Teachers Teaching with Technology™

Robert Cabane et Laurent Didier

Activités algorithmiques avec Python en spécialité mathématiques Lycée général – Classes de 1ère et de Tle

www.t3france.fr

© 2021 T³ France / Texas Instruments

Activités algorithmiques avec Python en spécialité Mathématiques

Table des matières

1. Mises à jour système et compléments logiciels....2
2. Avant-propos....3
3. Interpolation polynomiale....5
4. Déchiffrement d'Al Kindi....9
5. Comment (presque) résoudre une équation....13
6. Le flocon de Von Koch, courbe fractale....17
7. Marches aléatoires....21
8. Vitesses de croissance....25
9. L'approximation des intégrales....31
10. Le calcul des logarithmes....35
11. Le triangle de Pascal....39
12. La planche de Galton....45
13. Un problème de surréservation....49
14. La méthode de Monte Carlo....55
15. La combinatoire des parties....61
16. Quelques compléments sur la syntaxe du langage Python....67
17. Aide-mémoire sur les bibliothèques (modules) utilisées....71

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Téléchargements et mises à jour

Mises à jour système et compléments logiciels

Pour exécuter tels-quels les programmes présentés dans ce livret il est nécessaire de disposer d'une calculatrice Texas Instruments permettant de programmer en langage Python. Cela concerne les modèles suivants :

- TI-82 Advanced Édition Python (sans graphisme)
- TI-83 Premium CE + Python Adapter (sans graphisme ni mode examen)
- TI-83 Premium CE Édition Python
- TI-Nspire CX II-T

Pour les deux derniers modèles, il convient que le système d'exploitation de la calculatrice soit à jour.

Par ailleurs, deux des fiches proposées nécessitent de charger un module complémentaire pour accéder aux graphismes en mode « tortue » (Turtle). Nous décrivons ici l'utilisation du module Turtle proposé au moment de la conception de ce livret ; une mise à jour étant cependant prévue pour ce module, le livret sera lui-même ajusté en fonction.

L'installation des systèmes ou modules est très facile : il suffit de télécharger un fichier à l'adresse indiquée ci-dessous puis de copier le fichier téléchargé dans la machine à l'aide du logiciel ad hoc (soit TI-Connect CE prévu pour la TI-83 Premium CE, soit TI-Nspire CX prévu pour la TI-Nspire™ CX II).

Mise à jour des calculatrices TI-83 Premium CE	
Mise à jour des calculatrices et logiciels TI-Nspire™ CX	
Module Turtle pour TI-83 Premium CE Edition Python (module CE_TURTL publié en mai 2020, mise à jour Turtle prévue en 2022)	
Module Turtle pour TI-Nspire™ CX II	

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Activités algorithmiques avec Python en spécialité Mathématiques

Avant-propos

Ce livret s'adresse aux enseignants désirant travailler l'algorithmique incluse dans le **programme de spécialité mathématiques**, aussi bien en classe de Première et de Terminale, et propose des algorithmes très variés. L'idée de cet ouvrage a germé avec les nouveaux programmes, au contact des élèves et de l'outil sur lequel il est le plus facile de programmer au quotidien dans nos classes : la calculatrice. Nous avons souhaité proposer des activités progressives en expliquant le fonctionnement de certains de ces algorithmes.

Cet ouvrage permettra à chacun de concrétiser, de consolider et d'élargir ses connaissances algorithmiques au travers du langage Python inclus dans les calculatrices Texas Instruments.

Les treize fiches proposées sont classées par niveau, en commençant par celles de Première puis en enchaînant avec celles de Terminale. Chaque fiche (comprenant plusieurs activités) débute par une section introductive « **présentation et objectifs** », autour des buts visés ainsi que de leur lien avec les programmes officiels, avec les détours mathématiques nécessaires pour bien comprendre.

Aussi souvent que possible, nous avons proposé des **activités de groupe**, utiles pour que les élèves progressent avec plaisir, s'entraidant et mutualisant leurs découvertes respectives. Les calculatrices équipées du langage Python s'y prêtent bien, grâce à leur maniabilité et leur grande autonomie : ni câble, ni connexion à un réseau, ni identifiants à mémoriser !

La « **fiche méthode** » qui suit contient tous les détails de la réalisation.

Vous trouverez une section « **pour aller plus loin** » dans la plupart des fiches, permettant d'aborder des activités prolongeant certains algorithmes proposés en exemple dans le programme officiel avec des notions un peu plus poussées.

Le **choix de la calculatrice** a son importance. En général, la TI-83 Premium CE Edition Python (clavier sur fond blanc) constitue une bonne plateforme pour traiter une grande diversité de problèmes mathématiques et algorithmiques. Les utilisateurs de la TI-82 Advanced Edition Python (clavier sur fond noir) ne pourront pas traiter les activités comportant une partie graphique, ce qui sera plus ou moins limitant suivant les fiches. Deux des fiches (marches aléatoires et méthode de Monte Carlo) nécessitent une puissance de calcul plus importante : c'est une situation où la TI Nspire™ CX II-T démontrera sa supériorité. Nous avons indiqué dans le texte les sections spécifiques à chaque machine, en la désignant de manière très brève : **TI-83** ou **Nspire CX**.

Il est recommandé de **mettre à jour** le logiciel interne des calculatrices afin de disposer des correctifs et ajouts les plus récents, ce qui touche notamment l'implémentation du langage Python : pour cela, nous avons inséré page 2 (ci-contre) des liens et QR-codes permettant d'effectuer les mises à jour de votre calculatrice et/ou du logiciel associé.

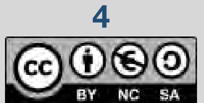
Vous pourrez retrouver une version numérique de cet ouvrage sur le site de Texas Instruments France à l'adresse https://education.ti.com/fr/enseignants (espace « ressources et cahiers d'activités »).

Nous vous souhaitons de prendre du plaisir à ces activités,

Les auteurs.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée


Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Interpolation polynomiale

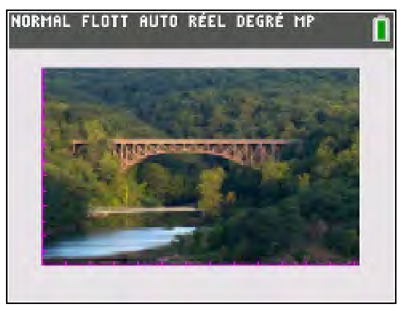
Interpolation par une fonction polynomiale du second degré

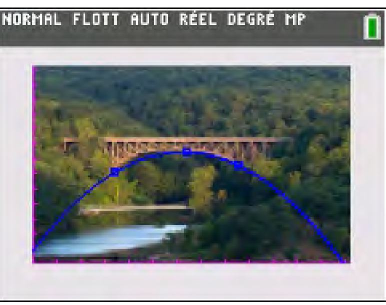
Dans le programme (spécialité Première)

Contenus	**Capacités attendues**
Fonction polynôme du second degré donnée sous forme factorisée. Racines, signe, expression de la somme et du produit des racines. Forme canonique d'une fonction polynôme du second degré. Discriminant. Factorisation éventuelle. Résolution d'une équation du second degré. Signe.	Choisir une forme adaptée (développée réduite, canonique, factorisée) d'une fonction polynôme du second degré dans le cadre de la résolution d'un problème.

Situation déclenchante

Dans le menu [format] de la calculatrice, on peut afficher un arrière-plan (par ex. numéro 4 ou numéro 1) et choisir dans le menu [zoom], A :Zquadrant1. On obtient alors l'un des écrans ci-dessous à gauche.





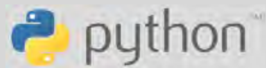
Dans notre vie quotidienne nous observons régulièrement des situations pouvant être modélisées par des paraboles (images de droite). Comment déterminer la fonction polynomiale permettant une « interpolation »[1] du modèle ?

On appelle « portée » d'un pont la longueur au niveau de l'eau entre deux piliers du pont. Comment utiliser le modèle pour déterminer la portée du pont ?

1 Il s'agit d'une fonction polynomiale prenant des valeurs imposées en un certain nombre de points (trois points non alignés quand on impose que la fonction soit de degré 2).

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

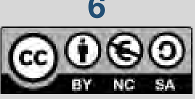


Interpolation polynomiale

L. DIDIER & R. CABANE

Buts à atteindre

1. Écrire un programme en langage Python prenant en entrée les coefficients d'un polynôme de degré 2 et renvoyant la liste des racines réelles de ce polynôme.
2. Utiliser la calculatrice pour réaliser une interpolation polynomiale de degré 2 à partir d'une photo.
3. Dans un programme Python, utiliser l'interpolation précédente pour déterminer la portée du pont au niveau de l'eau.



Ce document est mis à disposition sous licence Creative Commons

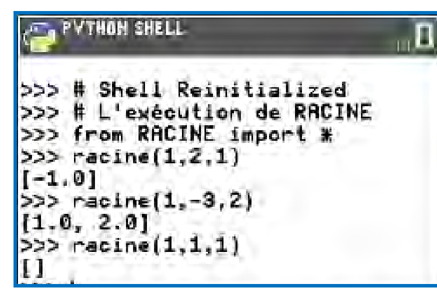
© Texas Instruments 2021 / Photocopie autorisée

Fiche méthode

Proposition de résolution

Pour atteindre l'objectif 1 :

Une fonction `racine` qui prend comme arguments trois nombres réels et qui renvoie la liste des racines réelles.

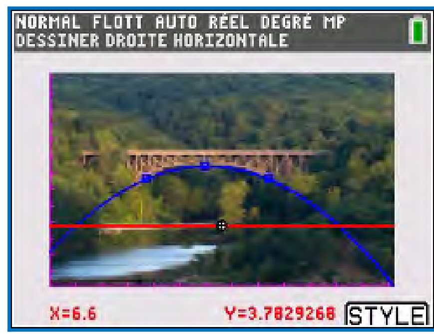


Pour atteindre l'objectif 2 :

Une fois l'affichage de la photo réalisé, à partir de l'écran graphique, appuyer sur la touche stats, puis sélectionner la rubrique CALC, puis enfin le menu E:TracéAjust-Éq.

Pour atteindre l'objectif 3 :

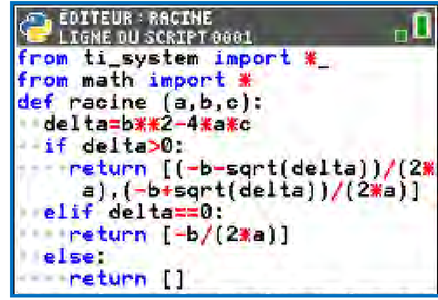
Une fois dans la fenêtre graphique, déterminer le niveau de l'eau avec le menu dessin (touche 2nde puis prgm) et sélectionner 3:Horizontal. Régler ensuite le segment horizontal au bon endroit. Il faudra ensuite créer un script qui permette d'utiliser l'interpolation précédente et qui utilisera la fonction `racine()`.



Étapes de résolution

▶ Pour atteindre l'objectif 1 :

Attention à ne pas oublier l'instruction `from math import *` pour pouvoir utiliser la fonction « racine carrée » `sqrt`.



▶ Pour atteindre l'objectif 2 :

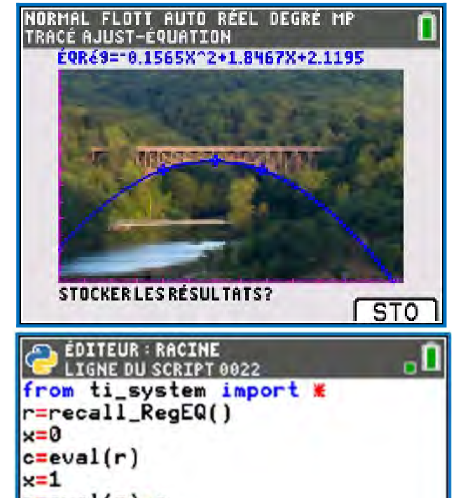
Une fois dans le menu E:TracéAjust-Éq placer trois points bien choisis puis sélectionner 3:RégDeg2 dans le menu ÉQRég. L'interpolation calculée apparaît au-dessus de la photo. Elle peut être stockée, en utilisant le menu STO, dans la fonction Y1 (fonction de la variable X), utilisable indépendamment de l'image.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

▶ Pour atteindre l'objectif 3 :

La difficulté réside dans le fait que la fonction polynomiale d'interpolation est calculée à l'extérieur du module de programmation Python.




```
ÉDITEUR : RACINE
LIGNE DU SCRIPT 0022
from ti_system import *
r=recall_RegEQ()
x=0
c=eval(r)
x=1
p=eval(r)-c
x=-1
m=eval(r)-c
a=(p+m)/2
b=(p-m)/2
hauteur =float(input("saisir le
    niveau de l'eau"))
print (fabs(racine(a,b,c-hauteur
    )[1]-racine(a,b,c-hauteur)[
    0]))_
Fns… | a A # | Outils | Exéc | Script
```

TI-83 Dans un script, nous allons devoir retrouver les coefficients du polynôme d'interpolation calculé précédemment. Les instructions suivantes :

```
from ti_system import *
r=recall_RegEQ()
x=0
c=eval(r)
```

Cette instruction ne peut pas être utilisée à l'intérieur d'une fonction, raison pour laquelle nous utiliserons ici les instructions *input* et *print* .

permettent de calculer l'image de 0 par la fonction polynomiale d'interpolation calculée à l'extérieur du module Python puis de stocker cette valeur dans la variable `c`.

En effectuant les mêmes opérations avec 1 et -1 on détermine les coefficients a,b,c du polynôme d'interpolation, car si $P(x)=ax^2+bx+c$ alors $P(1)=a+b+c$, $P(-1)=a-b+c$ et $b=\dfrac{P(1)-P(-1)}{2}$, etc.

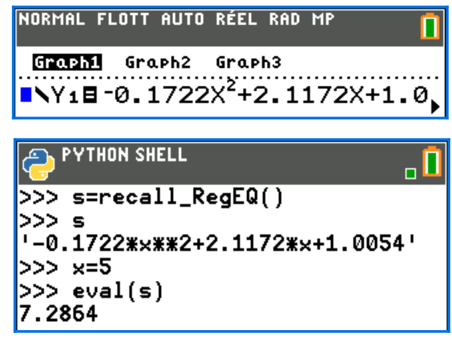
Pour déterminer la longueur du pont au niveau de l'eau, il suffit de saisir le niveau de l'eau déterminé dans l'objectif 2, et de calculer l'écart entre les racines en utilisant le bon polynôme.

Prolongement possible

> Imaginer une fonction réalisant une interpolation polynomiale de degré 2 à partir des coordonnées de 3 points.

Il s'agira ici dans un premier temps de résoudre à la main un système à 3 équations à 3 inconnues…

Les secrets …



La valeur retournée par `recall_RegEQ` est en fait une chaîne de caractères. Un petit exemple pour expliquer quelques mystères …

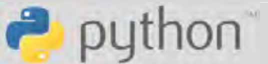
À la suite d'un processus comme indiqué précédemment, la fonction `Y1` a été initialisée comme il convient (interpolation du second degré sur trois points).

On récupère cette expression dans l'environnement Python, ici c'est le résultat de `recall_RegEQ` qui est rangé dans la chaîne de caractères `s`.

Pour que cela devienne une valeur, il faut affecter une valeur à `x` puis « évaluer » `s` par la fonction `eval()` (menu E/S dans l'éditeur Python).

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

Déchiffrement d'Al Kindi

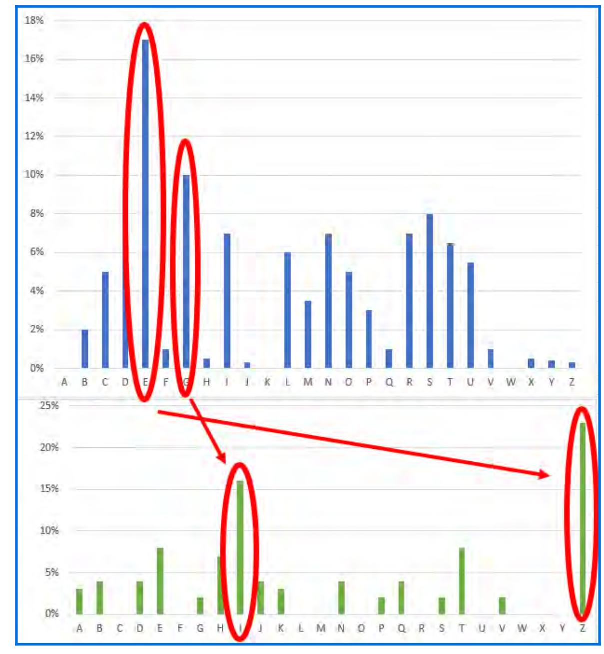
Présentation

Dans le programme (spécialité Première)

Exemples d'algorithmes Fréquence d'apparition des lettres d'un texte donné, en français, en anglais. **Notion de liste** La génération des listes en compréhension et en extension est mise en lien avec la notion d'ensemble. Les conditions apparaissant dans les listes définies en compréhension permettent de travailler la logique.	**Capacités attendues** Générer une liste (en extension, par ajouts successifs ou en compréhension). Manipuler des éléments d'une liste (ajouter, supprimer...) et leurs indices. Parcourir une liste. Itérer sur les éléments d'une liste.

Situation déclenchante

Al-Kindi[2], philosophe et savant arabe du IXe siècle, traduisit et adapta de nombreux ouvrages grecs, et fut le premier auteur à avoir écrit un traité de cryptographie. C'est à Al-Kindi que nous devons l'invention du chiffrement « monoalphabétique » qui consiste à remplacer chaque lettre[3] d'un texte par une autre lettre, sachant que deux lettres distinctes doivent être chiffrées par deux lettres distinctes pour permettre un déchiffrement du message sans ambiguïté. Comment décoder un message codé de cette manière sans en connaître le codage ?



Une façon d'attaquer un chiffrement par substitution mono-alphabétique est l'analyse fréquentielle si le texte à décoder est assez long. On compare la fréquence d'apparition de chaque caractère dans le texte codé avec la fréquence moyenne des lettres dans la langue de référence. On peut ainsi établir une première correspondance.

Les esprits curieux pourront aussi s'intéresser au concours de décryptage Al-Kindi[4].

But à atteindre

> Écrire un script Python permettant de trouver la fréquence d'apparition des lettres de l'alphabet dans un texte donné.

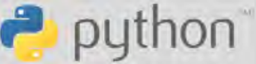
2 https://fr.wikipedia.org/wiki/Al-Kindi

3 Pour simplifier nous ne traiterons ici que des 26 lettres minuscules, ignorant tout autre lettre ou symbole.

4 https://concours-alkindi.fr

Ce document est mis à disposition sous licence Creative Commons

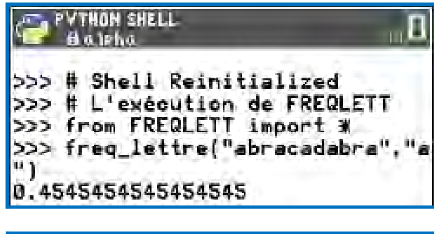
© Texas Instruments 2021 / Photocopie autorisée



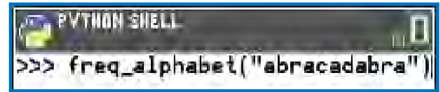
Fiche méthode

Proposition de résolution

On crée **trois fonctions** dans ce script :



- Une fonction freq_lettre qui prend comme arguments une chaîne de caractères et une lettre. Cette fonction renvoie la fréquence d'apparition de la lettre dans la chaîne de caractères.




```
909090909092], ['e', 0.0], ['f', 0.0], ['g', 0.0], ['h', 0.0], ['i', 0.0], ['j', 0.0], ['k', 0.0], ['l', 0.0], ['m', 0.0], ['n', 0.0], ['o', 0.0], ['p', 0.0], ['q', 0.0], ['r', 0.1818181818181818], ['s', 0.0], ['t', 0.0], ['u', 0.0], ['v', 0.0], ['w', 0.0], ['x', 0.0], ['y', 0.0], ['z', 0.0]]
```

- Une fonction freq_alphabet qui prend comme argument une chaîne de caractères et qui renvoie la liste des lettres de l'alphabet accompagné de leurs fréquences d'apparition dans la chaîne de caractères (c'est donc une liste de listes).

```
>>> affichage(freq_alphabet("abracadabra"))
[['a', 0.455], ['b', 0.182], ['c', 0.091], ['d', 0.091], ['r', 0.182]]
```

- Une fonction affichage qui prend comme argument la liste renvoyée par freq_alphabet et qui renvoie une liste composée des listes de lettres si la fréquence d'apparition est non nulle, avec une fréquence arrondie à 3 décimales.

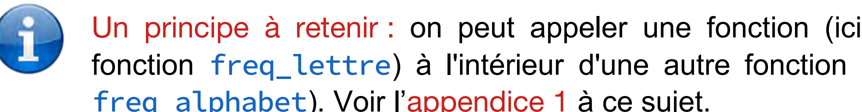
Étapes de résolution

```
def freq_lettre(texte,lettre):
  nbcar=len(texte)
  compteur=0
  for k in range(nbcar):
    if texte[k]==lettre:
      compteur=compteur+1
  return compteur/nbcar
```

- Pour freq_lettre, on commence par chercher le nombre de caractères à traiter : l'instruction len(texte) permet justement de déterminer la longueur de la chaîne de caractères (ici nommée texte). La boucle « for » permet ensuite de compter le nombre d'apparitions de la lettre dans le texte.

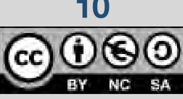
```
def freq_alphabet(texte):
  alphabet="abcdefghijklmnopqrstuvwxyz"
  frequences=[]
  for lettre in alphabet:
    frequences.append([lettre,freq_lettre(texte,lettre)])
  return frequences
```

- Pour la fonction freq_alphabet, on utilise plusieurs fois la fonction précédente.



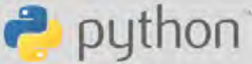
Un principe à retenir : on peut appeler une fonction (ici, la fonction freq_lettre) à l'intérieur d'une autre fonction (ici, freq_alphabet). Voir l'appendice 1 à ce sujet.

- On construit la liste appelée frequences, initialisée avec une liste vide. L'instruction frequences.append([lettre,freq_lettre(texte,lettre)]) permet à chaque passage de boucle d'ajouter une lettre de l'alphabet accompagnée de son pourcentage d'apparition dans le texte en faisant appel à la fonction freq_lettre.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



La fonction `affichage` permet d'obtenir des résultats plus lisibles en n'affichant que les fréquences non nulles et arrondies au millième.

```
def affichage(l):
  aff=[]
  for i in range (0,25):
    if l[i][1]!=0:
      l[i][1]=round(l[i][1],3)
      aff.append(l[i])
  return aff
```

L'instruction `round(x,3)` permet d'arrondir la valeur `x` à 3 chiffres après la virgule.

On note que la fonction `freq_alphabet` parcourt 26 fois le texte. Une autre approche algorithmique (code ci-contre) permettrait de parcourir une seule fois le texte et donc de gagner en efficacité. Cette approche utilise une boucle du type `for ch in txt`, revenant à faire parcourir à la variable `ch` la chaîne de caractères `txt`.

```
EDITEUR : FREQALPH
LIGNE DU SCRIPT 0009
def frqalpha(txt):
  freq=[0 for k in range(26)]
  for ch in txt:
    k=ord(ch)-ord('a')
    if (0<=k<=26):
      freq[k]+=1
  return freq
```

La fonction `ord` prend en paramètre un caractère et renvoie le numéro associé à cette lettre.

Lors de l'exécution de cette fonction, on observe le tableau des effectifs associés à chaque lettre de l'alphabet.

```
PYTHON SHELL
>>> # Shell Reinitialized
>>> # L'exécution de FREQALPH
>>> from FREQALPH import *
>>> frqalpha("portez ce vieux whisky au juge blond qui fume" )
[1, 1, 1, 1, 5, 1, 1, 1, 3, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 5, 1, 1, 1, 1, 1]
>>> |
Fns… | a A # | Outils | Éditer | Script
```

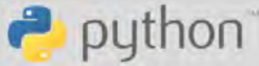
Complément : il est possible d'analyser des textes plus longs, sur plusieurs lignes.

Pour ce faire, on doit insérer le texte entre des triples guillemets (voir appendice 1) :

```
s="""Première ligne
Seconde
Troisième"""
print(s)
```

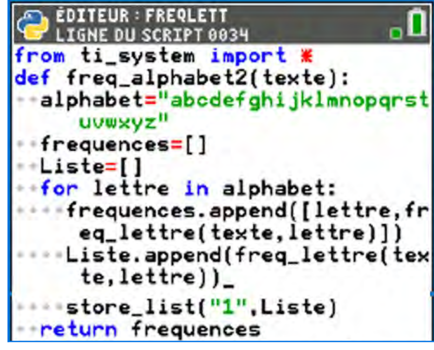
Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



Pour aller plus loin

Approfondissement possible



On peut représenter graphiquement les résultats à l'aide d'un histogramme. Pour cela, il faut créer une liste qui va être exportée en dehors de l'application Python.

