Il était une fois dans Losanges, des histoires informatiques

Jean-Marc Desbonnez

Société Belge des Professeurs de Mathématique d'expression française

2020

alors, raconte...

Préface

... Puis vint le temps de l'informatique

Au début des années 1980, l'enseignement dit « rénové » arrive dans les écoles secondaires, avec ses *cours d'option*. Mon très regretté directeur introduit l'informatique, et, en conséquence, budgétise l'achat de matériel, puisqu'il faut équiper une *salle d'info*.

Je suis à l'époque tout jeune professeur de mathématiques, le dernier arrivé et potentiellement le premier sorti [1] en cas de problèmes majeurs. Mes collègues ne se bousculent pas au portillon pour ajouter ces heures à leurs attributions... je prends. C'est le début de la grande aventure.

J'ai souvent eu l'impression que beaucoup d'enseignants étaient très réticents à propos des ordinateurs et de l'informatique. Peur de la nouveauté ? Peur du temps à consacrer ? Peur de devoir répondre « je ne sais pas » à un élève qui pose une question ? Sans doute un peu de tout cela. Je pense qu'on me considérais parfois comme *le grand sorcier de la salle d'info*.

J'ai débuté avec un *Commodore 64* (64 pour 64 Ko [2] de mémoire, dont 39 disponibles pour l'utilisateur) ; on enregistrait les programmes sur... une mini-cassette ; il y avait certes de magnifiques jeux, en couleur si on le branchait sur une télé couleur, mais aussi la possibilité de programmer en BASIC ; c'était le temps des ordinogrammes, des GOTO, des GOSUB, et de mon premier livre : Arthur Engel, *Mathématique élémentaire d'un point de vue algorithmique*, éditions CEDIC. 1979. Il y avait déjà dans la bibliographie un certain Donald Knuth, *The Art of Computer Programming*, Vol 1, 2 et 3, 1973.

Informatique rime avec Mathématique, c'est certain, mais je comprend très vite que celle-là est un outil extraordinaire pour celle-ci. Les dix dernières années de ma carrière, les cours de math se donnaient en salle d'info, puisque j'en étais, encore, quasi le seul locataire.

L'algorithmique n'est malheureusement pas au programme des cours de mathématiques en Belgique, contrairement à la France, par exemple. Pourtant, c'est un excellent apprentissage à la rigueur, ce qui, avouons-le, fait souvent défaut. Programmer c'est s'adresser à une machine, qui ne comprend rien (au sens humain du terme), et qui ne fera aucun effort pour essayer de comprendre ce que l'utilisateur a voulu écrire ; il y a une syntaxe à respecter dans les instructions ; on n'écrit pas n'importe quoi, n'importe comment, n'importe où ; la moindre erreur de frappe est fatale : le blocage est immédiat, et pas de résultat possible. C'est quelque chose de difficile à surmonter par les élèves.

Mettre l'informatique au service d'un cours de mathématiques est toujours bénéfique lorsqu'on peut illustrer l'un ou l'autre point du programme, comme un enseignant dans une autre branche passe une vidéo. Quand je pense à toutes ces choses que je n'ai pas eu la chance de connaître lors

1. En informatique, cela s'appelle une gestion de pile LIFO : Last In First Out.
2. Ce n'est pas une faute de frappe, c'est bien Ko et non Mo ni To...

de mes humanités, par exemple voir simultanément sur un écran le graphique d'une fonction et celui de sa dérivée, voir sur un écran le graphique d'un plan lorsqu'on encode une équation cartésienne, voir le résultat en quelques secondes de la simulation de dix millions de jets d'une pièce de monnaie, et la liste est longue, très longue. L'illustration informatique d'une notion mathématique peut aussi contribuer à fixer mieux la chose : voir à l'écran la valeur de *factorielle de 100 000*, et sa version papier, un document d'une centaine de pages, avec 456 574 chiffres, est un petit choc psychologique ; pourtant, on utilise souvent le vocabulaire « grand », « infini », mais personne ne s'en émeut. J'ai eu droit à une minute de silence !

Dans les cours de mathématiques à peu d'heures semaines, on pourrait penser qu'il n'y a pas le temps d'initier à l'outil informatique. C'est faux. Il ne faut que quelques heures pour prendre en mains *Géogébra*, les rudiments du tableur, et autres. Non seulement l'élève qui manipule retient mieux, mais il a alors les moyens de pousser un peu plus loin la chansonnette, s'il en éprouve le besoin, ou si on lui en donne l'envie... le bénéfice apporté en vaut la peine !

S'il est un peu mordu, le prof apporte ses propres « créations informatiques », et dans un cours « plus fort » ou une activité complémentaire, les élèves peuvent aussi être amenés à participer eux-mêmes à la création de cette illustration dans le cadre d'un exercice. Il ne s'agit pas d'un cours d'informatique, on ne cherche pas à optimiser au maximum un algorithme afin de gagner quelques secondes d'exécution. Le but est de faire au plus simple.

Et tout cela évolue très vite. Pour ne citer que trois nombres, en 40 années, la mémoire Ram de mon ordinateur est passée de 39 Ko à 40 Go... on ne s'en plaindra pas, puisque les outils à disposition pour illustrer un cours de mathématique sont de plus en plus nombreux et sophistiqués.

Ce document est ma modeste contribution à cette croisade. Il reprend tous mes articles consacrés à l'informatique, depuis Losanges 28 en mars 2015, fractals exceptés, ceux-ci ayant déjà fait l'objet d'une précédente publication [3].

L'utilisation du tableur occupe une large place. Que ce soit *OpenOffice Calc* ou *Microsoft Excel*, cette application, sans doute la plus ancienne depuis l'avènement de la micro-informatique, a quand-même pour spécialité... le calcul, ce qui est aussi le cas en mathématique.

Le plus compliqué est parfois de savoir quelle application choisir pour traiter tel ou tel aspect mathématique. Le présent document donne de nombreuses pistes.

Depuis le premier numéro en 2008, Losanges a hébergé bien d'autres articles liés à l'informatique. Citons Jean-Paul Houben et ses nombreuses interventions à propos de *Cabri-Géomètre*, Valérie Henry et Gérard Laloux à propos de *Géogébra*, Valérie Henry et les statistiques descriptives avec le tableur, Pascal Dupont qui a longtemps tenu la une avec ses articles sur LaTeX, essentiel pour écrire des mathématiques, Guy et Yolande Noël à propos de l'extraordinaire application *Apprenti Géomètre*.

Ce n'est pas l'endroit ici de parler de toutes les pierres qui ont jalonné le chemin informatique depuis le début des années 1980 à ce jour. Mais si cela peut convaincre l'un ou l'autre enseignant de sauter le pas, deux événements ont profondément marqué mon chemin ; le premier est l'abandon définitif de *Windows* au profit de *MacOs* (je sais que je ne vais pas me faire que des amis en écrivant cela, mais j'ai utilisé très longtemps *Windows* et je sais de quoi je parle), et le second l'abandon définitif de *Word* au profit de LaTeX. Mon seul regret est que ce ne soit pas arrivé plus tôt !

Jean-Marc Desbonnez

3. *Raconte-moi des histoires... de fractals*, Guy Noël et Jean-Marc Desbonnez. SBPM. 2020

La science est ce que nous comprenons suffisamment bien pour l'expliquer à un ordinateur.
L'art, c'est tout ce que nous faisons d'autre.
D. Knuth

Algorithmique élémentaire et langage Ruby

Résumé. *Les structures de base de l'algorithmique permettent d'expérimenter des propriétés, des définitions, des théorèmes mathématiques, et pourquoi pas, de conjecturer. Les instructions formant un algorithme doivent être traduites dans un langage informatique, afin d'être exécutées par un ordinateur. Mon choix s'est porté sur le langage Ruby.*

Cet article est un bref résumé du cours d'informatique (1 heure/semaine) donné à des élèves de cinquième et de sixième générale, option math6. Il est évidemment ponctué de nombreux exercices qui peuvent très vite être orientés vers le cours de mathématiques.

Le choix d'un langage de programmation simple permet d'être efficace très vite, et surtout de ne pas décourager les élèves par des difficultés informatiques d'installation et de mise en route.

Les programmes nous demandent d'utiliser calculatrices graphiques, ordinateurs, logiciels divers, pour illustrer les cours. Certaines matières s'y prêtent très bien, et nous avons à disposition des logiciels qui nous fournissent une aide précieuse, surtout lorsqu'il faut dessiner, ou faire de nombreux calculs. Toutes les écoles sont équipées d'ordinateurs, presque tous les élèves ont accès à un ordinateur à domicile (ce qu'ils en font n'est pas toujours ... mathématique, mais soyons bons princes, ils ne savent sans doute pas que cette machine peut aussi servir à autre chose ...). Soyons consommateurs de ces outils, sans modération !

Fantastique lorsqu'en quelques clics un logiciel calcule une valeur approchée de la racine de la fonction $f(x) = x^3 - x - 1$ *avec 15 décimales !*

Encore plus fantastique si on parvient à donner à un ordinateur les instructions qu'il doit exécuter pour parvenir à ce résultat. Là on met le doigt dans un engrenage qui s'appelle Algorithmique et programmation, et qui, intellectuellement parlant, permet de **structurer** *une pensée, et surtout d'apprendre la* **rigueur**, *ce qui, avouons-le, fait parfois un peu défaut.*
On « s'adresse » à une machine, et au moindre iota de travers, rien ne va plus.

1 Ruby, pourquoi ?

Faire de l'algorithmique sans langage de programmation, c'est un peu faire du solfège sans jouer d'un instrument : c'est frustrant. Il y beaucoup d'instruments de musique, et aussi beaucoup de langages de programmation.

Ruby est un langage de programmation conçu en 1993 par un informaticien japonais, Yukihiro Matsumoto. Rassurons-nous, il ne faudra pas apprendre le japonais.

Il est vrai que *Java* est pour l'instant à la mode. Mais pour écrire, par tradition, « Hello World »

sur un écran avec *Java*, il faut installer une petite usine à gaz et taper plusieurs lignes de codes assez indigestes pour un débutant.

Avec *Ruby*, il suffira de taper `puts "Hello World"`. La messe est dite. Le but visé n'est évidemment pas de former des programmeurs de haut vol, mais simplement d'écrire de petits programmes qui illustrent numériquement certaines parties du cours de mathématiques, et ce à moindre investissement.
Ruby donc, parce que

- Pour un apprentissage de base de la programmation, il est facile.
- Pour celui qui veut approfondir, il est puisant et performant, et n'a rien à envier aux langages plus connus.
- Il est gratuit (zéro euro).
- Il permet une utilisation *interactive* : on tape une instruction, et elle est immédiatement exécutée ; situation idéale pour la découverte.
- Il est facile à installer sur un ordinateur. Si vous avez la chance de travailler avec un *Mac*, c'est encore plus facile (comme d'habitude ...), *Ruby* est déjà installé ![1]

2 Ruby, comment ?

2.1 Installation et mise en place sous MacOsX

Hi hi hi. Il n'y a rien à faire, *Ruby* fait partie de l'OS.

2.2 Installation et mise en place sous Windows

Ruby peut être téléchargé à l'adresse `http://rubyinstaller.org`

- Choisir la version Ruby 1.9.3 (conseillé après la lecture des explications à droite de la boîte de dialogues).
- Exécuter le fichier *rubyinstaller ... .exe*.
- Cocher les cases *Add Ruby executables to your PATH* et *Associate .rb and .rbw files with this Ruby installation.*

2.3 Utilisation en mode interactif sous MacOsX

- Depuis le dossier Applications > Utilitaires, exécuter l'application Terminal.app
  Pour une utilisation soutenue, il est judicieux de placer l'application dans le dock où un seul clic suffit pour l'exécution.

1. Dire que depuis des années je l'avais sous la main, et je ne le savais même pas ...

- Au message d'invite, taper l'instruction *irb*[2] puis Enter .
  Et on peut taper n'importe quelle instruction, par exemple `puts "Hello World"`.

2.4 Utilisation en mode interactif sous Windows

Après l'installation, on trouve (en principe) *Ruby* dans le menu *Démarrer* ; si ce n'est pas le cas, le fichier **irb.bat** démarre le terminal de commandes *cmd.exe* avec *Ruby* en mode interpréteur.
Et on peut taper n'importe quelle instruction, par exemple `puts "Hello World"`.

2.5 Les opérations arithmétiques de base

Il y en a 6 : addition (+), soustraction (-), multiplication (*), division (/), exponentiation (**) et reste de la division entière (%).
Pas d'affront, on ne donnera pas un exemple d'addition !

14/11 ⇝ 1 (toute opération sur 2 entiers fournit un résultat entier, arrondi)
14.0/11 ⇝ 1.2727272727272727 (l'un des opérandes doit être réel pour un résultat réel)
8**9**4 ⇝ désolé, pas assez de place ...

2.6 Quelques calculs mathématiques un peu plus poussés

Fonction	Explication
`Math.sqrt(x)`	$\sqrt{x}$
`Math.cbrt(x)`	$\sqrt[3]{x}$
`Math.sin(x)`	sinus de x en radian
`Math.cos(x)`	cosinus de x en radian
`Math.tan(x)`	tangente de x en radian
`Math.exp(x)`	e^x
`Math.log(x)`	logarithme népérien de x
`Math.log10(x)`	logarithme décimal de x
`Math.log(x,a)`	logarithme en base a de x
`Math::E`	e=2.718281828459045...
`Math::PI`	π = 3.141592653589793...
`rand`	réel aléatoire (18 déc) dans [0,1[
`rand(a)`	entier aléatoire dans $[0,a[$
`rand(a..b)`	entier aléatoire dans $[a,b]$
`x.abs`	valeur absolue de x

2. Interactive Ruby

Attention : il est **impératif** de respecter la *casse* !

2.7 Ruby en mode « éditeur »

Écrire un programme ou un algorithme, c'est écrire une série d'instructions qui seront exécutées « en bloc ». Il est donc nécessaire d'utiliser un *éditeur de textes*[3]. Il y beaucoup de choix, gratuits ; opter de préférence pour un éditeur qui propose la coloration syntaxique, et la numérotation des lignes de code.

Sur les Macs des cyberclasses, il y a *TextWrangler*, donc rien à installer. Ce logiciel propose même une commande *Run in terminal* qui exécute l'algorithme en cours dans l'application *Terminal*.

Pour Windows, il y a certainement l'équivalent, chercher sur Internet (*Scite*, *Jedit*, *Wordpad2*, ...).

Principe de base : encoder les instructions *Ruby*, enregistrer le fichier **avec l'extension .rb** (nom sans espace, ni accents, ni caractères spéciaux), puis l'exécuter, et admirer les résultats ... s'il n'y a pas d'erreurs ni de frappe, ni de syntaxe, ni de raisonnement, rigueur oblige.

3 Algorithmique élémentaire et Ruby

L'algorithmique élémentaire tient en 5 mots : affectation, lecture, écriture, alternative, répétitive. Il est difficile d'en faire le tour en quelques pages, voici le minimum vital.

3.1 L'affectation

L'affectation consiste à mettre un contenu dans une variable ; le contenu peut être une donnée fixe, le résultat d'un calcul, le contenu d'une autre variable, etc.

Algorithmique	Ruby
variable ← contenu	variable = contenu

Avec l'*affectation*, c'est la personne qui écrit le programme qui « décide » du contenu de la variable (ne pas confondre avec la *lecture* où c'est l'utilisateur final qui décidera).

3.2 La lecture

La lecture est une instruction dite d'*entrée* ; elle permet de faire « entrer » une donnée de l'extérieur (l'utilisateur final, via le clavier) vers l'intérieur (une variable).

Il s'agit d'une « affectation de l'extérieur ».

La syntaxe Ruby dépend du type de contenu attendu : un nombre entier (integer), un nombre décimal (float), une chaîne de caractères (string).

3. Pas un traitement de textes, comme *celui dont on ne prononce pas le nom* ...

Algorithmique	Ruby
lire variable	variable = gets
	variable = gets.to_i
	variable = gets.to_f
	variable = gets.chomp

`gets` est à interpréter comme `get string` (recevoir une chaîne).
.to_i convertit la chaîne en entier.
.to_f convertit la chaîne en flottant.
L'instruction `gets` avale littéralement tout ce qui est tapé au clavier, y compris le code de la touche Enter qui termine un encodage au clavier. Dans le cas de caractères non numériques, **.chomp** enlève ce code pour ne garder que les caractères tapés.

Remarque importante

L'instruction de lecture **attend** que l'utilisateur encode une donnée via son clavier.

Encore faut-il qu'il sache ce qu'il doit encoder !

Il est donc indispensable que toute instruction de lecture soit précédée d'une instruction d'écriture : il faut au minimum poser une question !

3.3 L'écriture

L'écriture est une instruction dite de *sortie* ; elle permet de faire « sortir » un résultat vers l'extérieur, l'écran en ce qui nous concerne.

Algorithmique	Ruby
écrire donnée	puts ...

`puts` est à interpréter comme `put string` (mettre une chaîne). La syntaxe de l'instruction *puts* dépend du type de donnée à écrire.

instruction	effet
puts prix	affiche le contenu de la variable *prix*.
puts "blabla"	affiche le texte "blabla".
puts "blabla #{prix}"	affiche le texte "blabla" suivi du contenu de la variable *prix* (si *prix* est de type numérique).
puts "blabla" + prix	affiche le texte "blabla" suivi du contenu de la variable *prix* (si *prix* est de type texte).
puts "blabla" + prix.to_s	affiche le texte "blabla" suivi du contenu de la variable *prix* (numérique) converti en chaîne de caractère (string).

3.4 Les expressions booléennes

Une *expression booléenne*, encore appelée *condition*, est une expression dont la valeur est soit vrai (true) soit faux (false). On les retrouve dans les structures alternatives, ainsi que dans les structures répétitives.

Ruby	Explication
a == b	a est identique à b
a < b	a est strictement inférieur à b
a <= b	a est inférieur à b
a > b	a est strictement supérieur à b
a >= b	a est supérieur à b
a != b	a est différent de b
cond1 or cond2	cond1 ou cond2 (est fausse si les 2 sont fausses)
cond1 and cond2	cond1 et cond2 (est vraie si les 2 sont vraies)
not cond1	négation de cond1 (est vraie si cond1 est fausse, et vice-versa)

3.5 La structure alternative

C'est une structure qui permet de « prendre une décision » en fonction d'une *condition* qui est soit vraie soit fausse.

Première forme, alternative simple

```
si condition
   instruction(s) cas vrai
[sinon
   instruction(s) cas faux]
fin
```

```
if condition
   instruction(s) cas vrai
else
   instruction(s) cas faux
end
```

Exemple

```
lire nombre
si nombre est pair
   écrire "nombre pair"
sinon
   écrire "nombre impair"
fin
```

```
puts "encoder un nombre entier positif"
nombre = gets.to_i
if nombre %2 == 0
   puts "#{nombre} est pair"
else
   puts "#{nombre} est impair"
end
```

Remarques

- `[sinon`
  `instruction(s) cas faux]`

  est facultatif, et peut parfois ne pas exister.

- Ne pas confondre le signe = opérateur d'affectation avec le signe == opérateur de comparaison dans une expression booléenne.

Seconde forme, alternatives imbriquées

```
si cond1
   instruction(s)1
sinon si cond2
   instruction(s)2
sinon si cond3
   instruction(s)3
...
sinon
   autre(s) instruction(s)
fin
```

```
if cond1
   instruction(s)1
elsif cond2
   instruction(s)2
elsif cond3
   instruction(s)3
...
else
   autre(s) instruction(s)
end
```

Troisième forme, alternative multiple

```
selon que
   lorsque cond1 alors instr1
   lorsque cond2 alors instr2
   lorsque ...
   sinon autre(s) instruction(s)
fin
```

```
puts "que vaut le discriminant?"
d=gets.to_f
case
   when (d<0) then puts "2 sol dans R"
   when (d==0) then puts "1 sol dans R"
   else puts "2 sol dans C"
end
```

3.6 La structure répétitive, dite « boucle »

Une répétitive est une structure, qui comme son nom l'indique, permet d'exécuter plusieurs fois une ou plusieurs instructions, l'arrêt étant déterminé par une *expression booléenne*.

