Découverte de la divisibilité par les programmes de calcul

Les programmes de calcul, appliqués à des entiers, peuvent aider à conjecturer qu'un nombre de la forme 3n+2 a pour reste 2 quand on le divise par 3. Cela peut servir à découvrir ou expliquer les critères de divisibilité par 2 ou 5, et à introduire le reste euclidien. Cela sert également à démontrer l'irrationnalité de $\sqrt{2}$ par séparation des cas.

Les exemples de cet article sont donnés en Sophus, langage choisi pour sa concision et son caractère « naturel ». On peut les tester avec un interpréteur en ligne.

I/ Recette pour fabriquer des entiers de reste donné

1. Nombres pairs

Si on double un entier naturel, le résultat est pair :

```
n = nouvelle Variable 15
doubler n
montrer n
```

L'enrobage classique du magicien fonctionne bien ici : Le magicien ne peut pas facilement deviner le nombre choisi, mais il peut deviner la parité du résultat du calcul. L'exercice, répété avec d'autres valeurs entières de n, permet de mettre en œuvre le fameux raisonnement inductif, en favorisant l'émission de conjecture. Pour la faciliter, on peut tester cette variante :

```
n = nouvelle Variable 15
doubler n
montrer n.estPair( )
```

Pour démontrer le résultat conjecturé, on a besoin de la définition de l'adjectif « pair »… Se pose ensuite la question de l'obtention d'un nombre impair :

2. Nombres impairs

La suite est un exercice d'algorithmique : Que doit changer le magicien pour être certain d'avoir un nombre impair ? On peut comparer les versions choisies par les élèves, mais en voici une qui fonctionne :

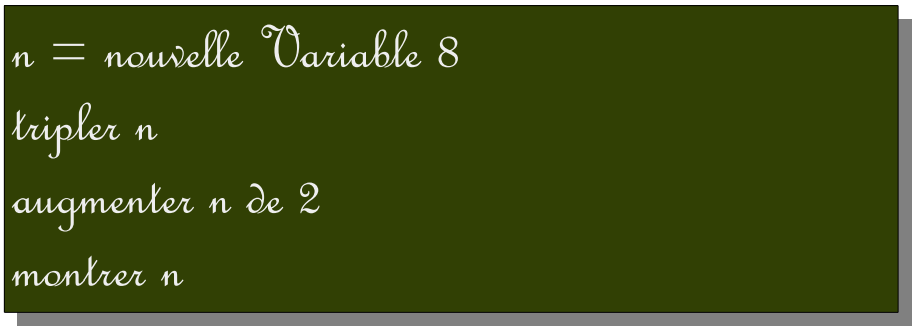
```
n = nouvelle Variable 8
doubler n
augmenter n de 1
montrer n.estImpair( )
```

On retiendra pour la fin que si n est un entier, 2n est pair et 2n+1 est impair, et même que ce sont des formes caractéristiques de ces entiers :

- Un entier pair est un entier de la forme 2n (par définition!)
- Un entier impair est de la forme 2n+1 (exercice : Que vaut n[1] pour l'entier impair 15 ?)

3. Division euclidienne par 3

L'activité peut être refaite avec 3 sauf que si un nombre n'est pas divisible par 3, il y a deux restes euclidiens possibles : 1 et 2 ; par exemple



