ALGORITHMES ET PROGRAMMES

Alain BUSSER
Irem de La Réunion

Introduction

Un *algorithme* est une méthode de résolution d'un problème rédigée sous la forme d'une suite d'instructions qui, lorsqu'on les suit dans l'ordre, mènent à la solution du problème, en un temps fini, de façon reproductible, et pour tout choix des données d'entrée. Par exemple, l'algorithme d'Euclide[1] a pour but de calculer un pgcd, et dépend de deux données d'entrée (les deux nombres dont on veut le pgcd), et, en un temps fini[2], donne le pgcd en répétant l'opération consistant à remplacer les deux nombres, respectivement par le plus petit d'entre eux et le reste de la division euclidienne :

```
Algorithme pgcd (a,b)
    répéter
        (a, b) ← (b, le reste de x dans
                  la division par y)
    jusqu'à ce que b = 0
    retourner a
```

1 En réalité Euclide procédait par soustractions itérées, la version avec divisions euclidiennes est une amélioration de son algorithme suite à une remarque d'Archimède (l'ensemble **N** des entiers naturels est archimédien)

Programmer un algorithme, c'est créer un programme permettant de tester l'algorithme et bien entendu tester le programme pour le plaisir de voir que ça marche.

En Python ça donne quelque chose comme ça[3] :

```
def pgcd(a,b):
    while b!=0:
        a,b = b, a%b
    return a

print(pgcd(24,32))
```

2 et même très court, le maximum étant atteint lorsqu'on calcule le pgcd de deux nombres de Fibonacci successifs, et vaut le rang du plus petit des deux dans la suite de Fibonacci.

3 Comment Python sait-il que a et b sont des entiers ? C'est le *duck typing* qui le permet : Comme on calcule à la ligne 3, le reste a%b de la division de a par b, cela suppose que a et b sont des entiers et Python fait cette inférence de type.

Or les activités d'« algorithmique », qu'elles soient menées en classe ou dans un sujet de contrôle/examen, concernent le plus souvent la programmation : on donne l'algorithme et les élèves sont chargés de le traduire sous la forme d'un programme sur ordinateur ou calculatrice, ou alors on donne un « algorithme » à corriger ou à compléter. La dernière ligne du programme Python ci-dessus comporte un affichage qui, à lui seul, montre qu'il s'agit d'un programme (qui réalise une action, à savoir l'affichage du pgcd) et non d'un algorithme, lequel ressemble plus, conceptuellement, à une fonction (celle qui, à deux entiers, associe leur pgcd).

Le propos de cet article est de montrer comment on peut, hors évaluation, insister sur la première étape (création de l'algorithme) *avant* de passer à la réalisation du programme. Des séquences pédagogiques très largement basées sur des TP où figuraient algorithmes *et* programmes ont permis, dans une optique de classe inversée, d'introduire le cours sur le logarithme et l'exponentielle.

1. — En Terminale

La plupart des élèves de terminale, lors du cours sur les primitives, semblent découvrir la formule pour les primitives de x^n. C'est l'occasion de leur montrer que si n est négatif, la formule s'applique aussi, par exemple avec $1/x^2 = x^{-2}$ dont une primitive est bel et bien $x^{-2+1}/(-2+1) = -x^{-1} = -1/x$, comme on peut le vérifier en dérivant $-1/x$.

Mais surtout, c'est l'occasion de constater que la formule ne fonctionne pas avec un exposant – 1 qui annulerait le dénominateur, et l'annonce est faite à ce moment du cours que si $1/x$ a bel et bien une primitive, ce n'est pas sous cette forme qu'on la "calcule". Le TP suivant a été fait juste après le cours sur les primitives.

1. *Introduction du logarithme*

Il s'agit là d'un TP de programmation : l'algorithme d'Euler a été décrit aux élèves qui ont ensuite eu le choix de l'outil (tableur, calculatrice, langage de programmation [4], …). Pour décrire la méthode d'Euler, il a été fait usage de la notation de Leibniz, en disant que si $1/x = \mathrm{d}f/\mathrm{d}x$ alors $\mathrm{d}x/x = \mathrm{d}f = f(x+\mathrm{d}x)-f(x)$ et finalement $f(x + \mathrm{d}x) = f(x) + \mathrm{d}x/x$.