TI-83 Cette opération nécessitera d'importer la bibliothèque `ti_system`. Les fréquences sont stockées dans une liste (Python) appelée `Liste` qui sera exportée au sein du menu *liste* dans la variable (système) L_1 grâce à l'instruction `store_list("1",Liste)` (voir l'appendice 2 pour des détails).

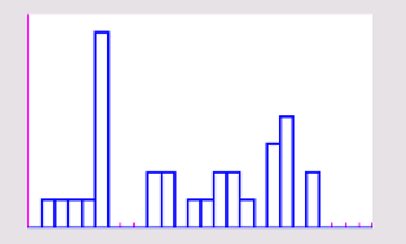
```
>>> # Shell Reinitialized
>>> # L'exécution de FREQLETT
>>> from FREQLETT import *
>>> freq_alphabet("bonjour je suis en classe de premiere")
```

Une fois le programme exécuté, il faut quitter l'application Python et aller dans la rubrique *graph stats* (touches [2nde] puis [f(x)]).

Il faut régler les paramètres d'affichage du graphique statistique (on peut par exemple mettre les numéros de 1 à 26 dans la liste L_2, la liste L_1 contenant les fréquences calculées par le programme).

Ne pas oublier de régler la fenêtre d'affichage (touche **fenêtre**) pour avoir un affichage adapté (exemple ci-contre) !

```
NORMAL FLOTT AUTO RÉEL DEGRÉ MP
FENÊTRE
 Xmin=0
 Xmax=26
 Xgrad=1
 Ymin=0
 Ymax=1
 Ygrad=1
 Xrés=1
 ΔX=0.098484848484848
 PasTrace=0.1969696969697
```



Prolongements possibles – et un défi

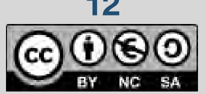
1. Utiliser ce script pour déterminer les fréquences des lettres employées en français (dans un texte de référence).
2. Appliquer à un problème de décodage.

Voici un texte codé :

```
fg ozjdgw v uldgvvg uxa gqvgzyquqwv lgvzduqw wdujuzoogd o uoyrdzwkpztxg
zqfoxvg luqv og sdryduppg lg vsgfzuozwg puwkgpuwztxgv uxvvz ezgq gq fouvvg lg
sdgpzgdg gw lg wgdpzquog gw sdrsrvg lgv uoyrdzwkpgv wdgv judzgv. og fkrza lg
ou fuofxouwdzfg u vrq zpsrdwuqfg.
```

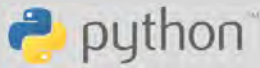
On suppose que le codage utilisé est une substitution mono-alphabétique. Quelle lettre code le « e » ? On pourra utiliser les programmes précédents ainsi que les pourcentages de référence d'apparition des lettres dans la langue française[5]. Et la suite du décodage … est votre défi !

5 https://fr.wikipedia.org/wiki/Fréquence_d'apparition_des_lettres_en_français
https://bibmath.net/crypto/index.php?action=affiche&quoi=chasseur/frequences_francais



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

Comment (presque) résoudre une équation

Présentation et objectifs

Dans les programmes

Première : exemple d'algorithme	**Terminale :** exemples d'algorithme
Méthode de Newton, en se limitant à des cas favorables.	Méthode de dichotomie. Méthode de Newton, méthode de la sécante.

Situation déclenchante

Imaginons qu'une fonction continue f ait été définie sur un intervalle $D=[a;b]$, et que cette fonction change de signe et s'y annule une fois. Par exemple, $f(x)=x^3-x-0{,}5$ sur l'intervalle $D=[0{,}5\ ;\ 1{,}3]$. Comment faire pour s'informer sur le nombre x de D tel que $f(x)=0$? Mais que veulent dire « s'informer » ou « trouver x » ? Comment faut-il répondre ?

Pour une équation polynomiale de degré 2, la réponse est claire : on cherche une « expression » de x faisant intervenir une racine carrée, tirée du modèle $\dfrac{-b+\sqrt{b^2-4ac}}{2a}$ pour une équation $ax^2+bx+c=0$ à discriminant positif. Pour d'autres équations (comme $x\ln(x)=1$) on ne trouve aucune « expression » bien que des solutions existent (théorème des valeurs intermédiaires). Faute de mieux, on cherche une **approximation** des solutions. Nous prendrons comme contexte une fonction f continue, croissante sur un intervalle $[a;b]$, telle que $f(a)<0<f(b)$.

Buts à atteindre

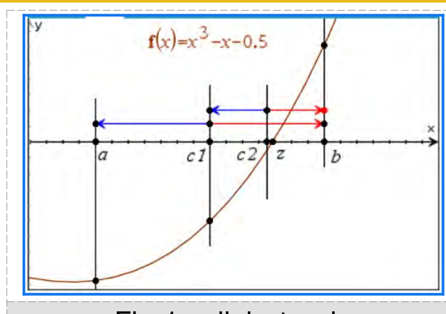


Fig.1 - dichotomie

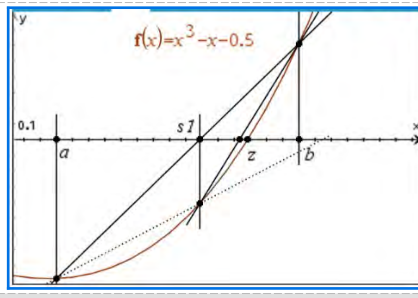


Fig. 2 - sécantes

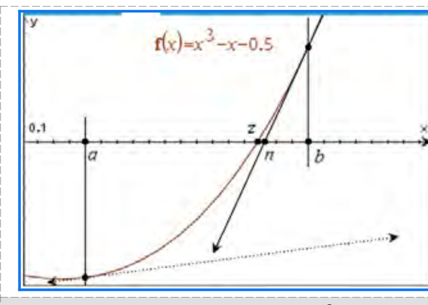


Fig. 3 - Newton[6]

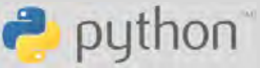
Suggestion : trois algorithmes étant en vue, on répartit le travail sur plusieurs groupes qui compareront leurs résultats sur un exemple (ici : f telle que $f(x)=x^3-x-0{,}5$ sur $[0{,}5\ ;\ 1{,}3]$).

1. Écrire une fonction Python approchant par dichotomie une solution de $f(x)=0$ avec une précision p, à partir de la connaissance de f, a, b.
2. Écrire une fonction Python faisant de même par la méthode des sécantes.
3. Écrire une fonction Python faisant de même par la méthode de Newton.

6 Figures réalisées avec le logiciel TI-Nspire CX

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

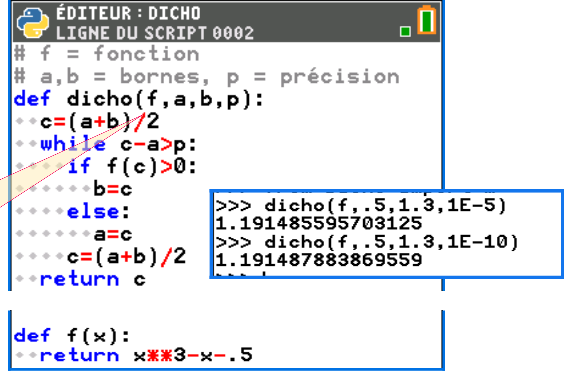
Fiche méthode

Étapes de résolution

▶ Objectif 1 : dichotomie

L'algorithme est essentiellement une boucle `while`, où les bornes `a` et `b` sont réajustées de sorte que la fonction `f` prenne toujours une valeur négative à gauche et positive à droite.

Ici le paramètre `f` est une <u>fonction</u> (Python), à ne pas confondre avec `f(x)` qui est un <u>nombre</u>.



▶ Objectif 2 : sécantes

L'équation de la droite passant par deux points distincts de coordonnées $(a\,;f(a))$ et $(b\,;f(b))$ est : $y=f(a)+\frac{f(b)-f(a)}{b-a}(x-a)$ (justification : remplacer x par a puis par b pour s'assurer que la formule est la bonne) ; cette droite coupe l'axe (Ox) au point d'abscisse s telle que

$0=f(a)+\frac{f(b)-f(a)}{b-a}(s-a)$ soit encore[7] $s=a-f(a)\frac{b-a}{f(b)-f(a)}$.

On recommence ensuite en prenant s à la place de a ou de b. On a donc essentiellement à calculer une suite récurrente du type $u_{n+1}=a-f(a)\frac{u_n-a}{f(u_n)-f(a)}$, avec $u_0=b$, ou bien $u_{n+1}=b-f(b)\frac{u_n-b}{f(u_n)-f(b)}$ avec $u_0=a$. Le choix de la formule qui convient dépend de la fonction f, et peut se faire en testant si on reste bien dans l'intervalle $[a\,;b]$. Dans l'exemple étudié, c'est la seconde formule qui doit être choisie (voir la figure 2 page précédente, la droite en pointillés ne convient pas).

```
ÉDITEUR : INTERPOL
LIGNE DU SCRIPT 0010
def interpol(f,a,b,p):
  u=a
  while True:
    u=b-f(b)*(u-b)/(f(u)-f(b))
    if f(u)<0 and f(u+2*p)>0:
      return u+p

def f(x):
  return x**3-x-0.5
```

```
>>> from INTERPOL import *
>>> interpol(f,.5,1.7,1E-5)
1.19147951169149
>>> interpol(f,.5,1.7,1E-10)
1.191487883966847
```

Nous nous limitons ici à la seconde formule, qui va donner une suite d'approximations croissante. Pour assurer la précision demandée, il faut tester le signe de la fonction « un peu plus loin » (en $u+2p$).

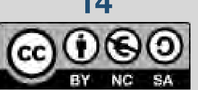
À noter :

1 La fonction utilisée s'annule sur l'intervalle $[0{,}5\,;1{,}3]$, mais aucune « formule » ne semble disponible pour la racine de l'équation $f(x)=0$ sur cet intervalle.

2 On utilise ici une « boucle infinie » `while True`, qui s'achève en fait au moment où la précision est atteinte (test du changement de signe de `f` entre `u` et `u+2p`), par une instruction `return`.

`True` est une constante toujours « vraie », exactement comme le serait un test `1==1`.

7 On pourrait aussi présenter s sous la forme $s=\frac{af(b)-bf(a)}{f(b)-f(a)}$; cela ne serait pas judicieux car si les valeurs $f(a)$ et $f(b)$ étaient proches, les deux soustractions se feraient avec une mauvaise précision.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

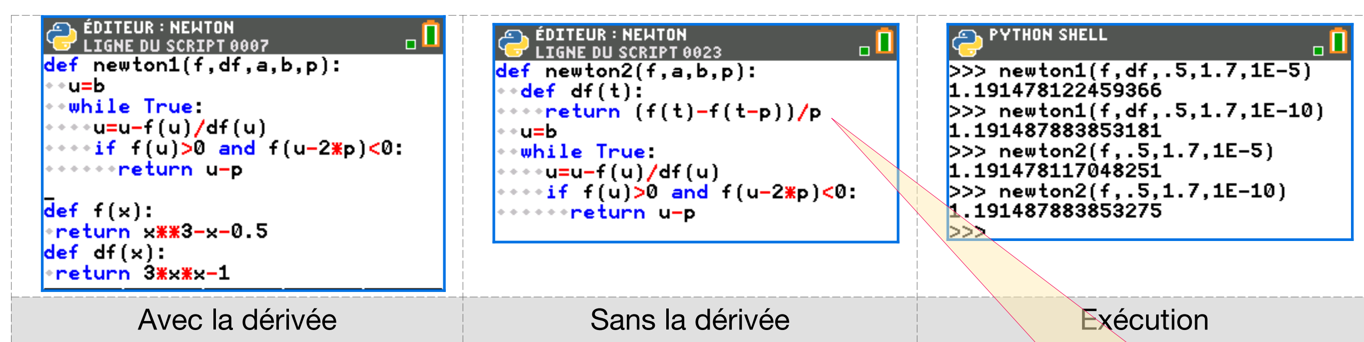
3 Par ailleurs, il faut reconnaître que le code proposé est bref mais non optimal ; ainsi, la valeur `f(b)` est recalculée de nombreuses fois, ce qui peut être long si le calcul de la fonction `f` est complexe.

Il serait préférable de « précalculer » `f(b)` avant la boucle `while` et de ranger cette valeur dans une variable. Même remarque pour la valeur `f(u)`, calculée deux fois.

▶ Objectif 3 : méthode de Newton

On suppose ici que f est dérivable sur $[a;b]$ et que f' ne s'annule pas sur $[a;b]$. La démarche est très similaire à celle de l'objectif 2. L'équation de la droite passant par le point de coordonnées $(b\,;f(b))$ et tangente à la courbe est $y=f(b)+f'(b)(x-b)$; cette droite coupe l'axe au point d'abscisse n telle que $0=f(b)+f'(b)(n-b)$ soit $n=b-\dfrac{f(b)}{f'(b)}$. On recommence ensuite en prenant n à la place de b, amenant une suite récurrente du type $u_{n+1}=u_n-\dfrac{f(u_n)}{f'(u_n)}$ avec $u_0=b$. On pourrait aussi imaginer de se baser sur le point a à la place de b, mais cela ne convient pas ici (voir la droite en pointillés sur la figure 3 page précédente).

Il y a cependant une difficulté nouvelle : comment accéder aux valeurs de la dérivée ? Deux solutions se présentent : ou bien définir la dérivée comme une nouvelle fonction, et la passer en paramètre de la fonction Newton, ou bien approcher les valeurs de la dérivée. Ces deux démarches sont présentées ci-dessous.



Avec la dérivée	Sans la dérivée	Exécution

Ici, la dérivée est *approchée* (ce qui ne ralentit pas la convergence) et traitée comme une fonction `df` *interne* à la fonction `newton2`. Le langage Python permet cela !

À l'usage : avec quelques tests, on s'apercevra que la méthode de Newton (sous n'importe laquelle des variantes ici présentées) converge beaucoup plus rapidement que les deux autres méthodes, la dichotomie étant la plus lente des trois. Cette question est abordée dans la section suivante.

Pour aller plus loin

Approfondissements et prolongements possibles

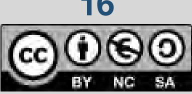
Améliorations : il faudrait adapter les algorithmes de manière à envisager les différents cas (fonction décroissante, etc.). Le cas d'une fonction décroissante peut se traiter en changeant f en $-f$ (le code montré ci-contre traite le cas de la dichotomie à titre d'exemple).

```
EDITEUR : DICHO
LIGNE DU SCRIPT 0027
def dicho(f,a,b,p):
  if a>b:
    a,b=b,a
  if f(a)>f(b):
    i=-1
  else:
    i=1
  c=(a+b)/2
  while c-a>eps:
    if i*f(c)>0:
      b=c
    else:
      a=c
    c=(a+b)/2
  return c
```

Pour la méthode des sécantes comme celle des tangentes, il faudrait tester celui des deux algorithmes (« à gauche » ou « à droite ») qui convient (c'est en fait une question de convexité de la fonction f).

Confrontations : il serait instructif de comparer le nombre d'étapes requises par chacun des algorithmes avant d'atteindre la précision demandée. La comparaison se fera en insérant un compteur `k` dans les boucles `while` et en terminant avec une instruction du genre `return c,k`. Il restera à faire des tests (dans la console Python) pour découvrir quel est le nombre d'étapes réellement consommées par chaque méthode.

Travail de recherche (en groupe si possible) : les trois algorithmes pourraient être « mixés » afin de produire plus rapidement un résultat précis. C'est ainsi qu'on remarque la propension des méthodes des sécantes et de Newton à donner des approximations de sens contraire (l'une donnant des valeurs par défaut et l'autre par excès) : si on les combine, on peut plus aisément obtenir des encadrements, et ainsi garantir une précision donnée.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Le flocon de Von Koch, courbe fractale

Dans le programme (spécialité 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. Suites géométriques : exemples, définition, calcul du terme général.	Sur des exemples, introduction intuitive de la notion de limite, finie ou infinie, d'une suite.

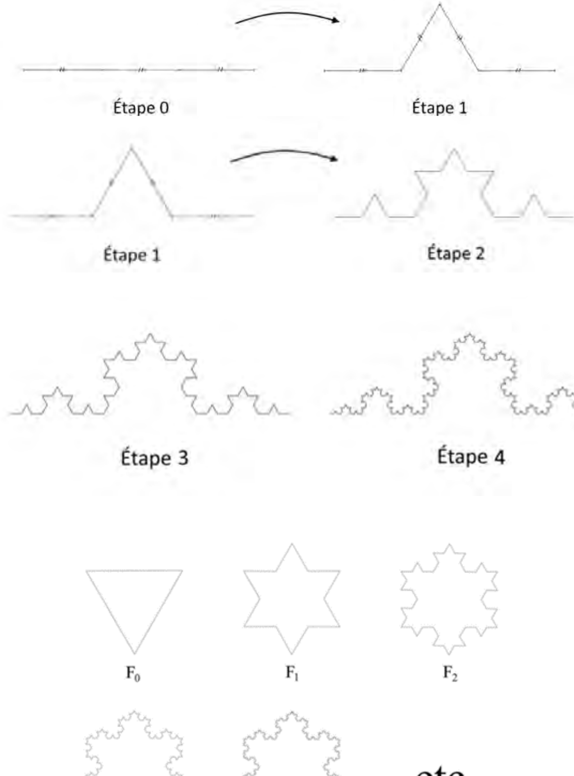
Situation déclenchante

Le flocon de Koch, imaginé en 1904 par le mathématicien suédois Helge Von Koch, est l'une des premières courbes fractales à avoir été décrites. Les fractales permettent notamment de modéliser le développement de certaines bactéries.

Dans cet exercice, nous allons montrer une propriété assez étonnante que possède le flocon de Koch.

La transformation consiste à découper le segment initial en trois segments de même longueur, puis à construire un triangle équilatéral ayant pour base le segment du milieu, et enfin à retirer ce segment servant de base.

On applique désormais ces transformations en prenant pour figure initiale un triangle équilatéral dont les côtés sont de longueur 1, et on applique la transformation de façon à ce que l'aire de la figure obtenue soit supérieure à celle de la figure précédente. Soit n un entier naturel, on note F_0 le triangle équilatéral initial et F_n la figure obtenue à l'étape n .



Buts à atteindre

1. Écrire une fonction Python qui prend comme paramètre un entier naturel n et qui renvoie le nombre de côtés de la figure F_n ainsi que la longueur de ses côtés.
2. Écrire une fonction Python qui prend comme paramètre un entier naturel n et qui renvoie le périmètre de la figure F_n ainsi que l'aire de cette figure. Quelle conjecture peut-on faire sur le périmètre et l'aire de cette figure pour de grandes valeurs de n ?
3. **TI-83** Écrire un script Python à l'aide du module Turtle qui permet de tracer la figure F_n.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Fiche méthode

Proposition de résolution

```
EDITEUR : FLOCON
LIGNE DU SCRIPT 0018
from math import *

def figure(n):
  cote=3
  longueur=1
  for i in range(n):
    cote=cote*4
    longueur=longueur/3
  return [cote,longueur]
```

Pour atteindre l'objectif 1 :

Une fonction `figure` qui prend comme argument un entier naturel *n*. Cette fonction renvoie dans une liste le nombre de cotés ainsi que la longueur d'un côté à l'étape *n*. Ce sont en fait des suites géométriques.

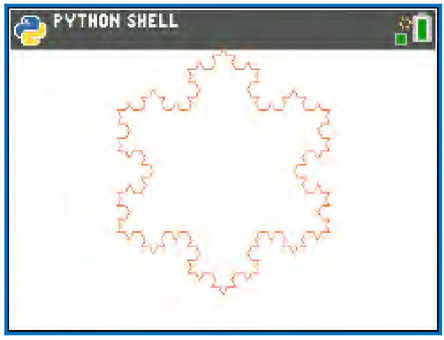
Pour atteindre l'objectif 2 :

```
PYTHON SHELL
>>> # Shell Reinitialized
>>> # L'exécution de FLOCON
>>> from FLOCON import *
>>> figure(4)
[768, 0.01234567901234568]
>>> flocon(4)
[9.481481481481481, 0.6826830343
504216]
```

Une fonction `floc` qui prend comme argument un entier naturel *n* et qui renvoie dans une liste le périmètre du flocon à l'étape *n* ainsi que l'aire de la figure à l'étape *n*.

TI-83 Pour atteindre l'objectif 3 :

- Une fonction `courbeVonKoch` qui permet de définir la transformation de Von Koch.
- Une fonction `trianglekoch` qui permet de définir la figure initiale sur laquelle on applique la transformation.
- Une fonction `trace` qui permet de régler les paramètres d'affichage et qui permet de tracer la figure souhaitée.



Étapes de résolution

Pour atteindre l'objectif 1 :

On initialise le nombre de côtés à 3 et la longueur du côté à 1. Lors de la transformation de Von Koch, à chaque étape, le nombre de côtés est multiplié par 4 et la longueur de chacun est divisée par 3 (voir le code ci-contre en haut).

```
ÉDITEUR : FLOCON
LIGNE DU SCRIPT 0018
def floc(n):
  rac=sqrt(3)/4
  aire=rac # aire 1er triangle
  f=fig(0) # [nbre côtés,long.]
  for i in range(n):
    g=fig(i+1)
    aire=aire+f[0]*rac*g[1]**2
    f=g
  peri=f[0]*f[1]
  return [peri,aire]
```

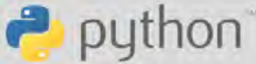
Pour atteindre l'objectif 2 :

On rappelle que l'aire d'un triangle équilatéral de côté *b* est égale à $\frac{\sqrt{3}}{4}b^2$.

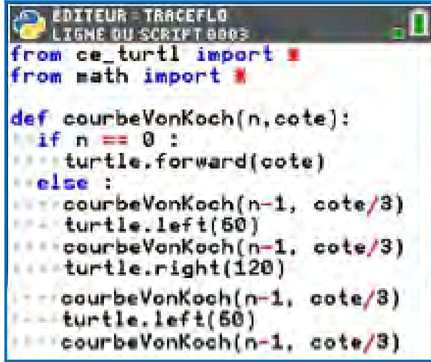
Pour calculer l'aire de la figure F_{n+1}, il faut additionner l'aire de la figure F_n avec l'aire des triangles équilatéraux de côtés de longueur L_{n+1} (où L_n désigne la longueur d'un côté de la figure F_n, valeur calculée grâce à l'instruction `figure(i+1)[1]`). Le nombre de ces triangles est calculé par `figure(i+1)[0]` (rappel : dans une liste `L=[3,8]`, le premier élément est `L[0]`, valant 3 ici).

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



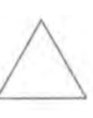
TI-83 Pour atteindre l'objectif 3 :

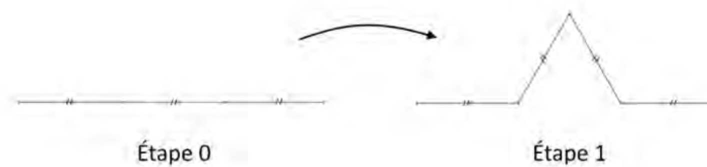


Pour effectuer le tracé il faut importer la bibliothèque `Turtle`[8].

La fonction `courbeVonKoch` permet de définir la transformation de Von Koch à l'aide d'un appel récursif (c'est-à-dire, quand la fonction s'appelle elle-même, ici quatre fois).

La fonction `trianglekoch` permet de définir la figure initiale sur laquelle on applique la transformation. Ici on applique la transformation sur chaque côté du triangle équilatéral montré ci-contre.





La fonction `trace` permet de régler les paramètres d'affichage et permet de tracer la figure souhaitée.

```
ÉDITEUR : TRACEFLO
LIGNE DU SCRIPT 0027
def trianglekoch(n,cote):
  for i in range(3) :
    courbeVonKoch(  n, cote  )
    turtle.right(120)
```

L'utilisation du module `turtle` est détaillée dans l'appendice 2.

Les instructions utilisées ici :

```
ÉDITEUR : TRACEFLO
LIGNE DU SCRIPT 0032
def trace(n):
  turtle.clear()
  turtle.penup()
  turtle.home()
  turtle.goto(-80,60)
  turtle.pendown()
  turtle.pensize(0)
  turtle.color(255,128,64)
  turtle.speed(1)
  trianglekoch(n,160)
  turtle.show()
```

`clear` : efface l'écran
`penup` : lève le stylet
`home` : tortue au centre, tournée vers la droite
`goto` : déplace la tortue vers …
`pendown` : baisse le stylet
`pensize` : taille du stylet (petite)
`color` : couleur du tracé (Rouge – Vert – Bleu)
`speed` : vitesse du tracé (rapide)

`show` : termine le tracé, se met en attente

8 Le module complémentaire `ce_turtl` ou `Turtle` doit être auparavant téléchargé puis installé (voir l'appendice 2).

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Pour aller plus loin

Approfondissements possibles

- Effectuer divers tracés en changeant couleurs, tailles, etc.
- Quand on teste `floc(n)` pour des valeurs croissantes de n, le périmètre de la figure F_n augmente et semble tendre vers l'infini avec n : pourquoi ?

```
PYTHON SHELL
>>> [floc(i) for i in range(16)]
[[3, 0.433], [4.0, 0.577], [5.3,
 0.642], [7.1, 0.67], [9.5, 0.68
3], [12.6, 0.6879999999999999],
[16.9, 0.6909999999999999], [22.
5, 0.692], [30.0, 0.692], [40.0,
 0.693], [53.3, 0.693], [71.0000
0000000001, 0.693], [94.70000000
000001, 0.693], [126.3, 0.693],
[168.4, 0.693], [224.5, 0.693]]
```

- L'aire de la figure F_n semble tendre vers une limite finie (0,692...) quand n tend vers l'infini : c'est le phénomène dit de la « longueur des côtes de la Bretagne » ! Mais pourquoi ?