La règle fondamentale :

Une répétitive DOIT s'arrêter !

Il est donc indispensable que parmi les instructions qui seront répétées, une au moins permette à l'expression booléenne de devenir soit *vraie* soit *fausse*, selon le type de répétitive.

Tous les langages de programmation proposent 2 types de répétitives :

- la répétitive *automatique* : le programmeur connaît à l'avance le nombre d'itérations.
- la répétitive *non automatique* : le nombre d'itérations dépend d'une condition.

3.6.1. La répétitive automatique

```
Pour compteur de debut à fin
   instruction(s)
fin
```

```
# calcul de 200!
n=1
for k in 1..200
   n=n*k
end
puts n
```

La variable *compteur* est initialisée à la valeur *debut* ; après chaque itération elle est incrémentée de 1 ; la répétitive s'arrête dès que la variable *compteur* a dépassé la valeur *fin*.

debut et *fin* doivent être des entiers (pas des flottants), et peuvent être négatifs.

Et bien évidemment, pour que la répétitive s'arrête, *debut* $\leqslant$ *fin* !

3.6.2. La répétitive non automatique

Ruby propose plusieurs formes :

```
Tant que condition
   instruction(s)
fin
```

```
k=3
while k<=12
   puts k.to_s
   k=k+1
end
```

```
Jusqu'à ce que condition
   instruction(s)
fin
```

```
k=3
until k>12
   puts k.to_s
   k=k+1
end
```

Les 2 exemples ci-dessus sont équivalents ; remarquer que les 2 conditions sont contraires.

Dans ces 2 types de répétitives, la condition est examinée avant l'exécution des instructions ; il se peut donc que les instructions ne soient jamais exécutées. Tout dépend de ce qui se passe avant ; on peut forcer une première exécution, ce qui a été fait dans les 2 exemples précédents.

```
Répéter
   instruction(s)
tant que condition
```

```
k=3
puts k.to_s
begin
   k=k+1
   puts k.to_s
end while k<12
```

Dans ce type de répétitive, les instructions sont exécutées avant d'évaluer la condition.

Ce 3e exemple a été rendu équivalent aux 2 premiers.

Le puriste dira que tout type de répétitive peut se ramener à la forme *Tant que*. C'est un bel exercice.

4 Valeur approchée de la racine de $f(x) = x^3 - x - 1$

Une manière de faire est d'utiliser la méthode de *Newton-Raphson* qui calcule une valeur approchée d'une racine d'une fonction f à partir d'une valeur de départ a_0 et de la formule de récurrence $a_{n+1} = a_n - \frac{f(a_n)}{f'(a_n)}$.

Par rapport à ce qui a été expliqué plus haut, on a juste besoin de la notion informatique de **fonction** (appelée jadis *routine* ou *sous-programme*), qui permet de calculer un résultat, à partir d'un ou plusieurs paramètres (ici l'argument de la fonction f et de sa dérivée f'), et de renvoyer ce résultat au programme appelant.

Cette notion informatique de *fonction* est d'ailleurs la même que que la notion mathématique : calculer $f(3)$ c'est remplacer x par 3 dans l'expression analytique de $f(x)$ et renvoyer la valeur obtenue.

```
# Algorithme de Newton-Raphson
# calcul de zéro(s) d'une fonction
# ---------------------------------------
def f(x) # expression analytique de f(x)
   return x**3-x-1
end

def df(x) # expression analytique de f'(x)
   return 3*x**2-1
end
# ------------------------------------------------
erreur=1e-15 # précision souhaitée
compteur=1 # compteur d'itérations
puts "valeur approchée de la racine ?"
a=gets.to_f # valeur initiale
while f(a).abs>erreur and df(a)!=0
   puts "valeur approchée n° #{compteur}: #{a} "
   a=a-f(a)/df(a)
   compteur=compteur+1
end
if df(a)==0
   puts "erreur, tangente horizontale, arrêt d'urgence"
else
   puts "terminé avec valeur approchée n° #{compteur} : #{a}"
end
```

Le caractère #, en dehors de l'instruction `puts`, introduit une *ligne de commentaire*, qui ne sera pas exécutée par *Ruby*.

Pour appliquer à une autre fonction (mathématique) $f(x)$, il suffit de faire les modifications nécessaires dans les 2 fonctions (informatiques) $f(x)$ et $df(x)$. Le calcul de la dérivée sera évidemment un atout majeur ...

5 Pour en savoir plus, en mathématique

1 En prenant 3 comme valeur initiale dans l'algorithme de Newton, l'algorithme s'arrête à la 7e itération avec 1.324717957244746 comme valeur approchée de la racine.

2 Si vous n'aviez jamais vu 200!, la voici :
7886578673647905035523632139321850622951359776871 7
3263294742533244359449963403342920304284011984623 9
0417721213891963883025764279024263710506192662495 2
8299311134628572707633172373969889439224456214516 6
4240254033291864131227428294853277524424075739032
4032125740557956866022603190417032406235170085879 6
1789222227896237038973747200000000000000000000000 0
00000000000000000000000

25978!, c'est très joli aussi, mais le papier coûte cher (environ 24 pages de 55 lignes de 80 caractères) !

Pour se consoler, le premier chiffre est 1, le dernier chiffre significatif est 4, et il est suivi de 6491 zéros.

3 Et pour terminer, un petit algorithme qui calcule le nombre de zéros qui terminent l'écriture de $n!$: chaque fois que $n!$ est divisible par 10, on incrémente un compteur de 1 unité, et on fait la division.

```
# calcul du nombre de zéros à la fin de n!
# ----------------------------------------
puts "valeur de n ?"
n=gets.to_i # supposé entier positif non nul
factn=1
for k in 1..n
   factn=factn*k
end
puts "#{n}! = #{factn}"
c=0 # compteur de zéros
while factn%10 == 0
   c=c+1
   factn=factn/10
end
puts "il y a #{c} zéro(s)"
```

Références

[1] Signac L., *Divertissements mathématiques et informatiques*. MiniMax, 2011.

[2] Thomas D. et Hunt A., *Programming Ruby : The Pragmatic Programmer's Guide*. 2002.

Algorithmique et Ruby, une intégrale définie... par hasard

Résumé. *La méthode de Monte-Carlo est une méthode de calcul numérique basée sur le calcul des probabilités. On l'utilise ici pour évaluer l'aire de régions planes du plan, et en particulier pour calculer une approximation de certaines intégrales définies.*

Cette méthode est particulièrement intéressante parce qu'elle est **très** *élémentaire, même lorsqu'on l'applique à des fonctions dont les primitives ne sont pas simples, ou n'existent tout simplement pas (du moins pas en termes de fonctions dites élémentaires), telles que celle associée à la loi normale de Gauss, ou à la fonction très oscillante* $f(x) = \left|\sin \frac{1}{x}\right|$.

Les algorithmes étant « ouverts », il est facile de les appliquer (ou de les transformer selon les besoins) pour traiter n'importe quelle fonction continue sur n'importe quel intervalle.

Un article sur la méthode de Monte-Carlo, par le même auteur mais traitée avec un tableur, a été publié précédemment dans la revue Math-Jeunes [2]. L'algorithmique permet d'aller beaucoup plus loin, puisqu'on n'est pas lié au fait qu'un point aléatoire correspond à une cellule de la feuille de calculs.

1 Monte-Carlo

Il existe des méthodes bien connues (et efficaces) pour calculer une valeur approchée d'une intégrale définie $\int_a^b f(x)\,\mathrm{d}x$: rectangles, trapèzes, Simpson, ?

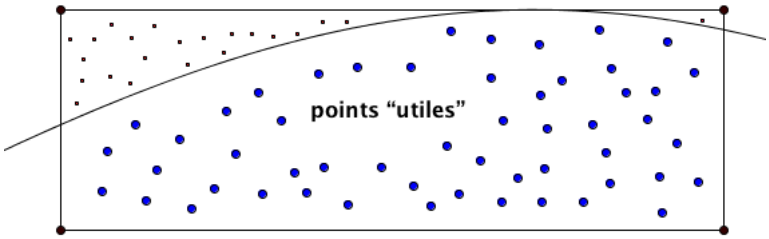
Moins traditionnelle, la méthode statistique dite « de *Monte-Carlo* », haut lieu des jeux de hasard, entre autres, permet de réaliser ce calcul à partir d'un générateur de nombres aléatoires (présent sur tous les moyens modernes de calcul, le tableur, la calculatrice, et autres logiciels mathématiques, et, bien sûr, les langages de programmation, ce qui nous intéresse particulièrement ici).

La méthode, simple de principe, consiste à arroser d'une pluie aléatoire un rectangle de dimensions connues, que l'on pourrait appeler aussi « champ de tir », circonscrit à une figure d'aire inconnue bordée par le graphique d'une fonction, et de calculer le rapport entre le nombre de gouttes arrosant la figure et le nombre de gouttes arrosant le rectangle.

Pour passer de la météo à la géométrie, on traduira *goutte de pluie* par un point dont l'abscisse x et l'ordonnée y sont engendrées aléatoirement dans un intervalle fixé. Pour la suite, on appellera « point utile » un point situé dans la région du plan dont nous évaluons l'aire.

Cette méthode ne donne pas d'aussi bons résultats que celles pré-citées (très loin de là), mais elle a le mérite de faire parler un peu d'algorithmique, de *Ruby*, et aussi de mathématique, tout

simplement.



Dans cette figure, on a représenté les « points utiles » de manière un peu plus enveloppée afin de les faire ressortir par rapport aux autres, mais il n'y a pas d'instruction dans le langage *Ruby* pour engendrer des points plus gros que les autres (on peut gaver un canard, une oie, mais pas un point) !

2 Des nombres pseudo-aléatoires

Une machine qui **calcule** un nombre qui serait **aléatoire**, ça n'existe pas ! Il s'agit en réalité de suites numériques calculées à partir de méthodes complexes aux noms non moins complexes comme *linéaire multi-récursif combiné, inversif, ….* Certaines sont moins mauvaises que les autres, et il existe des tests qui permettent de mesurer la qualité de ces générateurs de nombres pseudo-aléatoires. On n'a que très rarement des informations sur la manière dont sont engendrés ces nombres.

Il peut être toutefois intéressant de savoir que ces suites numériques sont souvent calculées à partir de différents paramètres dont l'un est appelé *la graine*. Lorsque ce nombre est basé par exemple sur l'horloge interne de la machine, il varie, forcément, et les suites générées sont toujours distinctes, d'où le sentiment d'*aléatoire*.

Le langage *Ruby* permet de fixer une *graine* au choix, ce qui permet d'utiliser toujours la même série de nombres (et donc de toujours avoir le même résultat aléatoire, si nécessaire) :

```
srand graine
```

Si l'algorithme contient par exemple l'instruction `srand 51` [1], on est sûr que le 100e nombre généré par l'instruction `rand` sera toujours 0.8895868292303459.
`srand 0` « remet les pendules à l'heure » et rend l'aléatoire plus aléatoire encore.

Les trois instructions de base qui permettent d'engendrer un nombre pseudo-aléatoire sous *Ruby* sont :

- `rand` : un réel dans $[0,1[$.
- `rand(a)` : un naturel dans $[0,a[$.
- `rand(a..b)` : un naturel dans $[a,b]$ (très pratique pour simuler le lancer d'un dé).

1. Le nombre 51 est bien connu dans le sud du pays des gaulois, mais on peut évidemment choisir n'importe quel naturel non nul.

Il faut manipuler un peu pour engendrer un réel dans $[a,b]$: `a+(b-a)*rand` avec bien entendu $b>a$.

On supposera pour la suite des calculs que $a<b$, et que $f(x)$ est continue sur l'intervalle $[a,b]$.

On se permettra également d'appeler *aléatoire* une valeur qui est en réalité *pseudo-aléatoire*.

3 $f(x)$ positive sur $[a,b]$

Dans ce cas, le calcul $\int_a^b f(x)\mathrm{d}x$ est celui de l'aire de la région du plan bordée par le graphique de $f(x)$, le segment $[ab]$ et les 2 droites d'équation $x=a$ et $x=b$, aire que nous noterons A.

On aura besoin de l'ordonnée du maximum de $f(x)$ sur $[a,b]$, que nous noterons max. La surface plane dont on calcule l'aire A est inscrite au rectangle dont un côté mesure $b-a$ et l'autre max.

Si max est connu, il suffit de le fixer dans l'algorithme (par exemple `max=5.1`) ; dans le cas contraire, on peut en calculer une valeur approchée via l'algorithme qui suit. Pour la facilité, on va fixer les valeurs de a et de b, mais on pourrait les faire encoder par l'utilisateur via l'instruction `a=gets.to_f` [1] et idem pour b.

Algorithme de calcul du maximum

Au départ, on assigne la valeur $f(a)$ à la variable max ; on parcourt l'intervalle $[a,b]$ *à petits pas* ; dès qu'on trouve une valeur de x telle que le nombre $f(x)$ dépasse max, c'est lui qui devient max. On peut aussi opter pour un pas qui dépend de l'intervalle $[a,b]$, par exemple $\frac{b-a}{100000}$.

Exemple ci-dessous pour $f(x)=-x^2+4$ sur l'intervalle $\left[-1,\frac{3}{2}\right]$.

On sait évidement que la réponse est 4, mais il vaut toujours mieux tester un algorithme sur un problème dont on connaît la réponse, avant de traiter des situations moins évidentes.

```
# max.rb
# calcul de l'ordonnée du maximum de f(x)
# positive sur [a,b]
# ---------------------------------------
def f(x)
    return -x*x+4
end
a=-1 ; b=1.5 ; pas=0.000001 # choix du petit pas
max=f(a)
x=a
while x<=b
    if f(x)>max
        max=f(x)
    end
    x=x+pas
end
puts "max= #{max}"
```

La formule de base de la méthode de Monte-Carlo, après la pluie aléatoire :

$$\frac{\text{nbre de points utiles}}{\text{nbre total de points}} \approx \frac{A}{\text{aire rectangle}}$$

d'où on tire aisément :

$$A \approx (b-a) \cdot max \cdot \frac{\text{nbre de points utiles}}{\text{nbre total de points}}$$

Il suffit donc de compter le nombre de « points utiles ».

Algorithme

Un point de coordonnée (x,y) est aléatoire si x est un réel aléatoire dans l'intervalle $[a,b]$ et y un réel aléatoire dans l'intervalle $[0,max]$;

un point est *utile* s'il est **sous le graphe** de $f(x)$, c'est-à-dire si $y < f(x)$. On pourrait écrire aussi $y \leqslant f(x)$, mais l'égalité est fort peu probable et on n'est pas à une goutte de pluie près, étant donné la drache nationale qui va arroser le rectangle ?

Au départ, un compteur de points utiles est initialisé à zéro ; chaque fois qu'un point est sous le graphe, on incrémente le compteur d'une unité.

```
# monte-carlo-fpositive.rb
# monte-carlo avec f(x) positive sur [a,b]
# ----------------------------------------
def f(x)
    return -x*x+4
end
a=-1
b=1.5
pas=0.000001 # choix du petit pas
max=f(a)
x=a
while x<=b
    if f(x)>max
        max=f(x)
    end
    x=x+pas
end
start=Time.now
ptutiles=0.0
nbrepoints=100000000 # la drache
for k in 1..nbrepoints
    x=a+(b-a)*rand
    y=0+(max-0)*rand #explication ci-dessous
    if y<f(x)
        ptutiles=ptutiles+1
    end
end
stop=Time.now
puts (b-a)*max*ptutiles/nbrepoints
puts "temps : #{stop-start} seconde(s)"
```

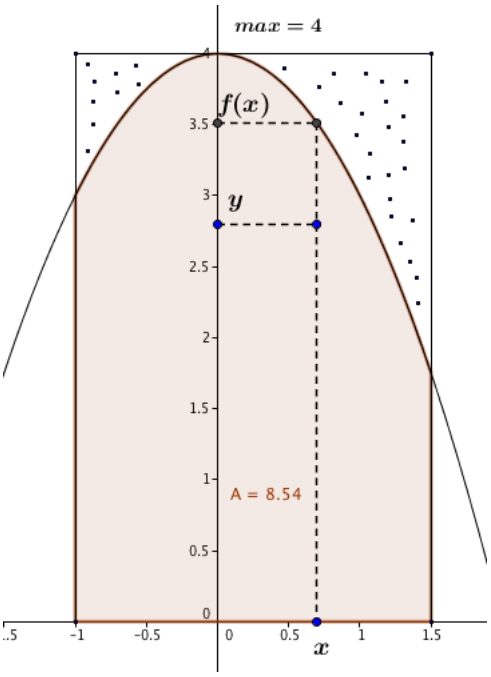
La syntaxe de l'instruction `y=0+(max-0)*rand` est volontaire pour montrer que y est engendré aléatoirement dans l'intervalle $[0,max]$, et ainsi la rendre similaire à l'instruction précédente qui exprime que x est engendré aléatoirement dans l'intervalle $[a,b]$.

Avec 100 millions de points aléatoires, et après 76 secondes de calcul (la durée de calcul dépend évidemment de l'ordinateur), j'ai obtenu comme valeur approchée $A = 8.541275$ pour une valeur exacte qui vaut $\frac{205}{24} = 8.541666?$

Pour tester d'autres exemples, il suffit de modifier ci-dessus l'expression de $f(x)$ (en respectant les règles de syntaxe *Ruby*) ainsi que les bornes a et b. C'est ce qui a été fait dans les 2 exemples qui suivent.

Remarque

La « hauteur » du rectangle peut même être supérieure au *maximum* de $f(x)$ sur $[a,b]$; en fixant $max = 10$ dans l'algorithme ci-dessus, on obtient une valeur approchée $A = 8.5403475$.



Cas particulier : une (mauvaise) valeur approchée de π

π au jeu de fléchettes, qui l'eût crû ?

Même si on lui connaît déjà plusieurs millions de décimales, en voici... 3.

Il suffit de prendre $f(x) = \sqrt{1-x^2}$, $a = 0.0$ [2], $b = 1.0$ et $max = 1$, pour obtenir

$$\frac{\pi}{4} \approx \frac{\text{nbre de points utiles}}{\text{nbre total de points}}$$

Avec 100 millions de points, on obtient $\pi \approx 3.1418465$.

Autre classique : la loi normale

Toujours avec 100 millions de points aléatoires, la méthode de Monte-Carlo donne $\int_1^3 \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}\,dx \approx 0,15727544?$ alors que les tables fournissent $0,1573$; ce n'est pas si mal !

Pour la définition de la fonction en *Ruby*, utiliser

2. La notation décimale est obligatoire pour obtenir un résultat décimal.

```
Math.exp(-0.5*x**2)/Math.sqrt(2*Math::PI)
```

et aussi $a = 1.0$, $b = 3.0$, $max = f(a)$.

4 $f(x)$ négative sur $[a,b]$

On se ramène au cas où $f(x)$ positive sur $[a,b]$ en utilisant la fonction $-f(x)$; et ne pas oublier de rendre le résultat négatif !

C'est un bel exercice... laissé au lecteur. On en donne à nos élèves, montrons l'exemple.

5 $f(x)$ change de signe sur $[a,b]$

On pourrait découper l'intervalle $[a,b]$ en sous-intervalles où $f(x)$ est soit positive soit négative, mais cela nécessiterait la connaissance (ou le calcul) des racines. La méthode de Newton (par exemple) est facile et efficace, à condition de savoir dériver, mais malheureusement ne les calcule pas toutes.

Adieu donc les racines !