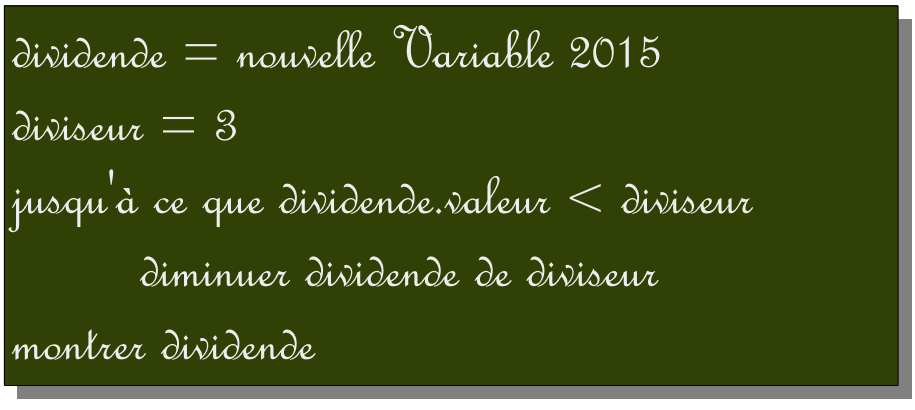
On retient alors que

- Si n est entier, 3n est divisible par 3 (et a donc 0 pour reste dans la division par 3) ;
- 3n+1 a pour reste 1 dans la division par 3 ;
- 3n+2 (et 3n-1) a pour reste 2.

C'est particulièrement avec 10 que la situation est intéressante, parce que le dernier chiffre saute aux yeux.

4. Propriété archimédienne

Pour connaître le reste d'une division euclidienne, il n'y a pas besoin d'effectuer une division, il suffit de soustraire le dividende au diviseur jusqu'à ce que la soustraction ne soit plus possible, c'est-à-dire jusqu'à ce que le reste soit inférieur au diviseur. Par exemple, pour savoir quel est le reste de la division de 2015 par 3, on peut faire



Cet algorithme de calcul de reste euclidien peut être justifié par la propriété ci-dessus : Que la réponse soit 2 veut dire que 2015=3n+2, et dans ce cas, 2015-3=3(n-1)+2 : On ne change pas la valeur du reste en soustrayant le diviseur au dividende[2].

1 Le quotient euclidien de 15 par 2 apparaît discrètement ici...

2 Cette propriété s'appelle un invariant de boucle, c'est avec un invariant de boucle qu'on démontre qu'un algorithme

La démonstration ci-dessus suggère une manière de calculer le quotient euclidien :

```
dividende = nouvelle Variable 2015
quotient = nouvelle Variable 0
diviseur = 3
jusqu'à ce que dividende.valeur < diviseur
        diminuer dividende de diviseur
        augmenter quotient de 1
montrer quotient
```

(c'est le nombre de fois qu'il a fallu effectuer la soustraction).

II/ La division euclidienne

1. Quotient

Les descriptions vues précédemment permettent de donner une définition :

Quand un nombre s'écrit sous la forme 7n+3, on dit que n est le quotient euclidien de ce nombre par 7, et 3 est le reste dans la même division euclidienne.

Voici un algorithme de calcul du quotient euclidien :

```
quotient = nouvelle Variable 2015
diviseur = nouvelle Variable 3
diviser quotient par diviseur
tronquer quotient
montrer quotient
```

On peut en effet définir le quotient euclidien comme troncature du quotient exact. Et faire la remarque que si le dividende est divisible par le diviseur, le quotient est exact.

2. Reste

Les activités du début tendent à donner plus d'importance au reste qu'au quotient : Cela justifie une notation issue de la programmation et assez courte, ainsi qu'une préparation aux algorithmes de calcul de PGCD. Cette notation étrange venue d'ailleurs est le symbole « % » :

fait vraiment ce qu'on attend de lui.

```
dividende = 2015
diviseur = 3
montrer dividende%diviseur
```

3. Divisibilité

La notion de reste euclidien est moins fondamentale que celle de divisibilité (« est divisible par » étant synonyme de « est un multiple de »). Mais on peut aussi définir la divisibilité à partir du reste euclidien :

*Lorsque le reste est nul, on dit que le dividende est **divisible** par le diviseur.*

Cette définition est pratique pour les calculs. Par exemple, comme on a vu que le dernier chiffre d'un entier est son reste euclidien dans la division par 10, on voit immédiatement le critère de divisibilité par 10 : Le dernier chiffre doit être nul.

4. Algorithmes

En imitant ce qui a été vu ci-dessus, on peut montrer que soustraire 10 ne change ni la parité ni la divisibilité par 5, ce qui fournit les critères de divisibilité par 2 et par 5. Pour la divisibilité par 3, on peut se contenter d'admettre[3] le résultat au travers d'expériences comme

```
pour c dans [0..9]
    pour d dans [0..9]
        pour u  dans [0..9]
            Si (c+d+u)%3 != (100*c+10*d+u)%3
                alert "Y'a un bug!"
```

(le fait qu'il ne se passe rien veut dire que le test échoue toujours, donc que la somme des chiffres n'a jamais un reste autre que le nombre[4]). Une fois ce résultat admis, le nombre est divisible par 3 si et seulement si la somme de ses chiffres est divisible par 3.

En résumé :

- Pour savoir si un nombre est pair, on regarde son dernier chiffre (les chiffres pairs ne sont pas nombreux) ;
- Pour savoir si un nombre est divisible par 3, on regarde la somme de ses chiffres (elle doit être divisible par 3 ; on peut recommencer récursivement l'opération) ;
- Pour savoir si un nombre est divisible par 5, on regarde son dernier chiffre (peu de chiffres sont divisibles par 5).

Or ces phrases ne sont ni plus ni moins que des ***algorithmes***. Voir ce mot assez tôt dans la scolarité ne peut qu'aider à l'apprivoiser.

3 On peut faire la démonstration sur des exemples comme par exemple pour prouver que 47 a le même reste dans la division par 3 que 4+7, on écrit 47 sous la forme 4×10+7 puis 10 sous la forme 3×3+1 et on développe les produits.

4 On peut préférer l'usage d'un tableur qui évite d'avoir à effectuer des tests.

III/ Le plus grand commun diviseur

1. Diviseurs d'un entier

Comme un diviseur d'un entier ne peut pas être plus grand que l'entier, on peut, contrairement aux multiples d'un entier, faire la liste exhaustive de ses diviseurs :

```
montrer (d pour d dans [2..24] quand 24%d == 0)
```

Ce script d'une ligne calcule et affiche la liste des diviseurs de 24 : [1 , 2 , 3 , 4 , 6 , 8 , 12 , 24]. Pour la suite, il est plus pratique de généraliser ce script en une fonction qui, à un entier n, associe la liste de ses diviseurs :

```
diviseurs = (n) ->
    (d pour d dans [1..n] quand n%d == 0)
montrer diviseurs 24
```

On peut maintenant regarder à quoi ressemble la liste des diviseurs d'une puissance. Mais aussi comparer la liste des diviseurs de deux entiers donnés :

2. Diviseurs communs à deux entiers

On voit rapidement que 1 est un diviseur de tout entier : Donc si on compare les diviseurs de deux entiers distincts, il y en a qui sont communs aux deux listes. Ce qui donne l'idée de calculer la liste des diviseurs communs :

```
diviseurs = (n) ->
    (d pour d dans [1..n] quand n%d == 0)
montrer (d pour d dans diviseurs(75) quand d dans diviseurs(60))
```

Calculer plusieurs listes de ce genre (ici celle des diviseurs communs à 75 et 60) est un bon exercice, surtout si on a la curiosité de vérifier que les nombres en question divisent bien les deux nombres de départ. Et cela amène assez naturellement à cette définition :

3. PGCD

Parmi les diviseurs communs à deux nombres, il y en a un qui est plus grand que les autres : On l'appelle le ***plus grand diviseur commun*** *des deux nombres.*

Cette définition a surtout pour avantage de donner du sens à l'abréviation PGCD (ou pgcd) qui sera utilisée par la suite.

On déduit de ce qui précède un premier algorithme de calcul du pgcd : On prend le plus grand élément de la liste :

```
diviseurs = (n) ->
    (d pour d dans [1..n] quand n%d == 0)
```

```
pgcd = (a,b) ->
    liste = (d pour d dans diviseurs(a) quand d dans diviseurs(b))
    liste[liste.length-1]
montrer pgcd 24, 16
```

Mais il y a des algorithmes plus rapides parce qu'ils ne nécessitent pas de construire toute la liste :

4. Calcul du PGCD par soustraction

```
pgcd = (a,b) ->
    jusqu'à ce que b.valeur == 0
        Si a.valeur > b.valeur
            diminuer a de b
        Sinon
            diminuer b de a
    a.valeur


m = nouvelle Variable 34
n = nouvelle Variable 21
montrer pgcd m, n
```

On peut aussi travailler sans variables, directement sur les nombres, à l'aide d'affectations :

```
pgcd = (a,b) ->
    jusqu'à ce que b == 0
        Si a>b
            a devient a-b
        Sinon
            b devient b-a
    a
```

```
montrer pgcd 24, 16
```

Cet algorithme présente deux intérêts :

- Un intérêt historique parce que c'est celui utilisé par Euclide ;
- Un intérêt pratique parce qu'il n'utilise que des soustractions, et peut donc être mis en œuvre avant la maîtrise de la division euclidienne.

Mais l'algorithme utilisant les divisions euclidiennes est plus court, tant à rédiger qu'à exécuter :

5. Algorithme dit « d'Euclide »

```
pgcd = (a,b) ->
    jusqu'à ce que b == 0
        [a,b] devient [b,a%b]
    a

montrer pgcd 55, 34
```

6. Simplification d'une fraction

Puisque le pgcd du numérateur et du dénominateur divise à la fois le numérateur et le dénominateur, on peut simplifier la fraction par le pgcd. Et comme c'est le plus grand des diviseurs communs, on est assuré que la fraction obtenue est irréductible. Pour programmer cela, on peut représenter une fraction par une liste, en convenant que son premier élément est le numérateur et son second élément, le dénominateur.

```
pgcd = (a,b) ->
    jusqu'à ce que b == 0
        [a,b] devient [b,a%b]
    a
```

```
simplifier = ([n,d]) ->
    g = pgcd n, d
    [n/g,d/g]


montrer simplifier [75,60]
```

On a là le départ d'une séquence possible sur le calcul des fractions programmé en Sophus.

IV/ Calcul sur les fractions

1. Addition

L'algorithme d'addition consiste à multiplier chaque fraction en haut et en bas par le dénominateur de l'autre fraction pour que les fractions soient au même dénominateur, puis additionner les numérateurs et simplifier le résultat :

```
pgcd = (a,b) ->
    [a,b] devient [b,a%b] jusqu'à ce que b == 0
    a
simplifier = ([n,d]) ->
    g = pgcd n, d
    [n/g,d/g]
somme = ([a,b],[c,d]) ->
    simplifier [a*d+c*b,b*d]
montrer somme [1,2], [1,3]
```

2. Soustraction

L'algorithme de soustraction est très similaire à l'algorithme d'addition parce que là aussi il faut d'abord mettre les deux fractions au même dénominateur :

```
différence = ([a,b],[c,d]) ->
    simplifier [a*d-c*b,b*d]
montrer différence [1,2], [1,3]
```

3. Multiplication

Le fait qu'il n'est pas nécessaire de mettre les fractions au même dénominateur facilite la programmation de la multiplication des fractions :

```
produit = ([a,b],[c,d]) ->
    simplifier [a*c,b*d]
montrer produit [40,1], [8,5]
```

Pour des fractions de ce genre (facteur entier, petit dénominateur), on peut vérifier avec les variables Sophus :

```
a = nouvelle Variable 40
multiplier a par 8 cinquièmes
montrer a
```

La comparaison des deux programmes et le passage de l'un à l'autre est un bon exercice pour donner plus de sens aux fractions, et pour travailler le changement de cadre.

4. Division

L'inverse d'une fraction est le plus facile à calculer :

```
inverse = ([a,b]) -> [b,a]
montrer inverse [8,3]
```

Ensuite diviser par une fraction c'est multiplier par l'inverse de cette fraction :

```
pgcd = (a,b) ->
    [a,b] devient [b,a%b] jusqu'à ce que b == 0
    a
simplifier = ([n,d]) ->
    g = pgcd n, d
    [n/g,d/g]
```

```
produit = ([a,b],[c,d]) ->
    simplifier [a*c,b*d]
inverse = ([a,b]) -> [b,a]

quotient = ([a,b], [c,d]) ->
    produit [a,b], inverse [c,d]



montrer quotient [40,1], [8,5]
```

Là encore on peut utiliser les spécificités de Sophus pour tester le même exemple dans un autre cadre :

```
a = nouvelle Variable 40
diviser a par 8 cinquièmes


montrer a
```

V/ Prolongements

1. Irrationnalité de $\sqrt{2}$

L'usage de programmes de calcul permet de conjecturer la partie la plus technique de la démonstration : n^2 a la même parité que n.

Vérification dans le cas d'un entier pair :

```
n = nouvelle Variable 15
doubler n
élever n au carré
montrer n.estPair( )
```

Et dans le cas où n est impair :

```
n = nouvelle Variable 8
doubler n
incrémenter n
élever n au carré
montrer n.estImpair()
```

La démonstration utilise les produits remarquables :

- Le carré de 2n est $4n^2$ qui est divisible par 4, *a fortiori* par 2 ;
- Le carré de 2n+1 est $4n^2+4n+1=2(2n^2+2n)+1$ qui est impair puisque, si n est entier, il en est de même pour $2n^2+2n$.

La suite est moins technique mais c'est une démonstration par l'absurde : On cherche p et q tels que $\sqrt{2}=\frac{p}{q}$; si ces deux entiers existent, on peut aussi bien choisir la fraction irréductible et on fait cette supposition. Alors on a, par passage au carré, $2q^2=p^2$. Mais ceci signifie que p^2 est pair et il en est donc de même pour p, d'après ce qu'on vient de prouver. Par définition, cela signifie qu'il existe un entier n dont le double 2n=p. On peut alors remplacer p par 2n dans l'égalité ci-dessus, ce qui donne $2q^2=(2n)^2=4n^2$. En divisant les deux membres par 2, on obtient $q^2=2n^2$ et donc q^2 est pair aussi. Donc q aussi, ce qui est contradictoire avec l'hypothèse selon laquelle la fraction est irréductible.

2. Jeu de Nim

La divisibilité par 4 permet de trouver une stratégie gagnante au jeu de Nim. Du moins avec la règle suivante :

Au départ on dispose d'un tas d'objets[5]. Par exemple 21. Chaque joueur à son tour enlève 1, 2 ou 3 objets du tas. Celui qui enlève le dernier objet gagne. La stratégie gagnante consiste à s'arranger pour laisser 4 objets à l'adversaire, ce qui fait qu'après qu'il a joué, il reste 1 à 3 objets qu'on peut enlever d'un coup. Mais comment laisser 4 objets à l'adversaire ? Si on part d'un nombre d'objets divisible par 4, on joue systématiquement ce qu'il faut pour que le nombre soit toujours divisible par 4, soit :

- 3 si l'adversaire vient de jouer 1
- 2 si l'adversaire vient de jouer 2
- 1 si l'adversaire vient de jouer 3

Ainsi, le total d'objets enlevés en deux tours de jeu est 4 et cela ne change pas la divisibilité par 4 du nombre d'objets restants. Voici alors un jeu de Nim difficile à gagner (il faut pour cela connaître la stratégie gagnante) :

5 Des haricots dans le livre de Berlekamp, Conway et Guy : « winning wyas for your mathematical plays »

```
tas = nouvelle Variable 21
jusqu'à ce que tas.valeur <=0
    alert "Il y a #{tas.valeur} objets; combien veux-tu en enlever: 1, 2 ou 3?"
    réponse = nouvelle Variable 0
    jusqu'à ce que 1 <=  réponse.valeur <= 3
        entrer réponse
        arrondir réponse à 1 # unité près
    diminuer tas de réponse
    Si tas.valeur <= 0
        alert "Tu as gagné"
    Sinon
        Si tas.valeur % 4 > 0
            jeu = tas.valeur % 4
        Sinon
            jeu = 4 - réponse.valeur
        alert "J'ai enlevé #{jeu} objets du tas qui en contenait #{tas.valeur}"
        diminuer tas de jeu
        Si tas.valeur <= 0
            alert "j'ai gagné"
```