Voici comment le justifier mathématiquement :

On note a_n l'aire de la figure à l'étape n.

1) On pourra justifier, en argumentant, que pour tout entier naturel n, $a_{n+1}=a_n+\frac{\sqrt{3}}{12}\left(\frac{4}{9}\right)^n$.
2) En déduire que pour tout entier naturel n, $a_n=\frac{\sqrt{3}}{4}+\frac{3\sqrt{3}}{20}\left(1-\left(\frac{4}{9}\right)^n\right)$.
3) Vérifier votre conjecture.

La figure complète :

Le programme ayant produit la figure du haut de cette page est reproduit ci-contre. Sa longueur est surtout due aux commandes graphiques.

```
ÉDITEUR : SNOWFLK2
LIGNE DU SCRIPT 0011
from turtle import *
t=Turtle()

def side(n,l):
  if n==1:
    t.forward(l)
    return
  else:
    side(n-1,l)
    t.left(60)
    side(n-1,l)
    t.right(120)
    side(n-1,l)
    t.left(60)
    side(n-1,l)

colors=[(255,0,0),(0,255,0),(0,0
,255),(200,0,120),(180,180,
0)]

def koch(n,l,a,b):
  for i in range(n-1):
    l=l/3
  t.penup()
  t.goto(a,b)
  t.pendown()
  t.pensize(0)
  cc=colors[n-1]
  t.pencolor(cc[0],cc[1],cc[2])
  for i in range(3):
    side(n,l)
    t.right(120)
  t.penup()

t.clear()
t.hideturtle()
t.speed(10)
koch(1,80,-140,90)
koch(2,80,-140,-25)
koch(3,80,-35,25)
koch(4,80,65,75)
koch(5,80,65,-30)
t.show()
```

Une sélection de couleurs pour les figures successives

`penup` : permet de ne pas tracer tous les déplacements qui seront effectués en dessous.

`pendown` : permet de tracer tous les déplacements qui seront effectués en dessous.

`goto` : envoie la souris à un point défini par coordonnées.

`clear` : pour effacer l'écran avant d'effectuer les tracés.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

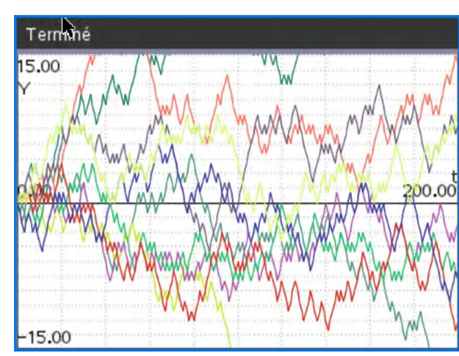
Marches aléatoires

Présentation et objectifs

Dans les programmes

Première : approfondissements possibles	**Terminale :**
• Exemples de succession de plusieurs épreuves indépendantes. • Exemples de marches aléatoires.	• Variables aléatoires indépendantes • Sommes de variables aléatoires. **Approfondissements :** marche aléatoire.

Situation déclenchante



Les marches aléatoires sont des modélisations de phénomènes de nature chaotique comme le déplacement des molécules d'un gaz dans une enceinte fermée, celui de petites particules en suspension dans un liquide ou encore les cours des marchés financiers.

La marche aléatoire unidimensionnelle peut s'expliquer comme un jeu. On place un pion à l'origine d'un axe gradué, et on le déplace avec cette règle : à chaque unité de temps, le pion avance d'un pas (1 unité de longueur), soit à gauche soit à droite et de manière équiprobable. Il s'agit d'étudier le mouvement du pion sur une durée longue, et d'abord de le simuler.

Objectifs 1D (spécialité en Première)

1. On note Y l'abscisse du pion à chaque instant. Simuler la variable Y à l'aide du module `random`.
2. En utilisant le module `ti_plotlib`, représenter la marche aléatoire sur un graphique avec le temps en abscisse et Y en ordonnée (comme le graphe[9] ci-dessus, mais avec une seule courbe).

Objectifs 2D (spécialité en Terminale)

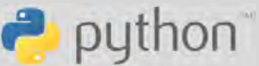
L'étude simultanée de plusieurs variables aléatoires permet d'envisager une marche aléatoire en 2 dimensions. Suivant la même idée, on déplace un pion dans le plan par « pas » suivant quatre directions possibles (équiprobables) : Nord, Sud, Ouest, Est, et on s'interroge sur le trajet suivi par le pion.

3. Simuler la marche aléatoire dans l'environnement Python. On pourra noter (X;Y) les coordonnées du pion.
4. Autre approche : justifier que (X;Y) peut aussi être obtenu à partir de deux marches aléatoires (U;V) en dimension 1, indépendantes, de la manière suivante : $X=(U+V)/2, Y=(U-V)/2$.
5. Représenter graphiquement la marche aléatoire en 2D (on pourra utiliser le module `turtle`).

9 Réalisé avec le logiciel Nspire CX II.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

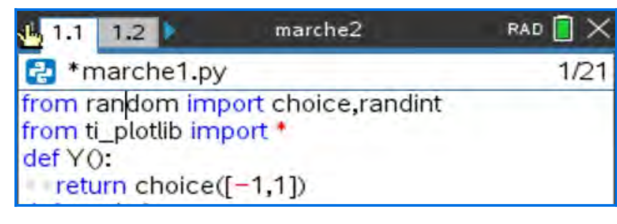


Fiche méthode

Marche aléatoire 1D

▶ **Objectif 1 :**

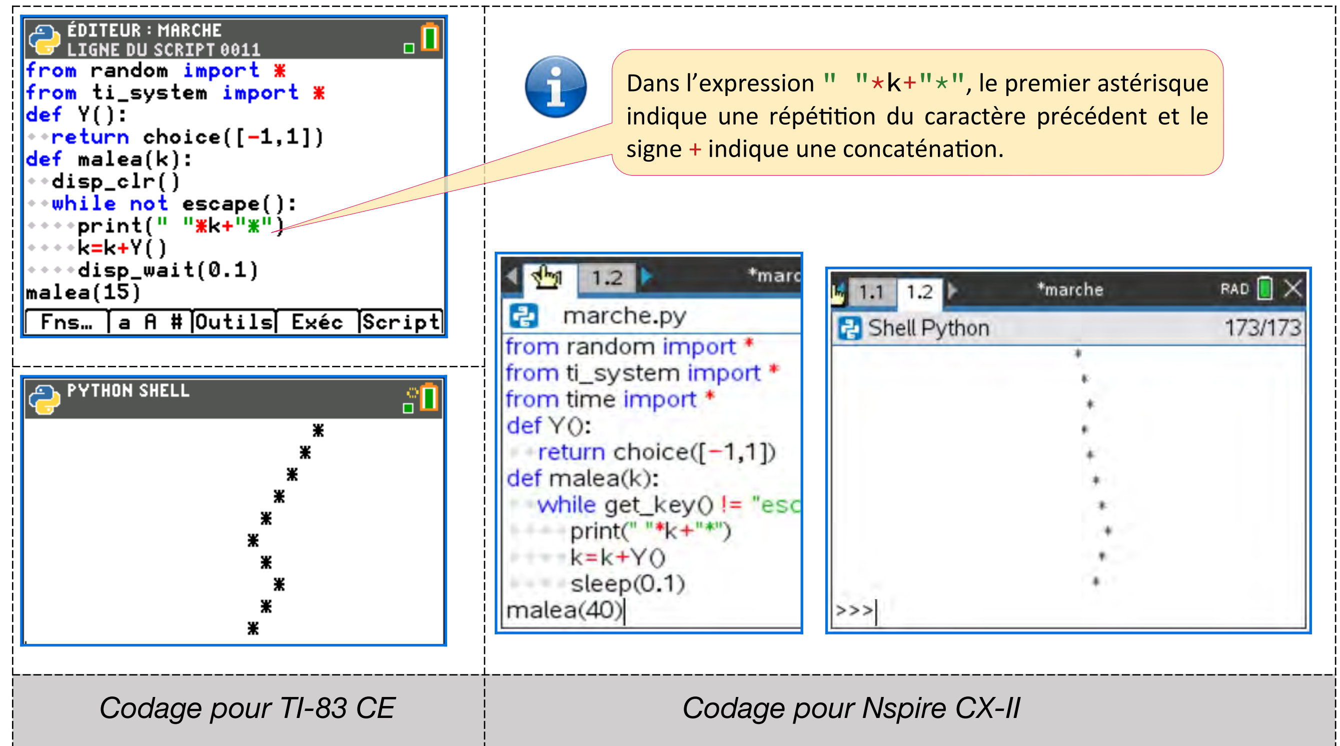
Pour simuler la variable aléatoire Y, on peut se servir de la fonction `randint` fournie par la bibliothèque (ou module) `random`, nécessitant deux paramètres pour préciser l'intervalle d'entiers entre lesquels les nombres au hasard seront fournis. Voir l'appendice 1 pour plus de détails.



Ici, on attend des nombres 0 ou 1 (probabilité ½) ; pour obtenir des nombres -1 ou 1, une transformation arithmétique suffit alors (`2*randint(0,1)-1`). L'espérance est évidemment nulle.

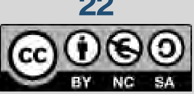
Une autre fonction fournie par ce module est `choice`, qui prend en paramètre la liste des valeurs entre lesquelles un tirage au hasard (équiprobable) est recherché. C'est le mécanisme du « pile ou face ».

Une représentation très simple de la marche aléatoire peut être obtenue en mode texte, en faisant défiler des lignes où une étoile est placée comme le pion. Pour positionner le caractère `*` dans la ligne, on le fait précéder d'un nombre variable d'espaces (c'est le rôle de la variable `k`).



Codage pour TI-83 CE — *Codage pour Nspire CX-II*

On constate que la marche aléatoire s'écarte assez peu du point de départ (traduisant l'espérance nulle), et y revient régulièrement (c'est un théorème dont la démonstration n'est pas au niveau du programme de spécialité).



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

▶ Objectif 2 :

Pour l'affichage graphique, il convient de définir une échelle, ici conventionnellement prise égale à 100 en abscisse et à 10 en ordonnées. À chaque « pas » on doit recalculer la position du pion (variable `z`), tracer le trait correspondant (instruction `line`) et mettre à jour abscisse (`t`) et ordonnée (`y`).

```
EDITEUR : MARCHE2
LIGNE DU SCRIPT 0011
from random import choice
from ti_plotlib import *
def Y():
  return choice([-1,1])
M=200
N=15
t=0
y=0
z=Y()
d=1
cls()
window(0,M,-N,N)
labels("t","Y",6,1)
grid(M/10,N/10)
axes()
color(0,0,255)
while t<M:
  line(t,y,t+d,z)
  t=t+d
  y=z
  z=y+Y()
show_plot()
Fns… | a A # | Outils | Exéc | Script
```

PYTHON SHELL (graphique : Y de -10 à 10, t de 0 à 100)

La fonction `line` fait tracer un segment entre le point de départ (ici de coordonnées (`t,y`)) et le point d'arrivée (ici (`t+d,z`)).

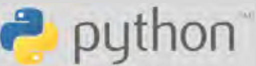
Le paramètre M est la largeur, et durée, de la simulation, et le paramètre N est sa hauteur (estimée).

```
marche1.py
from random import choice,randint
from ti_plotlib import *
def Y():
  return choice([-1,1])
def simul(M):
  t=0
  y=0
  while t<M:
    z=y+Y()
    line(t,y,t+1,z,"")
    t=t+1
    y=z
def courbes(M,N):
  cls()
  window(0,M,-N,N)
  labels("t","Y",6,2)
  grid(M/10,N/10,"dotted")
  axes("on")
  simul(M)
  show_plot()
courbes(200,15)
```

Code réalisant le « tracé » de la marche aléatoire, sur **TI-83**.	*Code esquissant le tracé de la figure illustrant le début de cette fiche, sur* **Nspire CX**.

Ce document est mis à disposition sous licence Creative Commons

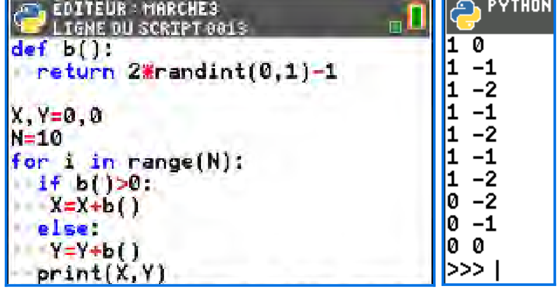
© Texas Instruments 2021 / Photocopie autorisée



Marche aléatoire en 2D

▶ Objectif 3 : une simulation de (X;Y)

Supposons que les coordonnées soient entières et que les « pas » du pion soient d'une unité de longueur. Il nous faut un premier tirage « pile ou face » (-1 ou 1) pour décider si le pion va se déplacer verticalement ou horizontalement, et un second pour décider de la direction qu'il prendra. Le code Python est alors très simple, ci-contre avec un exemple d'exécution associé.



À noter : les trois appels de b ne désignent pas la même valeur, puisque chaque appel à la fonction `randint` ou `choice` « relance la pièce ».

▶ Objectif 4 : autre approche

En posant $X=\frac{U+V}{2}, Y=\frac{U-V}{2}$, on a l'initialisation $(X;Y)=(0\,;\,0)$, et les évolutions possibles détaillées ci-contre, ce qui se trouve en accord avec les règles de déplacement du pion. Grâce au principe d'additivité de l'espérance, on en déduit que les espérances de X et Y sont nulles.

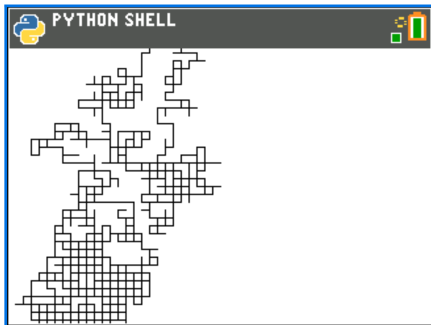
U	V	X	Y
+1	+1	+1	0
–1	–1	–1	0
+1	–1	0	+1
–1	+1	0	–1

▶ Objectif 5 : représentation graphique

Nous allons maintenant employer le module additionnel Turtle (présenté dans l'avant-propos et détaillé dans l'appendice 2).

Le programme ci-contre[10] est très simple : à chaque étape il tire un angle au hasard, multiple de 90°, oriente la tortue en conséquence puis provoque son avancement de 6 « pas » (ou pixels). La figure tracée ressemble à celle qu'on voit à droite. On constate que, cette fois, le « pion » ne repasse que très rarement, voire jamais, sur son point de départ ; d'autres conjectures peuvent être faites (éloignement, franchissement de droites, etc.).

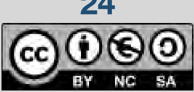
```
EDITEUR : MARCHE3
LIGNE DU SCRIPT 0009
from random import *
from ce_turtl import *
turtle.clear()
for i in range(2000):
  dir=90*randint(0,3) # degrés
  turtle.setheading(dir)
  turtle.forward(6)
```



D'autres expérimentations

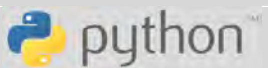
Il est dès lors facile de simuler une marche aléatoire évoluant avec des angles de 60° au lieu de 90°, ou avec des angles un peu quelconques. Autant le « retour à l'origine » est possible avec un angle de 60°, autant il n'a pratiquement plus jamais lieu avec d'autres angles. À essayer !

10 Ce programme devra être adapté selon la calculatrice employée et les mises à jour du module Turtle.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

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 <u>cumulés</u> des contaminations au CoVid-19 (ci-contre)[11]:

(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

11 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

croissance évoqué par Giancarlo (proportionnalité des accroissements).
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.

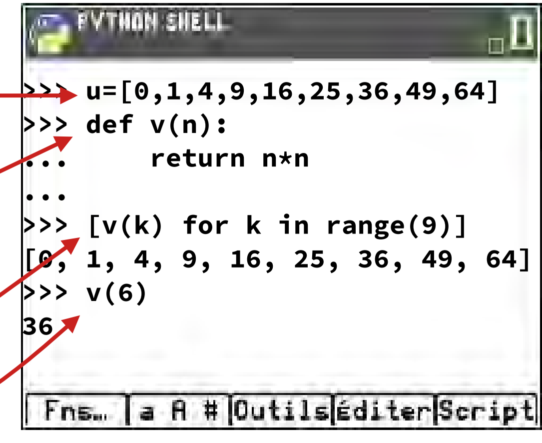
Coder une suite en Python

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

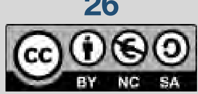
On peut représenter une suite (u_n) de nombres en portant n en abscisses et la valeur u_n en ordonnées. En représentant les suites à comparer sur le même graphique, on a un bon moyen pour conjecturer (visuellement) un comportement particulier : au vu de la liste des premiers termes, cette suite semble-t-elle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

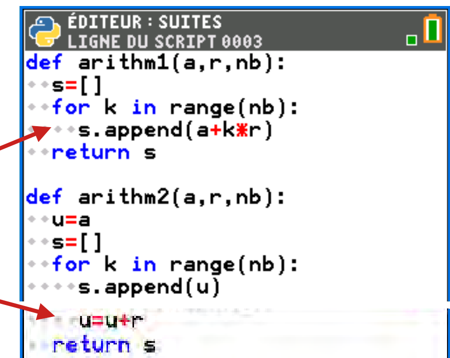


Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

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 !

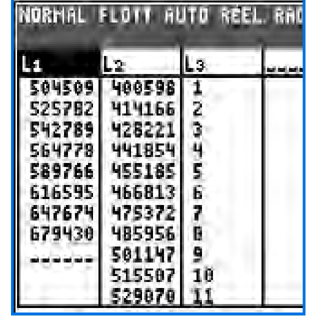
- Soit on s'appuie sur une formule explicite : $u_n = u_0 + n \times r$.
- Soit on calcule les termes en utilisant la relation de récurrence $u_{n+1} = u_n + r$.

Le codage Python suit ces idées.

```
PYTHON SHELL
>>> arithm1(1,3,9)
[1, 4, 7, 10, 13, 16, 19, 22, 25
]
>>> geom2(2,1.2,8)
[2, 2.4, 2.88, 3.456, 4.1472, 4.
97664, 5.971967999999999, 7.1663
61599999999]
```

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).

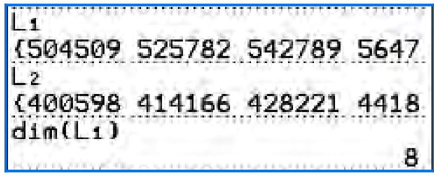
Objectif 2 : représenter des suites


NORMAL FLOTT AUTO RÉEL RAD

L1	L2	L3
504509	400598	1
525782	414166	2
542789	428221	3
564778	441854	4
589766	455185	5
616595	466813	6
647674	475372	7
679430	485956	8
------	501147	9
	515507	10
	529070	11

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

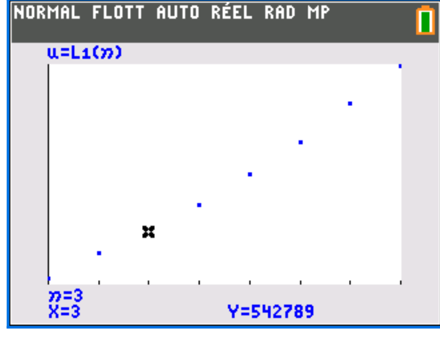
Pour ce faire, on peut commencer par saisir les listes a et b dans l'environnement natif de la calculatrice[12], disons dans les listes prédéfinies L_1 et L_2. Cela se fait commodément en appuyant sur la touche [stats], choix 1 Modifier (voir ci-contre). On ajoute en troisième colonne la suite des entiers (en prévision des représentations graphiques), ici associée à la liste L_3.

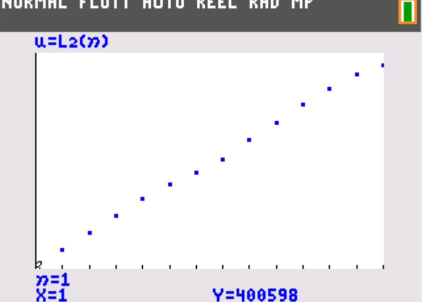


Pour afficher les suites, on règle le mode graphique (touche [mode]) en « suite » et « point épais », puis on choisit la suite à tracer (touche [f(x)]), on définit la fenêtre (touche [fenétre]) et on trace (touche [trace]).

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.

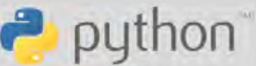




12 Pour alléger les affichages, nous avons saisi la suite b en lui soustrayant d'emblée 2 000 000.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

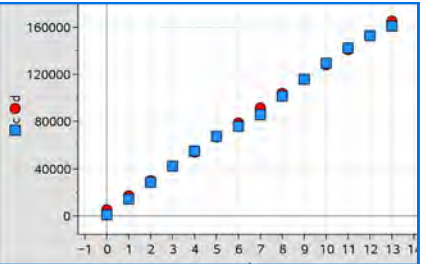
Il ne nous reste plus qu'à transférer ces listes dans l'environnement Python, ce qui se fait en important la bibliothèque `ti_system` et en appelant la fonction `recall_list`.

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 $c=(c_n)$).

```
EDITEUR : 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".

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.



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.

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

```
EDITEUR : 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
```

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[13].

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.

```
EDITEUR : 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
```

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.

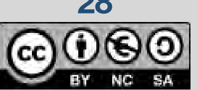
```
PYTHON SHELL
>>> cari(c)
[14166.0, 14110.5, 13951.33, 13796.25, 13362.6, 12562.0, 12279.43, 12643.38, 12834.11, 12907.0, 12889.36, 12752.67, 12381.31]
```

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).

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

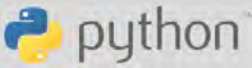
```
EDITEUR : SUITES2
LIGNE DU SCRIPT 0070
a=recall_list("1")
b=recall_list("2")
c=[]
d=[]
for i in range(len(b)):
  c.append(b[i]-400000)
  d.append(12300*i+5000)
trc(c,d)
store_list("4",c)
store_list("5",d)
```

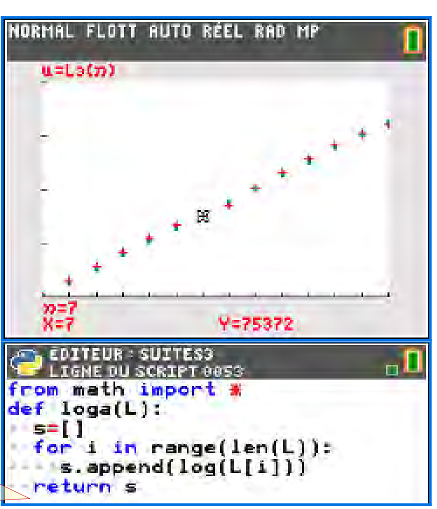
13 La définition de la variable `r` comme quotient n'est pas idéale : il vaudrait mieux formuler le test d'égalité avec deux produits faisant intervenir `L(0)` et `L[1]` de chaque côté du signe `==`.



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 ?*

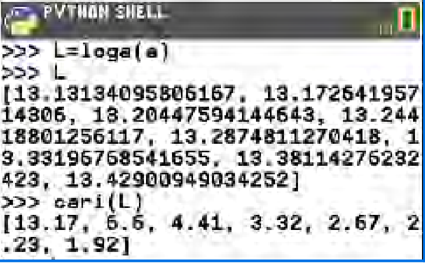
La croissance géométrique

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`.

On crée la liste des logarithmes des termes de L.

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

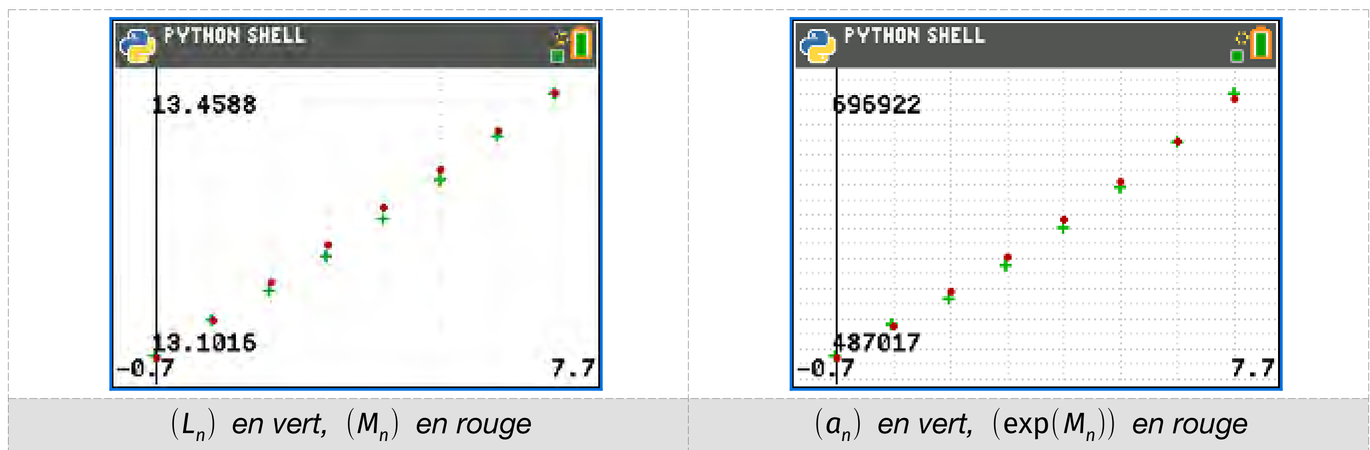


Pour en juger, on forme avec la fonction `loga` la liste des logarithmes des premiers termes de la suite `a`, puis on enchaîne avec la fonction `cari`. On obtient une liste suggérant une suite positive décroissante… peut-être convergente ! Il n'est en fait pas possible de conclure par cette approche,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

```
PYTHON SHELL
>>> M=[]
>>> for i in range(7):
...     M.append(L[i+1]-L[i])
...
>>> M
[0.0413009990813844, 0.031839984
30336482, 0.03971207111474406, 0
.04329311448062789, 0.0444865583
747589, 0.04917507690767309, 0.0
4786672801829184]
```

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 $M_n=13{,}1+0{,}043\,n$, conduisant à approcher a_n par $\exp(M_n)=\exp(13{,}1)\times\exp(0{,}043)^n\approx 503833\times 1{,}043^n$.

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

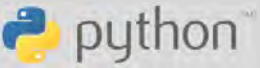


(L_n) *en vert,* (M_n) *en rouge* | (a_n) *en vert,* $(\exp(M_n))$ *en rouge*

Ainsi, Federico n'a pas tort non plus !

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



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.