S'ils ne sont pas connus, on peut calculer une valeur approchée de max et de min, respectivement ordonnées du maximum absolu et du minimum absolu de $f(x)$ sur $[a,b]$.

Algorithme

Initialement, on assigne $f(a)$ aux variables min et max ; x varie de a à b par petits pas ? si $f(x) > max$ alors il devient max, par contre si $f(x) < min$ alors il devient min.

Ces 2 valeurs étant connues, l'idée, finalement très simple, consiste à **translater le graphique** de $f(x)$ d'une amplitude de $|min|$ vers le haut, à savoir traiter la fonction $f(x) - min$. Celle-ci est positive sur $[a,b]$, l'ordonnée du maximum est $max - min$, et on applique la méthode de Monte-Carlo comme décrite précédemment pour une fonction positive. Le rectangle circonscrit a pour dimensions $(b-a)$ et $(max - min)$.

Le résultat obtenu, positif,

$$(b-a)\cdot(max-min)\cdot\frac{\text{ptutiles}}{\text{nbrepoints}}$$

doit être ajusté pour obtenir l'intégrale définie cherchée. Voici comment :

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x &= \int_a^b (f(x) - min + min)\,\mathrm{d}x \\ &= \underbrace{\int_a^b (f(x) - min)\,\mathrm{d}x}_{\text{Monte-Carlo}} + \underbrace{\int_a^b min\,\mathrm{d}x}_{\text{ajustement}} \\ &= (b-a)\cdot(max-min)\cdot\frac{\text{ptutiles}}{\text{nbrepoints}} + min\cdot(b-a)\end{aligned}$$

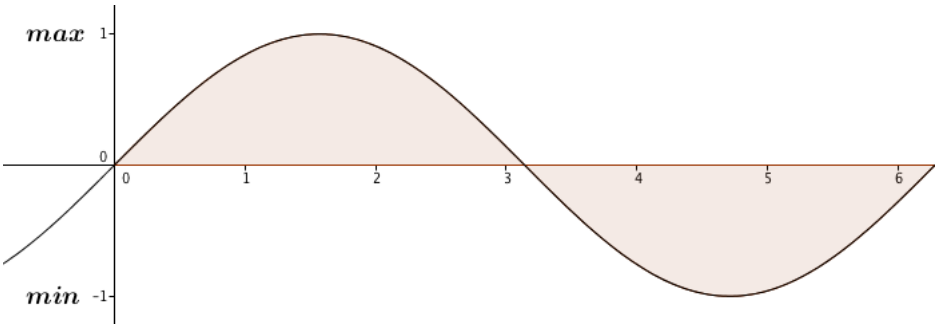
L'ajustement est une quantité négative. On peut encore mettre $(b-a)$ en évidence, mais cela ne rend pas la formule plus claire.

Illustration avec $f(x) = \sin x$ sur $[0, 2\pi]$

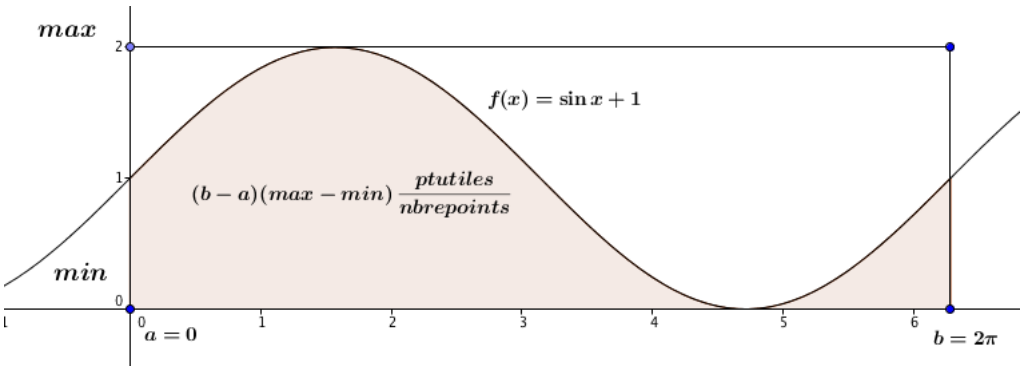
On sait que $\int_0^{2\pi} \sin x \, dx = 0$; on pourra ainsi facilement juger de la précision de la méthode, tout en sachant qu'il ne faut pas s'attendre à des miracles.

Pour rendre la situation plus naturelle, on a feint de ne pas connaître *min* et *max* (respectivement -1 et 1), et on en a calculé une valeur approchée en début d'algorithme par la méthode déjà décrite précédemment.
On trouve ainsi $min = -0,9999999999994802$ et $max = 0,9999999999932537$ pour un pas de $0,00001$.
Avec une pluie aléatoire de 100 millions de points, on obtient $0,000084822...$ comme valeur approchée de l'intégrale définie. Cette valeur change évidemment en fonction de la suite des nombres pseudo-aléatoires. Graphiquement :

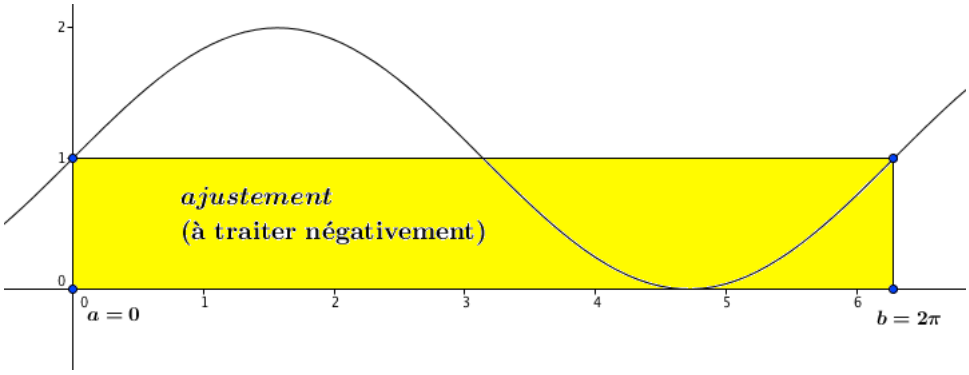
On veut évaluer



On translate le graphique et on applique Monte-Carlo



On ajuste



```
# monte-carlo-sinx.rb
# valeur approchée d'une intégrale définie
#  f(x)=sin(x)  sur [0,2pi]
# -------------------------------------
def f(x)
    return Math.sin(x)
end
# intervalle [a,b]
a=0 ; b=2*Math::PI
# valeur approchée de min et de max
pas=0.00001 ; min=f(a) ; max=f(a)
x=a
while x<=b
    x=x+pas
    if f(x)>max
        max=f(x)
    elsif f(x)<min
        min=f(x)
    end
end
puts "min = #{min}"
puts "max = #{max}"
# Monte-Carlo
ptutiles=0
nbrepoints=100000000 # la drache
for k in 1..nbrepoints
    x=a+(b-a)*rand
    y=min+(max-min)*rand
    xx=f(x)-min
    if y<xx
        ptutiles=ptutiles+1
    end
end
# ajustement et valeur approchée
puts (b-a)*(max-min)*ptutiles/nbrepoints + min*(b-a)
```

Pour traiter un autre exemple, il **suffit** de modifier dans l'algorithme ci-dessus l'expression de la fonction $f(x)$ ainsi que les valeurs des bornes a et b.

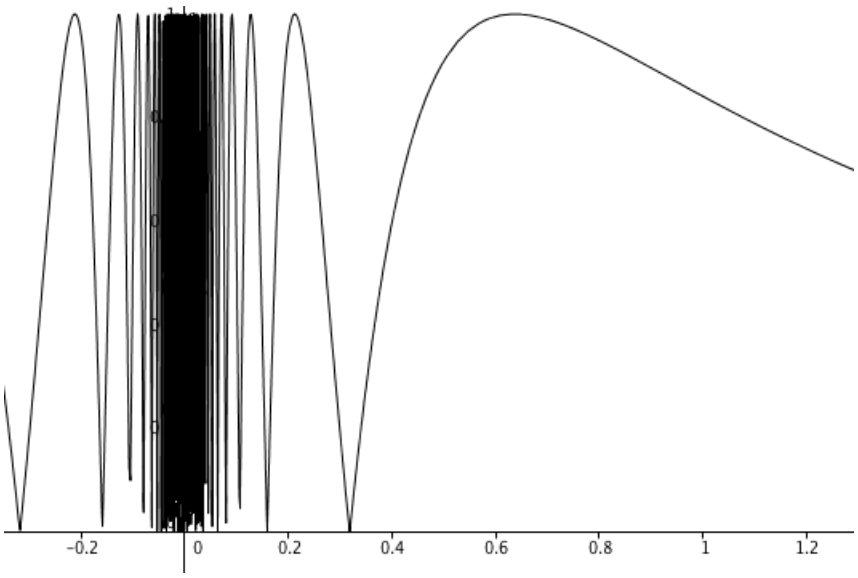
Remarquer que l'on peut appliquer cet algorithme dans les 2 cas précédents où $f(x)$ est soit positive, soit négative sur l'intervalle $[a,b]$.

Pour revenir à l'article de *Math-Jeunes* cité au début, on peut calculer facilement $\int_0^1 \left|\sin\frac{1}{x}\right| \mathrm{d}x$, fonction dont les primitives ne se calculent pas en termes de fonctions élémentaires.

Avec 100 millions de points, on obtient $\int_0^1 \left|\sin\frac{1}{x}\right| \mathrm{d}x \approx 0{,}774499...$

Dans l'algorithme ci-dessus, pour ne pas devoir calculer $f(0)$, on a pris $a = 10^{-15}$.

La fonction, en *Ruby*, s'écrit `(Math.sin(1/x)).abs` et son graphique ressemble à



En utilisant le logiciel *Maple*, nettement plus spécialisé pour ce genre de calculs, on obtient comme résultat la valeur 0,764298699621463...

La méthode de Monte-Carlo n'est de toute évidence pas à la hauteur, mais on a quand même la première décimale.

Il faut dire qu'en observant le graphique dans un voisinage de 0, il doit y avoir des *petits problèmes* d'arrondis lorsqu'il faut vérifier qu'un point aléatoire est **sous** la courbe.

6 L'aire de mon coeur

La méthode de Monte-Carlo permet aussi de traiter des courbes à partir d'une équation implicite. On aurait pu le faire pour le cercle trigonométrique ($x^2 + y^2 = 1$), mais varions les plaisirs.

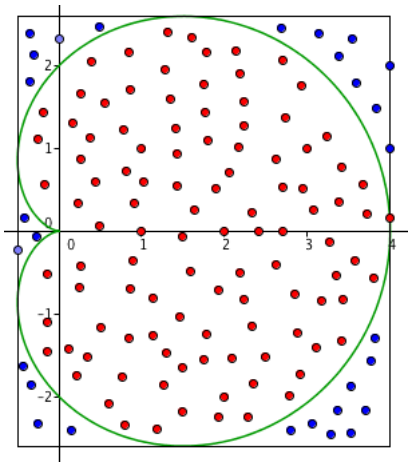
La cardioïde est une courbe représentant la trajectoire d'un point fixe situé sur un cercle de diamètre a qui roule sans glisser sur un second cercle de même diamètre.

Son équation implicite est

$$(x^2 + y^2 - ax)^2 = a^2(x^2 + y^2)$$

Pour les calculs qui suivent, on a choisi $a = 2$ (cercle trigonométrique). La courbe est inscrite dans un rectangle dont les sommets sont $\left(\frac{-1}{2}, \frac{-3\sqrt{3}}{2}\right), \left(\frac{-1}{2}, \frac{3\sqrt{3}}{2}\right), \left(4, \frac{-3\sqrt{3}}{2}\right), \left(4, \frac{3\sqrt{3}}{2}\right)$ et dont l'aire vaut $\frac{27\sqrt{3}}{2}$.

Un point (x, y) est intérieur à cette cardioïde s'il vérifie la condition $(x^2+y^2-2x)^2 \leqslant 4(x^2+y^2)$; il suffira donc de compter ces points.



```
# cardioide.rb
# -------------
m=0.0 # compteur de points intérieurs
n=10000000 # nbre de points aléatoires
xmin=-0.5
xmax=4
ymin=-3*Math.sqrt(3)/2
ymax=3*Math.sqrt(3)/2
aire_rect=(xmax-xmin)*(ymax-ymin)
for k in 1..n
   x=xmin+(xmax-xmin)*rand
   y=ymin+(ymax-ymin)*rand
   if (x**2+y**2-2*x)**2<=4*(x**2+y**2)
      m=m+1
   end
end
puts aire_rect*m/n
```

La valeur exacte est $6\pi = 18,8496\ldots$; l'algorithme ci-dessus donne $18,849063\ldots$ pour 10 millions de points aléatoires.

7 Remarque importante

Le « copier-coller » du code *Ruby* à partir de cet article (au format .pdf) vers un éditeur de textes ne donne pas de bons résultats ; certains sauts de lignes ne sont pas respectés, et parfois même des portions de texte disparaissent.

A la moindre faute de frappe, l'algorithme ne sera pas exécuté, ou donnera des résultats erronés. La vigilance est de rigueur !

Références

[1] Desbonnez J.-M., Algorithmique élémentaire et langage Ruby. *Losanges*, 28, p. 50. 2015.

[2] Desbonnez J.-M., Calcul statistique d'une aire. *Math-Jeunes*, 112S, pp. 16–19. 2005.

Algorithmique et Ruby, le hasard est probablement bien organisé !

Résumé. *Les probabilités sont des notions assez abstraites pour un non-initié. Lorsqu'on lance une pièce de monnaie, on comprend facilement que l'on a une chance sur deux d'amener pile. On accepte alors assez facilement que la* **probabilité** *d'obtenir pile se traduit par* $\frac{1}{2}$*. Mais, en pratique, lorsqu'on lance dix fois une pièce et que pile est amené neuf fois, on se pose des questions . . .*

Les quelques exemples développés ci-après ne vont pas révolutionner les mathématiques ; il ne faut pas s'attendre à une rencontre du troisième type avec Monsieur Kolmogorov*. Ils ont pour seule ambition de montrer, moyennant des processus de calculs assez simples, et surtout de* **nombreuses** *simulations, que l'on peut se représenter ce qu'est une probabilité.*

La rapidité de calcul d'un ordinateur va nous rendre de grands services : un humain peut-il envisager de lancer une pièce dix millions de fois ? Testons quelques nombreuses manipulations de pièces, de dés, de puces, et autres.

1 Les questions oubliées

On lance une pièce de monnaie parfaitement équilibrée.

Jusque là, c'est classique, mais il y a deux questions qu'on ne trouve jamais dans les livres de mathématiques :

1. Calculer la probabilité que la pièce retombe.
2. Calculer la probabilité que la pièce reste en l'air.

Pas besoin d'ordinateur pour répondre à ces questions, mais elles permettent d'ouvrir le débat à propos de l'événement certain, de l'événement impossible (sur la Terre, mais c'est notre milieu de vie), des événements contraires.

Pour les incrédules, on peut toutefois expérimenter : on lance 100 millions de fois une pièce, et on compte le nombre de fois qu'elle retombe, le nombre de fois qu'elle reste en l'air. On calcule les deux fréquences, et on peut conjecturer ?

C'est de cette manière que l'on va procéder pour faire des simulations de calculs de probabilités : effectuer de **nombreux** tirages aléatoires (ici l'ordinateur est indispensable), compter les *cas favorables*, puis calculer les fréquences en appliquant la sacro-sainte formule

$$\frac{\text{nombre de cas favorables}}{\text{nombre total d'expériences}}$$

J'allais oublier les réponses aux 2 questions fondamentales :

1 $\frac{100000000}{100000000} = 1$.

2 $\frac{0}{100000000} = 0$.

2 Pile je gagne, face tu perds

On lance une pièce de monnaie parfaitement équilibrée. Calculer la probabilité d'amener pile, *d'amener* face.

Quand j'étais en terminale (il y a bien longtemps déjà, soupir...), un élève lançait une pièce, un autre notait le résultat au tableau... pendant 2 heures [1]... (100 minutes en fait, puisque dans l'enseignement secondaire belge, 1 heure vaut 50 minutes) [2] ; heureusement, pour faire participer tous les élèves, le couple était remplacé régulièrement, sans doute pour s'assurer de l'indépendance des expériences ?

Sérieusement maintenant, avec *Ruby*.

L'instruction `rand` [3]génère un réel aléatoire dans l'intervalle $[0,1[$. On divise l'intervalle en deux parts égales, et on attribue à *pile* (par exemple) un nombre inférieur à $0,5$ et donc à *face* un nombre supérieur à $0,5$.

A chaque nombre aléatoire, il suffira d'incrémenter soit le compteur de *pile*, soit le compteur de *face*, selon le cas qui se présente. Et ce pour **beaucoup** de lancers [4].

```
# pile-face.rb
# ------------
nblancers=1000000000
pile=0.0 # compteur de pile
face=0.0 # compteur de face
for k in 1..nblancers
    if rand < 0.5
        pile=pile+1
    else
        face=face+1
    end
end
freqpile=pile/nblancers
freqface=face/nblancers
puts "fréquence pile : #{freqpile}"
puts "fréquence face : #{freqface}"
```

Résultats obtenus

Avec 100 millions de lancers :

- fréquence *pile*=0,49996718
- fréquence *face*=0,50003282

Avec 1 milliard de lancers (et un peu plus de temps de calcul), seules les 4 premières décimales restent inchangées [5].

Ces résultats permettent aussi de mesurer la qualité du générateur aléatoire.

Rappelons que l'initialisation des compteurs à 0.0 est indispensable pour obtenir un résultat de type réel (et non arrondi).

1. C'est la vérité !
2. Avec ce type de raisonnement, les 62 jours de vacances d'été sont réduits à 51,6 jours, soupir aussi.
3. Desbonnez J.-M., Algorithmique élémentaire et Ruby, *Losanges*, n°28, mars 2015, p.48.
4. Les nombres pseudo-aléatoires étant engendrés à partir de séries calculées, ils ne sont en réalité pas tout-à-fait indépendants.
5. On peut montrer qu'il faut multiplier par 100 le nombre de lancers pour gagner 1 décimale.

Si la pièce est truquée

Supposons que la pièce a deux fois plus de chance d'amener *pile* que *face*. Dans ce cas, il suffit de diviser l'intervalle [0,1[en trois parts égales et d'en attribuer deux à *pile*. Le test dans l'algorithme devient alors

```
if rand < 2/3
    pile=pile+1
else
    face=face+1
end
```

3 Pièce et dé(s)

3.1 Un dé, fréquence de chaque face

On peut diviser l'intervalle [0,1[en six parts égales.
On peut aussi engendrer un **entier** aléatoire dans [1,6] : il faut multiplier `rand` par 6, prendre la partie entière [6] et ajouter 1, ce qui donne `1+(6*rand).to_i`
Mais *Ruby* propose un petit luxe dont nous allons profiter, l'instruction `rand(1..6)`.

Pour calculer la fréquence d'apparition de chaque face, il faudra six compteurs (donc six noms de variables, et une alternative multiple [7] pour traiter les six cas). C'est un bel exercice.

On peut simplifier l'algorithme en utilisant une variable de type *tableau* [8] (appelée aussi *variable indicée*, une matrice ligne ou colonne, tout simplement) [9].

Le kit de survie

L'instruction `tab=Array.new(n){a}` crée un tableau nommé *tab* de n cellules initialisées avec la valeur a. Chaque cellule est accessible par les expressions `tab[0]`, `tab[1]`, ..., `tab[n-1]`.
Remarquer que la numérotation commence toujours à 0, mais il n'est pas obligatoire (parfois même plus facile) de ne pas utiliser `tab[0]`.

Pour traiter notre dé, on utilisera `compteur=Array.new(7){0.0}` pour initialiser six compteurs (`compteur[1]` pour compter les 1, ..., `compteur[6]` pour compter les 6).