La méthode d'Euler consiste essentiellement à remplacer une équation différentielle (ici, $y' = 1/x$) par une équation aux différences, obtenue en remplaçant l'infinitésimal $\mathrm{d}x$ par une approximation finie (ici, $y_{n+1} - y_n = \mathrm{d}x/x_n$ avec $x_n = x_0 + n \times \mathrm{d}x$).

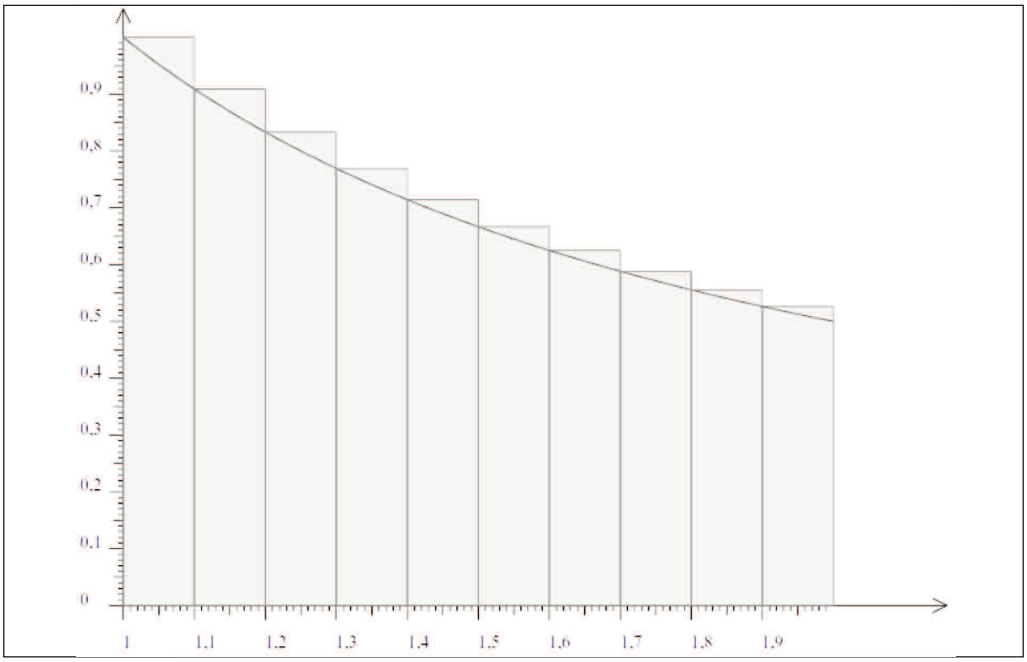
On remarque que dans ce cas on retombe sur la méthode des rectangles pour calculer l'intégrale de $1/x$ entre 1 et 2 : la suite récurrente y_n ci-dessus, avec $y_0 = 0$, peut se réécrire $y_{n+1} = y_n + \mathrm{d}x/x_n$ et on reconnaît l'algorithme de sommation utilisé par exemple plus bas pour calculer *e*.

La somme des $\mathrm{d}x/x$ est celle des aires des rectangles de largeur $\mathrm{d}x$ et de hauteur $1/x_n$ que l'on voit sur la figure de la page ci-contre. Et la somme des aires des rectangles est l'aire de l'histogramme, qui approche bien celle du domaine sous l'hyperbole.

En prenant une valeur petite et constante de $\mathrm{d}x$ on a une suite récurrente permettant de calculer les unes après les autres les valeurs de la fonction f (le logarithme népérien, si on commence par $x = 1$ et $y = 0$). Pour calculer $ln(2)$ par cette méthode on a cet algorithme [5] :

4 Essentiellement Sofus, Algobox, Python et C++

5 Remarque sur la notation: x←1 se lit parfois "*x prend la valeur 1*", ce qui a le défaut ne pas expliquer comment on fait pour que x prenne la valeur 1; la notation de la calculatrice 1→x se lit plus naturellement "*on met 1 dans x*"




```
dx ← 0.01
x ← 1
y ← 0
pour n allant de 0 à 99
      x ← x + dx
      y ← y + dx/x
fin du Pour
```

Le fait que la boucle doit être parcourue 100 fois si dx = 0,01 est intéressant à analyser. Et pour vérifier que les élèves ont compris le lien entre dx et le nombre de répétitions de la boucle, il leur a été demandé de chercher un résultat plus précis pour $f(2)$. Ce que la plupart d'entre eux ont trouvé seuls[6]. On trouvera en haut de la page suivante un tableau donnant les valeurs obtenues pour diverses valeurs de dx.

On découvre alors expérimentalement que, plus le pas est petit et plus l'algorithme est précis[7], et on imagine l'existence d'une valeur limite de $f(2)$ lorsque le pas est infinitésimal, comme somme d'un nombre infini de termes de la forme dx/x, et qu'il est donc justifié de noter :

$$\int_1^2 \frac{dx}{x} .$$

6 Avec l'ordinateur il est presque immédiat de boucler un million de fois. Avec la calculatrice c'est impossible. de toute façon en TP contrairement aux épreuves écrites, on dispose de l'ordinateur, si du moins le réseau le permet.

7 Sauf si le pas est vraiment trop petit, la précision ayant tendance à se détériorer dans ce cas. On constate ce phénomène lorsqu'on essaye de prendre un nombre trop petit (comme -1000000) comme approximation de $-\infty$ pour calculer la fonction de répartition d'une loi normale, alors que $\mu - 4\sigma$ permet parfois une meilleure précision (la probabilité que $X < k$ étant très proche de la probabilité que $\mu - 4\sigma < X < k$)

dx (incrément de x)	*nombre de répétitions*	*résultat*
0,1	10	0,668771403175
0,01	100	0,690653430482
0,001	1000	0,69289724306
0,000 1	10 000	0,693122181185
0,000 01	100 000	0,693144680565
0,000 001	1 000 000	0,693146930576

Mais surtout, en affichant les valeurs successives de f(x) calculées dans la boucle, on fabrique une table de logarithmes qui montre que ce n'est pas *un nombre* qu'on calcule ainsi mais *les valeurs d'une fonction*, définie dans le cours comme primitive de $1/x$ s'annulant en 1. C'est bien entendu sur cette fonction que portait le cours suivant. Mais rapidement il y a besoin dans ce cours de résoudre l'équation $ln(x) = 1$.

2. *Le nombre e*

Pour résoudre l'équation $ln(x) = 1$ à 6 décimales, les élèves n'ont pas reçu d'emblée l'algorithme à programmer mais au contraire, ont été incités à inventer le leur. Cette invitation, au début déroutante pour eux, est une excellente occasion d'expliquer ce que signifie la mention «*toute trace de recherche même incomplète devra figurer sur la copie*». En effet les élèves de terminale ne sont pas nécessairement habitués aux exercices à prise d'initiative ni aux narrations de recherche. Typiquement ils ont tendance à poser quelques mots sur la copie en croyant être exhaustifs. Par exemple, à la remarque selon laquelle 2,78 n'est pas une réponse parce qu'on ne sait pas comment ce 2,78 a été trouvé, la réponse orale est en général «je l'ai trouvé par tâtonnements avec la calculatrice». Les élèves ne sont tout simplement pas conscients que les tâtonnements font partie des traces de recherche à rédiger sur la copie! Mais une fois qu'ils acceptent que tous les tâtonnements doivent être rédigés par écrit, ils s'y mettent volontiers[8], jusqu'au moment où ils trouvent comment déléguer ce travail fastidieux à la calculatrice : Ils ont automatisé leurs tâtonnements suffisamment pour avoir *créé leur propre algorithme* !

Voici, dans le tableau de la page suivante, un exemple de tâtonnements rédigés…

Puis «blocage à 1,0002» mais avec annonce que l'algorithme par balayage permet de finir le travail. On constate sur cette narration de recherche que les nombres ne sont pas choisis totalement au hasard et qu'il y a bien élaboration d'un algorithme. Le choix de 2 (moyenne de 1 et 3) puis de 2,5 (moyenne de 2 et 3) suggère une approche dichotomique, mais la plupart des élèves abandonnent vite celle-ci parce qu'ils trouvent plus simples les nombres décimaux que les nombres binaires

8 D'autant qu'on leur a annoncé que les traces de recherche sont valorisées au bac.

valeur de x essayée	*son logarithme*
1	0
3	1,0986
2	0,69313
2,5	0,91629
2,7	0,99325
2,722	1,0014
2,71877	1,0002

(2,75 est un nombre apparemment assez compliqué). Voici l'algorithme par dichotomie dans ce cas de figure :

```
borneInf ← 1
borneSup ← 3
Tant que borneSup-borneInf > 10^-6
    moyenne ← (borneInf + borneSup)/2
    si ln(moyenne) < 1
        borneInf ← moyenne
    sinon
        borneSup ← moyenne
fin du Tant que
```

Mais la plupart des élèves ont opté pour l'algorithme par balayage, qu'ils ou leurs voisins ont inventé pendant leur narration de recherche. En commençant par constater que $ln(2) < 1$ [9] alors que $ln(3) > 1$, on calcule, en boucle, les logarithmes de 2 puis 2,1, etc. :

```
x ← 2
Tant que ln(x) < 1
    x ← x + 0,1
fin du Tant que
```

Seulement on constate que, puisqu'on est sorti de la boucle, la valeur de *x* affichée est par excès [10]. Du coup il ne faut pas boucler à partir de 2,8 mais à partir de 2,7 :

```
x ← 2,7
Tant que ln(x) < 1
    x ← x + 0,01
fin du Tant que
```

On peut compenser en rajoutant une instruction après la boucle :

```
x ← 2,7
Tant que ln(x) < 1
    x ← x + 0,01
fin du Tant que
x ← x – 0,01
```

9 au passage l'un des tâtonnements a fait constater à un élève que ln(0,5) est l'opposé de ln(2) ce qui est une bonne préparation au cours sur ln(1/*x*).

10 La plupart des élèves montrent, malgré un manque de confiance en leur talent de programmeur, une nette préférence pour la programmation plutôt que pour le tableur, dont l'usage est pourtant intéressant pour ce genre d'activité.

Ensuite on calcule à partir de la valeur 2,71 :

```
x ← 2,71
Tant que ln(x) < 1
    x ← x + 0,001
fin du Tant que
x ← x – 0,001
```

Et comme on est en train de se livrer à une activité répétitive, la tentation est désormais grande de résumer ces algorithmes en un seul:

```
x ← 2
p ← 0,1
répéter 6 fois
   Tant que log(x) < 1
     x ← x + p
   fin du Tant que
   x ← x – p
   p ← p/10
fin de la boucle
```

Ainsi qu'on l'a dit plus haut, cet algorithme a plus de succès que la dichotomie auprès des élèves.

Un ilôt d'élèves a créé et programmé seul cet algorithme:

```
x ← 2
Tant que ln(x) < 1
    x ← x + 0,000001
fin du Tant que
```

Comparé à la dichotomie, cet algorithme est beaucoup plus simple. Cependant il est très lent, même sur ordinateur.

Cette activité offre plusieurs avantages sur l'étude d'une fonction (sujet prévu initialement) :

— Les élèves manipulent, ce en quoi il sont plus à l'aise que pour des exercices traditionnels.

— étant actifs même en îlots, les élèves relâchent moins leur attention que d'habitude.

— Les élèves, mis au pied du mur "inventez votre propre algorithme", créent ; cela les intéresse mais cela intéresse aussi le prof, qui peut avoir de bonnes surprises.

— Avoir calculé un nombre le rend en quelque sorte plus concret et les élèves se sentent moins largués par le cours après ces activités.

— L'exercice est l'occasion de former les élèves à la narration de recherche et la prise d'initiative, donc *in fine* à l'épreuve du bac.

— Les compte-rendus du TP étant notés, c'est aussi une occasion pour certains des élèves de monter la moyenne trimestrielle, ce qu'ils apprécient parfois assez pour se mettre pour une fois au travail.

— C'est l'occasion aussi de montrer que si deux algorithmes sont équivalents au sens où ils donnent le même résultat, ils ne le sont pas forcément en durée d'exécution, ce qui est l'objet même de la science appelée algorithmique.

— C'est l'occasion aussi de montrer l'avantage en terme de concision, de la nouvelle notation algorithmique: Écrire 3 algorithmes de 4 lignes chacun, c'est plus rapide qu'écrire 3 programmes algobox de 10 lignes chacun !

La suite c'est évidemment le cours sur l'exponentielle présentée comme interpolation de la suite géométrique de raison e ; et il est prévu d'autres TP sur la méthode d'Euler pour résoudre les équations différentielles voire, si le temps le permet, sur les intervalles

de fluctuation et leur utilisation pour les tests d'hypothèse.

2. — En BTS

Une difficulté récurrente avec l'exponentielle se pose aux bacheliers ST2S, qui connaissent la fonction 10^x mais pas la fonction e^x. Il est urgent de les familiariser avec le nombre *e* parce que la fonction logarithme népérien est définie à partir de l'exponentielle et que celle-ci dépend de *e*. Comme la part de l'algorithmique est croissante aussi en BTS, il est également nécessaire de former à la programmation sur calculatrice ainsi qu'à l'usage du tableur les étudiants qui n'ont pas forcément fait grand usage de ces outils. D'où l'idée d'un TP basé sur le calcul de *e*.

1. *Le nombre e*

La série des inverses des factorielles converge très rapidement vers *e*. Mais

— c'est une série et pas une suite : il faut savoir comment additionner les termes ;

— les factorielles ne sont plus au programme : elles aussi il faut voir comment on les calcule.

C'est donc un TP de programmation plutôt que d'algorithmique qui a été proposé en début d'année scolaire. Le calcul des factorielles (appelées "produits des entiers successifs") se fait avec cet algorithme (ici pour la factorielle de 10) :

```
p ← 1
pour n allant de 1 à 10
    p ← p × n
fin du Pour
```

À partir de là il est possible de poser un problème de maths : comment additionner les inverses des valeurs successives de *p* ? Même s'il est possible de s'inspirer de l'algorithme de multiplication ci-dessus, le fait qu'il y ait deux affectations dans une même boucle tend à distancer certains étudiants qui préfèrent l'usage du tableur et de sa fonction $\sum$. Quoi qu'il en soit voici l'algorithme de calcul de *e* (p comme *produit* et s comme *somme*):

```
p ← 1
s ← 1
pour n allant de 1 à 10
    p ← p × n
    s ← s + 1/p
fin du Pour
```

L'intérêt de ce TP pour le cours est essentiellement qu'il montre un exemple de suite convergeant vers autre chose que 0 et permet d'introduire le nombre *e*. La durée (une heure) n'ayant pas été suffisante pour faire autre chose que de la programmation, le TP a surtout servi à montrer l'attendu en terme de *rapport* de TP.

2. *Vers la constante d'Euler*

Le cours sur les logarithmes népériens a permis de revoir l'algorithme de sommation, pour permettre de vérifier la ressemblance entre la série harmonique (somme des inverses des entiers) et la fonction logarithme népérien.

Voici comment on calcule les termes de la série harmonique :

```
s ← 0
pour n allant de 1 à 100
    s ← s + 1/n
fin du Pour
```

En représentant par un nuage de points les valeurs successives on constate une ressemblance avec la représentation graphique de la fonction *ln*, ce qui amène à représenter graphiquement, ou simplement calculer, la différence entre s et $ln(n)$. On trouve que celle-ci se rapproche quoiqu'assez lentement de 0,582 : la constante d'Euler.