```
show_plot()
```

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.

PYTHON SHELL 176993 15437.9 -1.3 14.3

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 …

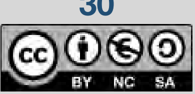
Revenons sur le dialogue entre nos amis italiens. Y a-t-il un modèle qui vaille mieux que les autres ? Tout dépend de l'intervalle de temps au long duquel on espère avoir une bonne adéquation du modèle, comme de la qualité de l'ajustement !

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

L. DIDIER & R. CABANE

L'approximation des intégrales

Comment « calculer » une intégrale ?

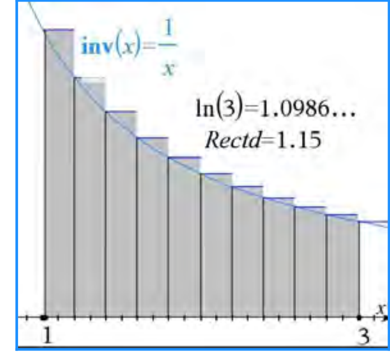
Dans le programme (spéciaité Terminale)

Contenus	**Capacités attendues**
Définition de l'intégrale d'une fonction continue positive définie sur un segment $[a,b]$, comme aire sous la courbe représentative de f. [Lien entre intégrales et primitives]	Estimer graphiquement ou encadrer une intégrale. Calculer l'aire entre deux courbes. **Approfondissements** Approximation d'une aire par l'utilisation de suites adjacentes. **Exemples d'algorithme** Méthodes des rectangles, des milieux, des trapèzes.

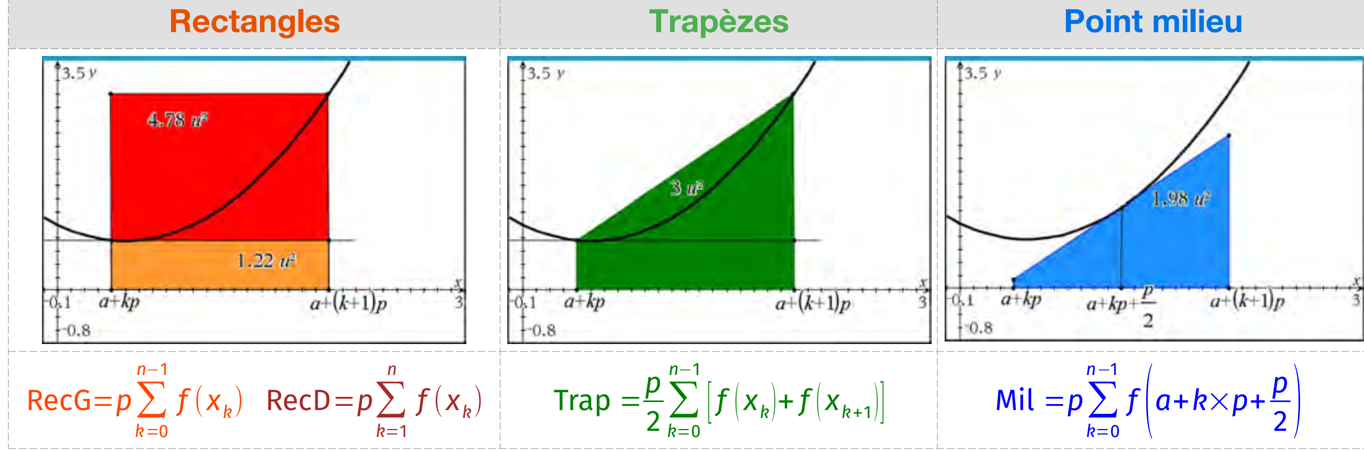
Situation déclenchante

– (l'élève) Madame, peut-on toujours calculer des intégrales avec des formules comme pour les dérivées ?
– Non, ce n'est pas toujours possible. Le mathématicien Liouville l'a démontré en 1835.
– Alors, on fait comment ?
– Si tu acceptes une petite erreur d'approximation, on peut y parvenir.
– D'accord. Comment ça se passe ?
– Tu pourrais déjà découper ton intégrale en « bandelettes » très fines, dont l'aire est bien proche de celle d'un rectangle …
– Ah ! Mais je peux même faire un peu mieux !

Il s'agit de déterminer une valeur approchée de l'intégrale d'une fonction f continue et positive sur un intervalle $[a\,;b]$ grâce aux algorithmes des rectangles, milieux ou trapèzes. Pour approcher $\int_a^b f(x)\,\mathrm{d}x$ on commence par la découper en n sous-intégrales sur des intervalles successifs, chacun de longueur $p=\frac{b-a}{n}$ (relation de Chasles, voir ci-contre). Les « points » du découpage sont $x_k=a+k\times p$ pour $0\leq k\leq n$.

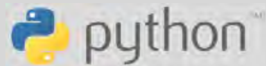


On approche cette dernière par l'aire d'un rectangle ou d'un trapèze, en suivant l'une des méthodes indiquées ci-dessous.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



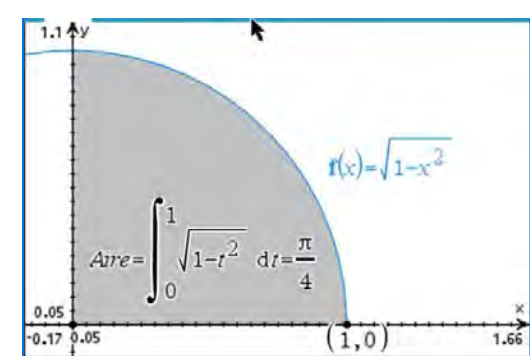
Objectifs

▶ Objectif 1 : programmer le calcul approché, travail de groupe

La classe se répartit en plusieurs petits groupes, qui choisissent l'un des quatre algorithmes cités, et les programment en langage Python.

À la suite, les groupes testent leurs programmes pour le calcul de $\ln(2) \approx 0.6931472$ (comme intégrale de la fonction inverse définie sur l'intervalle $[1\,;2]$ par $\text{inv}(x)=\frac{1}{x}$) et comparent leurs résultats : valeurs par excès ou par défaut, avec quel écart ?

Validation



Pour tester nos algorithmes, nous pouvons tenter d'approcher « au mieux » $I=\int_0^1 \sqrt{1-x^2}\,\mathrm{d}x$. Cette intégrale est bien connue[14] : elle mesure l'aire d'un quart de disque de rayon 1, valant $\frac{\pi}{4}$ unités d'aire (voir la figure ci-contre[15]).

▶ Objectif 2 : les primitives

Dernier objectif : tracer la courbe représentative d'une primitive F d'une fonction f définie et continue sur un certain intervalle. On pourrait évidemment approcher les valeurs de la primitive au moyen du travail antérieur, mais c'est peu efficace car on recalculerait de nombreuses fois les mêmes valeurs. Mieux vaut donc procéder « de proche en proche » : une fois $F(x)$ approchée, on peut approcher $F(x+h)$ (h étant le « pas », une petite valeur) à partir de la formule $F(x+h)=F(x)+\int_x^{x+h} f(t)\,\mathrm{d}t$, où l'intégrale peut être estimée par l'une des méthodes précédentes.

On propose donc d'écrire une fonction ayant pour appel : `prim(f,a,b,h,y0)`, `f` étant la fonction à primitiver, `a` et `b` les bornes, `h` le « pas » et `y0` la valeur de la primitive au point *a*, et renvoyant une liste formée de deux listes, celle des abscisses utilisées et celle des ordonnées correspondantes pour *F*.

En d'autres termes, il s'agit de calculer, de proche en proche, les valeurs de $F(x)=y_0+\int_a^x f(t)\,\mathrm{d}t$.

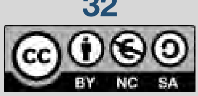
Et un exemple à traiter : le logarithme bien sûr (en partant de $\ln(1)=0$).

Attention : la fonction « logarithme népérien » (notée **ln**) est codée sous le nom `log` en Python !

---

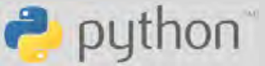
14 La primitive de la fonction $x \mapsto \sqrt{1-x^2}$ a une expression fort compliquée.

15 Cette figure, comme les précédentes, a été réalisée avec le logiciel TI-Nspire CX CAS.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

Une fonction = un objet

Dans nos fonctions d'approximation `recg`, `recd`, `mil`, `trap`, on a quatre paramètres : la fonction *f*, les bornes *a*, *b* et le nombre *n* de sous-intervalles à la base du découpage.

Notons qu'une fonction peut être passée comme paramètre à une autre fonction (voir l'Appendice 1 pour plus de détails).

Objectif 1 : programmes en Python

La programmation des approximations en rectangles à gauche (`recg`) et à droite (`recd`) suit exactement les prescriptions antérieures. On définit deux variables internes (locales) à la fonction, `s` pour accumuler les valeurs de la fonction `f` et `p` pour la largeur des intervalles (le « pas »).

Pour les trapèzes, on évite de transcrire directement le calcul des aires des trapèzes, soit $p\times\frac{f(x_k)+f(x_{k+1})}{2}$, car cela conduirait à calculer deux fois les valeurs de la fonction f excepté les valeurs extrêmes $f(a)$ et $f(b)$.

On garde ainsi le même schéma d'accumulation de valeurs dans une variable `s`, mais on initialise `s` avec les valeurs extrêmes.

On utilise ici une variante du mécanisme `range` : `range(1,n)` fait prendre à `k` les valeurs entières de 1 à `n-1` (inclus).

```
ÉDITEUR : RECTANGL
LIGNE DU SCRIPT 0013
def recg(f,a,b,n):
  s=0
  p=(b-a)/n
  for k in range(n):
    s=s+f(a+k*p)
  return s*p

def recd(f,a,b,n):
  s=0
  p=(b-a)/n
  for k in range(n):
    s=s+f(a+k*p+p)
  return s*p

def trap(f,a,b,n):
  s=(f(a)+f(b))/2
  p=(b-a)/n
  for k in range(1,n):
    s=s+f(a+k*p)
  return s*p

def mil(f,a,b,n):
  s=0
  p=(b-a)/n
  for k in range(n):
    s=s+f(a+k*p+p/2)
  return s*p

def f(x): return sqrt(1-x*x)

def inv(x): return 1/x
```

Pour le point milieu, on décale simplement le point d'évaluation de f d'une demi-longueur $\frac{p}{2}$.

Validations

On commence par intégrer la fonction « inverse » (`inv`) entre 1 et 2, en vue d'approcher $\ln(2)$; on voit de suite que pour un même nombre d'intervalles c'est l'approximation des milieux qui semble la meilleure. Cela dit, pour réaliser une approximation conséquente il faudrait augmenter substantiellement le nombre d'intervalles, au prix d'un temps de calcul considérable.

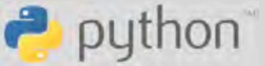
```
PYTHON SHELL
>>> recg(inv,1,2,100)-log(2)
0.002506249921878867
>>> recd(inv,1,2,100)-log(2)
-0.002493750078121137
>>> trap(inv,1,2,100)-log(2)
6.249921878809239e-06
>>> mil(inv,1,2,100)-log(2)
-3.124931644671314e-06
>>> mil(inv,1,2,1000)-log(2)
-3.124999337078549e-08
```

Pour l'aire du quart de disque, même constat : un meilleur algorithme peut conduire à de meilleures approximations, mais nous sommes encore loin de pouvoir atteindre des précisions de l'ordre de 10^{-9} ou mieux !

```
>>> pi/4
0.7853981633974483
>>> recg(f,0,1,100)
0.7901042579447612
>>> recd(f,0,1,100)
0.7801042579447612
>>> trap(f,0,1,100)
0.7851042579447612
>>> mil(f,0,1,100)
0.7854842144750021
```

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



Objectif 2 : primitives

Nous allons ici employer la méthode des milieux. En vue du tracé de courbe, il est plus simple de « remplir » deux listes (ici LX et LY), l'une avec les abscisses et l'autre avec les ordonnées des points de la courbe représentative de la primitive.

```
ÉDITEUR : RECTANGL
LIGNE DU SCRIPT 0035
def prim(f,a,b,p,y0):
  LX=[a]
  LY=[y0]
  x=a
  y=y0
  while x<b:
    x=x+p
    y=y+f(x+p/2)*p
    LX.append(x)
    LY.append(y)
  return [LX,LY]
Fns… | a A # | Outils | Exéc | Script
```

On initialise le processus avec la valeur attendue pour la primitive au point a, soit ici y0.

La fonction prim() renvoie un couple (une liste) formé de deux listes (abscisses, ordonnées).

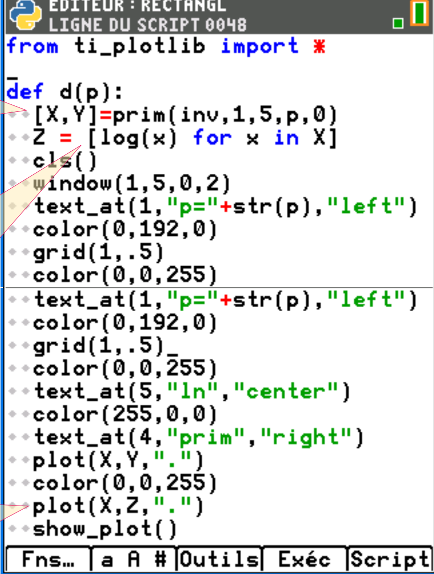
TI-83 La mise en œuvre graphique nécessite de faire appel au module ti_plotlib (ne fonctionne pas sur la TI-82). Pour comparer avec la « courbe » représentative du logarithme, on crée dans la liste Z les ordonnées correspondantes aux mêmes abscisses. Le reste n'est plus que mise en place graphique.

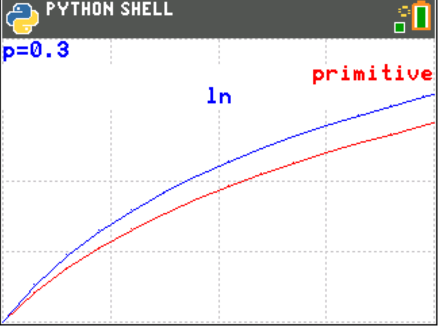
Les résultats, ci-dessous, montrent que l'approximation s'approche assez correctement de la courbe du logarithme. Compte tenu de la mémoire disponible sur la TI-83, il est impossible de faire mieux que p=0,045.

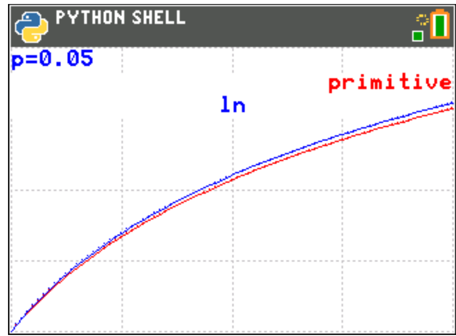
Le résultat de la fonction prim() est rangé dans une liste formée de deux listes.

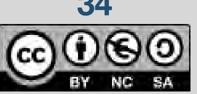
Une « liste en compréhension » permet ici d'avoir rapidement une liste Z formée des logarithmes des points de X (voir Appendice 1).

Une commande de tracé : la liste des points d'abscisses dans X, ordonnées dans Z, largeur fine.









Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

Le calcul des logarithmes

Présentation et objectifs

Dans le programme (spécialité Terminale)

Contenus	**Capacités attendues**
Fonction logarithme népérien. Propriétés algébriques du logarithme. Fonction dérivée du logarithme, variations.	Utiliser l'équation fonctionnelle de l'exponentielle ou du logarithme … **Exemple d'algorithme** Algorithme de Briggs pour le calcul du logarithme.

L'histoire du calcul des logarithmes

C'est l'Écossais John Napier (ou Neper) (1550-1617) qui inventa les logarithmes, initialement dans le but de faciliter les calculs propres aux activités de banque et de commerce, puis, assez rapidement, pour les très longs calculs trigonométriques nécessaires pour la navigation. Très vite s'est posée la question du calcul effectif (mais approché) en vue de construire des « tables de logarithmes ». Différentes méthodes furent aussitôt proposées, dont celle que nous allons étudier ici.

Henry Briggs (1561-1630) qui fut un mathématicien, astronome et géographe anglais très influent en son époque, s'attela au problème ; il inventa aussi les logarithmes décimaux.

Le principe

Imaginons un texte que Briggs aurait pu écrire, où il expliquerait comment calculer des logarithmes « naturels » (népériens)…

« Pour calculer le logarithme d'un certain nombre positif et non nul, je prends la racine carrée de ce nombre, de manière répétée jusqu'à obtenir un résultat suffisamment proche de 1. À la suite, je soustrais 1 et je double le résultat autant de fois que j'ai fait de racines auparavant. »

L'idée de l'**algorithme de Briggs** est donc la suivante : pour approcher $\ln(x)$ (pour $x>0$), on remplace x par $\sqrt{x}$ (n fois), l'entier n étant « grand » (on y reviendra plus loin). L'approximation à donner est alors $2^n(x-1)$.

ARITHMETICA
LOGARITHMICA
SIVE
LOGARITHMORVM
CHILIADES TRIGINTA, PRO
numeris naturali serie crescentibus ab vnitate ad
20,000 : et a 90,000 ad 100,000. Quorum ope multa
perficiuntur Arithmetica problemata
et Geometrica.

HOS NVMEROS PRIMVS
INVENIT CLARISSIMVS VIR IOHANNES
NEPERVS Baro Merchistonij : eos autem ex eiusdem sententia
mutavit, eorumque ortum et vsum illustravit HENRICVS BRIGGIVS,
in celeberrima Academia Oxoniensi Geometriæ
professor SAVILIANVS.

DEVS NOBIS VSVRAM VITÆ DEDIT
ET INGENII, TANQVAM PECVNIÆ,
NVLLA PRÆSTITVTA DIE.

I R

LONDINI,
Excudebat GVLIELMVS
IONES. 1624.

Frontispice de l'ouvrage publié par Briggs (crédit : Jacques Laporte)

Quelques explications

On peut formaliser l'algorithme de Briggs à l'aide d'une suite récurrente définie par $u_0=x$ et $u_{n+1}=\sqrt{u_n}$ pour tout entier naturel n, dont on démontre qu'elle converge vers 1.

On utilise alors l'approximation de la courbe représentative du logarithme par sa tangente au point

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

d'abscisse 1, soit : $\ln(x) \approx x-1$ pour des nombres réels x proches de 1. Dans le cas présent : si u_n est proche de 1 alors $\ln(u_n)$ est proche de $u_n - 1$; sachant que $\ln(u_n) = \frac{1}{2^n}\ln(x)$, on en vient à considérer que $\ln(x)$ est « à peu près égal » à $v_n = 2^n(u_n - 1)$.

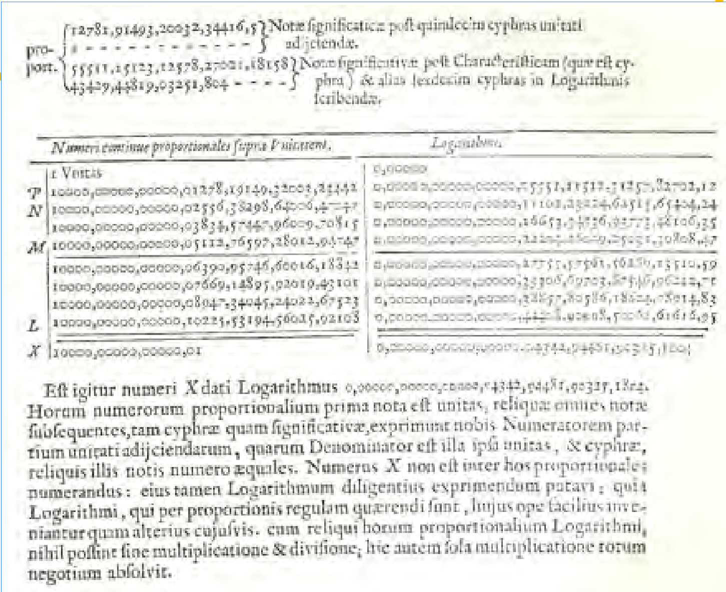
Il reste à choisir n de manière adéquate pour garantir la précision souhaitée. Vient alors une seconde idée : ce qui précède peut aussi bien être conduit avec $\frac{1}{x}$ à la place de x, aboutissant cette fois à une approximation de $\ln\left(\frac{1}{x}\right) = -\ln(x)$.

On définit donc la suite (U_n) par $U_0 = \frac{1}{x}$ pour $x > 0$ puis $U_{n+1} = \sqrt{U_n}$ pour tout entier naturel n, et enfin $V_n = 2^n(1 - U_n)$ qui converge aussi vers $\ln(x)$.

Nous admettrons que les suites (v_n) et (V_n) sont monotones et de sens contraires ; elles vont donc fournir des encadrements de $\ln(x)$, d'autant plus resserrés que n sera grand.

Objectifs

1. Écrire une fonction en langage Python donnant les approximations du logarithme de x pour un nombre d'itérations n donné, et comparer ces résultats avec la valeur fournie par la calculatrice, pour diverses valeurs de x : $x=2, x=20, x=404, x=0.05$ et diverses valeurs de n. Qu'observe-t-on quand n dépasse 40 ?
2. Écrire une fonction en langage Python donnant une approximation du logarithme de x pour une précision p donnée, et tester cette fonction pour $x=2, x=20, x=404, x=0.05$ et pour une précision $p=10^{-6}$. Pour ce faire, engager simultanément les algorithmes de Briggs pour le calcul de $\ln(x)$ et de $-\ln\left(\frac{1}{x}\right)$, et stopper dès que l'écart entre les deux approximations est suffisamment faible (par rapport à p).



pro-port. 12781,91493,20032,34416,5 } Notæ significaticæ post quindecim cyphras unitati adjiciendæ.
55511,15123,12578,27021,18158 } Notæ significativæ post Characteristicam (quæ est cyphra) & alias sedecim cyphras in Logarithmis scribendæ.
43429,44819,03251,804

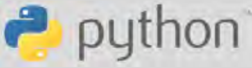
	Numeri continue proportionales supra Vnitatem.	Logarithmi.
	1 Vnitas	0,00000
P	10000,00000,00000,01278,19149,32003,23442	[illegible]
N	10000,00000,00000,02556,38298,64006,[illegible]	[illegible]
	10000,00000,00000,03834,57447,96009,70815	[illegible]
M	10000,00000,00000,05112,76597,28012,[illegible]	[illegible]
	10000,00000,00000,06390,95746,60016,18342	[illegible]
	10000,00000,00000,07669,14895,92019,43101	[illegible]
	10000,00000,00000,08947,34045,24022,67523	[illegible]
L	10000,00000,00000,10225,53194,56025,92103	[illegible]
X	10000,00000,00000,01	[illegible]

Est igitur numeri X dati Logarithmus [illegible]. Horum numerorum proportionalium prima nota est unitas, reliquæ omnes notæ subsequentes, tam cyphræ quam significativæ, exprimunt nobis Numeratorem partium unitati adjiciendarum, quarum Denominator est illa ipsa unitas, & cyphræ, reliquis illis notis numero æquales. Numerus X non est inter hos proportionales numerandus : eius tamen Logarithmum diligentius exprimendum putavi ; quia Logarithmi, qui per proportionis regulam quærendi sunt, hujus ope facilius inveniantur quam alterius cujusvis. cum reliqui horum proportionalium Logarithmi, nihil possint sine multiplicatione & divisione; hic autem sola multiplicatione totum negotium absolvit.

Ci-contre : une table de calcul de Briggs (crédit : Jacques Laporte)

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

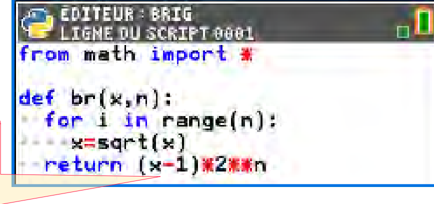


Fiche méthode

Objectif 1 : codage en Python

Il est très simple du moment que n est donné (voir ci-contre). Les résultats sont acceptables tant qu'on ne fait pas trop dépasser à n la valeur 30.

Pour le codage des nombres et des puissances, voir l'annexe 1.

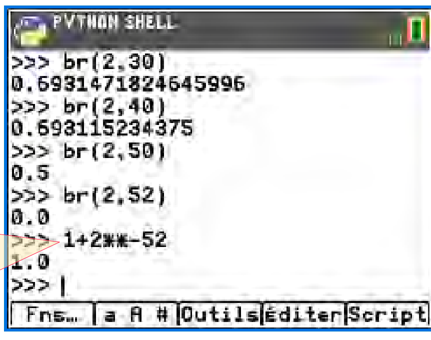


Une anomalie ?

Quand on prend $n=52$, on obtient une réponse aberrante.

Explication : les racines répétées finissent par produire des nombres extrêmement proches de 1, tellement que la machine les confond avec le nombre 1. En effet, les nombres réels « de la machine » (dits « à virgule flottante ») n'ont qu'un nombre limité de décimales (ou plutôt de bits) !

Dans l'environnement Python, les nombres 1.0 et $1+2^{-52}$ semblent égaux (annexe 1) .

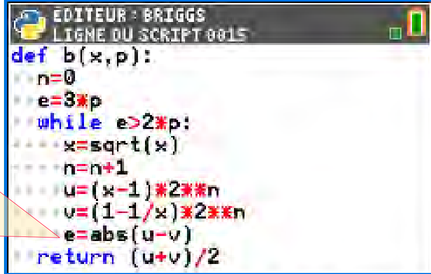


Objectif 2 : codage en Python

L'algorithme va s'inspirer du précédent, en conduisant simultanément les itérations pour l'approximation du logarithme de x (variable y), et celles pour l'opposé du logarithme de $\frac{1}{x}$ (variable z).

Comme ici n n'est pas fixé à l'avance, on continue tant que (boucle `while`) l'écart entre les variables y et z (variable e, valeur absolue de $y-z$) dépasse le double de la précision p donnée (on prend le double parce que la réponse finale se fait avec la moyenne des deux valeurs, divisant l'écart par 2). L'initialisation de l'écart e doit se faire à une valeur suffisamment grande pour que la boucle puisse « démarrer », on a ici choisi le triple[16] de la précision p.

On renvoie la moyenne des deux valeurs encadrantes u et v, ce qui assure un écart maximum de `e/2` par rapport à la valeur cible.