6. En *Ruby*, prendre la partie entière c'est convertir en entier.
7. Desbonnez J.-M., Algorithmique élémentaire et Ruby, *Losanges*, n°28, mars 2015, p.52.
8. Nom officiel : *Array*
9. Un article consacré aux tableaux est en cours de préparation.

```
# un-de.rb
# fréquence de chaque face
# ------------------------
nblancers=1000000000
compteur=Array.new(7){0.0}
for k in 1..nblancers
    jet=rand(1..6)
    compteur[jet]=compteur[jet]+1
end
for k in 1..6
    puts "fréq. de la face #{k} : "+
        (compteur[k]/nblancers).to_s
end
```

Résultats

Pour 100 millions de lancers, on obtient :

fréquence de la face 1 : 0.16670938
fréquence de la face 2 : 0.16662856
fréquence de la face 3 : 0.16663721
fréquence de la face 4 : 0.16666619
fréquence de la face 5 : 0.16670928
fréquence de la face 6 : 0.16664938
soit des valeurs relativement proches de $\frac{1}{6}$.

3.2 Deux dés

On lance simultanément deux dés ; calculer la probabilité que les deux faces amenées soient des entiers consécutifs.

Il faudra dans ce cas tester si la valeur absolue de la différence des deux valeurs est égale à 1 ; un seul compteur suffira.

```
# deux-des.rb
# proba 2 entiers consécutifs
# ---------------------------
nblancers=1000000000
compteur=0.0
for k in 1..nblancers
    jet1=rand(1..6)
    jet2=rand(1..6)
    if (jet1-jet2).abs==1
        compteur=compteur+1
    end
end
puts compteur/nblancers
```

Résultats

Pour 100 millions de lancers, on obtient : 0,27771426

La réponse exacte est $\frac{10}{36} = 0,2777777\ldots$ que l'on trouve facilement en faisant un tableau à deux entrées.

3.3 La variable aléatoire binomiale

On lance cinq fois une pièce de monnaie ; calculer la probabilité d'amener 0 fois pile, *1 fois* pile, *2 fois* pile, *?, 5 fois* pile.

Commençons par compter le nombre de *pile* sur cinq lancers :

```
# 1piece-5fois.rb
# --------------
c=0.0 # compteur de pile
for k in 1..5
    if rand<0.5
        c=c+1
    end
end
puts "nompre de pile : #{c}"
```

Aves le même canevas que dans l'exemple précédent, on va exécuter plusieurs parties, avec cumul des résultats dans six compteurs (les six valeurs possibles de nombre de *pile* amenés).

```
# binomiale5.rb
# --------------
nbparties=10000000
compteur=Array.new(5+1){0.0}
for j in 1..nbparties
    c=0
    for k in 1..5
        if rand<0.5
            c=c+1
        end
    end
    compteur[c]=compteur[c]+1
end
for k in 0..5
    puts "#{k} : #{compteur[k]/nbparties}"
end
```

Résultats

Pour 10 millions de parties :

nb pile	fréquence
0	0.0312355
1	0.1561776
2	0.3123901
3	0.312839
4	0.1561478
5	0.03121

Ces résultats sont à comparer avec ceux obtenus en utilisant la formule d'une loi binomiale de paramètres 5 et $\frac{1}{2}$

$$P(X=k)=\binom{5}{k}\left(\frac{1}{2}\right)^5$$

où X est la variable aléatoire « nombre de *pile* amenés sur cinq lancers ».

On peut facilement généraliser l'algorithme en y remplaçant 5 par n.

4 Deux dés encore, on corse un peu

On lance simultanément deux dés ; calculer le nombre de lancers nécessaires, en moyenne, avant d'amener un double six.

Commençons par gérer une seule partie : on compte le nombre de lancers jusqu'à ce qu'on obtienne un double six, en utilisant une répétitive à test d'arrêt final [(10)] puisqu'on doit lancer au moins une fois les deux dés.

10. Desbonnez J.-M., Algorithmique élémentaire et Ruby, *Losanges*, n°28, mars 2015, p.53.

```
# double-six.rb
# lancer jusqu'au premier double 6
# --------------------------------
compteur=0.0
begin
    jet1=rand(1..6)
    jet2=rand(1..6)
    compteur=compteur+1
end while !(jet1==6 and jet2==6)
puts compteur
```

Si on exécute plusieurs fois l'algorithme (à échelle humaine), on obtient des résultats très disparates, tout comme leurs moyennes. Voyons.

Programmons plusieurs parties (de plus en plus) ; pour calculer une moyenne, il suffit de cumuler la somme des nombres de jets (variable *somme*) et de diviser cette somme par le nombre de parties (variable *nbparties*).

```
# double-six-moyenne.rb
# lancer jusqu'au premier double 6
# --------------------------------
nbparties=50000000
somme=0.0
for k in 1..nbparties
    compteur=0.0
    begin
        jet1=rand(1..6)
        jet2=rand(1..6)
        compteur=compteur+1
    end while !(jet1==6 and jet2==6)
    somme=somme+compteur
end
puts somme/nbparties
```

Résultats

nbparties	moyenne
100	32,34
200	32,275
1000	35,346
2000	36,427
5000	36,6486
50000	36,12802
1000000	36,01693
50000000	35,994926

Il semble que 36 soit une bonne approximation de la réponse cherchée ?

Vérification

Le problème est relativement simple à traiter avec Ruby, sa vérification l'est un peu moins ; on est en effet ici face à une *variable aléatoire géométrique*, dite « du premier succès ». On vient en fait de calculer son *espérance mathématique*.

Notons X la variable aléatoire « Obtenir un double six après n expériences ».

$$P(X=1)=\frac{1}{36} \quad \text{(facile à voir sur un tableau à 2 entrées)}$$
$$P(X=2)=\frac{35}{36}\cdot\frac{1}{36} \quad \text{(1 échec puis 1 succès)}$$
$$P(X=3)=\left(\frac{35}{36}\right)^2\cdot\frac{1}{36} \quad \text{(2 échecs puis 1 succès)}$$
$$\dots$$
$$P(X=n)=\left(\frac{35}{36}\right)^{n-1}\cdot\frac{1}{36} \quad \text{(n-1 échecs puis 1 succès)}$$

On a bien une distribution de probabilité, car la somme des probabilités vaut 1 :

$$\frac{1}{36}\cdot\left\{1+\frac{35}{36}+\left(\frac{35}{36}\right)^2+\dots+\left(\frac{35}{36}\right)^{n-1}+\dots\right\}=\frac{1}{36}\cdot\frac{1}{1-\frac{35}{36}}=1$$

$\left(\text{somme des termes d'une suite géométrique de raison } \frac{35}{36}\right)$.

Pour le calcul de l'espérance mathématique :

$$E(X)=\sum_{n\geqslant 1} n\cdot P(X=n)=\sum_{n\geqslant 1} n\cdot\left(\frac{35}{36}\right)^{n-1}\cdot\frac{1}{36}=\frac{1}{36}\cdot\sum_{n\geqslant 1} n\cdot\left(\frac{35}{36}\right)^{n-1}$$

En utilisant le LEMME $\quad |x|<1 \Rightarrow \sum_{n\geqslant 1} nx^{n-1}=\frac{1}{(1-x)^2}$

dont voici une démonstration :

si $|x|<1$, alors $\frac{1}{1-x}=1+x+x^2+x^3+\dots+x^k+\dots$

donc

$$\begin{aligned}\frac{1}{1-x}\cdot\frac{1}{1-x}&=(1+x+x^2+x^3+\dots+x^k+\dots)\cdot(1+x+x^2+x^3+\dots+x^t+\dots)\\&=1+x+x^2+x^3+\dots+x\cdot 1+x\cdot x+x\cdot x^2+x\cdot x^3+\dots\\&\quad+x^2\cdot 1+x^2\cdot x+x^2\cdot x^2+x^2\cdot x^3+\dots+x^3\cdot 1+x^3\cdot x+x^3\cdot x^2+x^3\cdot x^3+\dots\\&=1+2x+3x^2+4x^3+5x^4+\dots+nx^{n-1}+\dots\end{aligned}$$

on a

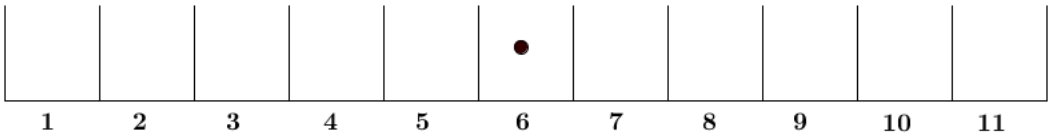
$$\frac{35}{36}<1 \Rightarrow E(X)=\frac{1}{36}\cdot\left(\frac{1}{1-\frac{35}{36}}\right)^2=36$$

5 La puce avait trop bu ?

... Ou une marche aléatoire de Pólya [11] qualifiée aussi de « marche de l'ivrogne ».

On dispose côte à côte onze boîtes numérotées de 1 à 11 [12] ; une puce s'installe dans la boîte centrale (n°6), commence à boire sans modération, puis démarre une ballade de cinq sauts aléatoires, soit d'une boîte à sa gauche, soit d'une boîte à sa droite.

11. George POLYA, mathématicien d'origine hongroise, 1887–1985.
12. Libre au lecteur d'imaginer la taille, la forme, la couleur et le contenu des boîtes.



Calculer la probabilité qu'elle revienne dans la boîte de départ. Et plus encore si affinité.

Algorithme

Comme d'habitude, on va observer les ballades de beaucoup de puces (variable *nbpuces*) ; à chaque saut, on mémorise la position (variable *pos*), initialement fixée à 6, on y retire 1 ou on y ajoute 1 selon qu'il y a un saut à gauche ou un saut à droite. Enfin, après les cinq sauts, on ajoute 1 au compteur correspondant au numéro de la boîte atteinte (variable *compteur*[pos]).

```
# polya-5puces.rb
# --------------
nbpuces=10000000
compteur=Array.new(12){0.0}
for j in 1..nbpuces
   pos=6
   for k in 1..5
      if rand<0.5 # saut à gauche
         pos=pos-1
      else # saut à droite
         pos=pos+1
      end
   end
   compteur[pos]=compteur[pos]+1
end
for k in 1..11
   puts "#{k} : #{compteur[k]/nbpuces}"
end
```

Résultats

Après observation de 10 millions de puces, [13]

k	probabilité de terminer dans boîte k
1	0.0312194
2	0.0
3	0.1561646
4	0.0
5	0.3124949
6	0.0
7	0.3124279
8	0.0
9	0.1564718
10	0.0
11	0.0312214

Curieux ?

Explication

Il semblerait que les puces soient allergiques aux boîtes paires.

En fait, il est impossible de rejoindre une boîte paire, et par conséquent de revenir dans la boîte de départ (n°6) ; la probabilité demandée est donc nulle.

13. Et beaucoup de démangeaisons ?

Notons G le nombre de sauts à gauche, et D le nombre de sauts à droite ; il est évident que $D = 5 - G$.
La position finale atteinte se calcule par l'expression

$$6 - G + D = 6 - G + (5 - G) = 11 - 2G$$

.
$2G$ est toujours pair, et donc $11 - 2G$ est toujours impair.

G	D	$11-2G$
0	5	11
1	4	9
2	3	7
3	2	5
4	1	3
5	0	1

Pour le calcul des probabilités de rejoindre les boîtes impaires, on applique une loi binomiale « nombre de sauts à gauches », par exemple, qui est une $Bin\left(5, \frac{1}{2}\right)$.

$$\text{P(rejoindre boîte n°1)} = \text{P}(5G \text{ et } 0D) = \binom{5}{5}\left(\tfrac{1}{2}\right)^5 = \tfrac{1}{32}$$
$$\text{P(rejoindre boîte n°2)} = \text{P}(4G \text{ et } 1D) = \binom{5}{4}\left(\tfrac{1}{2}\right)^5 = \tfrac{5}{32}$$
$$\text{P(rejoindre boîte n°3)} = \text{P}(3G \text{ et } 2D) = \binom{5}{5}\left(\tfrac{1}{2}\right)^5 = \tfrac{10}{32}$$
…

Pour qui se plait à généraliser,

$$\text{P(rejoindre boîte n°k)} = P\left(\frac{11-k}{2}G \text{ et } \frac{k-1}{2}D\right) = \binom{5}{\frac{11-k}{2}}\left(\frac{1}{2}\right)^5 \quad \text{avec } k \text{ impair } \in [1, 11]$$

6 Just for fun, une martingale ?

Les différents exemples traités précédemment montrent que l'algorithmique permet de faire des simulations rapides, en usant sans modération des nombres aléatoires.

Voici maintenant un calcul où toute ressemblance avec une situation existante [14] est tout sauf fortuite.

On choisit au hasard cinq nombres naturels distincts entre 1 et 50.
Quelle est, en moyenne, leur moyenne ?

Algorithme

La difficulté est d'engendrer cinq nombres **distincts** ; une solution consiste à placer tous les nombres dans un tableau, puis d'y supprimer le nombre qui a éte engendré aléatoirement, (*Ruby* propose l'instruction `tableau.delete_at(position)`) sans oublier de l'enregistrer dans le tableau des nombres retenus, et de mémoriser la taille du tableau réduit pour le tirage suivant.

L'utilisation d'un nombre important de tirages et le calcul de la moyenne ont déjà été utilisés dans les algorithmes précédents.

14. Une grille de l'euro-millions.

```
# just-for-fun.rb
# ---------------
nbessais=10000000
som=0.0
for t in 1..nbessais
   n=Array.new(51){0} # pour les 50 nbres de 1 à 50
   grille=Array.new(6){0} # pour les 5 nbres au hasard
   # remplissage du tableau des 50 nombres
   for k in 1..50
      n[k]=k
   end
   # tirage aléatoire de 5 nbres distincts
   taille=50
   for j in 1..5
      x=rand(1..taille)
      grille[j]=n[x]
      n.delete_at(x)# suppression du nbre choisi
      taille=taille-1 # ajustement de la taille du tableau
   end
   # calcul moyenne
   moy=0.0
      for j in 1..5
      moy=moy+grille[j]
   end
   som=som+moy/5
end
puts som/nbessais
```

Le résultat obtenu avec 10 millions d'essais est 25,5 ; il ne permet malheureusement pas d'augmenter la probabilité à laquelle tout le monde pense ! Le mathématicien, lui aussi, peut rêver !

Merci à Philippe TILLEUIL pour l'article — à paraître — sur la *loi symétrique*, qui explique et démontre le résultat obtenu.

On peut déjà lever un coin du voile : le résultat obtenu ne provient pas du calcul $\frac{1+50}{2}$. A suivre.

7 Un mouvement brownien

Robert BROWN (1773–1858), botaniste anglais, a observé un mouvement désordonné [15] de particules de pollen en suspension dans l'eau. De nombreux scientifiques s'y sont intéressés, dont Monsieur EINSTEIN, Albert de son prénom, excusez du peu...

Avec de modestes moyens, et pour rester dans le canevas des expériences précédentes, imaginons une particule qui se déplace k fois d'une distance rectiligne d dans une direction aléatoire α.

Calculons la distance moyenne parcourue entre sa position de départ et sa position d'arrivée.

15. D'où le nom de *mouvement brownien*.

Algorithme

Pour l'expérience, on va fixer d à 1 et k à 10, pour commencer. La direction α sera un nombre aléatoire dans l'intervalle $[0, 2\pi]$.

On supposera un repère orthonormé ; le point de départ est le point de coordonnée $(0,0)$. Si (x,y) est la coordonnée du point d'arrivée, la distance d séparant les deux points vaut $\sqrt{x^2+y^2}$.

A chaque « pas », on incrémente la position précédente de $d \cdot \cos\alpha$ pour l'abscisse et de $d \cdot \sin\alpha$ pour l'ordonnée.

Et, comme précédemment, pour obtenir une moyenne représentative, on observe un grand nombre de particules (variable *nbparticules*).

```
# mvt-brownien.rb
# ----------------
d=1 # distance parcourue à chaque déplacement
k=10 # nombre de déplacements élémentaires
nbparticules=1000000
moyenne=0.0
for i in 1..nbparticules
   x=0.0
   y=0.0 # position initiale
   for j in 1..k
      alpha=rand(0..2*Math::PI)
      x=x+d*Math.cos(alpha)
      y=y+d*Math.sin(alpha)
   end  # les k déplacements
   distance=Math.sqrt(x**2+y**2)
   moyenne=moyenne+distance
end
puts moyenne/nbparticules
```

On peut aisément faire varier les paramètres d et k, le temps d'exécution de l'algorithme étant directement proportionnel à k.

Résultats

nbparticules	k=10 d=1	k=100 d=1
100	2,819	9,2271
1000	2,8058	8,79303
10000	2,81058	8,812
100000	2,82085	8,8627
1000000	2,820827	8,8671

Pour les accrocs du chrono, et en guise de conclusion

Pour calculer le temps d'exécution d'un algorithme, il suffit de mémoriser (dans deux variables distinctes) le *temps-système* au début de l'algorithme, puis à la fin, la différence des deux fournissant un intervalle de temps en secondes.

```
start=Time.now
     ... toutes les instructions ...
stop=Time.now
puts "exécuté en #{stop-start} seconde(s)"
```

Algorithmique et langage Ruby, cuisiner les tableaux... bon appétit !

Résumé. *Lorsqu'il faut calculer (et mémoriser) de* **nombreux** *résultats, l'utilisation de variables simples est chose impossible, simplement à cause de leur nom. En effet, comment « inventer » mille noms de variables pour mémoriser, par exemple, mille nombres premiers, sans compter les mille lignes de code pour les affectations.*

Les tableaux sont des variables dont les composantes sont identifiées par un nom et un numéro appelé indice. Mathématiquement parlant, c'est une matrice ligne (ou colonne, selon la manière dont on la visualise).

*Les quelques exemples traités dans cet article relèvent à la fois du domaine de l'informatique (création de mots de passe, tri), du domaine de la gestion (recherche dichotomique, fusion de tableaux, recherche de doublons), et aussi, quand-même, du domaine mathématique (crible d'*Eratosthène*, une méthode originale de calcul d'un plus grand commun diviseur, et un peu de calcul vectoriel).*

Le tout accompagné d'algorithmique, et de sauce Ruby.

1 Mise en bouche

Un tableau est une liste d'éléments désignée par un *nom unique*, chaque élément étant repéré dans la liste par sa *position* (appelée aussi *indice*). On se limitera ici aux tableaux de dimension1.

On peut l'assimiler à une « commode à tiroirs », où chaque tiroir est une variable, qui porte le nom de la commode, et un numéro qui indique sa position dans la commode.

Dans le domaine informatique, on l'appelle *array* ; dans le domaine mathématique, on l'appelle *matrice* ou *vecteur*.

Si **x** est le nom générique du tableau, on notera **x[k]** l'élément d'indice **k**.
Soit **x** un tableau de taille 5, on peut le représenter par

x	x[0]	x[1]	x[2]	x[3]	x[4]

Tous les langages de programmation permettent la manipulation de tels objets, mais la syntaxe de leur utilisation diffère d'un langage à l'autre. Voici pour *Ruby*.

1.1 Remarques

- la numérotation des indices commence toujours à 0.
  Ainsi, le premier élément d'un tableau de taille n porte l'indice 0, le dernier élément porte l'indice $n-1$;

- il n'est pas obligatoire d'utiliser l'élément x[0], mais il faut alors penser à un tableau de taille 7 (par exemple) pour n'utiliser que x[1], x[2], ..., x[6].

1.2 Les points forts des tableaux

- l'utilisation d'un tableau permet de disposer d'un très grand nombre de variables en n'utilisant qu'un seul nom ;

- il y a un accès rapide à chaque élément du tableau si l'indice est le compteur d'une répétitive.

