PROGRAMMER AVEC DES NOMBRES : DES ENSEMBLES MATHEMATIQUES AUX TYPES INFORMATIQUES, IMPACT DU CODAGE SUR LA CONCEPTION DES ALGORITHMES

Christophe Declercq, Simon Modeste

Résumé. Les propositions d'aménagement du programme de seconde suite à la réforme du cycle 4 introduisent dans le programme de mathématiques une partie "algorithme et programmation" - qui préfigure certainement d'autres évolutions des programmes du lycée. Elle comprend l'utilisation d'un "langage de programmation simple d'usage à choisir parmi les langages interprétés, concis, largement répandus, et pouvant fonctionner dans une diversité d'environnements". Ce choix - qui semble converge vers le langage Python, Javascript - induit un choix d'un système de représentation des nombres influant la manière de concevoir les algorithmes de calcul. Entiers bornés ou à précision arbitraire, rationnels, flottants selon la norme IEEE 754, les ensembles de nombres - types - manipulés en programmation sont parfois éloignés des ensembles définis en mathématiques. Cet atelier a permis de débattre des enjeux épistémologiques et didactiques : Quelles différences dans la nature des nombres et leur traitement en informatique et mathématiques ? Quelles organisations des savoirs sont possibles selon les choix de langages de programmation et quelles connaissances sont alors en jeu ? Quelles situations didactiques pour aborder ces enjeux nouveaux ?

Sur la représentation des objets en informatique

Pour représenter les objets en informatique, il est nécessaire de passer par des "codages" finis mais pas nécessairement bornés. Cela permet de proposer des représentation des objets mathématiques tels ques les ensembles de nombres.

Ici, nous nous intéressons à deux "types" courants en informatique : les entiers - "int" en Python dont la taille est arbitrairement grande - et les nombres à virgule flottante - "float" en Python codés selon la norme IEEE 754 sur 64 bits.

Le fait de disposer de manière native en Python d'entiers de taille arbitraire, fait du langage un instrument adapté au calcul arithmétique, en évitant le double écueil du dépassement de capacité usuel avec les langages implémentant un type entier ou entier long sur 32 ou 64 bits et du passage implicite en notation scientifique, mécanisme habituel des calculatrices, qui occasionne une perte de précision pour les "grands entiers".

Problème 1 - Traitement et représentation des entiers

On cherche à calculer le nombre de 0 à droite de n!

Soit n un entier. Dans la représentation en base 10 de n!, combien de zéros consécutifs se trouvent à droite (en fonction de n) ? On appellera ce nombre z(n). La première activité de l'atelier a consisté à étudier z(n) et son comportement à la main, avec une calculatrice, et avec Python.

Les premiers calculs à la main voient apparaître le premier zéro pour 5! = 120, puis le 2eme pour 10! = 3628800, le troisième pour 15! = 1307674368000 et le quatrième pour 20! = 2432902008176640000. Une conjecture hative en déduirait une variation de Z(n) de l'ordre de n/5. Les calculatrices ne permettent pas d'aller au delà, les nombres entiers étant ensuite automatiquement convertis en notation scientifique sans possibilité de savoir si le nombre est exact ou approché.

La mise en oeuvre en Python permet de dépasser cette frontière et en fait un outil d'expérimentation pour les mathématiques.

Une première mise en oeuvre de la fonction factorielle peut être :

```
def fact(n):
    """Calcule n!"""
    if n == 0 :
        return(1)
    else :
        return(n * fact(n-1))
```

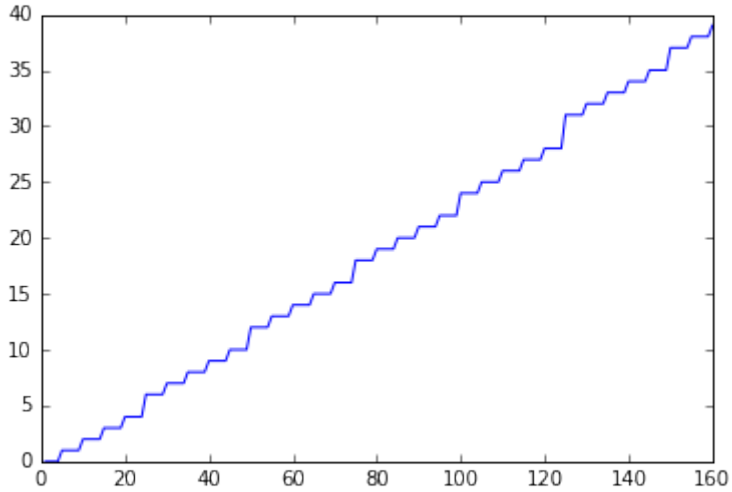
Ce qui permet alors de calculer : 100 ! =

```
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536979208272237582511852109168640000000000000000000000
```

avec 24 zéros, ce qui infirme la 1ere conjecture. Pour une étude plus approfondie, on instrumente le calcul en définissant une fonction calculant le nombre de 0 - pour ne pas devoir le compter à la main - puis on dessine le graphique représentant la variation de Z(n).

On itère alors le calcul des n! en retenant le nombre de 0, pour en dessiner le graphique.