```
PYTHON SHELL
>>> b(20,1e-4)-log(20)
1.043694464186729e-09
>>> b(20,1e-5)-log(20)
6.144018627196601e-11
>>> b(20,1e-6)-log(20)
-1.131827964684362e-10
>>> b(20,1e-5)
2.995732273615431
>>> b(404,1e-5)
6.001414877944626
```

On a choisi 20 et 404, qui sont des nombres entiers donc le logarithme est proche d'un entier !

16 Toute valeur supérieure à $2p$ conviendrait.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Pour aller plus loin

Objectif 3 : calculer des logarithmes ?

Briggs s'est surtout occupé de « calculer » les logarithmes des nombres premiers. Pourquoi ?

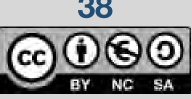
Calculer efficacement des logarithmes !

Il convient ici de se souvenir du fait que tout nombre entier est décomposable en un produit de nombres premiers (éventuellement répétés). Considérons par exemple un nombre entier s'écrivant $n=p^{\alpha}q^{\beta}r^{\gamma}$ avec p, q, r premiers et α,β,γ entiers positifs : on sait calculer son logarithme sous la forme $\ln(n)=\alpha\ln(p)+\beta\ln(q)+\gamma\ln(r)$, pour peu qu'on sache « calculer » les logarithmes des nombres premiers. Et à partir de ce point on sait aussi calculer les logarithmes des nombres rationnels (une fraction étant un quotient de deux nombres entiers), en particulier des nombres décimaux avec un dénominateur puissance de 10.

Prolongements

Quelques lectures pour aller un peu plus loin :

- Vies de John Napier et Henry Briggs (pages Wikipedia en français).
- « Histoire de la fonction logarithme », site de l'académie de Limoges
- « Histoires de logarithmes », publié chez Ellipses par la commission Inter-Irem d'Épistémologie et d'Histoire des Mathématiques (https://www.univ-irem.fr/spip.php?article667).



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

Le triangle de Pascal

Présentation

Dans le programme (spécialité Terminale)

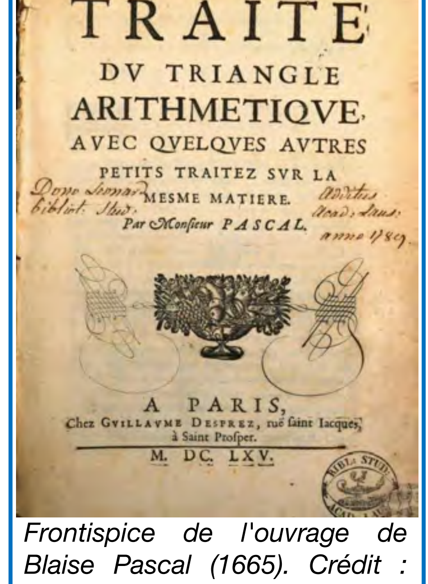
Contenus	**Capacités attendues**	**Algorithmes**
Pour $0 \leq k \leq n$, $\binom{n}{k} = \frac{n!}{k!(n-k)!}$. Relation et triangle de Pascal.	Dans le cadre d'un problème de dénombrement, utiliser une représentation adaptée et reconnaître les objets à dénombrer. Effectuer des dénombrements simples. Démonstration par dénombrement de la formule $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.	Génération de la liste des coefficients $\binom{n}{k}$ à l'aide de la relation de Pascal.

Situation déclenchante

En mathématiques, les coefficients binomiaux, définis pour tout entier naturel n et tout entier naturel k inférieur ou égal à n, donnent le nombre de sous-ensembles différents à k éléments que l'on peut former à partir d'un ensemble contenant n éléments. Les coefficients binomiaux sont utilisés dans de nombreux domaines : binôme de Newton, dénombrement, développement en série, probabilités, etc.…

En 1654, Blaise Pascal écrit son Traité du triangle arithmétique dans lequel il donne une présentation pratique en tableau des coefficients du binôme, le « triangle arithmétique », maintenant connu sous le nom de « triangle de Pascal ».

Il faut noter qu'un mathématicien chinois sous la dynastie des Qin, Yang Hui, avait travaillé quatre siècles plus tôt sur un concept semblable au triangle de Pascal.



Frontispice de l'ouvrage de Blaise Pascal (1665). Crédit : archive.org

Buts à atteindre

1 Écrire une fonction qui prend en paramètres deux entiers naturels n et k et renvoie le coefficient binomial k parmi n. Cette question est une opportunité pour effectuer une activité de groupe.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Travail de groupe : chaque groupe de 2 ou 3 élèves choisit une définition ou propriété caractéristique des coefficients binomiaux et en tire un algorithme de calcul de ceux-ci, puis le programme en langage Python. Ensuite, les groupes échangent leurs productions et en vérifient l'exactitude et l'efficacité. Les élèves pourront choisir parmi les définitions et propriétés suivantes :

$\binom{n}{k}=\frac{k!}{k!(n-k)!}$; $\binom{n}{0}=\binom{n}{n}=1$ et $\binom{n}{k}=\frac{n(n-1)\cdots(n-k+1)}{k!}$ pour $n\in\mathbb{N}^*, k\in\mathbb{N}^*$; $\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k}$; $\binom{n}{k}=\frac{n}{k}\binom{n-1}{k-1}$.

2 Nombre de parties à k éléments d'un ensemble de n éléments : écrire une fonction prenant en paramètre un entier naturel n, permettant de construire le triangle de Pascal jusqu'à la ligne de rang n, et renvoyant la liste des listes des coefficients binomiaux.

3 Écrire une fonction qui prenant comme paramètre un entier naturel n et renvoyant la liste des $\sum_{k=0}^{i}\binom{i}{k}$ (i variant de 0 à n). Quelle conjecture peut-on faire ?

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Fiche méthode

Proposition de résolution

Préalable informatique :

L'opérateur de division `/` en Python renvoie toujours un nombre à virgule (ou « flottante »). Comme on s'attend bien à ce que les coefficients binomiaux soient des entiers, on emploiera ici la « division entière » `//`, qui renvoie le quotient de la division euclidienne (`a//b` est la partie entière de `a/b`). De fait, nous ne l'emploierons que pour des divisions « tombant juste ».

Pour atteindre l'objectif 1 :

Dans cette partie nous allons présenter différentes manières pour calculer les coefficients binomiaux en essayant de les comparer. La fonction `binome` prend comme paramètres deux entiers naturels n et k et renvoie le coefficient binomial associé.

```
PYTHON SHELL
>>> trianglepascal(5)
[1]
[1, 1]
[1, 2, 1]
[1, 3, 3, 1]
[1, 4, 6, 4, 1]
[1, 5, 10, 10, 5, 1]
[1, [1, 1], [1, 2, 1], [1, 3, 3,
 1], [1, 4, 6, 4, 1], [1, 5, 10,
 10, 5, 1]]
```

Pour atteindre l'objectif 2 :

Une fonction `trianglepascal` qui prend comme argument un entier naturel n et qui renvoie la liste des listes des coefficients binomiaux en affichant le triangle de Pascal.

```
PYTHON SHELL
>>> # L'exécution de PASCAL
>>> from PASCAL import *
>>> conjecture(5)
[1] somme= 1
[1, 1] somme= 2
[1, 2, 1] somme= 4
[1, 3, 3, 1] somme= 8
[1, 4, 6, 4, 1] somme= 16
[1, 5, 10, 10, 5, 1] somme= 32
[1, 2, 4, 8, 16, 32]
```

Pour atteindre l'objectif 3 :

Une fonction `conjecture` prenant comme argument un entier naturel n et renvoyant la liste des $\sum_{k=0}^{i}\binom{i}{k}$ pour tout i entier naturel ($0 \leq i \leq n$).

Objectif 1 : étapes de résolution

Voici cinq propositions correspondant aux différentes définitions.

```
ÉDITEUR : BINOM
LIGNE DU SCRIPT 0010
def facto(n):
  f=1
  for i in range(1,n+1):
    f=f*i
  return f

def binom1(n,k):
  X=(facto(n)//facto(k))//facto(n-k)
  return X
```

La fonction `binom1` est fondée sur la formule de définition des coefficients. Elle pose le problème de la taille des factorielles dès que `n` dépasse 70. Noter l'usage de parenthèses pour fixer l'ordre des opérations arithmétiques.

```
ÉDITEUR : BINOM
LIGNE DU SCRIPT 0020
def binom2r(n,k):
  if k==0:
    return 1
  return binom2r(n,k-1)*(n-k+1)//k

def binom2i(n,k):
  X=1
  for i in range(1,k+1):
    X=X*(n-i+1)//i
  return X
```

La fonction `binom2r` est basée sur un appel récursif[17], au contraire de la fonction `binome2i` (itérative). Tout se fonde sur la formule usuelle de définition des coefficients.

```
ÉDITEUR : BINOM
LIGNE DU SCRIPT 0032
def binom3r(n,k):
  if k==0:
    return 1
  return binom3r(n-1,k-1)*n//k

def binom3i(n,k):
  X=1
  for j in range(1,k+1):
    X=(X*(n-k+j))//j
  return X
```

La fonction `binom3r` est basée sur un appel récursif inspiré de la formule $\binom{n}{k}=\frac{n}{k}\binom{n-1}{k-1}$.

Une version itérative est proposée dans `binom3i`.

17 C'est-à-dire, une fonction qui s'appelle elle-même.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

La fonction `binome4` (ci-contre) permet le calcul de coefficients binomiaux par la formule de Pascal avec des appels récursifs.

```
ÉDITEUR : BINOM
LIGNE DU SCRIPT 0042
def binom4(n,k):
  if n==0 or k==0 or n==k:
    return 1
  return binom4(n-1,k-1)+binom4(
    n-1,k)
```

La fonction `binome5` permet le calcul des coefficients binomiaux par la formule de Pascal et sans appels récursifs ; elle s'appuie sur un calcul du triangle de Pascal, ligne par ligne. Pour éviter d'avoir à garder en mémoire deux lignes du triangle, on réécrit la ligne en cours avec les coefficients nouvellement calculés, ce qui fonctionne si on procède *de droite à gauche* et en initialisant la liste `L` avec `n` valeurs toutes nulles.

`[0]*n` fournit liste de zéros de longueur n.

range(1,0,-1) produit ici des valeurs qui « descendent » (voir annexe 1).

```
ÉDITEUR : PASCAL2
LIGNE DU SCRIPT 0051
def LignePascal(n):
  n=n+1
  L=[0]*n
  L[0]=1
  for i in range(n):
    for j in range(i,0,-1):
      L[j]=L[j-1]+L[j]
  return L

def binome5(n,k):
  return LignePascal(n)[k]
```

Exécutons les différentes fonctions.

```
PYTHON SHELL
>>> binom1(26,17)
3124550
>>> binom1(23,17)
100947
>>> binom2r(23,17),binom2i(23,17
)
(100947, 100947)
>>> binom3r(23,17),binom3i(23,17
)
(100947, 100947)
```

Lors des différentes exécutions laquelle semble la plus rapide ?

Indication : la récursivité peut consommer beaucoup de temps de calcul en raison des appels de fonction qui se multiplient énormément.

▶ Pour atteindre l'objectif 2 :

Le programme utilise les listes que l'on construit ligne par ligne en utilisant la formule du triangle de Pascal. La liste `p` va contenir les coefficients binomiaux de i parmi k pour i allant de 0 à k.

La liste `p2` sera utilisée pour construire la ligne du dessous (en tant que liste de stockage) dans le triangle de Pascal. La liste `r` contiendra les listes des coefficients binomiaux.

```
ÉDITEUR : PASCAL
LIGNE DU SCRIPT 0002
def trianglepascal(n):
  p=[1]
  r=[1]
  print(p)
  k=1
  while k<=n:
    k=k+1
    p2=[1]
    for i in range(1,k-1):
      p2.append(p[i-1]+p[i])
    p=p2
    p.append(1)
    r.append(p)
    print(p)
  return r
```

Initialisations

Relation de Pascal

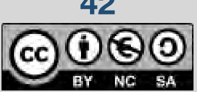
▶ Pour atteindre l'objectif 3 :

Pour calculer les différentes sommes, il suffit d'effectuer la somme des coefficients de chaque ligne du triangle de Pascal.

On peut donc reprendre le programme précédent et rajouter une liste `somme` qui va être construite en effectuant à chaque passage de boucle la somme des coefficients de la liste `p`.

```
ÉDITEUR : PASCAL
LIGNE DU SCRIPT 0054
def conjecture(n):
  p=[1]
  somme=[1]
  print(p,"somme=",1)
  k=1
  while k<=n:
    k=k+1
    p2=[1]
    for i in range(1,k-1):
      p2.append(p[i-1]+p[i])
    p=p2
    p.append(1)
    print(p,"somme=",sum(p))
    somme.append(sum(p))
  return(somme)
Fns… | a A # | Outils | Exéc | Script
```

Une commodité : pour une liste de nombres `L`, l'expression `sum(L)` donne la somme des termes de `L` (cf. appendice 1).



Ce document est mis à disposition sous licence Creative Commons

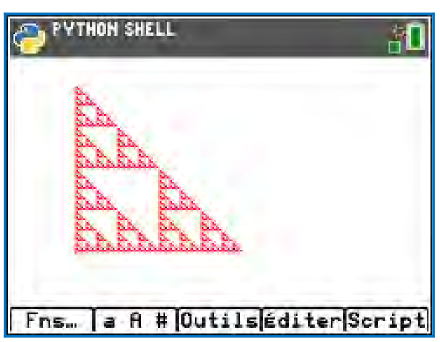
© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

Pour aller plus loin

Prolongement possible (spécifique TI-83)

Dans cette partie nous allons faire apparaître les triangles de Sierpinski en utilisant la parité des coefficients binomiaux.



> Écrire un script Python permettant d'afficher le triangle de Pascal dans lequel on remplacera les coefficients par un pixel rouge si le coefficient binomial est impair et un pixel blanc (ou rien) si le coefficient binomial est pair.

Code Python proposé

TI-83 Le module[18] `ti_draw` permet d'utiliser les instructions associées à l'affichage des pixels.

Les commandes graphiques employées sont simples, voir l'annexe 2 pour plus de détails.

EDITEUR : SIERPNS2
LIGNE DU SCRIPT 0010

```
from ti_draw import *
def trpsc(n):
  clear()
  set_color(255,0,0)
  p=[1] # ligne 0 du triangle
  for k in range(1,n):# ligne k
    p2=[1] # colonne 0
    for i in range(1,k): #col i
      p2.append(p[i-1]+p[i])
    p=p2+[1] # colonne k+1
    for i in range(k+1):
      if p[i]%2==1:
        plot_xy(50+i,50+k,7)
  show_draw()

trpsc(64)
```

On efface dans un premier temps l'écran (clear).

On fixe la couleur du tracé au rouge (set_color).

On calcule les coefficients binomiaux comme dans l'objectif 2.

Une fois les coefficients d'une ligne calculés, en fonction de la parité, on allume (plot_xy) certains pixels en rouge.

On met en attente (show_draw).

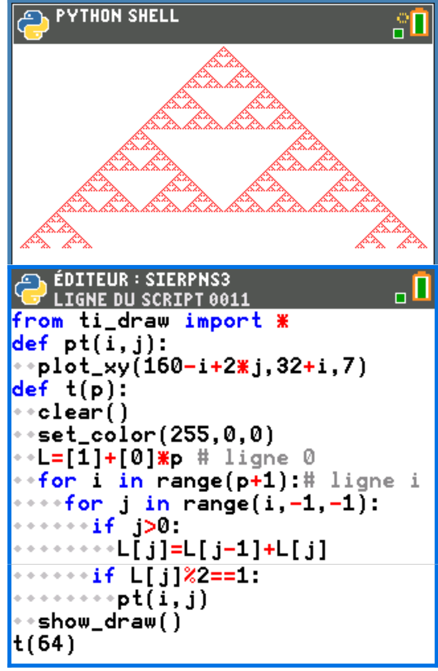
18 Ce module (ou bibliothèque) est accessible comme module complémentaire avec le système 5.7.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Un défi

On peut souhaiter mieux « orienter » le triangle de Sierpinski et obtenir le schéma ci-contre. Comment faire ?

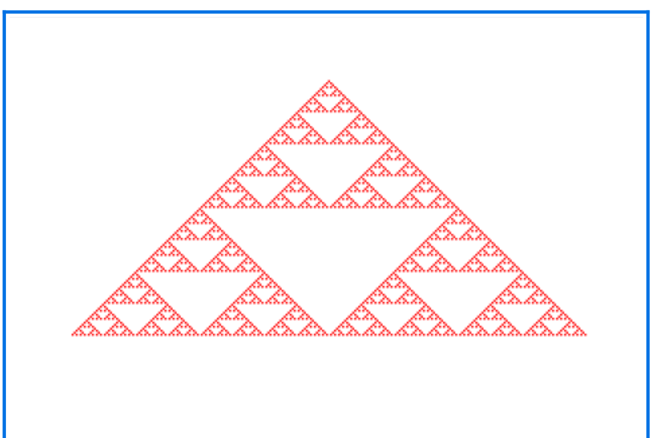


Une première idée est de créer une fonction d'accès aux « pixels » requis pour chaque couple (i,j) dont on veut représenter le coefficient binomial (fonction `pt`, ci-contre). Cela ne change pas l'algorithme et permet de mieux séparer le calcul de l'affichage.

Tant qu'à faire, le calcul est simplifié en recourant à l'algorithme de la fonction `binome5`.

Remarque : il n'est pas nécessaire d'afficher de point quand le coefficient est pair ; on teste donc l'imparité du nombre `L[j]` (qui contient la valeur de $\binom{i}{j}$) en testant si le reste de sa division par 2 est égal à 1, ce qui s'écrit `L[j]%2==1`.

Voici quelques copies d'écran réalisées avec le logiciel Nspire™ CX qui reprennent les algorithmes précédents déclinés sur la calculatrice TI Nspire™ CX II-T.



```
from ti_draw import *
def pt(i,j):
  plot_xy(160-i+2*j,32+i,7)
def t(p):
  clear()
  set_color(255,0,0)
  n=p+1
  L=[0]*n
  L[0]=1
  for i in range(n):
    for j in range(i,-1,-1):
      if j>0:
        L[j]=L[j-1]+L[j]
      if L[j]%2==1:
        pt(i,j)
```

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

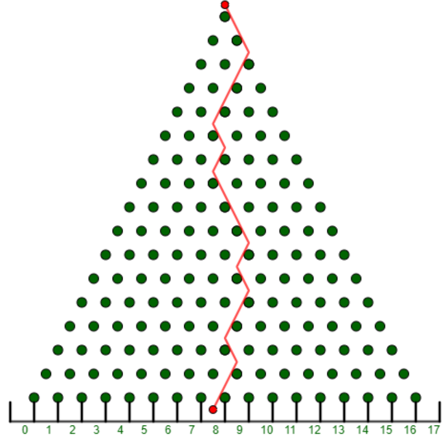
La planche de Galton

Présentation

Dans le programme (spécialité Terminale)

Contenus	**Capacités attendues**
Schéma de Bernoulli : répétition de n épreuves de Bernoulli indépendantes. Loi binomiale B(n, p) : loi du nombre de succès. Expression à l'aide des coefficients binomiaux.	Modéliser une situation par une succession d'épreuves indépendantes, ou une succession de deux ou trois épreuves quelconques. Représenter la situation par un arbre. Calculer une probabilité en utilisant l'indépendance, des probabilités conditionnelles, la formule des probabilités totales. Modéliser une situation par un schéma de Bernoulli, par une loi binomiale. Utilisation de boucle non bornée, utilisation des listes.

Situation déclenchante



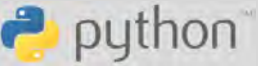
Une bille roule à la surface d'une planche inclinée sur laquelle sont disposés des clous en quinconce. La bille passe aléatoirement d'un côté ou de l'autre des clous, et on note à l'arrivée la position de la bille à la sortie de la planche. Cette planche a été inventée par Sir Francis Galton.

Buts à atteindre

1. Écrire une fonction Python permettant de simuler la chute de plusieurs billes. Cette fonction prendra en paramètres le nombre de lignes de clous de la planche, le nombre de billes et donnera en sortie la liste des effectifs des billes associés aux différents numéros de sortie.
2. Représenter graphiquement les résultats obtenus par la simulation précédente.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



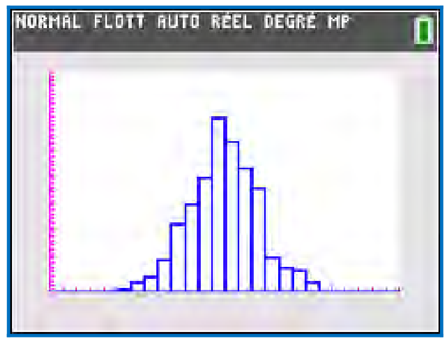
Fiche méthode

Proposition de résolution

On **crée trois fonctions dans ce script** :

1. Une fonction `chute` qui prend comme argument un entier naturel (qui représente le nombre de lignes de clous) et qui renvoie la position finale d'une bille après sa chute.
2. Une fonction `planche` qui prend comme argument deux entiers naturels (qui représentent respectivement le nombre de lignes de clous de la planche ainsi que le nombre de billes lâchées) et qui renvoie la liste des effectifs associés à chaque numéro de sortie.
3. **TI-83** Une fonction `aff3` qui prend comme argument une liste et qui affiche le diagramme en bâtons associé à cette liste.

```
PYTHON SHELL
>>> # Shell Reinitialized
>>> # L'exécution de GALTON
>>> from GALTON import *
>>> chute(10)
6
>>> planche(20,150)
[0, 0, 0, 0, 0, 2, 2, 11, 21, 19
, 33, 19, 20, 10, 9, 2, 2, 0, 0,
 0, 0]
```



Étapes de résolution

L'instruction `from random import *` permettra d'importer une bibliothèque pour utiliser la fonction `randint(0,1)` qui renvoie un nombre au hasard valant 0 ou 1 (de manière équiprobable)[19]. **La fonction `chute` simule en fait une loi binomiale de paramètres *n* et 0,5.**

```
EDITEUR : GALTON
LIGNE DU SCRIPT 0000
import ti_plotlib as plt
from random import *

def chute(n):
  t=0
  for k in range(n):
    t=t+randint(0,1)
  return t
```

La boucle permet de compter le nombre total de choix à droite lors des *n* rencontres entre la bille et les clous.

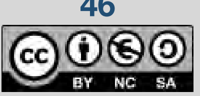
Quelques remarques :

1. L'instruction `L=[0]*k` permet de créer une liste `L` de longueur `k` et d'initialiser chaque élément de la liste à 0.
2. La boucle permet de simuler le lâcher de plusieurs billes et de stocker les différents résultats de sortie dans la liste `resultats`.
3. La fonction `aff3` a pour objectif de préparer la représentation graphique de la liste `l`.
4. L'instruction `from ti_system import *` permettra d'utiliser les fonctions associées à cette bibliothèque, et notamment l'instruction `store_list("1",x)` qui permet d'exporter la liste `x` dans le menu [listes] de la calculatrice sous le nom `L1`.

```
EDITEUR : GALTON
LIGNE DU SCRIPT 0018
def planche(etages,lances):
  resultats=[0]* (etages+1)
  for bille in range(lances):
    c=chute(etages)
    resultats[c]=resultats[c]+1
  return resultats_
```

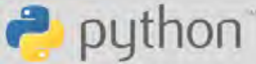
```
EDITEUR : GALTON
LIGNE DU SCRIPT 0060

from ti_system import *
def aff3(l):
  x=[]
  for i in range(len(l)):
    x.append(i)
  store_list("1",x)
  store_list("2",l)
```

19 L'appel `randint(0,1)` revient à simuler un jeu de « pile ou face ».

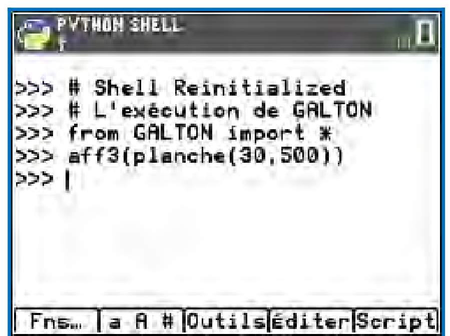


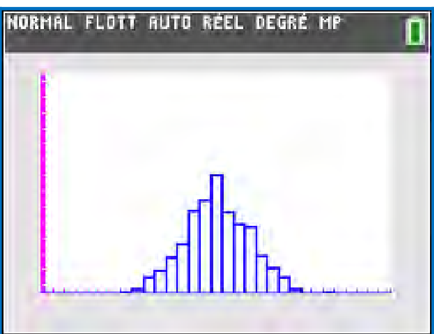
Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



On peut donc exécuter la fonction `aff3` avec `planche(30,500)` en paramètre. Une fois l'instruction tapée, il faut quitter l'application Python et aller dans le menu [graph stats] (touche [2nde] et [f(x)]), régler les paramètres comme ci-dessous puis enfin régler la fenêtre d'affichage (touche [fenêtre]).





On pourrait aussi faire une comparaison de nos résultats avec le modèle théorique. Voir la fiche « triangle de Pascal » pour déterminer un coefficient binomial à l'aide de la fonction `binome2`.

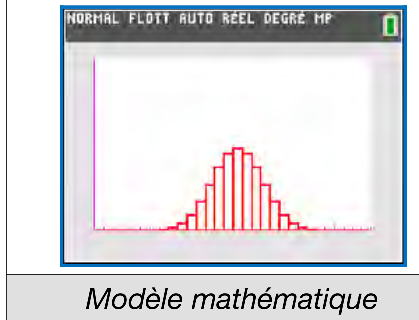
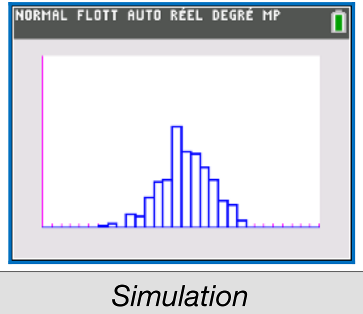
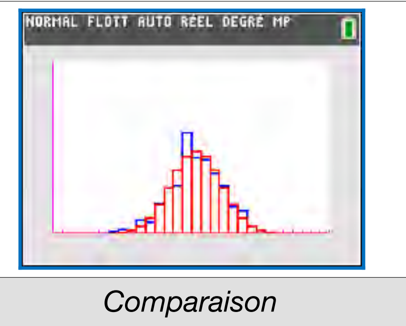
```
ÉDITEUR : GALTON3
LIGNE DU SCRIPT 0031
from ti_system import *
def aff4(etage,lance):
  x=[]
  y=planche(etage,lance)
  z=[]
  for i in range(etage+1):
    x.append(i)
    y[i]=y[i]/(lance)
    z.append(binome2(etage,i)*0.5**i*0.5**(etage-i))
  store_list("1",x)
  store_list("2",y)
  store_list("3",z)
Fns… a A # Outils Exéc Script
```

On lance une simulation.

On calcule les pourcentages associés à cette simulation

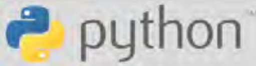
On détermine le modèle à l'aide de la loi binomiale.

On stocke les résultats pour les exporter dans menu graphstats et ainsi pouvoir les représenter.

		
Modèle mathématique	*Simulation*	*Comparaison*

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

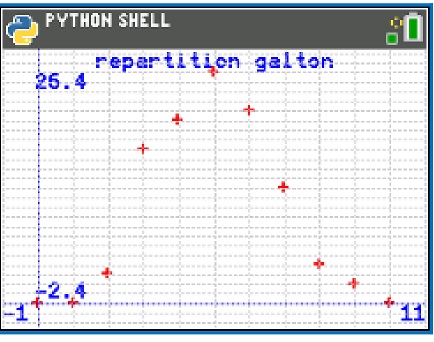


Pour aller plus loin

Approfondissement possible

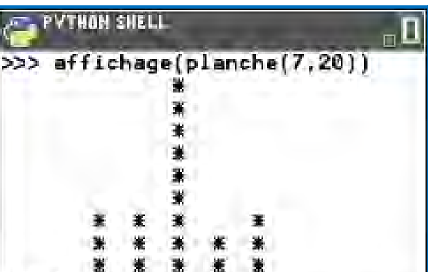
Une deuxième manière de représenter les résultats est d'utiliser la bibliothèque `plotlib`. On peut représenter graphiquement les résultats à l'aide d'un nuage de points. Pour cela, il faut créer une fonction `aff2` qui prend en paramètre une liste `l` et qui va régler les paramètres d'affichage.

```
def aff2(l):
  x=[]
  for i in range(len(l)):
    x.append(i)
  plt.cls()
  plt.auto_window(x,l)
  plt.color(255,0,0)
  plt.scatter(x,l,"+")
  plt.color(0,0,255)
  plt.axes("on")
  plt.color(0,0,255)
  plt.title("repartition galton"
    )
  plt.color(255,255,255)
  plt.grid(1,1,"dot")
  plt.show_plot()
```



Prolongement possible

Imaginer une fonction prenant en entrée une liste et représentant les billes dans la répartition finale de la planche de Galton.



Codes Python proposés

L'affichage de la calculatrice étant réduit par rapport à celui d'un ordinateur, on exécutera ce code avec de petites valeurs pour que cela reste lisible. On pourra être plus ambitieux sur un écran plus grand.

ÉDITEUR : GALTONF
LIGNE DU SCRIPT 0025

```
def affichage ( liste ):
  M = max(liste)
  for i in range( max(liste) ):
    ligne = ''
    for e in liste:
      if e == M:
        ligne += ' * '
        liste[liste.index(e)]-=1
      else:
        ligne =ligne + ' ' * 3
    print(ligne)
    M =M- 1
```

On repère le maximum.

On va construire le graphique ligne par ligne.

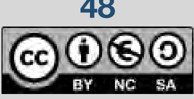
À chaque passage de boucle on initialise la ligne avec l'ensemble vide.

En balayant la liste on va construire les colonnes sur chaque ligne, si on rencontre le maximum de la liste, on met une étoile sur la ligne puis on diminue de 1 ce maximum...

... sinon on met 3 espaces sur la ligne.

On affiche la nouvelle ligne construite à chaque passage de boucle.

Le principe est de repérer la colonne ayant le plus de billes puis de représenter la première ligne à l'aide d'espace et du symbole `*`. Ensuite on passe à la ligne suivante en diminuant de 1 le maximum de la liste.



Indication : étant donné une liste `L`, l'appel `L.index(x)` renvoie l'index du premier élément de la liste `L` ayant pour valeur `x`.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

Un problème de surréservation

Présentation et objectifs

Dans le programme (spécialité Terminale)

Contenus	**Capacités attendues**	**Algorithmes**
Épreuve de Bernoulli, loi de Bernoulli. Schéma de Bernoulli : répétition de n épreuves de Bernoulli indépendantes. Loi binomiale $B(n, p)$: loi du nombre de succès. Expression à l'aide des coefficients binomiaux.	Modéliser une situation par un schéma de Bernoulli, par une loi binomiale. Utiliser l'expression de la loi binomiale pour résoudre un problème de seuil, de comparaison, d'optimisation relatif à des probabilités de nombre de succès. Dans le cadre d'une résolution de problème modélisé par une variable binomiale X, calculer numériquement une probabilité du type $P(X=k)$, $P(X \leq k)$, $P(-k \leq X \leq k)$, en s'aidant au besoin d'un algorithme ; chercher un intervalle I pour lequel la probabilité $P(X \in I)$ est inférieure à une valeur donnée α, ou supérieure à $1-\alpha$.	Problème de la surréservation. Étant donné une variable aléatoire binomiale X et un réel strictement positif α, détermination du plus petit entier k tel que $P(X>k) \leq \alpha$.

Positionnement du problème

Dans certains domaines et notamment dans le transport aérien, pour un vol donné, un certain nombre de passagers ayant procédé à une réservation ne se présentent pas à l'embarquement (maladie, retard, etc.).

Le taux de personnes qui ne se présentent pas semble se situer en moyenne autour de 5 %. Ce taux était plus important il y a quelques années car on pouvait annuler sa réservation sans pénalités.

Ce taux est confidentiel car ce taux de non-présentation est intimement lié à un autre, celui de la surréservation.

Pour chacun de leurs vols, afin d'améliorer le taux de remplissage de l'avion et donc la rentabilité du vol, les compagnies proposent un nombre de réservations supérieur au nombre de places de l'avion : c'est la surréservation, ou surbooking.

Dans cette pratique le risque est évident : que certains voyageurs ne puissent pas embarquer en raison d'un manque de places dans l'avion.

Les victimes de ce fonctionnement seront dédommagées financièrement.

Comment est déterminé le nombre de places en surréservation sachant que cela coûte de l'argent à la compagnie ? (dédommagement, prise en charge des frais d'hôtel, de restaurant, etc.)

À faire…

Étudions un exemple :

Un avion de la compagnie TI-Airline en provenance de Maths-en-ville et à destination de Lumière-ville

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

possède 300 places à bord. Les années précédentes, la compagnie a observé un taux de non présentation à l'embarquement de 4 % sur ce vol pour les personnes ayant effectué une réservation. Toutes les personnes ayant procédé à une réservation ont la même probabilité de ne pas se présenter

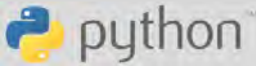
à l'enregistrement et on suppose que leurs comportements sont indépendants les uns des autres. On note α le pourcentage de risque accepté par la compagnie pour que certains voyageurs soient en situation de surbooking. Combien de places doit vendre la compagnie aérienne TI-Airline sur ce vol pour respecter le risque accepté par la compagnie mais aussi remplir au maximum l'avion ?

Buts à atteindre

1. Écrire une fonction qui prend en paramètres deux entiers naturels n et k et qui renvoie le coefficient binomial associé.
2. Écrire une fonction qui prend en paramètres trois nombres (deux entiers naturels n et k et un nombre décimal p compris entre 0 et 1) et qui renvoie $P(X=k)$, où X suit une loi binomiale de paramètres n et p.
3. Écrire une fonction qui permet de répondre à l'exemple étudié en supposant que la compagnie aérienne ne souhaite pas manquer de places dans l'avion dans plus de 1 % de ses vols.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

Fiche méthode

Étapes de résolution

▶ Objectif 1 :

Voir la fiche Triangle de Pascal qui présente diverses manières de calculer les coefficients binomiaux. Nous proposons ici une manière (efficace) parmi toutes celles présentées.

EDITEUR : SURBOOKI
LIGNE DU SCRIPT 0001

```
def coeff(n,k):
  if k>n:
    return 0
  e=1
  if 2*k>n:
    k=n-k
  for i in range(1,k+1):
    e=(e*(n-k+i))//i
  return e
```

Si $k>n$ on considère que le coefficient est nul.

On peut utiliser un argument de symétrie sur les coefficients pour gagner en efficacité.

▶ Objectif 2 :

On crée un fonction `binomiale` qui prend en paramètres trois nombres (deux entiers naturels n et k et p un nombre décimal compris entre 0 et 1) et qui renvoie $P(X=k)$, où X suit une loi binomiale de paramètres n et p. Le cas $k>n$ renvoie bien une probabilité nulle car `coeff(n,k)` renvoie 0 dans ce cas.

On calcule ici : $\binom{n}{k} p^k (1-p)^{n-k}$.

EDITEUR : SURBOOKI
LIGNE DU SCRIPT 0021

```
def binomiale(n,p,k):
  prob=p**k*(1-p)**(n-k)
  return coeff(n,k)*prob
```

Il s'agit ici de calculer une somme $\sum_{i=0}^{k} P(X=i)$ à l'aide d'une boucle.

▶ Objectif 3 :

Dans un premier temps on va créer la fonction de répartition de la fonction binomiale appelée `binomFrep` qui prend comme arguments deux entiers naturels k et n et qui renvoie $P(X \le k)$ où X suit une loi binomiale de paramètres n et k.

EDITEUR : SURBOOKI
LIGNE DU SCRIPT 0023

```
def binomFrep(n,p,k):
  S=0
  for i in range(k+1):
    S=binomiale(n,p,i)+S
  return S
```

Dans un second temps, nous créons une fonction `surbooking` qui renvoie le nombre de places répondant au problème posé et prend en paramètres :

- `alpha`, un nombre décimal qui représente le pourcentage de risque accepté en effectuant du surbooking (ici 0,01) ;
- `n`, entier représentant le nombre de places dans l'avion (ici 300) ;
- `p`, nombre décimal représentant la probabilité qu'une personne ayant effectué une réservation se présente (ici 0,96).

EDITEUR : SURBOOKI
LIGNE DU SCRIPT 0040

```
def surbooking(alpha,n,p):
  proba = 0
  i=n
  while proba<alpha:
    proba= 1-binomFrep(i,p,n)
    i = i+1
  return i-2
```

On cherche ici à déterminer le plus grand entier naturel i tel que si $X = Bin(i,p)$, $P(X \le 300) \ge 1-\alpha$, ce qui est équivalent à : $1-P(X \le 300) \le \alpha$. On commencera la recherche avec $i=n$ (au minimum lorsqu'on vend autant de billets que de places).

Soit X une variable suivant une loi binomiale de paramètres n et p=0,96. Le problème revient donc à chercher un entier n maximal tel que $P(X>300) \le 0{,}01$. On crée donc une boucle `while` qui fait appel à la fonction `binomFrep` pour répondre au problème posé.

PYTHON SHELL

```
>>> surbooking(0.01,300,0.96)
305
```

© Texas Instruments 2021 / Photocopie autorisée

Pour aller plus loin

Prolongement possible

La compagnie, en faisant du surbooking, cherche à optimiser son chiffre d'affaires. Cette optimisation est complexe au vu de tous les paramètres mis en jeu, nous allons étudier ce chiffre d'affaires sur un exemple simplifié.

Maintenant, sur ce même vol, la compagnie TI-Airline espère vendre le billet aller 500 €. Elle envisage de verser un dédommagement de 1000 € (remboursement de 500 € + prime de 500 €) aux personnes n'ayant pu embarquer.

Travail de groupe

La classe se répartit en plusieurs petits groupes, qui essayent de répondre à la question suivante :

Quel nombre n ($n \geq 300$) de billets doit vendre la compagnie afin d'optimiser son chiffre d'affaires ?

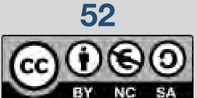
Chaque groupe devra présenter sa réponse à la classe en justifiant.

Éléments de réponse :

On pose n le nombre de billets vendus, X_n le nombre de personnes qui se présentent, et Y_n le nombre de personnes en surbooking ($Y_n = \max(X_n - 300, 0)$). On note G_n le chiffre d'affaires en euro : $G_n = 500n - 1000Y_n$.

On peut répondre à l'aide d'un tableur, simulant les diverses variables aléatoires dans des colonnes et répétant le tout sur autant de lignes successives que nécessaire pour « stabiliser » les fréquences et pouvoir ainsi observer les résultats. Le tableau reproduit ci-dessous montre que l'optimum semble se réaliser avec une vente de 310 ou 311 places vendues.

n	X(n)	Y(n)	G(n)	Moyenne nb places manquantes	Moyenne du gain pour n places	n	Moyenne du gain sur 10 000 simulations
315	304	4	144000	2,95	145316	300	144012
	303	3	145500			301	144475
	300	0	150000			302	144943
	305	5	142500			303	145473
	305	5	142500			304	145898
	308	8	138000			305	146386
	306	6	141000			306	146852
	298	0	149000			307	147255
	298	0	149000			308	147574
	302	2	147000			309	147813
	304	4	144000			310	147838
	301	1	148500			311	147807
	303	3	145500			312	147487
	304	4	144000			313	146894
	302	2	147000			314	146194
	302	2	147000			315	145305
	299	0	149500			316	144218
	301	1	148500			317	143113
	303	3	145500			318	141821



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Une réponse probabiliste est aussi envisageable ; elle consisterait à faire un calcul d'espérance mathématique, et ajuster n de sorte que l'espérance de G_n soit maximale, mais c'est assez compliqué.

Une réponse algorithmique consiste à simuler la situation suffisamment de fois pour que la moyenne des valeurs simulées de G_n se stabilise autour de l'espérance de G_n (loi des grands nombres), avant d'ajuster n pareillement.

Nspire CX Le nombre de calculs requis est assez élevé, nécessitant une bonne puissance de calcul ; c'est une situation où une machine rapide comme la TI-Nspire™ CX II-T permet d'obtenir une réponse dans un délai raisonnable. Le script Python correspondant est montré ci-contre.

Quelques remarques sur la programmation : l'essentiel est de simuler une variable aléatoire suivant une loi binomiale. Pour ce faire, on commence la simulation `bern` d'une variable de Bernoulli, ce qui oblige à employer la fonction `random` du module standard `random`. Pour simuler une variable de Bernoulli de paramètre 0,96 on pourrait aussi bien faire appel à `randint(0,96)/100`. Dès lors, la simulation de la variable binomiale revient à répéter des appels à la fonction `bern` autant de fois que nécessaire.

On peut dès lors faire la simulation du gain sur une répétition de `r` vols avec pour chacun `n` billets vendus ($n>n_0=300$ places), et une probabilité `p`=0,96 pour chacun des passagers de se présenter à l'embarquement. À la suite, on teste le gain simulé avec une série d'hypothèses portant sur `n`, évoluant entre 300 et 319.

SSurbook.py

```
from random import random
def bern(p):
    if random()<p:
        return 1
    return 0
def binomiale(n,p):
    s=0
    for _ in range(n):
        s = s + bern(p)
    return s

def gain(n,p,n0,p1,p2,r):
#n  = nb de billets vendus
#n0 = nb de places
#p1 = prix de vente des billets
#p2 = dédommagement / surbook
#r  = répétitions
    g = 0 # gain total
    for _ in range(r):
        m = binomiale(n,p) # alea
        g = g + m * p1 # ventes
        if m > n0: # ! surbooking
            g = g - (m - n0) * p2
    return int(g/r) # moyenne

L,p=[],0.96 # initialisation
for k in range(300,320):
    L.append(gain(k,p,
    300,500,1000,5000))
m=max(L)
print("p=",round(p,2),
" optim : ",m," places : ",
L.index(m)+300)
```

Le test réalisé ici sur 5000 vols simulés suggère plutôt de vendre 312 billets. Que cette réponse soit évolutive provient du fait que les simulations ne donnent pas toujours les mêmes valeurs (d'autant que le nombre de répétitions n'est pas énorme).

`max(L)` permet de calculer la valeur maximum dans la liste `L`.

`L.index(m)` renvoie l'index du premier terme de la liste `L` ayant la valeur `m`.

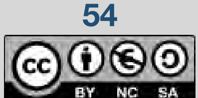
1.1 1.2 Ssurbook

Shell Python

```
>>>#Running SSurbook.py
>>>from SSurbook import *
p= 0.96  optim : 148624  places : 312
```

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

La méthode de Monte Carlo

Présentation et objectifs

Dans le programme (spécialité Terminale)

Probabilités	**Calcul intégral / Exemples d'algorithme**
Espérances. Loi des grands nombres.	Méthode de Monte-Carlo.

Une histoire atomique

La méthode dite de Monte-Carlo est une méthode visant à approcher une solution d'une équation mathématique, voire toute valeur numérique, en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes. Le nom de ces méthodes, qui fait allusion aux jeux de hasard pratiqués à Monte-Carlo, a été inventé en 1947 par Nicholas Metropolis, et publié en 1949 dans un article coécrit avec Stanislaw Ulam lors du développement de l'arme nucléaire[20].

JOURNAL OF THE AMERICAN STATISTICAL ASSOCIATION

Number 247 SEPTEMBER 1949 *Volume 44*

THE MONTE CARLO METHOD

NICHOLAS METROPOLIS AND S. ULAM
Los Alamos Laboratory

We shall present here the motivation and a general description of a method dealing with a class of problems in mathematical physics. The method is, essentially, a statistical approach to the study of differential equations, or more generally, of integro-differential equations that occur in various branches of the natural sciences.

Situation déclenchante

Comme exemple de recherche de valeur numérique, nous allons voir comment trouver des valeurs approchées du nombre π en traitant le problème d'un point de vue probabiliste. La précision du résultat sera directement liée au nombre de répétitions qui seront réalisées. De ce fait, la précision du résultat sera liée à la durée de la simulation.

Le nombre π, considéré comme la surface délimitée par un cercle de rayon 1 unité, vaut approximativement 3,14159 unités. Comment la déterminer par simulation de Monte-Carlo ? Il s'agit de représenter cette valeur comme une proportion qui sera considérée comme l'espérance d'une variable aléatoire.

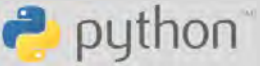
Pour ce faire, nous traçons un disque de rayon 1 unité et l'englobons dans un carré de côté 2 unités, donc d'aire 4 unités d'aire, tangent au cercle en 4 points, comme sur la figure ci-contre.

Nous considérons alors l'aire du quart de disque grisé divisée par l'aire du carré ABDG, valant $\pi/4$ unités d'aire. Si nous disposons une grille très fine sur ce carré, formée par M lignes horizontales et M lignes verticales (M étant un très grand entier), nous avons M^2 petits carrés qui se répartissent en deux sortes : les carrés « rouges » dont le coin en bas à gauche se trouve à l'intérieur du quart de

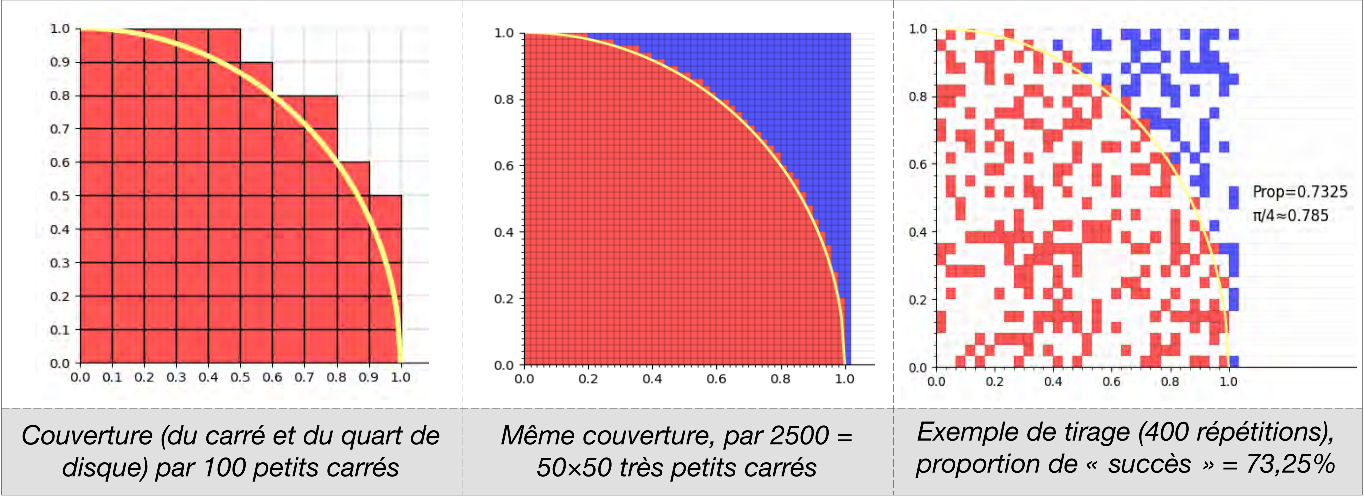
20 Plus précisément, il s'agissait de modéliser la trajectoire moyenne des neutrons dans un réacteur ou pendant une explosion nucléaire.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



disque, et les carrés « bleus » qui ne sont pas de ce type. On admet aisément que le total des aires des carrés rouges est proche (et supérieur) à l'aire du quart de disque grisé (et d'autant plus proche que M est grand), de sorte que la proportion des carrés rouges parmi l'ensemble des petits carrés (notons-la p) est une approximation de π/4 (d'autant meilleure que M est grand).



Couverture (du carré et du quart de disque) par 100 petits carrés	*Même couverture, par 2500 = 50×50 très petits carrés*	*Exemple de tirage (400 répétitions), proportion de « succès » = 73,25%*

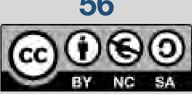
Or, approcher des proportions au moyen d'expériences aléatoires est exactement l'objectif de la Statistique. Nous procédons donc à un tirage aléatoire de petits carrés au sein du carré ABDG (avec une loi uniforme sur les M^2 petits carrés) ; soit la variable aléatoire Z valant 1 si le petit carré tiré est rouge et 0 sinon. La probabilité d'avoir Z=1 est exactement p, et c'est aussi l'espérance de Z. Pour estimer p, on considère un grand nombre n de tirages indépendants, suivant la même loi que Z, de sorte que quand n tend vers l'infini, la proportion de carrés rouges obtenus converge vers l'espérance de Z, soit p. Le nombre π peut donc être approché par 4 fois la proportion de carrés rouges.

Pour tirer un petit carré au hasard, il suffit de tirer l'abscisse et l'ordonnée de son coin en bas à gauche ; pour cela, on trouve dans le module `random` une fonction `random` qui réalise ce dont nous avons besoin.

En effet, cette fonction tire une valeur au hasard parmi $M=2^{53}$ valeurs régulièrement réparties entre 0 et 1 : c'est bien un très grand nombre (valant environ 10^{16}).

Objectifs

1. Écrire un script permettant de générer une approximation du nombre $\pi/4$ à l'aide de l'approche probabiliste décrite ci-dessus.
2. Examiner si les approximations semblent converger lorsque n tend vers l'infini : tester l'écart entre $\pi/4$ et l'approximation trouvée pour un nombre de tirages variant de 1000 en 1000, ou croissant plus rapidement encore.
3. Modifier le script pour approcher l'aire comprise entre la courbe représentative de la fonction carré, l'axe des abscisses, les droites d'équation $x=0$ et $x=1$.



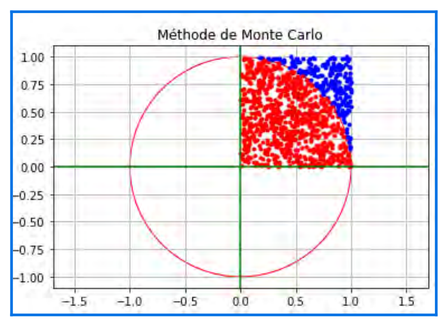
Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Fiche méthode

Objectif 1 : proposition de résolution

Stratégie



On crée deux fonctions dans ce script :

- Une « fonction de décision » `d` qui prend comme arguments deux réels `x` et `y` et qui renvoie le nombre réel correspondant au carré de la distance OM entre O(0,0) et M(x,y) diminué de 1 (l'usage de la racine carrée n'est ici pas utile et consomme du temps de calcul) ; on teste si `d(a,b)` est négatif pour savoir si on est dans la zone souhaitée.
- Une fonction `mc` qui prend comme argument un entier naturel *n* et renvoie un nombre réel correspondant à la fréquence (ou proportion observée) des carrés générés aléatoirement se situant dans le quart de cercle rouge sur le dessin ci-contre.

```
EDITEUR : MONTECAR
LIGNE DU SCRIPT 0012
from random import random
def d(x,y):
  return x*x+y*y-1
def mc(n):
  c=0
  for i in range(n):
    a=random()
    b=random()
    if d(a,b)<=0:
      c=c+1
  return c/n
Fns… | a A # | Outils | Exéc | Script
```

Étapes de résolution

On commence par assurer la nécessaire importation de la bibliothèque `random`, avant de définir les fonctions `d` et `mc`.

On peut afficher la valeur approchée de $\pi/4$ pour comparer avec la simulation et ainsi observer la précision du résultat de la simulation.

On notera que l'approximation n'est pas bonne pour des petites valeurs de n (il faut prendre au moins $n=10000$ pour une approximation au centième près). Par ailleurs, le temps de calcul peut être très important, déjà 35 secondes pour `mc(80000)` sur une TI-83 Premium CE Edition Python, moins de 3 secondes sur une TI-Nspire™ CX II-T. C'est la caractéristique des méthodes de Monte Carlo : pour atteindre une précision correcte il faut un nombre d'itérations très élevé et accepter des temps de calcul énormes (ou disposer de matériel très performant)[21].

```
from math import *
from random import *
from ti_system import *
def d(x,y):
  return x*x+y*y-1
def mc(n):
  c=0
  for i in range(n):
    a=random()
    b=random()
    if d(a,b)<=0:
      c=c+1
  return c/n
def arithm():
  X1=[k*1000 for k in range(2,80)]
  store_list("X1",X1)
  L1=[mc(n) for n in X1]
  store_list("L1",L1)
```

Objectif 2

Nous pouvons créer un script Python qui va appeler de manière répétée la fonction `mc` et tester l'écart avec la valeur cible. Pour se rendre compte de la convergence le mieux est de prendre des valeurs de *n* assez rapidement croissantes, par exemple de 1000 en 1000 ou suivant une suite quadratique (basée sur les carrés des

21 Le matériel utilisé par Metropolis et Ulam (ENIAC) était électromécanique, car les circuits intégrés n'existaient pas encore, pas plus que les transistors ; d'où des temps de calcul énormes (des heures ou des jours), imposant d'éviter absolument les erreurs logicielles (ou « bugs »). L'usage du mot « bug » provenait des insectes qui entraient dans les relais et provoquaient des faux contacts …

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

entiers).

Sur une calculatrice TI-Nspire™ CX II-T, on peut aisément explorer les valeurs de `mc(n)` pour n allant jusqu'à 80000 (ou plus).

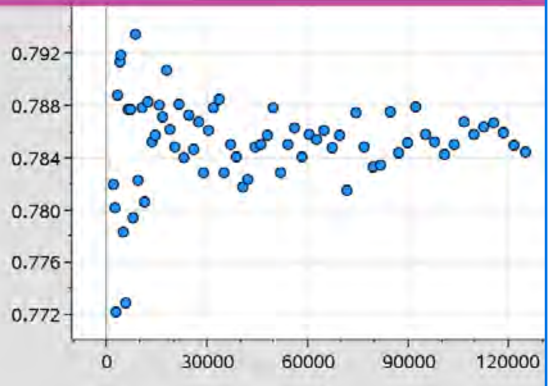
Une représentation graphique permet de dégager une idée intuitive du phénomène. Le plus commode pour cela est de créer une liste et de la traiter dans l'environnement « natif » de la calculatrice grâce à la fonction `store_list` prévue à cet effet dans la bibliothèque `ti_system`.

La représentation graphique est alors facile à faire à l'aide des outils fournis par la machine.

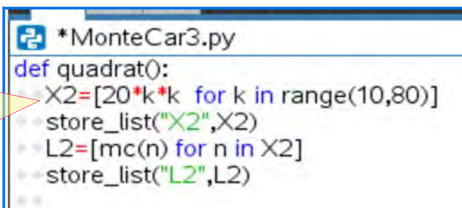
Nspire CX L'instruction `store_list("L1",L)` crée une liste nommée L1 dans le système, liste qui peut s'afficher dans l'environnement « données et statistiques » ; on crée similairement une liste X pour les abscisses.

On constate que les approximations ne convergent que très lentement, et assez irrégulièrement (côté aléatoire du processus).

Si on veut explorer plus loin, il est intéressant de donner à n des valeurs croissant plus rapidement, en suite quadratique plutôt qu'arithmétique (ici, $n=20k^2$, k entier). Le temps de calcul peut alors devenir important ; exemple ci-dessous.



On emploie ici une « liste en compréhension » (voir Appendice 1 pour plus de détails).

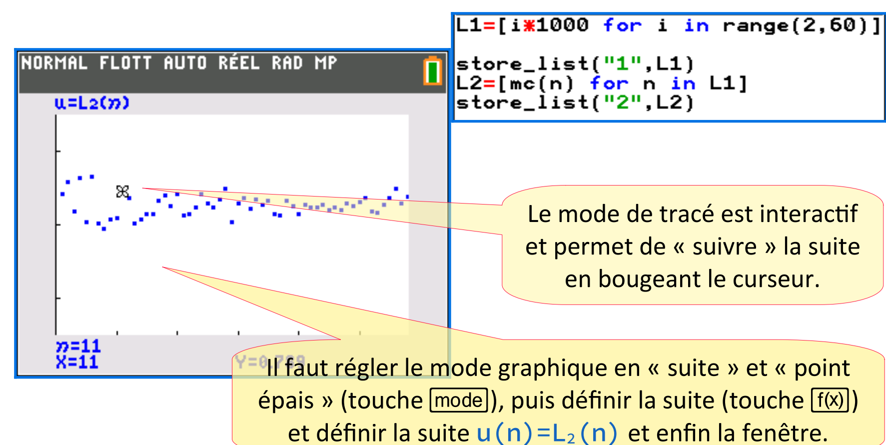


```
*MonteCar3.py
def quadrat():
    X2=[20*k*k  for k in range(10,80)]
    store_list("X2",X2)
    L2=[mc(n) for n in X2]
    store_list("L2",L2)
```

TI-83 On procède de manière semblable. Ici, l'instruction `store_list("1",L)` permet de copier le contenu de la liste L dans la variable système L_1.

La représentation graphique est alors facile à faire à l'aide des outils fournis par la machine avant de tracer.

```
L1=[i*1000 for i in range(2,60)]

store_list("1",L1)
L2=[mc(n) for n in L1]
store_list("2",L2)
```



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

Il faut régler le mode graphique en « suite » et « point épais » (touche [mode]), puis définir la suite (touche [f(x)]) et définir la suite $u(n)=L_2(n)$ et enfin la fenêtre.

La lenteur de cette calculatrice ne permet pas d'explorer autant qu'avec une TI-Nspire™ CX II, à moins d'une grande patience !

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Objectif 3

On va modifier le script précédent pour approcher l'aire comprise entre la courbe représentative de la fonction carré, l'axe des abscisses, les droites d'équation $x=0$ et $x=1$, c'est-à-dire l'intégrale $\int_0^1 x^2 \, \mathrm{d}x = \frac{1}{3}$.

Il suffit en fait de modifier la « fonction de décision » d en y codant l'équation de la parabole ; le reste du programme est inchangé.

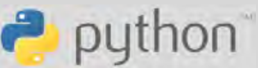
```
EDITEUR : MONTECA2
LIGNE DU SCRIPT 0012
from random import random
def d(x,y):
  return y-x*x
def mc(n):
  c=0
  for i in range(n):
    a=random()
    b=random()
    if d(a,b)<=0:
      c=c+1
  return c/n
Fns… | a A # | Outils | Exéc | Script
```

```
PYTHON SHELL
>>> from MONTECA2 import *
>>> mc(80000)
0.3360375
```

Comme annoncé, la précision est mauvaise à moins de faire un très grand nombre de calculs …

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



Pour aller plus loin

Approfondissement (TI 83 ou Nspire CX II)

TI-83 **Nspire CX-II** En approfondissement, on peut travailler sur la représentation graphique de la simulation. Il faudra donc importer la bibliothèque `ti_plotlib` de manière à pouvoir effectuer la représentation graphique. On rajoutera donc au code précédent uniquement des lignes qui vont gérer les paramètres de la représentation graphique (voir l'Appendice 2) :

- en préliminaire le cadrage de la fenêtre,
- puis à chaque itération le choix de la couleur selon le positionnement,
- puis l'affichage d'un pixel correspondant au tirage au hasard,
- et à la fin l'affichage du graphique.

PYTHON SHELL

0.781

8192 tirages de points au hasard

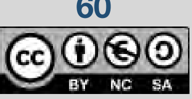
EDITEUR : MONTECA3
LIGNE DU SCRIPT 0002

```
from random import random
import ti_plotlib as plt
def d(x,y):
  return x*x+y*y-1
def mc(n):
  plt.cls()
  plt.window(0,1.6,-.15,1)
  plt.axes("off")
  plt.grid(.2,.2,"dash")
  c=0
  for i in range(n):
    a=random()
    b=random()
    if d(a,b)<=0:
      c=c+1
      plt.color(255,0,0)
    else:
      plt.color(0,0,255)
    plt.plot(a,b,".")
  plt.color(0,0,0)
  s=str(round(c/n,3))
  plt.text_at(12,s,"right")
  plt.show_plot()
  return c/n
```

Remarque : dans l'instruction :

`plt.text_at(12,str(c/n),"right")`

on met l'instruction `str` devant `c/n` de manière à le convertir en chaîne de caractères car c'est le type attendu dans l'instruction `plt.text_at`.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

L. DIDIER & R. CABANE

La combinatoire des parties

Présentation et objectifs

Dans le programme (spécialité Terminale)

Contenus Nombre des parties d'un ensemble à n éléments. Lien avec les n-uplets de {0,1}, les mots de longueur n sur un alphabet à deux éléments, les chemins dans un arbre, les issues dans une succession de n épreuves de Bernoulli.	Combinaisons de k éléments d'un ensemble à n éléments : parties à k éléments de l'ensemble. Représentation en termes de mots ou de chemins. **Exemple d'algorithme** Génération des parties à 2, 3 éléments d'un ensemble fini.

Situation déclenchante

Lundi

– *(le chef de service) Alex, je voudrais former une équipe avec quelques membres du service qui s'entendent vraiment bien. Peux-tu me faire la liste de toutes les équipes possibles ? Peu importe le nombre d'équipiers.*
– *Votre liste va être longue, non ?*
– *Peu importe…*
– *Bien chef.*

Mardi

– *Alex, tu avais raison, il y en a trop….*
– *Chef, on pourrait décider que l'équipe est formée de quatre personnes*
– *Bonne idée, fais-moi ça au plus vite.*
– *Oui chef.*
– *Et fais-moi une proposition d'équipe à partir de ta nouvelle liste, n'importe laquelle.*

Il s'agit donc ici de parties (ou sous-ensembles) d'un ensemble à n éléments, par exemple $E=\{0,1,\ldots,n-1\}$, non pour les dénombrer (voir la fiche sur le triangle de Pascal) mais pour les <u>énumérer</u> c'est-à-dire en faire une liste soit complète soit limitée aux parties à k éléments (combinaisons).

- La représentation informatique d'un ensemble peut se faire de diverses manières, la plus simple étant fournie par les listes, en convenant de ne pas prêter attention à l'ordre des éléments. Pour plus de facilité dans la saisie comme l'affichage, les chaînes de caractères sont commodes d'autant qu'elles peuvent être parfois traitées avec la même syntaxe que les listes[22] .
- Manipulations de listes : voir l'appendice 1 pour plus de détails sur les syntaxes appropriées.
- Les sous-ensembles seront donc pris comme des sous-listes ou des sous-chaînes (selon le contexte), et la collection des sous-ensembles sera formée comme une liste de listes ou une liste de chaînes. Par sous-liste d'une liste `L` on entend une liste formée de certains éléments distincts de `L` (dans le même ordre que dans `L`), et par sous-chaîne d'une chaîne de caractères `s` on entend une chaîne formée de certains caractères de `s`, dans le même ordre. Le respect de l'ordre nous évitera d'avoir des « doublons » comme `"ab"` vis-à-vis de `"ba"`.
- Enfin, au lieu de produire une liste des parties, nous pouvons aussi chercher à en tirer une au hasard, en veillant à ce que ce tirage soit « équiprobable ».

22 Attention : au contraire des listes, les chaînes ne sont pas « mutables » en Python, au sens où on ne peut modifier leurs éléments.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

La combinatoire des parties

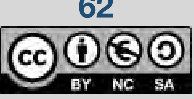
L. DIDIER & R. CABANE

Un outil essentiel : la récursivité

Pour ces diverses tâches, il est essentiel de pouvoir se servir du principe de récursivité, admis par le langage Python : une fonction peut s'appeler elle-même, sous réserve de ne pas aboutir à une « boucle infinie ». Voir l'appendice 1 pour plus de détails.

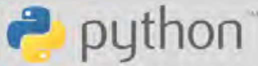
Objectifs

1. Écrire une fonction Python récursive opérant sur une chaîne de caractères distincts s et renvoyant une liste contenant toutes les sous-chaînes de s. L'idée pourrait être de combiner toutes les sous-chaînes contenant le premier caractère de s avec toutes les sous-chaînes ne le contenant pas.
2. Écrire une fonction Python récursive opérant sur une chaîne de caractères distincts s et renvoyant une liste contenant toutes les sous-chaînes de longueur k de s.
3. Écrire une fonction Python récursive opérant sur une liste de caractères distincts L et renvoyant une sous-liste de longueur k de L choisie au hasard.



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



L. DIDIER & R. CABANE

Fiche méthode

Objectif 1 : la liste des parties

La logique à suivre est de considérer le premier élément `s[0]` de l'ensemble `s` (ici pris comme une chaîne de caractères) et de distinguer les parties contenant cet élément de celles qui ne le contiennent pas.

```
ÉDITEUR : PARTIES
LIGNE DU SCRIPT 0010
def parties(s):
  if s=='':
    return ['']
  if len(s)==1: # cas simple
    return ['',s] # sortie
  t=s[1:] # s sauf 1er élt
  L=parties(t) # appel récursif
  L2=[s[0]+u for u in L]+[u for
    u in L]
  return L2
```

Une fois que ce choix est effectué, il n'y a plus qu'à « recommencer de même » sur les éléments restants, ce qui se note `s[1:]` ; nous avons ici une fonction récursive !

On utilise ici des « listes en compréhension », syntaxe concise permettant de créer le résultat à partir des éléments d'une liste.

Pour que cela s'achève, il faut traiter préalablement les cas où l'ensemble possède 0 ou 1 élément (l'ensemble à 1 élément admet deux parties, la partie vide et la partie pleine). Le principe de récursivité montre, au passage, que le nombre de parties double quand on ajoute un élément à l'ensemble ; en d'autres termes, le nombre de parties d'un ensemble à n éléments est 2^n .

```
PYTHON SHELL
>>> parties("")
['']
>>> parties("a")
['', 'a']
>>> parties("ab")
['a', 'ab', '', 'b']
>>> parties("abc")
['ab', 'abc', 'a', 'ac', 'b', 'b
c', '', 'c']
```

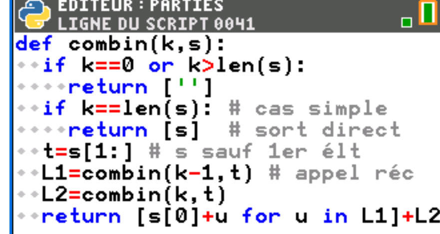
Objectif 2 : la liste des combinaisons

Commençons par un cas très simple, suggéré par le programme : lister les sous-ensembles de taille 2 d'un ensemble donné. Cela peut se faire de manière très simple en Python grâce au principe des « listes en compréhension », voir ci-contre.

```
PYTHON SHELL
>>> L=[1,2,3,4]
>>> [[x,y] for x in L for y in L
 if x<y]
[[1, 2], [1, 3], [1, 4], [2, 3],
 [2, 4], [3, 4]]
```

On pourrait procéder de même pour les parties à trois éléments, mais la commande commence à être pénible à écrire, aussi convient-il de chercher une réponse plus générale. Si on s'autorise à écrire une fonction doublement récursive, on peut suivre un peu la même idée que pour le listage de toutes les parties : celles qui contiennent le premier élément et celles qui ne le contiennent pas.

Il faut prendre garde à l'initialisation et bien prévoir les deux cas extrêmes : lorsqu'on ne cherche que des parties vides (en ce cas, on renvoie le vide sous la forme `['']`), et lorsqu'on ne cherche que la partie pleine (en ce cas, on renvoie `s` pour seule réponse).



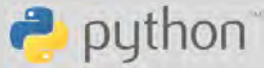
Détail à surveiller : le symbole `+` désigne ici la concaténation (mise bout à bout) dans deux contextes <u>différents</u> :

- l'expression `s[0]+u` est une <u>chaîne</u> de caractères, commençant par `s[0]` et continuant avec `u`,
- et l'opération `+` qui suit concatène deux <u>listes</u>.

Nous reviendrons plus loin sur l'inefficacité de ce codage (défi « modérer la récursivité »).

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



Objectif 3 : une combinaison au hasard

```
ÉDITEUR : PARTIES
LIGNE DU SCRIPT 0067
def combalea(k,L):
  if k==0:
    return []
  ch=L.pop(randint(0,len(L)-1))
  K=combalea(k-1,L)
  return sorted([ch]+K)
```

Pour tirer une partie de `k` éléments d'un ensemble donné (ici décrit par une liste `L`), nous pouvons aussi procéder de manière récursive. S'il s'agit de tirer une partie vide (`k=0`), c'est immédiatement réglé.

Si `L` est une liste comme `[2,3,5]`, `L.pop(1)` fait deux choses :
– retirer le deuxième élément de `L`
– renvoyer cet élément (ici, 3).

Si `L` est une liste, l'appel `sorted(L)` renvoie une liste triée ayant les mêmes éléments que `L`.

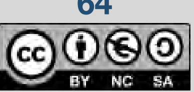
Sinon, on tire au hasard un élément de `L`. Ce choix est effectué par l'instruction `randint(0,len(L)-1)` ; puis on le retire de `L` (grâce à l'appel `L.pop`).

Il ne reste plus qu'à tirer `k-1` éléments de la liste restante, ce que est réalisé par l'appel récursif `combalea(k-1,L)`.

Pour fournir la réponse, on met l'élément tiré au hasard en tête (ce que fait l'instruction `[ch]+K`) puis on trie le tout (appel de la fonction `sorted`) pour éviter de présenter comme différentes des réponses ne se distinguant que par l'ordre.

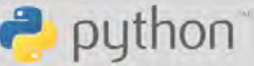
Une fois que l'idée est formulée, il n'y a aucune difficulté à reprendre l'algorithme et à le programmer de manière itérative (non récursive) : on accumule de proche en proche les éléments tirés (au hasard) de `L` dans une liste `C` initialement vide ; il ne reste plus qu'à renvoyer cette liste, triée.

```
ÉDITEUR : PARTIES2
LIGNE DU SCRIPT 0063
def combaleait(k,L):
  C=[]
  for i in range(k):
    r=randint(0,len(L)-1)
    C.append(L.pop(r))
  return sorted(C)
```

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée



Pour aller plus loin

Et la représentation binaire ? (objectif 4)

Il s'agit de ce qui est évoqué au programme par le « lien avec les n-uplets de {0,1} ». L'idée est la suivante. Tout nombre entier m peut se décomposer en une somme de puissances de 2, de manière analogue à la représentation décimale avec les puissances de 10. Les exposants de ces puissances sont appelés les « bits[23] » de m ; ainsi, on a $11=2^3+2+1$. En choisissant certains des n bits de rangs $0,1,\cdots,n-1$, on peut représenter tous les entiers de 0 à 2^n-1 ; c'est ainsi que le choix d'un sous-ensemble de $\{0,1,\ldots,n-1\}$ revient à choisir un entier inférieur ou égal à 2^n-1 et à déterminer ses « bits », c'est-à-dire à trouver sa représentation binaire.

Un nouvel **objectif** apparaît alors : il s'agit d'écrire une fonction Python <u>non récursive</u> renvoyant une liste de toutes les parties (sous-listes) de $\{0,1,\ldots,n-1\}$, en s'appuyant sur la représentation binaire de l'entier n.

Objectif 4 : codage en Python

Commençons par la recherche des bits d'un nombre entier `m`. Le bit le plus aisé à trouver est le bit « de poids faible » c'est-à-dire associé à $1=2^0$ dans l'écriture binaire de `m` : ce bit vaut 1 si m est impair et 0 sinon. Pour accéder aux autres bits, il suffit de diviser `m` par 2 : cela fait perdre le bit de poids faible et décale tous les autres bits. La fonction ci-contre réalise ce principe.