2 Préparation

2.1 Par affectation « manuelle » ou « explicite »

nom_du_tableau=[liste des valeurs séparées par une virgule]

impair=[1,3,5,7,9]
crée un tableau nommé *impair*, contenant les cinq premiers nombres impairs :

impair	1	3	5	7	9

Sans oublier que **impair[1]** contient le nombre 3.

jours=["lundi" , "mardi" , ... , "dimanche"]
crée un tableau nommé *jours* contenant les sept noms de jours [1] :

jours	lundi	mardi	mercredi	...	dimanche

nbre_premier = [false,false,true,true,false,true,false,true,false,false]
crée un tableau de 10 expressions booléennes, nommé *nbre_premier*, indiquant si l'indice est un nombre premier ou non :

nbre_premier	false	false	true	true	false	true	false	true	false	false
	0	1	2	3	4	5	6	7	8	9

1. On supposera le lecteur capable de compléter les mots manquants.

2.2 Tableau vide, puis ajout d'éléments

Cette manière de faire est utilisée lorsque les contenus ne sont pas « connus à l'avance », ou lorsqu'il y en a beaucoup ; ils seront « ajoutés au fur et à mesure » à un tableau initialement vide.

nom_du_tableau=[ ]

L'ajout d'une donnée à la fin du tableau tableau se fait par l'instruction

nom_du_tableau << donnée

ou en appliquant au tableau la méthode **push** par l'instruction

nom_du_tableau.push(donnée)

Exemple 1

Calcul (et mémorisation) des nombres premiers compris entre 2 et 50.

Pour rappel, un nombre entier est premier s'il est au moins égal à 2 et n'a pour seuls diviseurs que 1 et lui-même.

```
# premiers.rb
# nombres premiers de 2 à 50
# --------------------------
nb_prem=[ ] # tableau vide
for n in 2..50 # parcourir les nbres de 2 à 50
    prem=true # à priori n est premier
    for div in 2..n-1 # diviseurs potentiels
        if n%div==0
            prem=false # on a trouvé un diviseur
            break # inutile de continuer à chercher
        end
    end
    if prem
        nb_prem << n # ou bien nb_prem.push(n)
    end
end
puts nb_prem
```

Algorithme

On suppose qu'un nombre est premier (`prem = true`) ; on le divise par tous les nombres qui le précèdent (sauf 1 et lui-même) ; dès que l'on trouve un diviseur, on conclut qu'il n'est pas premier (`prem = false`) et on sort de la boucle des divisions.
A la sortie de la boucle, il suffit de tester le contenu de la variable `prem`.

Exemple 2

Encodage au clavier et mémorisation dans un tableau.

Algorithme

On initialise un tableau vide ; chaque élément encodé est ajouté au tableau tant que l'on n'a pas encodé une valeur d'arrêt (choisie à l'avance, évidemment).
Ci-contre l'encodage de nombres réels (arrêt lorsqu'on encode 0).

```
# encodage-tableau.rb
# -------------------
tableau=[ ]
puts "encoder nombre (0 pour arrêter)"
nombre=gets.to_f
while nombre !=0
    tableau.push(nombre)
    puts "encoder nombre (0 pour arrêter)"
    nombre=gets.to_f
end
puts tableau
```

Si la valeur 0 peut faire partie des nombres à encoder (par exemple le résultat d'une interrogation), il faut choisir une autre valeur d'arrêt (un nombre négatif par exemple dans ce cas précis).

2.3 Tableau initialisé avec des valeurs identiques

nom_du_tableau=Array.new(n){a}

Crée un tableau de n cellules de contenu a. Si l'option {a} est omise, les cellules contiennent l'élément « nil », c'est-à-dire « rien ».

3 Quelques épices : méthodes s'appliquant aux tableaux

Il y en a beaucoup... Pour se faire une petite idée, il suffit de créer, en mode interactif, un petit tableau nommé t, par exemple, puis de lui appliquer la méthode **methods** :

t.methods

On peut appliquer celà à n'importe quel objet du langage Ruby.

Soit donc un tableau nommé t. Voici un petit échantillon :

Méthode	Explication
t.size	calcule la taille du tableau
t.length	idem
t.delete_at(n)	supprime l'élément numéro n
t.pop	supprime le dernier élément
t.last	calcule le dernier élément
t.push(donnée)	ajoute *donnée* à la fin du tableau
t.min	calcule le plus petit élément
t.max	calcule le plus grand élément
t.sort	classe les éléments en ordre croissant
t.reverse	inverse les éléments (du dernier au premier)
t.clone	crée une copie du tableau
t.each	initialise une répétitive avec tous les éléments

Deux autre méthodes **split** et **join** sont utilisées et expliquées ci-après.

4 Les plats consistants

4.1 Générateur de mots de passe syllabiques (francophones)

Un mots de passe syllabique est un mot de passe facilement prononçable (et donc plus facile à retenir) ; il est formé d'une succession de couples de lettres consonne-voyelle.
Dans un premier temps, on génère un mot de deux lettres (une syllabe) ; il suffit de reproduire plusieurs fois le couple pour des mots plus longs.

Algorithme

Un tableau de taille 20 contiendra les vingt consonnes, un tableau de taille 5 contiendra les cinq voyelles (Y et I se prononçant de la même manière, on choisira l'une ou l'autre).
On choisit un nombre aléatoire dans [0,19] qui sera l'indice d'un élément dans le tableau des consonnes et un nombre aléatoire dans [0,4] qui sera l'indice d'un élément dans le tableau des voyelles ; puis on concatène [2] (opérateur +) les deux éléments.

```
# mots-passe.rb
# mots de passe syllabiques
# -------------------------
v=['a','e', 'i', 'o','u']
c=['z','r','t','p','q','s','d']
c=c+['f','g','h','j','k','l']
c=c+['m','w','x','c','v','b','n']
syll=2 # nombre de syllabes
pw=""
for k in 1..syll
    pw=pw+c[rand(0..19)]+v[rand(0..4)]
end
puts pw
```

Remarques

- on peut aussi utiliser des lettres majuscules... ;
- les délimiteurs de chaînes de caractères sont soit les guillemets, soit l'apostrophe ;
- on a composé le tableau *c* en trois lignes pour que le code *Ruby* tienne dans le cadre... ;
- on peut aussi « compliquer » le mot de passe en lui ajoutant un nombre de 4 chiffres aléatoires (par exemple) ; dans la dernière ligne, on doit convertir le nombre aléatoire en chaîne de caractères (méthode `.to_s`) pour le concaténer à la chaîne `pw` ;

```
for k in 1..syll
    pw=pw+c[rand(0..19)]+v[rand(0..4)]
end
pw=pw+rand(1000..9999).to_s
```

2. Concaténer deux chaînes de caractères consiste à les mettre bout à bout.

- et pourquoi pas mettre en majuscule l'un des $(2 * syll)$ caractères...
  on commence par transformer le mot de passe en tableau (où chaque cellule contient un caractère du mot) ; on choisit au hasard le numéro de l'une des cellules, et on transforme le contenu correspondant en majuscule (méthode `.upcase`) ;

```
pw=pw+rand(1000..9999).to_s
work=pw.split('')
i=rand(0..2*syll-1)
work[i]=work[i].upcase
pw=work.join('')
puts pw
```

- ci-dessous le nécessaire pour comprendre `split` et `join`.

Les méthodes split et join

- la méthode **split** décompose une chaîne de caractères en un tableau où chaque cellule contient un ou plusieurs caractères, selon un séparateur éventuel.

  Soit t un tableau et c une chaîne de caractères.

  Deux exemples explicatifs :

  Exemple 1

  ```
  c="25/09/1957"
  t=c.split('/')
  ```
  va donner **t**

"25"	"09"	"1957"

  Dans c, l'oblique sera considérée comme séparateur.

  Exemple 2

  ```
  c="aeiou"
  t=c.split('')
  ```
  va donner **t**

"a"	"e"	"i"	"o"	"u"

  Dans c, il n'y a pas de caractère particulier qui peut faire office de séparateur.

  Dans le code de *mots-de-passe.rb*, on pourrait remplacer la construction des tableaux v et c en frappant les lettres qui se suivent sur le clavier, ce qui est plus facile à écrire.

```
voyelles='aeiou'
v=voyelles.split('')
consonnes='zrtpqsdfghjklmwxcvbn'
c=consonnes.split('')
```

- la méthode **join** est la réciproque de la méthode **split** ; elle crée une chaîne de caractères avec tous les éléments d'un tableau en les reliant par un éventuel séparateur choisi.

  Soit **t**

"d"	"e"	"s"	"b"	"o"	"n"	"n"	"e"	"z"

  `c=t.join('-')` va créer la chaîne "d-e-s-b-o-n-n-e-z"

  `c=t.join('')` va créer la chaîne "desbonnez"

4.2 Le tri à bulles

Les algorithmes de tri (classement alphabétique ou numérique) ont toujours été des applications très importantes en informatique.

En effet, il y a quantité de situations où une liste triée est indispensable pour la **recherche rapide** d'informations : dictionnaire, annuaire de téléphone, ... de manière générale toute liste dans laquelle il faut repérer le plus rapidement possible une information précise.

Il existe de très nombreux algorithmes de tri (faire une recherche avec votre moteur de prédilection...), impossible de ne pas citer le *tri à bulles*, le *tri rapide* (quicksort), le *tri par tas* (heapsort), ... la quête étant toujours la recherche de la méthode *la plus rapide*.

L'algorithme du *tri à bulles* est certes peu performant, mais il est facile à programmer. Just for fun !

Certes, il y a la *méthode* **tableau.sort**, bien plus performante, mais derrière toute *méthode*, il y a un algorithme qui ne dort pas !

Idée

On compare le premier élément du tableau avec le second ; si le second est plus petit, on les permute ; et ainsi de suite avec le troisième, le quatrième, ...
après toutes les comparaisons, le plus petit élément du tableau est *remonté* en première place, comme une **bulle** dans certains liquides ...

On recommence ensuite avec le second élément du tableau, que l'on compare avec tous les suivants, en permutant éventuellement, ..., et après toutes les comparaisons, le « second plus petit » élément du tableau se retrouve en seconde place.

Et ainsi de suite, c'est simple, mais long, vu le nombre de comparaisons et autant de permutations potentielles.

Permuter les contenus de deux variables

Explication imagée : pour permuter les contenus d'un verre de coca et d'un verre de jus d'orange, il faut un troisième verre, et trois manipulations :

verre trois ← verre de coca
verre de coca ← verre jus orange
verre jus orange ← verre trois

Algorithme

Commençons par énumérer les différentes comparaisons dans un tableau **t** de cinq éléments (et bien entendu, à chaque comparaison, il y a éventuellement permutation).

étape 0

$t[0] \leftrightarrow t[1]$
$t[0] \leftrightarrow t[2]$
$t[0] \leftrightarrow t[3]$
$t[0] \leftrightarrow t[4]$

t[0] bien placé

étape 1

$t[1] \leftrightarrow t[2]$
$t[1] \leftrightarrow t[3]$
$t[1] \leftrightarrow t[4]$

t[1] bien placé

étape 2

$t[2] \leftrightarrow t[3]$
$t[2] \leftrightarrow t[4]$

t[2] bien placé

étape 3

$t[3] \leftrightarrow t[4]$

t[3] et t[4] bien placés

Il faudra donc un compteur (`etape`) pour énumérer les différentes étapes, et, dans chaque étape, un second compteur (`comp`) pour énumérer les comparaisons.
Il est donc alors aisé de généraliser l'algorithme pour un tableau t de taille n.

```
pour etape de 0 à 3
   pour comp de etape+1 à 4
      si t[etape] > t[comp]
         permuter t[etape] et t[comp]
      fin
   fin
fin
```

```
pour etape de 0 à n-2
   pour comp de etape+1 à n-1
      si t[etape] > t[comp]
         permuter t[etape] et t[comp]
      fin
   fin
fin
```

Dans l'algorithme complet ci-dessous, on a créé un tableau contenant 10000 nombres aléatoires, et calculé le temps nécessaire à son tri (environ 6.6 secondes [3]).

```
# tri-bulle.rb
# ------------
n=10000
t=Array.new(n){rand}
start=Time.now
for etape in 0..n-2
    for comparaison in etape+1..n-1
      if t[etape] > t[comparaison]
         a=t[etape]
         t[etape]=t[comparaison]
         t[comparaison]=a
      end
    end
end
stop=Time.now
puts "duree : #{stop-start} seconde(s)"
puts t
```

Et je me souviens du temps (en l'an de grâce 1983) où il fallait, avec cette méthode, plus d'une heure pour trier alphabétiquement les 450 noms des élèves de l'école...

4.3 La recherche dichotomique

Lorsqu'une liste (un tableau) n'est pas triée, et qu'il faut y rechercher un élément particulier, la seule méthode est la recherche dite *séquentielle* ; elle consiste à parcourir les éléments de la liste

3. Dépend de l'ordinateur, évidemment.

à partir du premier, et de s'arrêter soit lorsqu'on a trouvé l'élément cherché, soit lorsqu'on a atteint la fin de la liste. Cela peut être très long ; pourrait-on imaginer un annuaire de téléphone non classé ?

Lorsque la liste est triée, il en va tout autrement : on divise la liste initiale en deux sous-listes en son milieu ; l'élément cherché est soit au milieu, soit dans la liste « inférieure », soit dans la liste « supérieure ». S'il n'est pas au milieu, on divise en deux soit la liste « inférieure », soit la liste « supérieure » ; et ainsi de suite jusqu'au moment où l'élément a été trouvé, ou lorsqu'il n'est plus possible de diviser, auquel cas il n'existe pas dans la liste.

Cette méthode de recherche, très rapide, est appelée *dichotomie*, du grec *dikhotomia*, division en deux.

Au départ, les bornes de l'intervalle de recherche sont 0 (min) et $taille-1$ (max) ; [0,10] dans l'exemple ci-contre. Rappelons que les indices des tableaux commencent à 0.
La position *milieu* est 5 ; si l'élément cherché n'est pas en 5e position, il est soit en $[0,4]$, soit en $[6,10]$.
La boucle de dichotomie doit s'arrêter soit lorsque l'élément cherché est trouvé, soit lorsque les bornes se sont « croisées ».

```
# dichotomie.rb
# -------------
t=[9,15,21,31,40,57,68,72,89,90,112]
# le tableau t DOIT être trié !!
min=0
max=t.size-1
trouve=false
x=31 # élément cherché
while trouve==false and min<=max
   milieu=(min+max)/2
   case
      when x==t[milieu]
         trouve=true
         position=milieu
      when x<t[milieu]
         max=milieu-1
      else
         min=milieu+1
   end
end
if trouve
   puts "trouve en position #{position}"
else
   puts "#{x} n'est pas dans le tableau !"
end
```

Cette méthode peut être également utilisée pour rechercher une valeur approchée d'une racine d'une fonction $f(x)$ continue sur un intervalle $[min,max]$ où l'on sait que $f(min)\cdot f(max)<0$.

4.4 Fusion de 2 tableaux triés

À partir de deux tableaux triés (et contenant des éléments distincts), il faut en créer un troisième, trié lui aussi, contenant les éléments du premier, ceux du second, mais une seule fois les éléments communs aux deux.

Algorithme

Soit $t1$ et $t2$ à fusionner en $t3$; les index $i1$ et $i2$ (initialisés à 0) vont parcourir respectivement $t1$ et $t2$ jusqu'au moment où la fin de l'un des tableaux a été atteinte. A chaque comparaison entre deux éléments, le plus petit est copié dans $t3$ et l'index correspondant est incrémenté de une unité. On avance ainsi pas à pas dans chacun des deux tableaux. En cas d'égalité, l'un des deux éléments (au choix) est copié dans $t3$, et les deux index sont incrémentés. Lorsque la fin de l'un des tableaux a été atteinte, il reste à copier le reste de l'autre dans $t3$.

```
# fusion.rb
# ---------
t1=[7,9,14,17,51,60]
t2=[5,9,10,14,17,60,61,62]
t3=[ ]
i1=0
i2=0
fin1=t1.size-1
fin2=t2.size-1
while i1<=fin1 and i2<=fin2
   case
      when t1[i1]<t2[i2]
         t3.push(t1[i1])
         i1=i1+1
      when t1[i1]>t2[i2]
         t3.push(t2[i2])
         i2=i2+1
      else # égalité
         t3.push(t1[i1]) #ou t2[i2]
         i1=i1+1
         i2=i2+1
   end
end
# fin de l'un des 2 tableaux
# ou des deux simultanément
if i1>fin1 and i2<=fin2# fin t1  pas t2
   for k in i2..fin2
      t3.push(t2[k])
   end
elsif i2>fin2 and i1<=fin1# fin t2 pas t1
   for k in i1..fin1
      t3.push(t1[k])
   end
end
puts t1; puts t2; puts t3
```

4.5 Suppression des doublons dans un tableau trié

On appellera « doublon » un élément qui est présent **au moins deux fois** dans le tableau. Comme le tableau est supposé trié, les doublons se suivent nécessairement.

Algorithme

On regarde chaque élément du tableau et le suivant ; s'ils sont identiques, on supprime le suivant, et on fait un « pas en arrière » si jamais le suivant avait plus de deux occurrences ; dans le cas contraire, on fait un « pas en avant ». Et ainsi de suite jusqu'à la fin du tableau.

La suppression d'un élément d'un tableau se fait par l'instruction

tableau.delete_at(position)

Il s'agit bien d'une suppression physique, et la taille du tableau est ajustée en conséquence.

```
# doublons.rb
# -----------
t=[9,9,11,11,12,13,13,13,13]
t=t+[13,13,20,25,25,25]
index=0
while index <=t.size-1
   if t[index]==t[index+1]
      t.delete_at(index+1)
      index=index-1
   end
   index=index+1
end
puts t
```

Une application « scolaire » (avant l'ère d'internet) : une école doit envoyer un courrier aux parents ; dans le même établissement, il y a souvent des frères et sœurs, habitant sous le même toit, et pour lesquels un seul courrier suffit.

On copie le « fichier élèves » dans un tableau, on le trie selon les adresses, on supprime les doublons, puis on imprime des étiquettes à partir du tableau résultant.

4.6 Trouver les éléments communs à 2 tableaux triés

A partir de deux tableaux triés, en créer un troisième qui contiendra les éléments communs aux deux.

Algorithme

Cet algorithme ressemble très fort à celui de la fusion de deux tableaux triés ; on avance pas à pas dans chacun des deux tableaux, et en cas d'égalité entre deux éléments, on ajoute l'un d'eux au troisième tableau (qui sera forcément trié lui aussi). Lorsque la fin de l'un des tableaux est atteinte, c'est terminé.

```
# communs.rb
# ----------
t1=[7,9,14,17,51,60]
t2=[5,9,10,14,17,60,61,62]
t3=[ ]
i1=0
i2=0
fin1=t1.size-1
fin2=t2.size-1
while i1<=fin1 and i2<=fin2
   case
      when t1[i1]<t2[i2]
         i1=i1+1
      when t1[i1]>t2[i2]
         i2=i2+1
      else # égalité
         t3.push(t1[i1]) # ou t2[i2]
         i1=i1+1
         i2=i2+1
   end
end
puts t1; puts t2; puts t3
```

5 Les desserts

5.1 L'original : le plus grand commun diviseur

L'idée n'est pas de concurrencer la *méthode d'Euclide*, loin de là, mais simplement d'appliquer quelques manipulations de tableaux expliquées dans les quelques pages qui précèdent.

Algorithme

Soit a et b deux naturels non nuls dont on souhaite calculer le plus grand commun diviseur.
Dans un tableau $t1$ on calculera tous les diviseurs de a (sa construction fera en sorte qu'il sera trié) ; de même, le tableau $t2$ contiendra tous les diviseurs de b.
Dans un troisième tableau *divcommuns* on mettra les éléments communs à $t1$ et $t2$.
Ce tableau est trié, et donc son dernier élément est le plus grand commun diviseur.

```
# pgcd.rb
# --------
puts "premier nombre ?"
a=gets.to_i
t1=[ ]
for i in 1..a
   if a%i == 0
      t1.push(i)
   end
end
# ----------------------
puts "second nombre ?"
b=gets.to_i
t2=[ ]
for i in 1..b
   if b%i == 0
      t2.push(i)
   end
end
# -------------------------
t3=[ ]
i1=0
i2=0
fin1=t1.size-1
fin2=t2.size-1
while i1<=fin1 and i2<=fin2
   case
      when t1[i1]<t2[i2]
         i1=i1+1
      when t1[i1]>t2[i2]
         i2=i2+1
      else # égalité
         t3.push(t1[i1]) # ou t2[i2]
         i1=i1+1
         i2=i2+1
   end
end
puts "le pgcd est #{t3.last}"
```

5.2 La recette de grand mère : le crible d'Eratosthène

Eratosthène est un mathématicien de l'antiquité grecque.
La méthode dite du *crible d'Eratosthène*, encore appelée *tamis d'Eratosthène* est une manière élégante de calculer une liste de nombres premiers, qui jouent un rôle important dans les algorithmes de cryptage, entre autres.

Le premier nombre premier est 2. Pour les autres, il suffit de regarder ce qui suit.

Idée générale

Dans un tableau, on mémorise les nombres naturels de 2 à N.
On commence par supprimer tous les multiples de 2, sauf lui-même, puisque, par définition, ils ne sont pas premiers.
Puis on supprime tous les multiples de 3, sauf lui-même, pour la même raison.
4 n'est déjà plus dans la liste.
Puis tous les multiples de 5, sauf lui-même, et ainsi de suite.
A la fin de l'opération, il ne reste plus que les nombres premiers compris entre 2 et N.

Illustration pour les nombres premiers entre 2 et 50 :

2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28
29	30	31	32	33	34	35	36	37
38	39	40	41	42	43	44	45	46
47	48	49	50					

multiples de 2
multiples de 3
multiples de 5
multiples de 7

Préparation de l'algorithme

Décrivons les premières étapes afin de délimiter les différentes répétitives, avec un tableau contenant les 100 premiers naturels (exceptés 0 et 1).

2 est premier, on supprime tous ses multiples, sauf lui-même :
2×2 , 2×3 , 2×4 , 2×5 ,...., 2×50
Il faut s'arrêter à 50, puisque 2×51 dépasse la limite du tableau (100).
Remarquons que $50 = \frac{100}{2}$.

3 n'est pas supprimé ; il est premier, on supprime tous ses multiples, sauf lui-même :
3×2 , 3×3 , 3×4 , 3×5 ,...., 3×33
Il faut s'arrêter à 33, puisque 3×34 dépasse la limite du tableau.
Remarquons que $33=\frac{100}{3}$ (la partie entière de $\frac{100}{3}$, puisqu'au sens *Ruby* du terme, le quotient de deux entiers est un entier).

4 est déjà supprimé (multiple de 2) ; on passe donc au suivant.

5 est premier, on supprime tous ses multiples, sauf lui-même :
5×2 , 5×3 , 5×4 , 5×5 ,...., 5×20
Il faut s'arrêter à 20, puisque 5×21 dépasse la limite du tableau.
Remarquons que $20=\frac{100}{5}$.

6 est déjà supprimé...

7 est premier, on supprime tous ses multiples, sauf lui-même :
7×2 , 7×3 , 7×4 , 7×5 ,...., 7×14
Il faut s'arrêter à 14, puisque 7×15 dépasse la limite du tableau.
Remarquons que $14=\frac{100}{7}$, au sens *Ruby* du terme.

8 (multiple de 2), 9 (multiple de 3) et 10 (multiple de 2) sont déjà supprimés...

Dans un premier temps, on voit assez aisément se profiler la « boucle de suppression » : si k est le nombre traité, on supprime tous ses multiples de 2 à $\frac{100}{k}$

Reste à voir combien de valeurs de k il faut traiter...

11 est en théorie le suivant, et il faudrait supprimer tous ses multiples :
11×2 , 11×3 , 11×4 , 11×5 ,...., 11×9
mais ils le sont déjà !

Le dernier nombre à traiter est 10, soit $\sqrt{100}$
$\underbrace{10\times 2 \quad , \quad 10\times 3 \quad , \quad 10\times 4 \quad , \quad 10\times 9}_{\text{déjà supprimés}} \quad , \quad \underbrace{10\times 10}_{\text{maximum}}$

On voit se profiler également la « boucle de traitement des nombres k » : il faut traiter tous les nombres k de 2 à $\sqrt{100}$, c'est à dire tous les nombres k supérieurs à 2 et tels que $k^2\leqslant 100$

Algorithme

$k \leftarrow 2$
tant que $k^2 \leqslant 100$
    si k « pas encore supprimé »
        pour i de 2 à $\frac{100}{k}$
            « supprimer l'élément d'indice $i \times k$ »
        fin
    fin
    $k \leftarrow k+1$
fin

Plus subtil...

On utilise un tableau de *valeurs booléennes* (true, false) où toutes les cellules sont initialisées à *true* (par défaut, tous les nombres sont supposés premiers).
Lorsque l'**indice** de la cellule n'est pas un nombre premier, on affecte à la cellule correspondante la valeur *false*.
Avec cette convention, on ne traite que les indices du tableau ; on a donc les équivalences suivantes, en supposant que le tableau s'appelle **t** :

si k « pas encore supprimé » $\Leftrightarrow$ if (t[k])
« supprimer élément d'indice $i \times k$ » $\Leftrightarrow$ t[i*k] = false

Et pour écrire la liste des nombres premiers, il suffira d'afficher les indices des cellules dont le contenu est *true*.

```
# eratosthene.rb
# --------------
t=Array.new(100){true}
k=2
while k**2<=100
   if t[k]
      for i in 2..100/k
         t[i*k]=false
      end
   end
   k=k+1
end
for i in 2..100
   if t[i]
      puts i
   end
end
```

6 Le pousse-café : un peu de calcul vectoriel

Dans l'espace, on peut assimiler la composante d'un vecteur $\overrightarrow{v}(x_v, y_v, z_v)$ à une variable tableau de taille 3 :

v	v[0]	v[1]	v[2]

Dans le plan, il suffit de supprimer la troisième composante.

Les formules traditionnelles, dans un repère orthonormé :

$\|\overrightarrow{v}\| = \sqrt{x_v^2 + y_v^2 + z_v^2}$

$\overrightarrow{v} \bullet \overrightarrow{w} = x_v \cdot x_w + y_v \cdot y_w + z_v \cdot z_w$

$\cos(\widehat{\overrightarrow{v}, \overrightarrow{w}}) = \frac{\overrightarrow{v} \bullet \overrightarrow{w}}{\|\overrightarrow{v}\| \cdot \|\overrightarrow{w}\|}$

Algorithme

Pour simplifier, on assigne directement les coordonnées des deux vecteurs dans deux tableaux nommés v et w ; il suffira de modifier ces deux lignes pour traiter d'autres vecteurs.
On calcule ensuite les normes, le produit scalaire et le cosinus de l'angle déterminé par les deux vecteurs.
Enfin, on utilise une alternative multiple pour étudier les positions relatives des deux vecteurs.

```
# vectoriel.rb
# ------------
v=[1,5,-8]
w=[7,0,7.6]
nv=Math.sqrt(v[0]**2+v[1]**2+v[2]**2)
nw=Math.sqrt(w[0]**2+w[1]**2+w[2]**2)
ps=v[0]*w[0]+v[1]*w[1]+v[2]*w[2]
cos=ps/(nv*nw)
puts "norme de v : #{nv}"
puts "norme de w : #{nw}"
puts "produit scalaire : #{ps}"
puts "angle (radian) : #{Math.acos(cos)}"
puts "angle (degre) : #{Math.acos(cos)*180/Math::PI}"
case
   when ps==0 then puts "vecteurs orthogonaux"
   when cos==-1 then puts "vecteurs paralleles de sens contraires"
   when cos==1 then puts "vecteurs paralleles de memes sens"
   else puts "vecteurs quelconques"
end
```

7 La digestion : sauvegarde/réutilisation des résultats

Les résultats affichés dans le terminal de commandes disparaissent à la fermeture de la fenêtre. Que faire si on souhaite les mémoriser ou les utiliser à d'autres fins ?

Sous MacOsX

On peut exporter le contenu complet de la fenêtre sous la forme d'un fichier de type « texte » :
`Shell > Exporter le texte sous... > fichier.txt`

Il suffit ensuite d'ouvrir ce fichier, et de faire un « copier-coller » des lignes à conserver dans un autre document de type traitement de texte, ou tableur, ou...
Attention : s'il s'agit d'une liste de nombres décimaux (en *Ruby* le séparateur décimal est le point), ceux-ci auront le statut de « texte » dans les cellules d'un tableur, et il faudra peut-être remplacer le point par une virgule pour leur donner le statut de « nombre » afin de pouvoir les impliquer dans d'autres calculs éventuels :

`Edition > Rechercher et remplacer...`

Sous Windows (8, sûr, les autres, à tester)

`Clic droite dans la fenêtre > Sélectionner > sélectionner les lignes`

Puis faire un « copier-coller » dans un autre document, et idem que ci-dessus.

Dériver avec Géogébra

Résumé. *En cinquième année, au cours de mathématiques 4 périodes par semaine, le chapitre consacré aux limites est celui qui passe le plus mal (expérience personnelle) ; ϵ et δ sont depuis longtemps passés à la trappe, et quelques semaines plus tard il faudra parler de dérivées. Prof et élèves s'arrachent les cheveux.*
À l'aide du logiciel Géogébra, il est possible d'introduire assez facilement le concept de pente d'une tangente, en tout cas pour une fonction « simple » comme $f(x) = x^2$, sans parler immédiatement de limite. Par contre, lorsqu'il faudra introduire le concept d'un nombre « proche de 0 », il faudra forcer un peu pour obtenir par exemple 0,00001 ; il semble assez surnaturel pour un élève de choisir un nombre autre qu'un naturel non nul supérieur à 10 . . .

1 Géogébra, un « super-tableau » blanc

Tous les élèves ont à ce jour accès à un ordinateur (si pas à la maison, ils peuvent utiliser ceux de l'école), le logiciel *Géogébra* est gratuit et facile à installer, donc pas d'excuse.

En 50 minutes de cours, il est possible d'en expliquer les rudiments nécessaires aux manipulations qui vont suivre ; il faut savoir :

- dessiner le graphique d'une fonction (*) ;
- afficher un point à partir de ses coordonnées (*) ;
- calculer et représenter la pente d'une droite (**) ;
- dessiner une droite passant par 2 points donnés (**) ;
- utiliser les outils zoom avant et zoom arrière (**) ;
- déplacer le graphique sur l'écran (**) ;
- afficher la grille sur la zone de dessin (***) ;
- afficher une précision de calcul d'au moins 10 décimales (***).

(*) frappe d'une commande dans la zone de saisie, (**) utilisation d'un outil de la barre du même nom, (***) des options à cocher.

Les outils, menus et options pouvant varier d'une version à l'autre du logiciel, ils ont été volontairement non détaillés ici.

Pour bien profiter de la suite, l'élève doit être capable de calculer une équation d'une droite à partir d'un point et de la pente (c'est vu l'année précédente, mais . . . nous connaissons la chanson).

2 Première approche

On place un point choisi sur le graphique de $f(x) = x^2$; on fait un zoom avant assez fort en ce point jusqu'à ce que l'on voit une droite, dont il est aisé d'estimer la pente grâce à la grille ; on calcule une équation de cette droite et on la dessine ; on fait enfin un zoom arrière pour revenir à la situation de départ.

La majorité des élèves voient qu'ils ont dessiné une tangente au graphique au point choisi.

Détails des commandes et des manipulations

- si ce n'est déjà fait, afficher la grille (***) ;
- dessiner le graphique de la parabole $f(x) = x^2$ (*) ;
- afficher un point sur le graphique $(1.5, f(1.5))$ (*) ;
- zoom avant sur le point (qui a été nommé A) (**) ;
- faire calculer la pente de la droite (repérer un triangle rectangle dans lequel on peut évaluer Δx et Δy grâce aux coins du quadrillage ; la pente vaut 3 ;
- faire calculer une équation de la droite ; $y = 3x - 2.25$;
- dessiner le graphique de la droite $y = 3x - 2.25$ (*) ;
- zoom arrière (**) ;
- regarder la tangente.

Il suffit de recommencer le processus avec une dizaine de points bien répartis sur la parabole, par exemple des points d'abscisses -3, -2.5, -1, -0.5, 0, 0.5, 1, 2, 3, ...

Les élèves voient assez vite le lien entre la pente de la tangente et l'abscisse du point ($pente = 2 \times abscisse$). Il reste alors à mettre en place le vocabulaire *nombre dérivé*, *fonction pente*, *dérivée*, et la notation $\left(x^2\right)' = 2x$.

En prime, c'est le cas de le dire, lorsqu'on masque la graphique de la parabole, il reste l'*enveloppe*, l'ensemble des tangentes.

3 Seconde approche

Cette seconde approche permet d'illustrer le passage d'une sécante à une tangente, et donc un échauffement précédant le calcul $f'(x_A) = \lim_{h \to 0} \frac{f(x_A+h)-f(x_A)}{h}$.

Il faut ajouter aux pré-requis la manière de définir une variable : en tapant dans la zone de saisie l'expression $h = 1$ par exemple, on définit la variable h ayant pour valeur 1. Rien n'est dessiné, mais on la voit apparaître dans la zone *algèbre* de l'écran [1].

1. Dans la version 5 de *Géogébra*, l'activation de l'affichage crée sur l'écran un *curseur* qui permet de faire varier le contenu de la variable ; en modifiant les propriétés du curseur, on peut le faire varier, par exemple, de 0 à 1 par pas de 0,001.

On fixe un point sur le graphique, à partir de coordonnées déterminées par une variable, ainsi qu'un second point à partir de coordonnées déterminées également à partir de variables, mais dépendantes du premier point (les deux points sont ainsi liés) ; on dessine la sécante passant par ces deux points et on affiche la valeur de la pente de cette sécante ; on s'amuse alors à faire varier la coordonnée du second point, et on regarde l'évolution de la pente.

Détails des commandes et des manipulations

- activer l'option de calcul avec 10 ou 15 décimales (***) ;
- dessiner le graphique de la parabole $f(x) = x^2$ (*) ;
- définir la variable xA qui sera l'abscisse du premier point, $xA = 1.5$ par exemple (*) ;
- afficher sur le graphique le point de coordonnées $(xA, f(xA))$, $(xA, f(xA))$ (*) ;
- définir la variable h, $h = 1$ (*) ;
- afficher sur le graphique le second point, $(xA + h, f(xA + h))$ (*) ;
- dessiner la droite (la sécante) qui passe par ces 2 points (**) ;
- calculer et afficher la pente de cette droite (**) ;
- donner ensuite à h des valeurs de plus en plus proches de 0, $h = 0.1$, puis $h = 0.01$, $h = 0.00001$ (*), etc. et regarder évoluer la pente de la sécante (pente à droite) ;
- il est aussi intéressant de taper $h = 0$: la droite disparaît et la pente n'est plus définie, puisque les 2 points sont identiques ;
- pour la pente à gauche, il suffit de prendre comme second point $(xA - h, f(xA - h))$ (*) ;

En notant les différents résultats intermédiaires des valeurs des pentes en fonction de h, on voit la pente de la sécante se transformer en pente de la tangente, ce qui permet de faire passer plus facilement les pilules $f'(x_A) = \lim_{h\to 0} \frac{f(x_A+h)-f(x_A)}{h}$ et $f'(x_A) = \lim_{h\to 0} \frac{f(x_A-h)-f(x_A)}{-h}$.

La première approche devient plus difficile si la fonction $f(x)$ devient plus complexe, les droites ne passant plus nécessairement par des coins du quadrillage, d'où la difficulté d'estimer les pentes. La seconde approche donne, quant à elle, toujours de bons résultas numériques, mais le lien entre l'abscisse d'un point et la pente de la tangente n'est pas toujours aussi évident à voir qu'avec $f(x) = x^2$.

On peut par contre s'en servir pour vérifier si un $f'(x)$ et un $f'(a)$ calculés au moyen des formules de dérivation correspond à celui évalué avec *Géogébra*.

von Koch, la Tortue et Ruby, une rencontre du troisième type

Résumé. *Helge* VON KOCH *(1870–1924) est un mathématicien suédois.*

Il a décrit une courbe « sans tangente » appelée « ligne fractale de VON KOCH *» à partir de laquelle on construit un autre objet fractal, appelé « flocon de* VON KOCH *».*

Ces deux objets sont construits en appliquant une infinité de fois les mêmes transformations géométriques (élémentaires) respectivement à un segment et à un triangle équilatéral.

Ils constituent aussi une belle application sur les suites géométriques.

Dans cet article, on se propose de les dessiner en utilisant la Géométrie de la tortue ([1]) *et de faire des calculs avec le langage de programmation Ruby.*

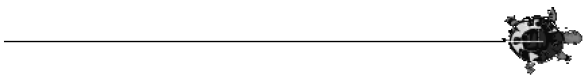
1 La transe de la Tortue

1.1 La ligne fractale

Un début en douceur ...

- étape 0 : on dessine un segment de longueur choisie, que nous noterons :*long* en accord avec le langage Tortue.

```
av :long
```

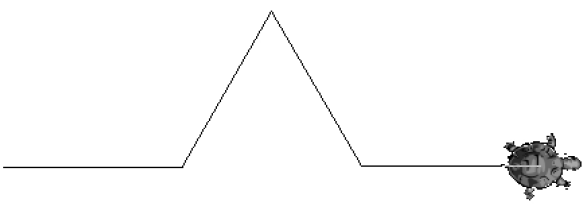


- étape 1 : on partage le segment de l'étape 0 en 3 parts égales (donc de longueur :*long*/3), on ôte la partie centrale et on la remplace par les deux tiers d'un triangle équilatéral (voir image).

  Les instructions pour la Tortue sont élémentaires ([1] pp. 45–52), le débutant choisit une valeur pour :*long*, et doit calculer :*long*/3.

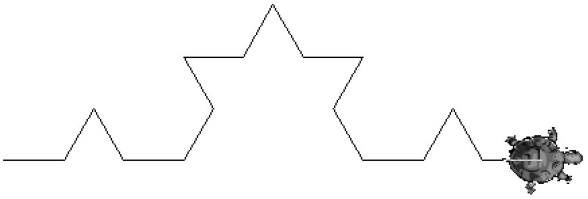
```
av :long/3
tg 60
av :long/3
td 120
av :long/3
tg 60
av :long/3
```

ligne 1



- étape 2 : on applique à chaque segment de l'étape 1 le même processus que celui utilisé pour passer de l'étape 0 à l'étape 1, en divisant par 3 la longueur de l'étape 1, celle-ci étant devenue la nouvelle longueur de référence.

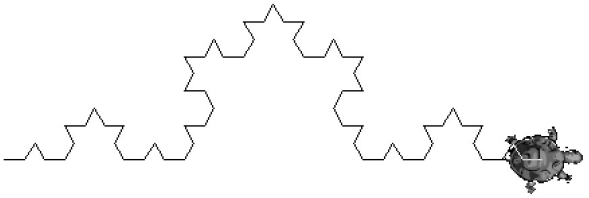
```
ligne 1
tg 60
ligne 1
td 120
ligne 1
tg 60
ligne 1
```
} ligne 2



Si on reste au niveau « débutant », en ayant fixé une valeur initiale :*long*/3, on en est à une longueur qui vaut :*long*/9 ; la procédure « ligne 2 » contient 31 instructions (4×7+3), certes simples, mais ...

- étape 3 : on applique à chaque segment de l'étape 2 le même processus que celui utilisé pour passer de l'étape 1 à l'étape 2, en divisant par 3 la longueur de l'étape 2, celle-ci étant devenue la nouvelle longueur de référence.

```
ligne 2
tg 60
ligne 2
td 120
ligne 2
tg 60
ligne 2
```
} ligne 3



La procédure « ligne 3 » contient 127 instructions $(4\times 31+3)$, et ainsi de suite pour les étapes 4 (511 instructions), 5 (2047 instructions), 6 (8191 instructions), ... de quoi décourager un débutant ; il est donc temps de changer de vitesse.

Puis un coup d'accélérateur ...

Il faut être bien conscient qu'au départ, on décide de dessiner une ligne, d'ordre 3 (par exemple). Pour ce faire, il faut savoir dessiner la ligne d'ordre 2, et pour dessiner la ligne d'ordre 2 il faut savoir dessiner la ligne d'ordre 1, et pour dessiner la ligne d'ordre 1 il faut savoir dessiner la ligne d'ordre 0. Stop.

À chaque étape, on enlève 1 au numéro d'ordre, et on divise une longueur par 3. On voit ainsi se profiler une procédure *ligne* qui dépend de 2 variables, longueur et ordre, que nous noterons :*longueur* et :*ordre*, ce qui se traduira en langage Tortue :

```
pour ligne :longueur :ordre
```

Mais, la ligne d'ordre 3 doit faire appel à la ligne d'ordre 2, qui doit faire appel à la ligne d'ordre 1, qui doit faire appel à la ligne d'ordre 0. Ce processus s'appelle (informatiquement) *la récursivité*. C'est facile à comprendre : la *récursivité*, c'est la *récursivité*.

Trêve de plaisanteries. La *récursivité* est le fait qu'une procédure peut s'appeler elle-même ; c'est un délire d'informaticien, extrêmement puissant, mais qui consomme beaucoup de mémoire.

Principe de base : la première instruction d'une procédure récursive doit être un test d'arrêt, sinon c'est parti pour l'éternité !

Dans le cas qui nous préoccupe, la procédure *ligne* doit arrêter de s'appeler lorsqu'on a atteint l'étape 0.

Le kit de dépannage : Tortue et la structure alternative

si condition [instruction(s) cas vrai] [instruction(s) cas faux]

ce qui va donner :

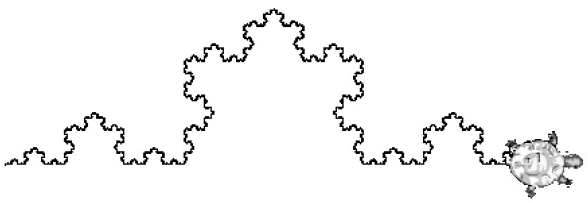
```
pour ligne :longueur :ordre
      si :ordre = 0 [av :longueur] [(*)]
fin
```

Et tant qu'on n'a pas atteint l'ordre 0, il faut diviser la longueur par 3, et retirer 1 à l'ordre, ce qui se traduit par :

```
ligne :longueur/3 :ordre-1
tg 60
ligne :longueur/3 :ordre-1
td 120
ligne :longueur/3 :ordre-1
tg 60
ligne :longueur/3 :ordre-1
```

(*)

Voici le résultat pour l'instruction « ligne 400 6 »



Remarques

- l'unité de longueur est le *pixel* ; le dessin sera illisible si on choisit une valeur trop petite pour la variable :*longueur*, d'autant plus qu'elle est divisée par 3 à chaque étape ;

- le temps de calcul et la quantité de mémoire nécessaire augmentent très vite avec la grandeur de la variable :*ordre* ;

  pour info, avec 64 Mo de mémoire allouée au logiciel, la Tortue s'arrête à mi-chemin de « ligne 400 10 » ; en lui accordant 1Go, elle a bien voulu continuer, mais il lui a fallu environ 15 minutes, ce qui est énorme pour une tortue bionique.

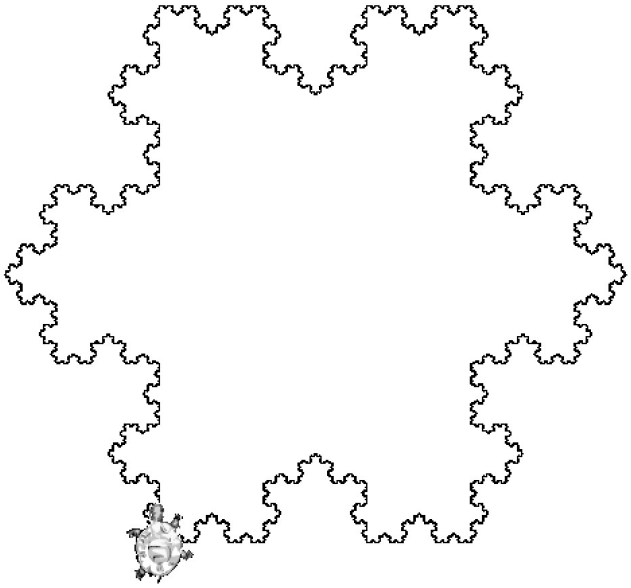
- on ne sait pas le montrer sur une image fixe, mais durant le tracé, la Tortue, animal réputé calme et lent, est animée d'un *mouvement vibratoire frénétique*, impressionnant à voir ; de là à dire que la courbe n'a pas de tangente, il n'y a qu'un pas de Tortue à franchir.

1.2 Le flocon

Le flocon de VON KOCH est obtenu en appliquant le principe de la ligne du même nom aux 3 côtés d'un triangle équilatéral. Le flocon dépendra donc aussi des 2 variables :*longueur* et :*ordre*. Sa construction est très facile, à partir du moment où l'on sait tracer une ligne :

```
pour flocon :longueur :ordre
      repete 3 [ligne :longueur :ordre td 120] fin
```

Ci-contre « flocon 400 6 ».



2 Les calculs

2.1 La ligne

Les quantités que l'on peut calculer sont, pour chaque étape, et en prenant 1 comme longueur initiale,

- le nombre de segments $(A_n) \equiv \begin{cases} A_0 = 1 \\ A_{n+1} = 4A_n \end{cases}$ suite géométrique de raison 4 ;
- la longueur d'un segment $(B_n) \equiv \begin{cases} B_0 = 1 \\ B_{n+1} = \frac{1}{3}B_n \end{cases}$ suite géométrique de raison $\frac{1}{3}$;
- la longueur de la ligne $(C_n) \equiv C_n = A_n \cdot B_n$.

Bien sûr, on peut programmer (ou calculer directement) ces valeurs à l'étape n en utilisant les formules issues de la théorie des suites, à savoir $A_n = 4^n$, $B_n = \left(\frac{1}{3}\right)^n$ et $C_n = \left(\frac{4}{3}\right)^n$.

On profite cependant de l'occasion pour montrer l'algorithme de calcul des termes successifs d'une suite définie par récurrence.

```
longueur_init ← 1.0
etape ← ...   (⩾ 2)
a ← 1
b ← longueur_init
pour k de 0 à etape − 1
    a ← a ∗ 4
    b ← b/3
fin
écrire etape, a, b, a ∗ b
```

```
# ligne-von-koch.rb
# -----------------
longueur_init=1.0 ; etape = 100
a = 1 ; b = longueur_init
for k in 0..etape-1
   a = a*4 ; b = b/3
end
puts "apres #{etape} iterations,"
puts "nombre de segments = #{a}"
puts "longueur d'un segment = #{b}"
puts "longueur de la ligne = #{a*b}"
```

Pour 100 itérations, on obtient les résultats suivants :

nombre de segments = 1606938044258990275541962092341162602522202993782792835301376 [1]
longueur d'un segment = 1.9403252174826334e-48
longueur de la ligne = 3117982410207.943

2.2 Les prolongations

mathématiques les suites A_n et B_n ont respectivement pour limite $(+\infty)$ et (0) ; la suite C_n a pour limite $(+\infty)$, ce qui permet d'illustrer le cas d'indétermination $(0)\cdot(+\infty)$.

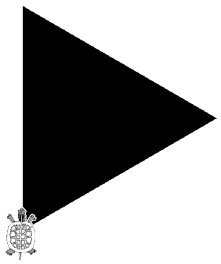
techniques les fractals sont d'excellents procédés pour miniaturiser les antennes de nos « chers » GSM ; après 20 étapes par exemple, un segment de longueur initiale 1 cm est devenu 3,15 m.

2.3 Le flocon

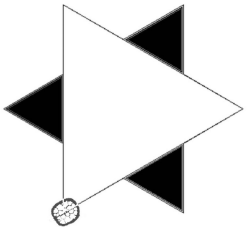
En ce qui concerne son périmètre à l'étape n, il suffit de multiplier par 3 les résultats obtenus pour la ligne. On laissera donc au lecteur le soin de s'en occuper.

Intéressons-nous à l'*aire* du flocon à l'étape n.

À l'étape 0, partons d'un triangle équilatéral de côté 1, et donc d'aire égale à $\frac{1}{2}\cdot 1^2\cdot\sin\frac{\pi}{3}$ (par formule de l'aire par le sinus).

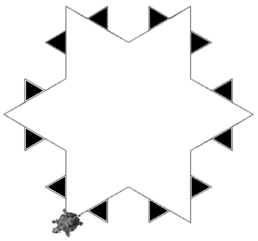


À l'étape 1, on ajoute l'aire de 3 triangles équilatéraux de côté $\frac{1}{3}$, soit une aire supplémentaire de $3\cdot\frac{1}{2}\cdot\left(\frac{1}{3}\right)^2\cdot\sin\frac{\pi}{3}$.



1. . . .six.

À partir de l'étape 2, on ajoute l'aire d'autant de triangles équilatéraux qu'il y a de de segments (le nombre de segments est multiplié par 4 à chaque itération) dont la longueur est divisée par 3 à chaque itération, soit, pour l'étape 2, une aire supplémentaire de $12 \cdot \frac{1}{2} \cdot \left(\frac{1}{9}\right)^2 \cdot \sin\frac{\pi}{3}$. Et ainsi de suite.



À l'étape n, on y ajoute l'aire de $(3 \cdot 4^{n-1})$ triangles équilatéraux de côté $\frac{1}{3^n}$, soit une aire supplémentaire de $(3 \cdot 4^{n-1}) \cdot \left(\frac{1}{3}\right)^{2n} \cdot \frac{\sqrt{3}}{4}$.

2.3.1. L'aire par Ruby

```
longueur_init ← 1.0
etape ← ...   (⩾ 2)
a ← 1
b ← longueur_init
aire ← 1/2 ab² sin π/3  (étape 0)
a ← 3
b ← b/3
aire ← aire + 1/2 ab² sin π/3  (ét 1)
pour k de 2 à etape
    a ← a * 4
    b ← b/3
    aire ← aire + 1/2 ab² sin π/3
fin
écrire etape, aire
```

```
# aire-flocon.rb
# ---------
longueur_init=1.0
etape = 100
a = 1
b = longueur_init
aire = 0.5*a*b*b*Math.sin(Math::PI/3)
a = 3
b = b/3
aire = aire + 0.5*a*b*b*Math.sin(Math::PI/3)
for k in 2..etape
  a = a*4
  b = b/3
  aire = aire + 0.5*a*b*b*Math.sin(Math::PI/3)
end
puts "apres #{etape} iterations,"
puts "aire de la figure = #{aire}"
```

après 100 itérations, l'aire calculée vaut 0.6928203230275508...

2.3.2. L'aire par les suites géométriques

En additionnant toutes les aires décrites ci-avant, on a :

$$\begin{aligned}
\text{aire} &= \frac{\sqrt{3}}{4} + 3\cdot\left(\frac{1}{3}\right)^2\cdot\frac{\sqrt{3}}{4} + 12\cdot\left(\frac{1}{3}\right)^4\cdot\frac{\sqrt{3}}{4} + 48\cdot\left(\frac{1}{3}\right)^6\cdot\frac{\sqrt{3}}{4} + \dots \\
&= \frac{\sqrt{3}}{4} + 3\cdot 4^0\cdot\left(\frac{1}{3}\right)^2\cdot\frac{\sqrt{3}}{4} + 3\cdot 4^1\cdot\left(\frac{1}{3}\right)^4\cdot\frac{\sqrt{3}}{4} + 3\cdot 4^2\cdot\left(\frac{1}{3}\right)^6\cdot\frac{\sqrt{3}}{4} + \dots \\
&= \frac{\sqrt{3}}{4} + \frac{\sqrt{3}}{4}\cdot\left(3\cdot 4^0\cdot\left(\frac{1}{3}\right)^2 + 3\cdot 4^1\cdot\left(\frac{1}{3}\right)^4 + 3\cdot 4^2\cdot\left(\frac{1}{3}\right)^6 + \dots\right) \\
&= \frac{\sqrt{3}}{4} + \frac{\sqrt{3}}{4}\cdot\left(\underbrace{\left(\frac{1}{3}\right) + 4^1\cdot\left(\frac{1}{3}\right)^3 + 4^2\cdot\left(\frac{1}{3}\right)^5 + \dots}_{\text{somme des termes d'une suite géométrique de raison } \frac{4}{9}}\right) \\
&= \frac{\sqrt{3}}{4} + \frac{\sqrt{3}}{4}\cdot\left(\frac{1}{3}\cdot\frac{1-\left(\frac{4}{9}\right)^n}{1-\frac{4}{9}}\right) \\
&= \frac{\sqrt{3}}{4} + \frac{\sqrt{3}}{4}\cdot\left(\frac{1}{3}\cdot\frac{1}{\frac{5}{9}}\right) = \frac{\sqrt{3}}{4} + \frac{\sqrt{3}}{4}\cdot\left(\frac{3}{5}\right) = \frac{2\sqrt{3}}{5} \approx 0,692820323\dots
\end{aligned}$$

Références

[1] Honclaire B. et Noël-Roch Y., Géométrie de la tortue. *Losanges*, 32, pp. 44–52. 2016.

Les nombres premiers, une aubaine pour l'algorithmique

Résumé. *2, 3, 5, 7, 11, 13, ... banal, des nombres premiers, arithmétique élémentaire, exactement deux diviseurs, ..., mais il y a aussi 1 066 818 132 668 207, ..., et aussi* $2^{74\,207\,281}-1$*, nombre de 22 338 618 chiffres ... on est loin, très loin, d'avoir tout exploré. Voici une toute petite incursion dans le monde fascinant des nombres premiers, éclairée par quelques algorithmes en langage Ruby.*

Jumeaux, cousins, sexys, nombres de Mersenne*, ..., il y en a pour tous les goûts ; conjectures de* Goldbach*, de* Proth-Gilbreath*, ..., il y a encore beaucoup de résultats à démontrer ; des tests de primalité pour lesquels un super ordinateur ne suffit plus, il en faut des milliers, ...*

Marek WOLF*, physicien polonais, né en 1956, « ... en regardant les nombres premiers, on a l'impression d'être en présence d'un des secrets inexplicables de la création ... ni théorèmes, ni preuves, seulement des résultats d'ordinateurs, des conjectures ... »*

Cet article est extrait de l'exposé fait au congrès de la SBPMef à Gembloux, en août 2016.

1 Il faut bien commencer par le commencement

Depuis l'Antiquité, les nombres premiers font couler beaucoup d'encre ; des mathématiciens célèbres s'y sont frottés : Platon, Aristote, Euclide, Ératosthène, Nicomaque, Diophante, Fibonacci, Mersenne, Fermat, Pascal, Goldbach, Euler, Bézout, Lagrange, Wilson, Legendre, Gauss, Dirichlet, Tchebychev, Riemann, Lucas, Hadamard, de la Vallée Poussin, ... pour n'en citer que quelques-uns, mais ils sont encore nombreux, et moins connus, ceux qui s'y intéressent toujours, notamment avec les progrès de l'informatique.

Les deux définitions de base

$a \in \mathbb{N}$ et $b \in \mathbb{N}_0$: b est diviseur de $a \Leftrightarrow \exists k \in \mathbb{N} : a = k \cdot b$
$\Leftrightarrow$ le reste de la division de a par b est nul
(notation : $a \bmod b = 0$)

$n \in \mathbb{N}_0$ est premier $\Leftrightarrow$ n a exactement 2 diviseurs, 1 et lui-même.

Un nombre qui n'est pas premier est dit *composé* ; 1, premier naturel non nul, n'est donc pas

premier ; 2 est le premier premier, mais le dernier premier pair ; 3 est le deuxième premier ; les jeux de mots s'arrêtent ici.

Premier algorithme, la division euclidienne

La division euclidienne n'est qu'une suite de soustractions. Comme on ne s'intéresse ici qu'au reste de la division de a par b, avec bien entendu $a \geqslant b$, il suffit d'ôter b de a jusqu'à ce que le résultat devienne négatif.

$a \leftarrow \cdots$
$b \leftarrow \cdots \quad (b \leqslant a)$
Tant que $a \geqslant b$
    $a \leftarrow a - b$
fin
écrire a

```
# division-euclidienne.rb
# -----------------------
a=1234567 # dividende
b=12 # diviseur
while a>=b
   a=a-b
end
puts "le reste de la division est #{a}"
```

Le langage de programmation *Ruby* propose un opérateur arithmétique de base qui fait ce calcul, l'opérateur `%`.

L'instruction `1234567%12` donne 7 comme résultat, ce qui nous dispense au passage de devoir exécuter l'algorithme ci-dessus.

Une application concrète de la division euclidienne : en Belgique, les 2 derniers chiffres d'un compte bancaire ou du numéro national d'identité forment un nombre qui est le reste de la division par 97 du nombre formé des 10 premiers chiffres.

Le second, un test de primalité élémentaire, mais ...

On souhaite vérifier si un naturel n est premier :

- informatiquement parlant, on utilise la méthode dite du « drapeau » : on suppose au départ que n est premier (drapeau baissé), et on le divise par tous les naturels k tels que $2 \leqslant k < n$; dès qu'un diviseur est trouvé (drapeau levé), on peut conclure que n est composé, et on arrête les divisions ;

- mais on peut réduire considérablement le nombre de divisions :
  en effet, n *composé* $\Leftrightarrow n = a \cdot b$ où $a \leqslant \sqrt{n}$ ou $b \leqslant \sqrt{n} \quad (a \neq b)$
  $(a > \sqrt{n}$ et $b > \sqrt{n} \Rightarrow a \cdot b > n)$

- on suppose donc au départ que n est premier ; on le divise par tous les naturels k tels que $2 \leqslant k < \sqrt{n}$; dès qu'un diviseur est trouvé, on peut conclure que n est composé, et on arrête les divisions.

Dans l'algorithme qui suit, la variable *prem* est une variable dite *booléenne* car elle contient soit la valeur *true*, soit la valeur *false*.

L'instruction `if prem` est donc équivalente à *si c'est vrai* ou *si c'est faux*, selon le contenu de la variable *prem*.

```
n ← ···
prem ← vrai
k ← 2
Tant que k² ⩽ n
    si n mod k = 0
        k est diviseur de n
        prem ← faux
        sortie de boucle
    fin
    k ← k + 1
fin
si prem
    n est premier
sinon
    n est composé
fin
```

```
# n-premier.rb
# vérifier si n est premier
# -------------------------------
n=2**313-1
prem=true
k=2
while k*k<=n
   if n%k == 0
      puts "#{k} est diviseur de #{n}"
      prem=false
      break
   end
   k=k+1
end
if prem
   puts "#{n} est premier"
else
   puts "#{n} n'est pas premier"
end
```

Mais... pour tester un nombre de 50 chiffres, on peut être amené à faire environ 10^{25} divisions ; mon ordinateur [1] fait environ 17×10^6 divisions par seconde ; il lui faudrait donc environ 5×10^{17} secondes de calcul(s) ;
une année comptant $60\times60\times24\times365$ secondes, on est donc parti pour plus de 18 milliards d'années...

