nous les présentons ci-contre.



On crée ici la liste des abscisses (comme liste en compréhension).

L'appel `auto_window` ajuste la fenêtre d'après une liste d'abscisses et une liste d'ordonnées.

L'appel `scatter` trace un nuage de points d'après une liste d'abscisses et une liste d'ordonnées.




```
def trc(L,M):
  cls()
  p=len(L) # taille des données
  X=[i for i in range(p)]
  auto_window(X,L) # fenêtre
  grid(1,10000,"point")
  axes("on")
  color(0,180,0) # vert
  scatter(X,L,"+") # 1re liste
  color(180,0,0) # rouge
  scatter(X,M,"o") # 2de liste
  show_plot()
```

Nspire CX Les commandes graphiques sont identiques, excepté une : il faut écrire `grid(1,10000,"dotted")` au lieu de `grid(1,10000,"point")` et tout fonctionne.

Et la modélisation …

Revenons sur le dialogue entre nos amis italiens. Y a-t-il un modèle qui vaille mieux que les autres ? Tout dépend de l'intervalle de temps au long duquel on espère avoir une bonne adéquation du modèle, comme de la qualité de l'ajustement !

Pour développer un modèle disposant d'une meilleure adéquation sur une longue durée, il faut changer de théorie et renoncer aux suites définies par une simple relation de récurrence. Le lecteur intéressé pourra consulter les références ci-dessous, écrites par de bons spécialistes français du secteur.

https://interstices.info/modeliser-la-propagation-dune-epidemie

https://www.inrae.fr/actualites/modelisation-pratique-gestion-dune-epidemie

http://images.math.cnrs.fr/Modelisation-d-une-epidemie-partie-1.html

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée