Enseigner des mathématiques liées à l'informatique au (futur) lycée

Philippe Marquet
Univ. Lille

Résumé

L'enseignement d'informatique apparaît aujourd'hui dans les programmes du secondaire. Comment s'appuyer sur les savoirs et savoir-faire des élèves en ce domaine pour enseigner des mathématiques, pour enseigner des mathématiques utiles à de l'informatique. Nous présentons des propositions issues de travaux menés par les quatre sociétés savantes de mathématiques et d'informatique sur ce sujet [1]. Nous proposons ensuite quelques activités de mathématiques en lien avec l'informatique, activités exploitant aussi des connaissances et savoirs d'informatique exigibles des élèves.

Les sciences telles qu'elles se font aujourd'hui ne sont pas cloisonnées comme elles sont présentées dans les cursus scolaires. La richesse des liens entre les sciences est multiple. Cette richesse peut être exploitée quand il s'agit d'enseigner les sciences. Un enseignement de sciences doit permettre aux élèves de comprendre et mettre en œuvre une réelle démarche scientifique. La mise en œuvre d'une telle démarche repose sur des "outils", connaissances et savoir-faire, dans les différentes disciplines.

Dans la perspective d'un cursus comprenant des enseignements de mathématiques et d'informatique, la complémentarité de l'enseignement des deux disciplines est bénéfique à chacune d'elles.

C'est dans cet esprit qu'un groupe de travail des sociétés savantes de mathématiques et d'informatique a travaillé à des propositions d'enseignements qui concernent à la fois les mathématiques et l'informatique, dans la perspective de la refonte des programmes du lycée qui allait naturellement suivre celles des programmes du collège et du primaire.

1 Mathématiques et informatique dans le secondaire

Des enseignements d'informatique sont progressivement introduits dans les cursus, au collège comme au lycée. Aujourd'hui, la situation est la suivante :

— le programme du cycle 4, collège, en vigueur depuis la rentrée 2016 comporte un enseignement d'informatique dispensés à la fois dans la cadre des mathématiques et de la technologie. Il s'agit d'un enseignement de l'algorithmique et de la programmation au sein du programme de mathématiques, et d'un enseignement d'informatique et de programmation au sein du programme de technologie ;
— le programme de mathématiques du lycée aborde la notion d'algorithme dans un périmètre restreint, ne manipulant que des objets mathématiques.

1. Cette publication reprend les résultats des travaux réalisés par le groupe de travail des sociétés savantes de mathématiques et d'informatique, SFdS – Société française de statistique, SIF – Société informatique de France, la SMAI – Société de mathématiques appliquées et industrielles, et SMF – Société mathématique de France, qui a proposé des éléments d'un programme de mathématiques liées à l'informatique pour le lycée : *Quatre sociétés savantes de mathématiques et d'informatique font des propositions pour le futur programme de mathématiques du lycée*, octobre 2016, accessible à societe-informatique-de-france.fr/2016/10/propositions-maths-info-lycee/.

L'aménagement du programme de mathématiques de la classe de seconde, en vigeur depuis la rentrée 2017, introduit une section "Algorithmique et programmation".
L'enseignement de spécialité de mathématiques de la classe de terminale de la série économique et sociale aborde la notion de graphes et quelques algorithmes associés ;

— l'enseignement d'ISN – Informatique et science du numérique, enseignement de spécialité de terminale S, introduit il y a quelques années dans certains lycées, propose un large programme qui n'est couvert qu'en partie. Ces classes d'ISN sont assurées par des professeurs de différentes disciplines dont la moitié environ par des professeurs de mathématiques ;
— un enseignement exploratoire d'ICN – Informatique et création numérique, est proposé en tant qu'enseignement d'exploration en classe de seconde depuis la rentrée 2015. Sous ce même intitulé, est proposé dans certains lycée un enseignement facultatif en classes de première (sections S, ES et L) et en classes de terminale (sections L et ES) depuis la rentrée 2016.

Les programmes du futur lycée tels qu'ils se profilent en ce début 2018 suite aux annonces du Ministre, comporteront des enseignements d'informatique :

— un enseignement de tronc commun "Sciences numériques et technologie" en classe de seconde permettra à raison d'une heure et demie par semaine d'assoir les connaissances et savoir-faire d'informatique découverts au collège ;
— une discipline de spécialité "Numérique et sciences informatiques" sera proposée aux élèves de première à raison de 4 heures hebdomadaires, pouvant être prolongée en terminale à raison de 6h par semaine.

Les connaissances et compétences en science informatique de tous les lycéens vont rendre possible l'enseignement de mathématiques jusqu'alors absentes des programmes en s'appuyant sur des savoirs et savoir-faire informatiques. Elles vont aider à l'enseignement de ces mathématiques et plus généralement des mathématiques. Elles incitent aussi à enseigner des mathématiques en lien avec l'informatique qui ne sont pas abordées aujourd'hui.

2 Proposer des contenus pour les mathématiques au futur lycée

Anticipant la refonte des programmes du lycée qui nécessairement suivrait celle du collège, un groupe de travail d'une dizaine de personnes des quatre sociétés savantes de mathématiques et informatique a élaboré durant un an des propositions pour le futur lycée.

Il s'est agit de proposer des contenus pour des enseignements qui concernent les mathématiques et l'informatique, des contenus de mathématiques "en lien" avec l'informatique.

Ces contenus se devaient être profitables aux lycéens en tant que tels, soit que les notions étudiées sont essentielles, soit que leur étude répond à des objectifs généraux des enseignements de mathématiques ou d'informatique tel le développement des compétences de modélisation, de raisonnement, de logique. Ces contenus se devaient aussi d'être la base de prolongements en mathématiques, en informatique, plus généralement en sciences.

Les programmes de mathématiques pourraient par exemple intégrer des notions telles que des graphes, de la combinatoire ou de la logique, qui relèvent d'abord des mathématiques et sont aussi des notions fondamentales en informatique. La modélisation, déjà abordée au collège, devra être approfondie au lycée. Il faudra également sensibiliser les lycéens aux aspects mathématiques et informatiques liés à la représentation des nombres et aux calculs approchés sur machine.

Quand des notions de mathématiques nécessaires pour l'informatique qu'il convient d'enseigner au lycée sont identifiées, ne perdons pas de vue que ces mathématiques « liées » à l'informatique sont des mathématiques, et peuvent être étudiées pour elles-mêmes.

Cette proposition visait aussi à montrer les apports de la discipline informatique à l'enseignement des mathématiques, notamment dans une approche de mathématiques expérimentales.

L'approche du groupe de travail a été, pour chacun des quatre domaines identifiés, de présenter les notions clés qui semblent pouvoir et devoir être enseignées au lycée, ainsi que quelques exercices permettant de les illustrer. La démarche envisagée vise à introduire des notions fondamentales à partir d'exemples simples et d'expérimentations.

Enfin, une présentation sous forme de progression dégage les éléments qui concernerait plus particulièrement la classe de seconde générale, donc tous les élèves, ou les futurs scientifiques, ou les autres lycéens.

3 Quatre thèmes de mathématiques en lien avec l'informatique

Quatre thèmes ont été identifiés pour lesquels des propositions concrètes de contenu ont été élaborées. Il s'agit de

- la logique ;
- les graphes ;
- la combinatoire ;
- la représentation et modélisation de l'information.

Ces quatre thèmes ne couvrent pas l'ensemble des sujets de mathématiques pouvant être traités dans cet esprit, la géométrie pourrait par exemple être considérée, mais offrent néanmoins un panorama varié de notions de mathématiques en lien avec l'informatique.

Sur chacun des thèmes travaillés, quatre aspects ont fait l'objet d'une attention particulière :

- la définition des notions mathématiques couvertes par le thème pouvant être introduites dans un enseignement de mathématiques au lycée ;
- la pertinence des notions : ce sont bien des mathématiques, et ces mathématiques sont pertinentes pour l'informatique ;
- le support : en quoi l'informatique peut aider à enseigner ces notions ;
- la proposition d'exercices afin d'illustrer les notions, de calibrer ce qui peut être attendu de leur enseignement, de concrétiser le lien mathématiques/informatique sous-tendu.

L'élaboration de ces propositions a été éclairée par des retours d'expérience en classe. Cette étape essentielle est toutefois délicate aujourd'hui, le profil actuel des élèves et le cadre scolaire ne correspondent pas à ceux envisagés par les propositions de contenu. En effet, les connaissances et savoir-faire en informatique supposés des élèves et des enseignants (de mathématiques) pour la mise en place de ces enseignements ne sont pas réunies, l'enseignement d'informatique étant encore balbutiant.

L'accent a été mis, pour chacun des thèmes, sur la résolution de problèmes concrets, une éventuelle phase de modélisation – transcription dans un modèle du problème –, et sur de possibles réalisations informatiques – le plus souvent en terme de programmes –, dont l'analyse des résultats se fait au regard du problème.

Les quatre thèmes sont très rapidement décrits ci-après. On se référera au document produit par le groupe de travail pour plus de détails.

Logique

En ce qui concerne le thème de la logique, la motivation repose sur deux éléments essentiels :

- approfondir la notion de démonstration qui est à la base de tout raisonnement scientifique, et

— aborder la notion de connecteurs logiques qui sont utilisés en programmation, en particulier dans les instructions conditionnelles.

Il s'agira alors de travailler progressivement

— la notion de démonstration, en particulier d'un point de vue relativement intuitif, par exemple en découvrant les algorithmes de chaînage avant ou de chaînage arrière qui illustrent les notions d'axiomes et de règles d'inférences ;
— les notions de proposition et d'énoncé logique, dont la formalisation est justifiée par l'ambiguïté d'énnoncés en langage naturel ;
— la notion de modèle et de vérité dans un modèle.

Graphes

Sur le thème des graphes, il s'agit d'insister sur l'outil de modélisation que sont les graphes,

— de partir de problèmes concrets, de les modéliser par des graphes, puis
— d'identifier ou définir les algorithmes qui sur ces graphes permettent de résoudre le problème.

On s'attachera aussi

— à implémenter ces algorithmes, par exemple à l'aide de bibliothèques ;
— à travailler sur de grands graphes pour aborder le problème de la complexité en temps des algorithmes, mais aussi les aspects liés à la validation et à l'interprétation des résultats.

Les travaux du groupe IREM de Luminy [2] ont servi de base à ces propositions.

Combinatoire

Sur le thème de la combinatoire, la motivation est

— d'introduire les notions et savoir-faire permettant de compter les éléments d'ensembles finis, et
— de les transférer au calcul élémentaire de complexité en temps d'algorithmes.

On couvrira les notions

— de couples et n-uplets, leur dénombrement (et par exemple en générant les éléments d'ensembles à dénombrer) ;
— la fonction factorielle ;
— le triangle de Pascal.

Il sera ainsi possible de traiter

— d'identités remarquables et de probabilités ;
— de compter des objets informatiques ;
— de distinguer complexité en temps d'un algorithme et complexité intrinsèque d'un problème.

Cette partie repose largement sur une approche mathématique expérimentale dans laquelle le développement de programmes informatiques amène la découverte des notions, leur acquisition, en confrontant résultats expérimentaux et concepts mathématiques.

Représentation et modélisation de l'information

Concernant la représentation de l'information, la motivation est

2. *Graphes pour la Terminale ES*, octobre 2002, accessible à ens-info.irem.univ-mrs.fr/?p=261

— de relier des notions importantes de mathématiques (de représentation des nombres, combinatoire, probabilité, etc.) aux concepts fondamentaux de l'information numérique (compression des données, codes correcteurs d'erreurs), et
— d'appliquer ces concepts sur des données de grande taille pour apprécier les gains en taille et la robustesse aux erreurs.

On abordera par exemple

— la représentation des entiers en base n ;
— un peu d'arithmétique et de calcul modulaire ;
— de l'algèbre linéaire de base ;
— la représentation approchée des nombres réels et impact sur la stabilité des calculs numériques.

4 Quelques activités de mathématiques en liens avec l'informatique

Sont ici présentées quelques activités d'algorithmique et de programmation Python permettant de manipuler des objets mathématiques qui pourront ensuite être introduits ou revisités sur la base de ces manipulations.

Cette illustration ne couvre qu'une toute petite part des contenus mathématiques en lien avec l'informatique pouvant être abordées sur la base de connaissances en informatique des élèves.

Nous proposons dans la suite

— de découvrir les fonctions logarithmes ;
— de manipuler les puissances ;
— de toucher les polynômes.

Nombre de comparaisons, nombre de chiffres, logarithme

La fonction logarithme peut être définie de multiples manières. Une des définitions en motive également l'introduction dans un enseignement de mathématiques en lien avec l'informatique. Il s'agit de présenter le logarithme comme le nombre d'étapes de la recherche par dichotomie, un algorithme général aux applications multiples (dont, en mathématiques, la méthode de dichotomie de recherche d'un zéro d'une fonction).

Étant donné un ensemble d'éléments organisés dans un tableau trié, il s'agit de trouver la position d'un élément de valeur donnée (que l'on suppose présente dans le tableau).

Le principe est de

— considérer l'élément central dans le tableau ;
— comparer la valeur de cet élément à la valeur recherchée ;
— si les deux valeurs sont égales, on a trouvé la position de l'élément recherché, on s'arrête ;
— si l'élément recherché est inférieur à la valeur de l'élément central, on poursuit la recherche dans la première moitié du tableau ;
— sinon, l'élément recherché est supérieur à la valeur de l'élément central, on poursuit la recherche dans la seconde moitié du tableau.

À partir de cet algorithme, on se pose la question du nombre d'étapes nécessaires à la recherche d'une valeur dans un ensemble trié. On remarque que la recherche parmi n valeurs va consister en une étape suivie d'une recherche parmi $n/2$ valeurs, qui elle même va consister en une étape suivie d'une recherche parmi $n/4$ valeurs, et ce, au pire, jusqu'à la recherche parmi une valeur.

Le nombre d'étapes est donc égal au nombre de fois que l'on peut diviser la taille n de l'ensemble initial par 2 jusqu'à obtenir la valeur 1.

On peut décider de définir cela comme une fonction que l'on nomme logarithme, disons *fonction logarithme entière de base 2* pour préciser que la valeur de la fonction est entière et que les divisions considérées sont des divisions par 2.

Considérons maintenant le problème du calcul du nombre de chiffres d'un entier n, en base 10.

Un algorithme possible pour mener ce calcul est de procéder par divisions successives par 10 de cette valeur n, et de compter le nombre de divisions nécessaires pour obtenir une valeur inférieure à 10.

Nous avons à faire à un procédé de même nature que le comptage du nombre d'étapes de l'algorithme de recherche dichotomique.

Décidons de définir cela comme une fonction que l'on nomme logarithme, disons *fonction logarithme entière de base 10* pour préciser que la valeur de la fonction est entière et que les divisions considérées sont des divisions par 10.

Poursuivons avec ce problème et proposons de définir des fonctions Python pour calculer le nombre de chiffres d'une valeur entière donnée.

Une première version calque le comptage du nombre de division par 10 :

```
def nbc_div (n) :
    """Nombre de chiffres d'un entier (base 10) donné
    par comptage des divisions successives.
    """
    nbc = 1
    while n >= 10 :
        nbc = nbc + 1
        n = n // 10
    return nbc
```

On peut réaliser quelques tests :

```
>>> nbc_div(0)
1
>>> nbc_div(1)
1
>>> nbc_div(9)
1
>>> nbc_div(1000)
4
>>> nbc_div(9999)
4
>>> nbc_div(10**100)
101
```

Une autre approche est de considérer que le nombre de chiffre d'un entier plus grand que 9 est de un supérieur au nombre de chiffres de cet entier divisé par 10.

On en vient à définir une nouvelle fonction Python :

```
def nbc_rec(n) :
    """Nombre de chiffres d'un entier (base 10) donné,
    version récursive.
    """
    if n < 10 :
```

```
        return 1
    else :
        return 1 + nbc_rec(n // 10)
```

On peut alors annoncer que ce que l'on a nommé "logarithme" est un ensemble de fonctions mathématiques bien connues, parmi lesquelles nous allons nous intéresser à la fonction "logarithme décimal". Cette fonction notée log ou $\log_{10}$ est disponible dans la bibliothèque `math` de Python sous le nom `log10()`.

Et que le calcul du nombre de chiffres, notre fonction logarithme entière de base 10, est équivalente à $1 + \lfloor \log_{10} \rfloor$.

Ce que l'on peut vérifier en proposant

```
def nbc_log (n) :
    """Nombre de chiffres d'un entier (base 10) donné,
    version formule '1 + partie entière de log base 10'."""
    log = log10(n)
    loge = floor(log)
    nbc = 1 + loge
    return nbc
```

Une fonction de comparaison des trois versions peut être proposée :

```
def compare_nbc(n):
    """Compare les différentes fonctions nombre de chiffres sur la valeur n."""
    return nbc_div(n) == nbc_rec(n) == nbc_log(n)
```

à partir de quoi des tests systématiques (sur des valeurs de `n` contenu dans un intervalle ou dans une liste par exemple) ou interactifs (sur des valeurs de `n` lues au clavier) proposés.

Un prolongement peut proposer une version pour le calcul de la fonction "logarithme entière de base 2".

Globalement, cet exercice propose le calcul de fonctions (complexité de la recherche dichotomique, ou nombre de chiffres) par des algorithmes pouvant être mis en œuvre simplement en Python, introduit les logarithmes comme la fonction mathématique associée à ces calculs, et amène à des définitions concrètes de cette notion qui pourront être complétées par d'autres aspects.

Cet exercice propose donc de s'appuyer sur quelques savoir-faire des élèves en informatique pour motiver la découverte des logarithmes, notion de mathématiques des plus utiles en informatique, que ce soit lors de l'étude de la complexité des algorithmes, ou de la représentation des données.

Calculs (rapides) d'élévation à la puissance

Le calcul "naïf" de 5^{13} soit $5 \times 5 \times 5 \times 5 ... \times 5$ nécessite 12 multiplications.

Un possible calcul "rapide" de cette élévation à la puissance considère :

- le codage en base 2 de 13 : $13 = 1101b$
- soit $13 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$
- et donc $5^{13} = 5^{(2^3+2^2+2^0)}$
  ou $5^{13} = 5^{2^3} \times 5^{2^2} \times 5^{2^0}$

On réalise alors les calculs suivants :

	–	5^1
$5^1 \times 5^1 = 5^2$	–	—
$5^2 \times 5^2 = 5^4$	–	$\times 5^4$
$5^4 \times 5^4 = 5^8$	–	$\times 5^8$

soit 3 multiplications pour le calcul des puissances successives de 5, et 3 (potentiellement 4) multiplications pour le calcul des produits de ces puissances de 5.

Ces deux méthodes peuvent être programmées :

```
def pow_product(i,n):
    """calcul de la puissance comme un produit"""
    p = 1                           # le produit
    for _ in range(0,n):
        p *= i
    return p

def pow_fast(i, n):
    j = 0                           # la puissance courante
    ip2j = i                        # i puissance 2 puissance j
    res = 1                         # le résultat
    while (n >> j) != 0:
        if (n >> j) % 2 == 1:       # j-ieme bit de n à 1 ?
            res = res * ip2j
        ip2j = ip2j * ip2j
        j = j+1
    return res
```

Et les valeurs retournées comparées et validées sur un exemple (** est l'opérateur d'élévation à la puissance de Python) :

```
>>> pow_product(5,13)
1220703125
>>> pow_fast(5,13)
1220703125
>>> 5**13
1220703125
```

Au dela du résultat, les coûts des deux méthodes peuvent être comparés. Pour la méthode “naïve” on compte $n-1$ multiplications.

Pour la méthode “rapide”, on observe que la boucle **while** comporte autant d'itérations que l'écriture en base 2 de n comporte de chiffres. On retrouve notre logarithme entier base 2. Pour chaque itération on effectue au plus deux multiplications et quelques autres opérations. L'ordre de grandeur du nombre d'opérations, la complexité de la fonction, varie donc comme le logarithmique (entier base 2) de n.

La performance des différentes méthodes de calcul peut être interrogée.

```
from time import process_time

def duration2 (f, x, y) :
    """Le temps d'exécution d'une fonction à deux paramètres."""
    start = process_time()
    f(x, y)
    duration = process_time() - start
    return duration

def pow_python(i, n) :
    """élévation à la puissance directement avec l'opérateur Python"""
    return i**n
```

```
a = 92982829
b = 50000

print(" pow_product() :", duration2(pow_product, a, b))
print(" pow_fast()    :", duration2(pow_fast, a, b))
print(" pow_python()  :", duration2(pow_python, a, b))
```

Une exécution donne

```
 pow_product() : 1.79841
 pow_fast()    : 0.255379
 pow_python()  : 0.093863
```

La version “rapide” est bien plus efficace que la version “naïve”, mais l'opérateur Python d'élévation à la puissance est encore plus rapide !

Une autre méthode dite d'“Exponentiation rapide” [3] repose sur le principe suivant de calcul de i^n

- si n est pair alors $i^n = (i^{n/2})^2$, on se ramène au calcul d'une puissance $n/2$-ième et d'un carré [4] ;
- si n est impair et supérieur à 1, alors $i^n = i \cdot (i^{(n-1)/2})^2$, on se ramène au calcul d'une puissance $(n-1)/2$-ième, d'un carré et d'un produit.

Le calcul de i^n se ramène systématiquement au calcul d'une puissance de degré inférieur. L'algorithme récursif de calcul de $p(i, n)$, puissance n-ième de i, s'écrit donc :

$$\begin{cases} 1 & \text{si } n \text{ est égal à } 0, \\ (i^{n/2})^2, & \text{si } n \text{ est pair,} \\ i \cdot (i^{(n-1)/2})^2 & \text{si } n \text{ est impair.} \end{cases}$$

chaque élévation à la puissance de i étant elle-même calculée à l'aide de cet algorithme (récursif), on écrit que $p(i, n)$ se calcule comme :

$$\begin{cases} 1 & \text{si } n \text{ est égal à } 0, \\ p(i, n/2)^2, & \text{si } n \text{ est pair,} \\ i \cdot p(i, (n-1)/2)^2 & \text{si } n \text{ est impair.} \end{cases}$$

Une traduction immédiate en Python permet de définir la fonction `pow_sqrt()` :

```
def pow_sqrt(i, n):
    if n == 0:                      # n nul, pair ou impair ?
        return 1
    elif n%2 == 0:
        return pow_sqrt(i, n//2)**2
    else:
        return i * pow_sqrt(i, (n-1)//2)**2
```

On peut en mesurer la complexité, nombre d'opérations. Il y aura autant d'appels à la fonction `pow_sqrt()` que l'on peut diviser n par deux, on retrouve notre logarithme. Chaque exécution de la fonction consiste en une élévation au carré, et en une éventuelle multiplication.

L'odre de grandeur du nombre d'opérations, la complexité, est donc logarithmique en n.

Prenons quelques mesures de temps pour comparer nos trois versions et l'opérateur Python :

3. voir par exemple sur Wikipedia fr.wikipedia.org/wiki/Exponentiation_rapide, ou la version anglaise plus complète en.wikipedia.org/wiki/Exponentiation_by_squaring.

4. le lecteur intéressé pourra mener la comparaison avec la transformation en $i^n = (i^2)^{n/2}$ jusqu'à en confronter les temps d'exécution des implementations Python...

```python
def print_pow_durations(a, b) :
    """Affiche les temps d'exécution des différentes versions de a^b."""
    print(a, "^", b)
    print(" pow_product() :", duration2(pow_product, a, b))
    print(" pow_fast()    :", duration2(pow_fast, a, b))
    print(" pow_sqrt()    :", duration2(pow_sqrt, a, b))
    print(" pow_python()  :", duration2(pow_python, a, b))
    print()

print_pow_durations(92,          50)
print_pow_durations(92982829, 50000)
print_pow_durations(9298,     99997)
print_pow_durations(9298,    200018)
```

Une exécution donne :

```
92 ^ 50
 pow_product() : 1.7999999999990246e-05
 pow_fast()    : 1.0000000000010001e-05
 pow_sqrt()    : 1.1000000000011001e-05
 pow_python()  : 3.0000000000030003e-06

92982829 ^ 50000
 pow_product() : 1.9254399999999996
 pow_fast()    : 0.262448
 pow_sqrt()    : 0.09677900000000017
 pow_python()  : 0.0951679999999997

9298 ^ 99997
 pow_product() : 3.69949
 pow_fast()    : 0.2818990000000001
 pow_sqrt()    : 0.1037069999999991
 pow_python()  : 0.10533500000000018

9298 ^ 200018
 pow_product() : 10.180958
 pow_fast()    : 0.774832
 pow_sqrt()    : 0.28559900000000127
 pow_python()  : 0.2977869999999996
```

Nous pouvons faire quelques observations rapides. Une campagne de tests plus importante serait bien entendu nécessaire pour analyser finement et prudemment ces données.

Première observation, les performances de notre meilleure implémentation sont identiques à celles de l'opérateur Python, sauf pour les "petites" valeurs.
Deuxième observation, c'est bien la valeur de l'exposant qui influence le plus les performances.
Troisième observation, le temps de calcul de `pow_product()` croît grossièrement comme n.
Quatrième observation, les performances des deux fonctions de complexité logarithmiques semblent liées par un facteur multiplicatif constant.

Cet exercice propose donc de mener des transformations algébriques d'exponentiations. Ces transformations sont guidées par la mise en œuvre de l'évaluation de ces exponentiations, opération naturelle en informatique. En effet, en informatique, le traitement habituellement réalisé sur les expressions est de les "évaluer", de les réduire à une valeur.

Cet exercice illustre que différents algorithmes peuvent être proposés pour mener un même calcul, pour résoudre un même problème, que ces algorithmes peuvent être comparés. C'est un point essentiel en algorithmique.

La comparaison peut se faire en terme de complexité (dont le calcul nous ramène ici par deux fois à notre fonction logarithme), ou en terme de mesure du temps de calcul d'une exécution sur sur machine, on parle de performance. Enfin, la complexité et la performance peuvent être confrontées.

Programmation Python, polynôme

Plusieurs méthodes, algorithmes, peuvent être mis en œuvre pour l'évaluation d'un polynôme en une valeur donnée.

La réalisation informatique passe par une phase préalable de représentation du polynôme. Il s'agit de mémoriser les coefficients et les puissances associées. De multiples représentations sont possibles. Un choix simple peut être de mémoriser les coefficients dans une liste, l'index du coefficient dans la liste correspondant à la puissance. Le coefficient constant est donc l'élément 0 de la liste, le coefficient de plus haut degré, le dernier élément de la liste. Par exemple, le polynôme $5x^4 + 12x^2 + 3x + 2$ sera représenté par la liste `[2, 3, 12, 0, 5]`. On notera la présence du zéro qui indique l'absence de monôme de degré 3 dans cet exemple.

Un calcul de la valeur d'un polynôme en une valeur donnée x se fait "naïvement" en calculant chacune des puissances de x, multipliant celle-ci par son coefficient, et faisant la somme de ces produits.

Une fonction bâtie sur ce principe peut être :

```
def val(p, x):
    """Valeur d'un polynôme p en une valeur x donnée"""
    px = 0                                  # le résultat, p en x
    n = 0                                   # les puissances successives
    for coeff in p:
        px += coeff * x**n
        n += 1
    return px
```

Dans un shell Python [5] nous pouvons utiliser cette fonction pour évaluer notre polynôme en 7 :

```
>>> poly = [2, 3, 12, 0, 5]
>>> v = 7
>>> val(poly, v)
12616
```

Chacune des puissances de x étant utiles, nous pouvons les calculer progressivement par multiplication successives, et proposer une nouvelle fonction, optimisant la précédente :

```
def valopt(p, x):
    px  = 0
    n   = 0                                 # les puissances successives
    xpn = 1                                 # les x**n successifs
    for coeff in p:
        px  += coeff * xpn
        n   += 1
        xpn *= x
    return px
```

5. on parle aussi de console, boucle interactive, *read-eval-print-loop*, en.wikipedia.org/wiki/Read–eval–print_loop

Un polynôme comme $2 + 3x + 12x^2 + 5x^4$ peut être réécrit $2 + x(3 + 12x + 5x^3)$, lui même réécrit $2 + x(3 + x(12 + 5x^2))$. En poursuivant ainsi, on obtient finalement $2 + x(3 + x(12 + x(x(5))$. Cette réécriture est à la base de la méthode dite de Horner d'évaluation d'un polynôme [6].