3. *Un sujet en devoir maison*

Voici un extrait du sujet de BTS groupement D 2017:

> Soit D une variable aléatoire. D suit la loi normale de paramètres $m = 46$ et $\sigma = 0{,}3$.
>
> Déterminer, par la méthode de votre choix, une valeur approchée à 10^{-1} près du nombre réel a tel que $P(46 - a \leq D \leq 46 + a) = 0{,}95$.

Plusieurs méthodes ont été utilisées en devoir maison (c'est le principe des exercices à prise d'initiative) :

— Le cours ("c'est un intervalle à 2σ" ou "à $1{,}96\sigma$"),

— Le calcul de la demi-largeur d'un intervalle de confiance,

— Le recours à la fonction de répartition avec $P(D \leq 46 + a) = 0{,}975$,

— mais aussi cet algorithme, qui marche bien puisqu'on demande a à 10^{-1} près :

```
a ← 0
Tant que P(46 – a ≤ D ≤ 46 + a) < 0,95
        a ← a + 0,1
fin du Tant que
```

Conclusion

Laissés face à leur esprit créatif, même les pas trop matheux apprécient de tenter d'inventer l'algorithme, une fois qu'ils savent que c'est sur les *traces de recherche* qu'ils seront évalués. La notation algorithmique avec les flèches est suffisamment concise pour permettre d'écrire rapidement l'algorithme et soit multiplier les algorithmes (étapes de la recherche), soit passer à la suite du sujet ou programmer l'algorithme. Un autre avantage non négligeable est que si on met des algorithmes courts dans le sujet on peut s'attendre à moins d'abandon de la question (consistant à compléter l'algorithme incomplet, corriger l'algorithme avec des erreurs, ou même demander comment modifier l'algorithme pour qu'il réponde à un autre problème).

Sitographie

— Sofus pour programmer avec des blocs :
https://alainbusser.github.io/Sofus/Sofus_fr.html

— SofusPy, idem mais avec passage automatique à Python :
http://irem.univ-reunion.fr/spip.php?article924

— Pour obtenir automatiquement l'algorithme avec notation moderne, à partir de Python ou d'Algobox :
http://irem.univ-reunion.fr/spip.php?article953

— Comptes-rendus des TP en BTS :
http://irem.univ-reunion.fr/spip.php?article950

ANNEXE

Les scripts Python

Logarithme de 2 par la méthode d'Euler[11] :

```
dx=0.01
x=1
y=0
for n in range(100):
   x = x+dx
   y = y+dx/x
print(y)
```

dichotomie

```
from math import *
borneInf = 1.0
borneSup = 3.0
while borneSup-borneInf>1e-6:
   moyenne = (borneInf+borneSup)/2
   if log(moyenne)<1:
      borneInf = moyenne
   else:
      borneSup = moyenne
   print(moyenne)
```

balayage à pas de 0,1

```
from math import *
x = 2
while log(x)<1:
   x = x+0.1
print(x)
```

balayage à pas variable:

```
from math import *
x = 2
p = 0.1
for n in range(6):
    while log(x)<1:
      x = x+p
    x = x-p
    p = p/10
    print(x)
```

11 L'outil en ligne SofusPy permet de créer ce genre de script Python en déplaçant des blocs comme avec Scratch ce qui a pour effet d'engendrer automatiquement le code Python, moyennant quelques retouches dans l'éditeur qu'il contient.

L'algorithme de l'ilôt créatif:

```
from math import *
x = 2
while log(x)<1:
   x = x+0.000001
print(x)
```

somme des inverses des factorielles:

```
p = 1
s = 1
for n in range(1,11):
   p = p*n
   s = s + 1.0/p
   print(s)
```

calcul de la constante d'Euler:

```
from math import *
s = 0
for n in range(1,101):
   s = s+1/n
   print(s-log(n))
```