```
ÉDITEUR : PARTIES
LIGNE DU SCRIPT 0021
def bits(m): # liste bits de m
  L=[] # réceptacle réponse
  k=0
  while m>0:
    if m % 2==1: # impair ?
      L.append(k)
    k=k+1 # bit suivant
    m=m//2 # décalage
  return L
```

```
PYTHON SHELL
>>> bits(13)
[0, 2, 3]
```

C'est ainsi que le choix du nombre 13, dont les bits ont pour rangs 0, 2 et 3, est associé au choix de la partie $\{0,2,3\}\subset\{0,1,2,3\}$. En récupérant les ensembles de bits de tous les entiers compris entre 0 et 2^n-1 on aura la liste de toutes les parties possibles, ce que la fonction `partbin` réalise.

```
ÉDITEUR : PARTIES
LIGNE DU SCRIPT 0032
def partbin(n):
  R=[] # réceptacle réponse
  for k in range(2**n):
    R.append(bits(k))
  return R
```

```
PYTHON SHELL
>>> partbin(4)
[[], [0], [1], [0, 1], [2], [0,
2], [1, 2], [0, 1, 2], [3], [0,
3], [1, 3], [0, 1, 3], [2, 3], [
0, 2, 3], [1, 2, 3], [0, 1, 2, 3
]]
```

Les parties apparaissent dans un ordre peu intuitif, mais ce n'est qu'une question de tri.

23 Abréviation anglaise pour « binary digits », c'est-à-dire chiffres binaires.

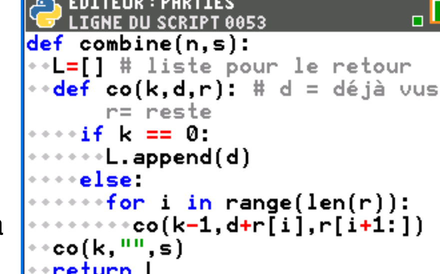
Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Défi : peut-on modérer un peu la récursivité ?

La récursivité présente l'avantage de la simplicité mais l'inconvénient d'amener parfois une « explosion » de la consommation de mémoire, très inefficace. C'est le cas dans la fonction `combin` ci-dessus : lors de l'appel `combin(2,"abcd")`, se déclenchent les appels `combin(1,"bcd")` et `combin(2,"bcd")` ; le premier va appeler `combin(1,"cd")` et le second de même : il y a un « doublon ». Dans l'appel `combin(2,"abcde")` les doublons sont bien plus fréquents…

Une fonction « simplement récursive » (qui s'appelle elle-même une seule fois) serait bien préférable. Nous en présentons un exemple ci-dessous.



Les lecteurs sont invités à en analyser le comportement en décrivant pas à pas (sur papier !) les actions qui se suivent lors de l'appel `combin(2,"abcd")`.

Indication : nous avons ici une « définition de fonction à l'intérieur d'une fonction ». Cette manière de procéder est tout à fait légitime en Python ; la fonction interne `co` (qui est récursive) va pouvoir traiter la variable `L` comme « externe » mais modifiable.

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021 / Photocopie autorisée

Appendice 1 – compléments Python

Quelques compléments sur la syntaxe du langage Python

Généralités

Nombres

- Les nombres en notation scientifique : $5{,}1\times10^{-3}$ se note `5.1e-3`. Le « e » comme « exposant » peut être tapé comme lettre e ([alpha] + [e]) ou symbole EE ([2nde] + [EE]).
- Pour arrondir un nombre à un certain nombre de décimales, on emploie la fonction Python `round`. Ne pas confondre avec la fonction `int` ou « partie entière » qui convertit des valeurs à virgule (« flottantes ») en entiers.
- La division : `a/b` donne toujours une valeur flottante, tandis que `a//b` donne le quotient entier.
- Les puissances : l'opérateur « puissance » se code `**` : x^5 se note `x**5`.
- Le nombre `1e3` ne vaut pas 1000 (entier) mais 1000.0 (flottant). Le plus petit nombre strictement supérieur à 1 est $1+2^{-52}$: avec `X=1+2**-53`, le test `X==1` donne `True`.

Itérateur range

Syntaxe	Exemples

La fonction Python range ne crée pas de liste mais un objet interne qui peut servir pour des boucles (`for`) ou autres itérations. Pour avoir une liste il faut recourir à la fonction `list`.
Variantes : `range(a,b)` produit tous les entiers depuis `a` (inclus) jusqu'à `b` (exclu), et `range(a,b,-1)` fait de même en descendant, sous réserve d'avoir a>b.
	`>>> list(range(10))`
	`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
	`>>> list(range(1,10))`
	`[1, 2, 3, 4, 5, 6, 7, 8, 9]`
	`>>> list(range(9,0,-1))`
	`[9, 8, 7, 6, 5, 4, 3, 2, 1]`

Listes

Listes en extension

Premier moyen de définition de listes : on donne tous les éléments, *in extenso*.

Syntaxe	Exemples
`L=[élément1,élément2,… élémentN]` `L=[élément]*n` (répétition)	`L=[0,1,4,9,16,25,36,49]` `L=[0]*5` donne `[0,0,0,0,0]`

Listes en adjonction

Second moyen : on agrandit la liste petit à petit. Ainsi, l'ensemble des carrés de 0 à 7 : $\{k^2 / k\in\mathbb{N}, k\leq7\}$.

Syntaxe	Exemple

`L=[] # liste vide`
`for k in range(N):`
`  L.append(élément_k)`
	`L=[]`
	`for k in range(8):`
	`  L.append(k**2)`

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

Appendice 1 – compléments Python

Listes en compréhension

Troisième moyen de définir des listes, proche de la notation mathématique des ensembles.

Syntaxe	Exemple
`L=[élément_k for k in range(N)]`	`L=[k**2 for k in range(8)]`

Manipulations de listes

Syntaxe	Exemples

Les listes de Python sont indexées à partir de 0. Si on a une liste `L`, sa longueur est donnée par `len(L)`, son premier terme est `L[0]`, le second `L[1]` et le dernier `L[len(L)-1]`.	`>>>L=[2,3,5,7]`
	`>>>L[1]`
	`3`
	`>>>len(L)`
	`4`

Pour ajouter un terme à la fin de la liste `L`, on passe par l'instruction `L.append(element)`. Pour retirer le terme de numéro `k`, on passe par l'instruction `L.pop(k)` qui supprime `L[k]` de `L` et renvoie sa valeur (la liste `L` est ainsi raccourcie).	`>>>L.append(11)`
	`>>>L.pop(0)`
	`2`
	`>>>L`
	`[3,5,7,11]`

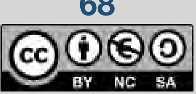
Pour avoir la sous-liste débutant au terme d'indice `k` et sans limite on code `L[k:]`, alors que la sous-liste commençant au début et s'arrêtant au terme d'indice `k-1` se code `L[:k]`. Quant à chercher l'indice d'un terme valant `p` (s'il existe), on l'obtient avec `L.index(p)` (si aucun terme de `L` n'a la valeur `p`, cela provoque une erreur).	`>>>L.append(11)`
	`>>>L.pop(0)`
	`2`
	`>>>L`
	`[3,5,7,11]`

Pour mettre « bout à bout » (concaténer) deux listes `L1` et `L2`, on code cela : `L1+L2`.	`>>>L2=[13,17,19]`
	`>>>L+L2`
	`[3,5,7,11,13,17,19]`

Pour trier une liste `L`, on code : `L.sort()`. Pour le tri à l'envers on fait : `L.reverse()`.
Si on veut créer une liste triée `LS` à partir de `L`, sans écraser `L`, on code : `LS=sorted(L)`. Et à l'envers : `reversed(L)`.
	`>>>L.reverse()`
	`>>>L`
	`[11,7,5,3]`
	`>>>sorted(L)`
	`[3,5,7,11]`

Connaître la valeur la plus élevée dans une liste se fait avec la fonction `max`, la plus faible avec la fonction `min`, la somme avec la fonction `sum`.	`>>>min(L),max(L),sum(L)`
	`(3,11,26)`
	`>>> def moy(l):`
	`...     return sum(l)/len(l)`
	`>>> moy(L)`
	`6.5`

Parcourir une liste se fait avec le mot-clé `for` :
`for k in L:`
	`>>> p=1`
	`>>> for k in L: p=p*k`
	`...`
	`>>> p`
	`1155`



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

Appendice 1 – compléments Python

Chaînes de caractères

Définition

Syntaxe	Exemples

Les textes ou chaînes de caractères sont simplement des suites ordonnées de caractères. On les introduit entourées d'apostrophes simples ou doubles.	`>>> s="Six cent"`
	`>>> t=' six scies'`
	`>>> s+t`
	`'Six cent six scies'`

Un texte sur plusieurs lignes peut être introduit entouré de triples guillemets. Les fins de ligne sont incluses dans la chaîne.	`s="""Une ligne`
	`Seconde ligne"""`
	`print(s) #fin de ligne en plein milieu`

Manipulation de chaînes de caractères

L'extraction, la concaténation de chaînes de caractères fonctionnent comme pour les listes, excepté `pop` et `sort` qui ne fonctionnent pas.

Les chaînes de caractères, comme les listes, peuvent être « parcourues » par itération au moyen de boucles `for`.

Fonctions

Logarithme

Attention : la fonction « logarithme népérien » est notée `log` en Python.

Enchaînements de fonctions

Le langage Python permet d'écrire des fonctions appelant d'autres fonctions et ainsi de suite (jusqu'à un certain niveau). L'usage intensif de fonctions rend le code plus lisible et adaptable, à l'opposé du « code-spaghetti » que l'on voit trop souvent sur les calculatrices.

Une fonction peut s'appeler elle-même, cela s'appelle de la « **récursivité** » … à condition de ne pas aboutir à une boucle infinie. Voici deux exemples :

<table>
<tr><th>Multiplication</th><th colspan="2">Puissance</th></tr>
<tr>
<td>def mul(x,y) :
    if x==1 :
        return y
    else :
        return mul(x-1,y)+y
Cette fonction réalise la multiplication de deux nombres entiers naturels (par additions successives) ; cependant, si x n'est pas entier ou est négatif, la</td>
<td>L'appel de la fonction puiss pour calculer les puissances d'un nombre crée des appels « en cascade » jusqu'à ce que puiss(x,0) renvoie 1. La variante ppuiss (mettant à profit la parité de l'exposant pour raccourcir les calculs) est plus performante ; elle</td>
<td>EDITEUR : PUISS
LIGNE DU SCRIPT 0005
def puiss(x,n):
  if n == 0:
    return 1
  return x*puiss(x,n-1)

def ppuiss(x,n):
  if n == 0:
    return 1
  if n % 2 == 0:
    return ppuiss(x*x,n//2)
  return x*ppuiss(x,n-1)</td>
</tr>
</table>

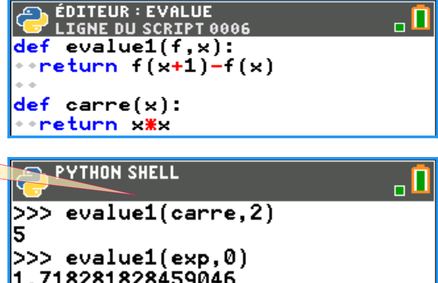
Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

boucle devient infinie, amenant une erreur à l'exécution.	montre aussi qu'une fonction peut s'appeler elle-même en plusieurs endroits.

Une fonction peut être passée comme paramètre à une autre fonction. L'exemple ci-contre confirme cette possibilité.

Ici carre désigne une fonction Python.



Bibliothèque (ou module) random

Quand on importe le module random, on obtient quelques fonctions Python utiles pour simuler des lois de probabilité, notamment :

Fonctions	Exemples

`randint(a,b)`
Tire au hasard un nombre entier compris entre a et b (inclus).
	`>>> randint(0,1)`
	`0`
	`>>> randint(0,1)`
	`1`

`choice(liste)`
Tire au hasard un élément de la liste donnée.
	`>>> choice([-1,1])`
	`-1`
	`>>> choice([-1,1])`
	`1`

`random()`
Tire au hasard un nombre compris entre 0 et 1 suivant une « distribution uniforme ».
	`>>> random()`
	`0.3208598777888272`
	`>>> random()`
	`0.4957570280169452`

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

Appendice 2 – modules Python additionnels

Aide-mémoire sur les bibliothèques (modules) utilisées

time

Le module `time` donne accès à quelques fonctions liées au temps.

TI-83 Premium CE Edition Python	TI Nspire CX II-T
`sleep(t)` interrompt l'exécution pendant `t` secondes avant de poursuivre.	
`monotonic()` renvoie la valeur d'un « compteur de temps » (en secondes) permettant de mesurer le temps écoulé.	`clock()` a le même effet.

ti_system

Le module « système » `ti_system` donne accès à quelques fonctions spécifiques de la machine.

TI-83 Premium CE Edition Python	TI Nspire CX II-T
`disp_wait()` permet d'interrompre l'exécution et d'attendre que l'utilisateur appuie sur la touche [annul].	Même résultat (attendre [esc]) avec `while get_key() != "esc":`
`sleep(t)` interrompt l'exécution pendant `t` secondes avant de poursuivre.	
La fonction `recall_list` permet de récupérer le contenu d'une liste du système natif de la calculatrice. L'interface de la fonction `recall_list` est particulière : seules les listes système L_1 à L_6 sont accessibles, et désignées par les chaînes de caractères "1" à "6". Exemple : `uneliste=recall_list("2")`. À noter : tandis que les listes en Python sont indexées à partir de 0, dans l'environnement natif c'est à partir de 1.	On utilise le nom d'une variable (de type liste) déjà définie dans le classeur en cours. Exemple : `uneliste=recall_list("L2")`. À noter : tandis que les listes en Python sont indexées à partir de 0, dans l'environnement natif c'est à partir de 1.
La fonction `store_list` fait l'inverse, copiant une liste dans l'environnement natif de la calculatrice, avec quelques limitations (en taille notamment : au maximum 99 éléments). Exemple : `store_list("2",maliste)`.	On donne un nom de liste arbitraire, qui sera créée dans le système d'exploitation. Exemple : `store_list("L5",[1,2,42])`
La fonction `recall_RegEQ` permet de récupérer les expressions des fonctions calculées comme « régressions » (touche [stats], rubrique CALC, menu E:TracéAjust-Éq). La valeur `s` retournée par `s=recall_RegEQ()` est en fait une chaîne de caractères contenant une expression mathématique dépendant d'une variable nommée `x`. En ayant assigné une valeur à la variable `x`, `eval(s)` renvoie la valeur de l'expression mathématique pour la valeur voulue de la variable `x`.	Pas d'équivalent mais on peut « importer » une fonction Nspire dans l'environnement Python avec `eval_function("mafonction",valeur)` où `mafonction` est une fonction Nspire (d'une seule variable) déjà définie ; par exemple une fonction d'interpolation ...

Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

Appendice 2 : modules Python additionnels

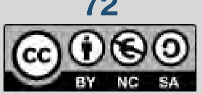
ti_plotlib

Le module `ti_plotlib` donne accès à de nombreuses fonctions de tracé graphique dans une « fenêtre » correspondant à l'écran (de format 3:2) mais avec des coordonnées définies par l'utilisateur.

Instructions	Exemples
`plt.cls()` permet d'effacer l'écran.	`plt.cls()`
`plt.window(xmin,xmax,ymin,ymax)` permet le cadrage de l'affichage.	`plt.window(-2,2,0,5)` ajuste la fenêtre, [–2;2] en abscisses etc.
`plt.autowindow(x-list,y-list)` ajuste le cadrage de l'affichage en fonction d'une liste d'abscisses et ordonnées (afin que tous les points soient visibles)	`plt.autowindow([0,4.5],[1,9.8]))` fixe une fenêtre d'affichage avec des « coins » de coordonnées (0;1) / (4,5;9,8).
`plt.axes("off/on")` permet d'afficher ou non les axes du graphique.	`plt.axes("off")` permet de ne pas afficher les axes du graphique.
TI-83 : `plt.grid(xscal,yscal,type)` affiche la grille en réglant l'écartement sur les axes et le « type » de la grille, valant `"dot"`, `"dash"`, `"solid"` ou `"point"`. **Nspire CX** : Type vaut `"dotted"`, `"dashed"`, ou `"solid"`.	`plt.grid(0.2,0.5,"solid")` pour une grille avec des lignes continues `plt.grid(1,5,"dash")` pour des pointillés.
`plt.color(R,G,B)` fixe la couleur du tracé à partir du code RGB (3 nombres entre 0 et 255 pour les composantes de couleur rouge, vert, bleu de la couleur globale).	`plt.color(255,0,0)` sélectionne une couleur rouge « vif ». `plt.color(0,0,0)` sélectionne le noir.
`plt.show_plt()` finit d'afficher le graphique et se met en attente (appuyer sur annul pour continuer).	`plt.show_plt()`
`plt.text_at(ligne,"texte","left/right/center")` permet d'afficher du texte en définissant la ligne d'affichage, le texte à afficher et le centrage sur la ligne sélectionnée. Il y a 12 lignes d'affichage possibles.	`plt.text_at(12,"hello","left")` affiche sur la ligne 12 à gauche le texte : hello. **Note :** l'arrière-plan des caractères est effacé.
`plt.plot(a,b,"marque")` place sur le graphique un point aux coordonnées (a,b) du style de la « marque ».	`plt.plot(2,3,"+")` affiche **+** au point de coordonnées (2;3).
`plt.scatter(x-list,y-list,"marque")` trace un nuage de points d'après une liste d'abscisses et une liste d'ordonnées, avec le style de la « marque ».	`plt.scatter([1,2],[3,7],"o")` affiche une "bulle" aux points de coordonnées (1;3) et (2;7).

`plt.plot(x-list,y-list,"marque")` trace un nuage de points d'après une liste d'abscisses et une liste d'ordonnées et le style de la « marque », et les segments connectant ces points.	`plt.window(-1.6,1.6,-.1,2)`
	`plt.plot([-1,0,1,-1],`
	`[0,1.732,0,0],"o")`
	affiche un triangle équilatéral avec des « bulles » aux sommets.

`plt.line(x1,y1,x2,y2,"mode")` trace un segment d'après les coordonnées de ses extrémités et ajoute éventuellement une flèche au bout.	`plt.line(0,0,2,3,"arrow")` trace le vecteur partant de l'origine et allant vers le point de coordonnées (2;3).



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

Appendice 2 – modules Python additionnels

turtle

La « tortue » peut être imaginée comme un stylet dessinant sur une feuille de papier et se déplaçant suivant des instructions du type « à gauche », « à droite », « en avant », etc. La « taille » de la feuille est de 316×208 pixels, avec un référentiel centré au milieu.

TI-83 Pour utiliser le module Turtle, il faut préalablement télécharger vers la calculatrice le fichier `CE_TURTL.8xv` . Il est possible de mettre ce fichier en « archive » (touche [mém], choix 2 et 1) si la mémoire fait un peu défaut. Ensuite, on doit passer dans l'environnement Python, créer un nouveau script vide et y écrire la ligne d'importation :

```
from ce_turtl import *
```

Dès que cette ligne est écrite, on trouve[24] dans le menu Modul une nouvelle ligne

```
8: ce_turtl …
```

donnant accès à l'ensemble des commandes et fonctions faisant partie de ce module.

Nspire CX Pour utiliser le module Turtle, il faut préalablement le télécharger dans la calculatrice le fichier `turtle.tns`, puis l'ouvrir comme classeur. Des instructions figurent à la page 1.1. Appuyer sur [ctrl] + [►] pour passer au shell Python figurant à la page 1.2, puis sur [menu] et sélectionner "Outils ► 8 installer en tant que module Python".

Une fois l'installation terminée, le message "**Installation terminée**" apparaît, confirmant l'installation[25]. Pour accéder à la programmation Python avec Turtle et aux menus adaptés, il convient de créer un nouveau document : depuis l'écran d'accueil, sélectionner "Nouveau / Ajouter Python / Nouveau...".

À ce stade, on est dans une page d'édition de programme Python. Quand on appuie sur la touche [menu], on peut sélectionner "A: Plus de modules ► 6 Turtle Graphics", les commandes Turtle y sont.

Dans tous les cas, la première ligne du programme consistera à créer une « instance » de Turtle, sous la forme `t=Turtle()` ou `turtle=Turtle()` selon le système et les mises à jour.

Voici quelques commandes fournies grâce à l'importation du module `turtle` :

- `t.clear()` permet d'effacer l'écran.
- `t.home()` permet de remettre la tortue au centre de l'écran et dirigée vers la droite (azimut 0°).
- `t.penup()` ou `t.pendown()` permet de lever ou baisser le stylet.
- `t.goto(a,b)` permet de se positionner aux coordonnées (a,b) (en pixels).
- `t.show()` « montre» la tortue et se met en attente d'un appui sur la touche [annul].
- `t.left(d)` et `t.right(d)` permettent de faire « tourner » la tortue (elles prennent comme paramètre un angle `d` en degrés).
- `t.forward(p)` permet de faire « avancer » la tortue d'un nombre `p` de pixels.
- `t.circle(r)` permet de tracer un cercle de rayon `r` pixels, la tortue étant au centre.
- `t.dot(s)` permet de tracer un disque de rayon `s` pixels.

24 Sous réserve d'avoir mis à jour la calculatrice avec un système 5.7 au moins.

25 Une fois installé, le fichier du module Turtle est déplacé dans le dossier Pylib, dans le dossier Pylib.

Ce document est mis à disposition sous licence Creative Commons

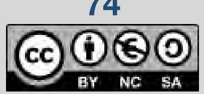
© Texas Instruments 2021-2022 / Photocopie autorisée

- `t.color(r,g,b)` (ou `t.pencolor(r,g,b)` selon le système et les mises à jour) permet de fixer la couleur des traits en composantes de rouge, vert, bleu. Le code RGB permet de définir la couleur d'un pixel à l'aide de trois nombres entiers compris entre 0 et 255 décrivant des intensités lumineuses pour les composantes rouge, verte et bleue de la lumière (0 indiquant une absence de lumière).
- `t.pensize(p)` permet d'avoir des tracés de largeur fine (p=0), moyenne (p=1) ou épaisse (p=2).
- `t.speed(v)` permet d'avoir des tracés lents (v=0) ou rapides (v=1 ou v=10 selon le système).
- `t.setheading(d)` permet de « diriger » la tortue dans l'azimut (ou direction) `d`, `d` étant un angle en degrés compté à partir de l'horizontale dans le sens trigonométrique.
- `t.position(v)` renvoie un couple (a,b) donnant la position de la tortue (en pixels).
- `t.heading(v)` renvoie la valeur de l'azimut de la tortue (en degrés).

ti_draw

Ce module donne accès à des fonctions de tracé graphique « ponctuel » (bitmap) à l'écran avec des coordonnées « absolues » (pixels).

Instructions	Exemples
Sur l'écran de la calculatrice il y a des colonnes et des lignes de pixels. Les pixels de coordonnées x et y sont repérés à l'aide de ces lignes et colonnes, avec $0 \leq x \leq 319$ et $0 \leq y \leq 209$. Le « coin » en haut à gauche a les coordonnées (0;0).	On commence par effacer l'écran : `clear()`
`set_color(R,G,B)` fixe la couleur pour les traits d'après des intensités de rouge, vert, jaune (comprises entre 0 et 255, 255 étant l'intensité la plus forte).	`set_color(128,128,128)` choisit un « gris moyen » et `set_color(255,0,0)` choisit un rouge « vif ».
`plot_xy(x,y,type)` permet d'allumer le pixel aux coordonnées $(x;y)$ sur l'écran de la calculatrice avec un « type » de point compris entre 1 et 13 (plus ou moins gros).	`plot_xy(159,105,7)` allume le pixel du centre de l'écran dans la couleur choisie.
`draw_line(x1,y1,x2,y2)` trace un segment d'après les coordonnées de ses extrémités.	`draw_line(0,209,319,0)` trace la diagonale montante de l'écran.
`draw_circle(x,y,r)` trace un cercle d'après les coordonnées de son centre et son rayon.	`draw_circle(159,104,105)` trace le plus grand cercle entier possible.
`fill_circle(x,y,r)` trace un disque (cercle plein) d'après les coordonnées de son centre et son rayon.	`fill_circle(212,104,105)` trace le plus grand disque possible, calé à droite.
`draw_text(x,y,"texte")` trace un texte commençant aux coordonnées indiquées.	`draw_text(0,0,"TITRE")`
TI-83 : `show_draw()` finit le dessin et se met en attente de l'appui sur annul.	



Ce document est mis à disposition sous licence Creative Commons

© Texas Instruments 2021-2022 / Photocopie autorisée

Plus d'informations sur le site internet de T[3] France :

- Des ressources pédagogiques pour votre classe
- Un programme de formations gratuites sur site et en ligne
- Des vidéos d'aide à la prise en main de la technologie

Un service après-vente est également accessible depuis le site **education.ti.com/fr/csc**