Une implémentation pour mener à bien ce calcul peut partir du coefficient du plus grand degrés, qu'il convient de multiplier par la valeur de x, auquel on ajout le coefficient de degrés inférieur, le tout est multiplié par x, etc jusqu'à ajouter le coefficient constant :

```
def horner(p, x):
    px = 0
    for coeff in reversed(p):
        px = coeff + x*px
    return px
```

On remarquera le `reversed()` qui permet de parcourir les coefficients à partir du plus grand.

Une autre implémentation possible de la méthode de Horner considère que l'évaluation du polynôme $2 + 3x + 12x^2 + 5x^4$ réécrit $2 + x(3 + 12x + 5x^3)$, peut être obtenue par le calcul de $2 + 3P$, P étant l'évaluation en x du polynôme facilement obtenu à partir du polynôme initial en oubliant le coefficient constant, et en abaissant le degré des monômes restants.

Avec notre représentation par une liste de coefficients, la représentation de ce polynôme P correspond simplement à la liste privée de son premier élément. On écrit `p[1:]` en Python.

La valeur d'un polynôme constant étant égale à la valeur de la constante, on en vient à une version récursive de l'évaluation d'un polynôme par la méthode de Horner :

```
def hornerr(p, x):
    if len(p) == 1:
        return p[0]
    return p[0] + x * hornerr(p[1:], x)
```

Les complexités, c'est-à-dire nombre de multiplications, additions et élévations à la puissance de ces différents algorithmes peuvent être calculées et comparées. En intégrant les résultats précédents de complexité de l'opération d'élévation à la puissance, on pourra se ramener à un calcul du nombre de multiplications ou additions.

La mesure des temps de calcul sur quelques polynômes des différentes implementations pourra donner lieu à des comparaisons avec ces complexités.

Cet exercice dans la même veine que le précédent, illustre la manière dont des transformations algébriques permettent de mener effectivement des calculs en vue de l'évaluation d'expressions. Il illustre donc que plusieurs algorithmes peuvent être proposés pour résoudre un même problème, que leur complexité et performance peuvent être comparées.

Au delà de ces éléments déjà mentionnés, cet exercice apporte un éclairage sur une élément essentiel de l'informatique : pour manipuler des "objets", que ce soit de la vie réelle ou des objets mathématiques, il faut en passer par une représentation de ces "objets". Ici les polynômes sont représentés par des listes de coefficients. C'est sur ces représentations que les programmes opèrent.

De plus, ces représentations résultent de choix arbitraires que nous faisons. Pour notre exemple, des alternatives à la représentation choisie sont envisageables. Une liste de couples [7] coefficients / degrés peut être un bon choix pour des polynômes dont de nombreux coefficients sont nuls.

Le fait d'en passer par une représentation informatique des polynômes, de pouvoir nommer des variables qui représentent ces polynômes, de pouvoir les passer en paramètre à des fonctions, permet aussi, du point de vue mathématiques, de questionner la nature même des polynômes.

6. voir par exemple la page Wikipedia fr.wikipedia.org/wiki/Méthode_de_Ruffini-Horner
7. un couple pouvant être représenté en Python par une liste ou un tuple de 2 éléments.

Les polynômes sont des objets mathématiques, que l'on peut manipuler. Nous les avons simplement évaluer en un point, ils peuvent par exemple être dérivés. Une prolongation possible est alors de proposer la définition d'une fonction (informatique) qui renvoie le polynôme dérivé d'un polynôme donné en paramètre.

5 Conclusion

La contribution du groupe de travail des sociétés savantes de mathématiques et d'informatique d'éléments pour un programme de mathématiques au lycée, en lien avec l'informatique a été résumée ici.

Cette contribution s'envisageait dans le contexte d'un enseignement de mathématiques au lycée qui s'appuierait aussi sur un enseignement autonome de l'informatique.

Avec l'introduction d'une discipline "Sciences numériques et technologie" dans le tronc commun du lycée, ce contexte sera celui du futur lycée.

Le contenu des programmes de mathématiques va donc pouvoir évoluer en ce sens, abordant d'autres notions que les programmes actuels – par exemple des notions en lien avec l'informatique, mais aussi d'autres approches de l'enseignement – par exemple des approches reposant sur des mise en œuvre informatiques.