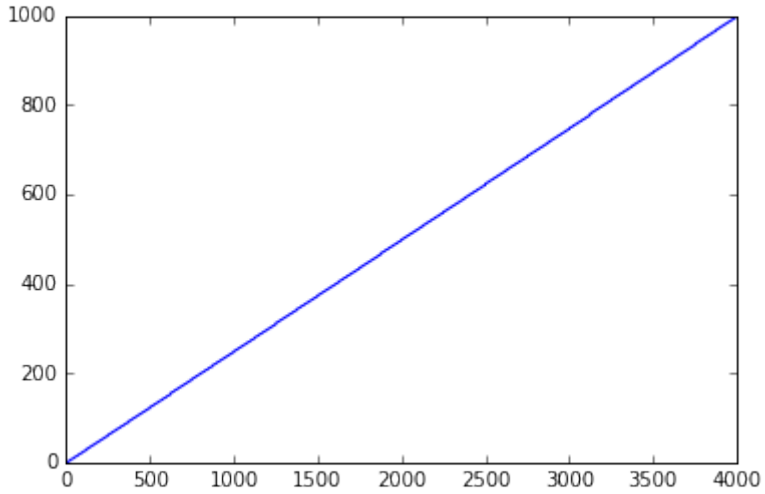
```
import matplotlib.pyplot as plot
  def plotnb0fact(n):
      f = 1
      X,Y = [],[]
      for i in range(1,n+1):
          f = f * i
          nb = nb0(f)
          X.append(i)
          Y.append(nb)
      plot.plot(X,Y)
      plot.show()
  plotnb0fact(160)
```



A cette échelle, on observe le "gain" d'un zéro pour chaque multiple de 5, de 2 zéros pour les multiples de 25, puis de 3 zéros pour 125... C'est effectivement ce que l'on pourrait démontrer de manière formelle. Le nombre de 0 est donné par la formule : $z(n)=\Sigma_{i=1}^{k} PartieEntiere(n/5^i)$ avec k le plus grand entier tel que $5^k \leq n$.

En réitérant le calcul pour des valeurs de n beaucoup plus grandes, on obtient une courbe d'une allure qui semble quasi-linéaire.

On peut en fait montrer que la suite $z(n)$ calculant le nombre de 0 à droite dans la représentation base 10 de n! est équivalente à la suite $n/4$, dans le sens où le rapport des 2 suites tend vers 1 quand n tend vers $+\infty$. La démontration utilise un encadrement de $z(n)$ et le calcul de la somme de la série géométrique $\Sigma^{n}_{i=1} 1/5^{i}$ qui tend vers $1/4$ quand n tend vers $+\infty$.



En conclusion on a pu dire que l'ensemble des nombres entiers disponibles en Python, de par sa réelle proximité avec $\mathbb{Z}$ permet donc d'en faire un instrument pertinent pour l'expérimentation mathématique.

Problème 2 - Limite d'une série réelle : somme des inverses des puissances de p

Soient p et n deux entiers. Quelle est la limite lorsque $n \to \infty$ (si elle existe) de la série $S_p(n) = \sum_{k=1}^{n} 1/p^k$

La deuxième activité de l'atelier a consisté à étudier la limite de $S_p(n)$ à la main, avec une calculatrice, et avec Python.

La calcul à la main permet de montrer que la limite de $S_p(n)$ est : $1/(p-1)$.

L'activité avec Python a permis de mettre en évidence deux phénomènes : une meilleure précision du résultat quand p est une puissance de 2, et quand les termes de la somme sont calculés du plus petit au plus grand.

Pour comprendre, il suffit d'examiner en détail le codage des flottants selon la norme IEEE 754. Un nombre flottant est représenté par : son signe **s** sur 1 bit, sa mantisse **m** sur 52 bits en binaire normalisée sous la forme $1.b_1b_2...b_{52}$ où chaque b_i vaut 0 ou 1 (le premier 1 n'est en fait pas représenté), son exposant **e** en puissance de 2 représenté sur 11 bits en binaire (exposants -1022 à 1023). Le nombre flottant représenté est : $s.m.2^e$.

La meilleure précision obtenue en commençant le calcul par la fin est évidente quand on sait que les flottants sont représentés avec une précision relative de 2^{-52} soit environ $2.\ 10^{-16}$. Il vaut mieux ajouter des termes d'ordre de grandeur comparable entre eux pour limiter les approximations.

Pour expliquer la meilleure précision pour p puissance de 2, il suffit de remarquer que seuls sont codés de manière exacte en flottant, les nombres dont la mantisse est un multiple de 2^{-52}.

Nombre flottant	Mantisse en codage base 2 à virgule

0.25	0.01000000000000000000000000000000000000000000000000000000
0.2	0.00110011001100110011001100110011001100110011001100110011

Le nombre 1/4 peut être représenté de manière exacte en base 2, mais pas le nombre 1/5 dont la représentation est cyclique et donc nécessairement tronquée.

Discussion collective

En conclusion de l'atelier, la discussion a permis d'échanger sur les enjeux mathématiques et informatiques et leur articulation. Les ensembles mathématiques et les types en informatique sont loin d'être alignés : hormis le cas particulier du type entier en Python - qui permet réellement d'expérimenter sur les entiers relatifs, les types de nombres flottants présents quels que soient les langages n'ont rien à voir avec l'ensemble des nombres réels défini en mathématiques.

Les questions didactiques pour la classe sont de proposer soit des situations évitant les difficultés, soit d'éclairer les enjeux du calcul approché en explicitant les approximations effectuées par le codage.

Ces enjeux doivent, dans tous les cas, être explicités dans la formation des enseignants, dans un contexte où les enseignants de mathématiques ne pourront plus se contenter d'algorithmes abstraits énoncés au tableau, mais devront accompagner leurs élèves dans la mise en oeuvre de programmes manipulant les nombres tels qu'ils existent dans les langages de programmation.