Le langage *Ruby* propose une bibliothèque d'instructions relatives aux nombres premiers ; pour l'activer, exécuter l'instruction require 'prime'.

L'instruction qui permet de vérifier si un nombre est premier est nombre.prime? (attention, pas d'espace avant le point d'interrogation).

Exemple d'utilisation : 2017.prime? ; la réponse retournée est true.

Cette instruction a donc le statut d'une expression booléenne, et on peut par exemple la retrouver sous la forme if n.prime?...

Il est difficile de savoir quel algorithme « se cache derrière » une instruction d'un langage de programmation [2]. Toujours est-il que l'instruction (2**313-1).prime? donne pour résultat false, tout comme l'algorithme ci-dessus, heureusement.

L'algorithme « fabrication maison » affiche par contre que 10 960 009 est diviseur de 1668739871813211001871110707944962589533362908091134976521126256111109160766125429705439130419 1, ce que ne fait pas l'instruction *Ruby*, en tout cas pas celle-là.

Pour être honnête, il faut dire que l'algorithme « fabrication maison » fait le calcul en 4.787817 secondes, tandis que l'instruction *Ruby* le fait en 2.705436 secondes.

1. MacBook Pro 2,8 GHz Core i7, 16 Go de mémoire.
2. Personnellement je n'ai pas trouvé, mais peut-être n'ai-je pas cherché assez longtemps...

2 Y-a-t-il beaucoup de nombres premiers ?

Les Grecs (de l'Antiquité) ont donné une preuve qu'il y en a une infinité ; on trouve trace de cette affirmation dans les *Éléments* d'Euclide.
Il y en a une infinité, mais lorsqu'on les cherche sur l'axe des réels (positifs), on en trouve de moins en moins !

La clé de répartition des nombres premiers serait la répartition des zéros non triviaux de la fonction zêta de Riemann, célèbre conjecture non démontrée à l'heure actuelle (l'un des sept problèmes du millénaire, doté d'un prix de 1 million de dollars [3]).

Des déserts sans nombres premiers

Il est facile d'observer des intervalles aussi grands que l'on veut, et ne contenant aucun nombre premier :

$1+1000!$ n'est pas premier (6563 est diviseur ; il suffit de lui appliquer l'algorithme précédent, ou l'instruction *Ruby*).

$$\begin{aligned}
2+1000! &= 2\cdot(1+3\cdot4\cdot5\cdot\ldots 999\cdot1000) && \text{n'est pas premier non plus, 2 est diviseur}\\
3+1000! &= 3\cdot(1+2\cdot4\cdot5\cdot\ldots 999\cdot1000) && \text{non plus, 3 est diviseur}\\
999+1000! &= 999\cdot(1+2\cdot3\cdot4\cdot\ldots 998\cdot1000) && \text{non plus, 999 est diviseur}\\
1000+1000! &= 1000\cdot(1+2\cdot3\cdot4\cdot\ldots 998\cdot999) && \text{non plus, 1000 est diviseur}
\end{aligned}$$

$\Rightarrow$ l'intervalle $\left[1+1000!, 1000+1000!\right]$ ne contient donc aucun nombre premier.

Je ne sais pas si $1+10^{999}!$ est un nombre premier, mais l'intervalle $\left]1+10^{999}!, 10^{999}+10^{999}!\right]$, de longueur $10^{999}-1$, n'en contient pas. Plus on va loin, plus il y a d'espaces de plus en plus grands sans nombres premiers… en se fiant à nos yeux, il est difficile de croire qu'il y en a une infinité !

3 La chasse aux grands nombres premiers

Ici on joue dans la cour des grands, là où les algorithmes et l'ordinateur de monsieur [4] Tout-lemonde ne suffisent plus.

Dans un journal du lundi 25 janvier 2016, on pouvait lire « $2^{74\,207\,281}-1$ est premier ! ». Ce « mammouth des maths » est le plus grand nombre premier de Mersenne connu à ce jour, il compte plus de 22 millions de chiffres. Pour se donner une petite idée, cela représente environ 16 années de Losanges où l'on n'imprimerait que 72 pages de 4800 chiffres [5].

C'est un nombre dit « de Mersenne », religieux français et mathématicien du XVI^{e} siècle. Les nombres de Mersenne sont les nombres de la forme $M_p = 2^p - 1$; quelques-uns sont premiers (probablement 49 connus à ce jour).

3. À vos crayons et calculettes…
4. ou madame
5. Cool pour le comité de rédaction.

M_n premier $\Rightarrow$ n premier : si $n = pq$, alors $2^{pq} - 1$ est divisible par $2^p - 1$ et par $2^q - 1$,
mais n premier $\not\Rightarrow$ M_n premier : $2^{37} - 1 = 223 \times 616\,318\,177$.

Les nombres de MERSENNE sont une « usine à fabriquer des grands nombres premiers », pas beaucoup de concurrence, ni chômage, mais la production est très limitée !

Sur le site `www.mersenne.org`, on peut s'inscrire au projet GIMPS : the Great Internet Mersenne Prime Search, projet collaboratif qui utilise les capacités de calcul de votre ordinateur à ses moments perdus. On y trouve aussi la liste chronologique des records, dont voici un extrait (année de la découverte, auteur(s) de la découverte, n°d'ordre) :

- 1772, EULER, le 8e, M_{31}
- 1998, Roland CLARKSON, le 37e, $M_{3\,021\,377}$
- 1999, Nayan HAJRATWALA, le 38e, $M_{6\,972\,593}$
- 2001, Michael CAMERON, le 39e, $M_{13\,466\,917}$
- 2003, Michael SHAFER, le 40e, $M_{20\,996\,011}$
- 2004, Josh FINDLEY, le 41e, $M_{24\,036\,583}$
- 2005, Dr. Martin NOWAK, le 42e, $M_{25\,964\,951}$
- 2005, Curtis COOPER et Steven BOONE, le 43e, $M_{30\,402\,457}$
- 2006, Curtis COOPER et Steven BOONE, le 44e, $M_{32\,582\,657}$
- 2008, Hans-Michael ELVENICH, le 45e, $M_{37\,156\,667}$
- 2009, Odd Magnar STRINDMO, le 46e, $M_{42\,643\,801}$
- 2008, Edson SMITH, le 47e, $M_{43\,112\,609}$
- 2013, Curtis COOPER, le 48e, $M_{57\,885\,161}$
- 2016, Curtis CURTIS COOPER, le 49e, $M_{74\,207\,281}$ le dernier connu à ce jour (07/01/2016).

À partir du 46e, le classement est encore provisoire, tous les « candidats » n'ayant pas encore été éliminés.

Le langage *Ruby* peut calculer et afficher le 38e nombre de MERSENNE, $M_{6\,972\,593}$, en quelques secondes, le résultat est un document de 778 pages au format A4. J'ai essayé le suivant, mais après un temps d'attente qui semble interminable, on ne sait pas si l'ordinateur est bloqué, ou s'il faut attendre encore.

Bien évidemment, il y a des (grands) nombres premiers autres que les nombres de MERSENNE, il « suffit » de les chercher...

Citons, à tout hasard, $3\,756\,801\,695\,685 \times 2^{666\,669} - 1$, $208\,003! - 1$, $1\,098\,133\# - 1$ (voir *fonction primorielle* page 82), etc.

Si vous avez l'occasion de les placer dans une conversation, ...

D'autres tests de primalité

À côté du test élémentaire, dont on a expliqué les limites, il y en a d'autres, bien plus complexes, qui dépassent le cadre de cet article ; on peut citer, entre autres :

- le test de MILLER-RABIN ;
- le test de SOLOVAY-STRASSEN ;
- ...

Le test de Lucas-Lehmer

Le nom vient des mathématiciens qui l'ont mis au point : Edouard LUCAS (1842–1891), mathématicien français et Derrick LEHMER (1905–1991), mathématicien américain. Il est facile à programmer, mais ne convient qu'à des nombres de MERSENNE M_p « petits », car le nombre M_p doit être généré p fois dans l'algorithme.

$$M_p = 2^p - 1 \text{ est premier} \Leftrightarrow M_p \text{ divise } s_p \text{ où } \begin{cases} s_1 = 4 \\ s_{n+1} = \left((s_n)^2 - 2\right) mod\, M_p \\ \quad \text{pour } n \text{ de } 1 \text{ à } p-2 \end{cases}$$

```
p ← ...
mp ← 2^p − 1
s ← 4
pour i de 1 à p − 2
    s ← (s^2 − 2) % mp
fin
si s = 0
    écrire premier
sinon
    écrire composé
fin
```

```
# test-lucas-lehmer.rb
# --------------------
p=3217 #(record 1957)
mp=(2**p)-1
s=4
for i in 1..p-2
   s=(s**2-2) % mp
   puts s
end
if s==0
   puts "#{mp} est premier"
else
   puts "#{mp} est compose"
end
```

4 Comment engendrer des nombres premiers ?

4.1 Divers essais

Il n'y a (à l'heure actuelle) aucune « formule » qui permet de générer les nombres premiers. Certains ont cependant fait des essais.

Citons tout d'abord Marcel PAGNOL, surtout célèbre au cinéma, mais aussi féru de mathématiques. D'après lui, « si x et $x+2$ sont 2 impairs consécutifs, alors $x + (x+2) + x(x+2)$ est premier ». La formule engendre en effet des nombres premiers, mais pas pour tous les nombres impairs ; il suffit de faire un petit tableau de calculs avec le tableur pour voir que les impairs

1, 3, 5, 7, 11, 13, 17, 19 génèrent des nombres premiers, mais pas 9, 15, 21, 23, etc. La formule n'est donc pas tout-à-fait au point.

L. Euler donne un polynôme du second degré $x^2 - x + 41$ qui engendre des nombres premiers pour $x = 0, 1, 2, \ldots, 40$.

R. Ruby donne lui aussi un polynôme du second degré $36x^2 - 810x + 2753$ qui engendre des nombres premiers pour $x = 0, 1, 2, \ldots, 44$.

Il y en a d'autres. La production (en terme de nombres premiers) est... faible.

Et lorsque les factorielles s'en mêlent :

$3! - 2! + 1! = 5$ nombre premier
$4! - 3! + 2! - 1! = 19$ nombre premier
$5! - 4! + 3! - 2! + 1! = 101$ nombre premier
$6! - 5! + 4! - 3! + 2! - 1! = 619$ nombre premier
$7! - 6! + 5! - 4! + 3! - 2! + 1! = 4421$ nombre premier
$8! - 7! + 6! - 5! + 4! - 3! + 2! - 1! = 35899$ nombre premier
mais ça ne marche plus à partir de $9! - 8! + \ldots$

4.2 Le crible d'Ératosthène

C'est sans doute la méthode la plus ancienne et la plus simple ; le problème n'est pas de calculer les nombres premiers, mais de les mémoriser !

L'algorithme a déjà été décrit dans un numéro précédent de Losanges [6], seule la version *Ruby* est rappelée ci-dessous.

```
# eratosthene.rb
# -----------------
n=100000
t=Array.new(n+1){true}
prem=[ ]
k=2
while k**2<=n
   if t[k]
      for i in 2..n/k
         t[i*k]=false
      end
   end
   k=k+1
end
# ----------------------
for i in 2..n
   if t[i]
      prem << i
   end
end
```

Quelques chiffres qui interpellent

Le calcul des nombres premiers dans l'intervalle $[2, 10^9]$ par la méthode du crible d'Ératosthène dure 247 secondes, il y a 50 847 534 nombres premiers dans cet intervalle, le dernier est 999 999 937.

Le même calcul dans l'intervalle $[2, 2 \times 10^9]$ dure 83 minutes (l'attente est longue...), le dernier est 1 999 999 973, et le fichier (au format .txt) pour les mémoriser a une taille de 1,02 Go ; en y faisant un double clic pour l'ouvrir, il faut attendre pas moins de 10 minutes ! Mais il s'ouvre au début, et pour le faire défiler jusqu'à la fin, c'est très, très long !

Nous supposerons donc, pour la suite de cet article, que chaque algorithme commence par mémoriser suffisamment de nombres premiers dans un tableau nommé *prem*, de la forme

2	3	5	7	11	13	17	19	23	29	31	37	41	43	...

Il s'interprète comme `prem[0]=2, prem[1]=3, prem[2]=5, ...`

5 Des histoires de nombres premiers

5.1 Jumeaux, cousins et sexys

Soit a et b deux nombres premiers tels que $a < b$; on dit que

- a et b sont jumeaux $\Leftrightarrow b - a = 2$;
- a et b sont cousins $\Leftrightarrow b - a = 4$;
- a et b sont sexys $\Leftrightarrow b - a = 6$.

premiers de 2 à	paires de jumeaux	paires de cousins	paires de sexys
10^3	35	40	44
10^4	205	202	299
10^5	1 224	1 215	1 950
10^6	8 169	8 143	13 549
10^8	440 312	440 257	768 752
10^9	3 424 506	3 424 679	6 089 791

```
# jum-cou-sex.rb
# -----------------
# ici eratosthene
nbprem = prem.size
jum=0
cou=0
sex=0
for i in 0..nbprem-2
   diff=prem[i+1]-prem[i]
   case
      when diff==2
          jum=jum+1;
      when diff==4
         cou=cou+1;
      when diff==6
          sex=sex+1;
   end
end
puts "de 2 a #{n}, il y a :"
puts "#{jum} jumeaux"
puts "#{cou} cousins"
puts "#{sex} sexys"
```

Autour du même thème

- projet TPS (Twin Prime Search)
  projet associatif (partage du temps de calcul d'ordinateurs) pour la recherche des plus grands nombres premiers jumeaux :
  - → record 1999 : $361\,700\,055 \times 2^{39\,020} - 1$ et $361\,700\,055 \times 2^{39\,020} + 1$
  - → record 2007 : $2\,003\,663\,613 \times 2^{195\,000} - 1$ et $2\,003\,663\,613 \times 2^{195\,000} + 1$
  - → record 2011 : $3\,756\,801\,695\,685 \times 2^{666\,669} - 1$ et $3\,756\,801\,695\,685 \times 2^{666\,669} + 1$
- conjecture de Polignac (Alphonse de, 1826–1863) :
  quel que soit $k > 0$, les paires de premiers dont l'écart est $2k$ sont en nombre infini ;
- dans l'ensemble des nombres premiers compris entre 2 et 10^{15}, l'écart le plus fréquent est 6 ; à partir de 1.37×10^{36}, c'est 30 (Marek Wolf).

5.2 La fonction $\pi(x)$

$\pi(x)$ = le nombre de nombres premiers $\leqslant x$

GAUSS : (théorème des nombres premiers) $\lim_{x\to+\infty} \pi(x)\cdot\frac{\ln x}{x} = 1$

```
# pi-de-x.rb
# ----------
n=1000000
# ici eratosthene
nbprem=prem.size
puts "pi(#{n}) = #{prem.size}"
puts "Gauss : #{n/Math.log(n)}"
puts "#{prem.size}*#{n/Math.log(n)}"
```

x	$\pi(x)$	$\frac{x}{\ln(x)}$	$\pi(x)\cdot\frac{\ln x}{x}$
10^5	9 592	8 685.88...	1.104...
10^6	78 498	72 382.41...	1.084...
10^7	664 579	620 420.68...	1.071...
10^8	5 761 455	5 428 681.68...	1.061...
10^9	50 847 534	48 254 942.43...	1.053...

5.3 La fonction primorielle

$n\#$ est le produit des nombres premiers inférieurs à n : $n\# = \prod_{\substack{2\leqslant p_k\leqslant n\\ p_k \text{ premier}}}^{n} p_k$

C'est en quelque sorte une *factorielle*, mais avec des nombres premiers.

```
# primorielle.rb
# --------------
n=10000
# ici Eratosthene
nbprem=prem.size
indice=50 # < nbprem-1
primorielle=1
for k in 0..indice
   primorielle=primorielle*prem[k]
end
puts "primorielle de #{prem[indice]} = #{primorielle}"
```

Dans l'algorithme ci-dessus, on choisit l'indice 50 dans le tableau des nombres premiers ; $prim[50] = 233$, le 51e nombre premier (les indices commencent à 0).

Le résultat calculé est 233# = 44452361852721854381692407942913125574322226427271838090264514387041601034796008004320294642 70.

On peut aussi fabriquer des nombres premiers avec la fonction primorielle : $1 + n\#$ est premier pour n=2, 3, 5, 7, 11, 31, 379, 1 019, 1 021, 2 657, 3 229, 4 547, 4 787, 11 549, 13 649, 18 523, 23 801, 24 029, pour aucune autre valeur inférieure à 35 000, ...

5.4 La décomposition en facteurs premiers

Théorème fondamental de l'arithmétique :
« Tout naturel > 1 se décompose de manière unique comme produit de facteurs premiers, à l'ordre des facteurs près. »

Le langage *Ruby* possède une instruction qui calcule les diviseurs premiers d'un naturel n. Elle nécessite l'activation de la bibliothèque *prime*, via l'instruction require 'prime' et a pour syntaxe Prime.prime_division(n).

Exemple :

Prime.prime_division(12 345 678 987 654 321) affiche comme résultat $\Rightarrow [[3,4],[37,2],[333\,667,2]]$, ce qui signifie que $12\,345\,678\,987\,654\,321 = 3^4 \times 37^2 \times 333\,667^2$.

Est-il vraiment nécessaire de disserter longtemps sur l'importance de ce théorème dans le calcul fractionnaire ?

5.5 La conjecture de Proth-Gilbreath

F. Proth (1852–1879) est un mathématicien français et N. L. Gilbreath (1958), américain, est amateur de mathématiques, d'informatique, et de ... tours de magie.

- on prend une liste de nombres premiers consécutifs, par exemple :

2	3	5	7	11	13	17	19	23

- on soustrait deux à deux le plus grand du plus petit, jusqu'à épuisement de la liste.

Toutes les lignes commencent par 1 (sauf la première) ; cette conjecture est vérifiée jusqu'aux nombres premiers inférieurs à 3×10^{11}.

```
# proth-gilbreath.rb
# ------------------
n=100
# ici Eratosthene
nbreprem = prem.size
for k in 0..nbreprem-2
   for i in 0..nbreprem-2-k
       prem[i] = (prem[i]-prem[i+1]).abs
       print "#{prem[i]}-"
   end
   puts
end
```

2	3	5	7	11	13	17	19	23
1	2	2	4	2	4	2	4	
1	0	2	2	2	2	2		
1	2	0	0	0	0			
1	2	0	0	0				
1	2	0	0					
1	2	0						
1	2							
1								

L'instruction puts est une instruction d'écriture écran suivie d'un saut de ligne, tandis que print est une instruction d'écriture écran sans saut de ligne.

5.6 La conjecture de Goldbach

Christian Goldbach (1690–1764) est un mathématicien russe.

La conjecture forte : « Tout naturel pair > 2 est la somme de 2 nombres premiers ».

```
# goldbach2.rb
# --------------
nombre = 460 # nombre à décomposer
n=100000
# ici Eratosthene
require 'prime'
position = 0
while prem[position]<= nombre
   position = position+1
end
puts "position = #{position}"
compteur = 0
for i in 0..position
   for j in 0..position
      if prem[i]+prem[j] == nombre and prem[i]<=prem[j]
         compteur=compteur+1
         puts "#{compteur} : #{prem[i]} + #{prem[j]} = #{nombre}"
      end
   end
end
puts "il y a #{compteur} couples distincts"
```

Il y a (par exemple) 16 manières de décomposer 460 en somme de deux premiers : 3 + 457, 11 + 449, 17 + 443, 29 + 431, 41 + 419, 59 + 401, 71 + 389, 101 + 359, 107 + 353, 113 + 347, 149 + 311, 167 + 293, 179 + 281, 191 + 269, 197 + 263, et 227 + 233.

Après avoir calculé « suffisamment » de nombres premiers (variable n), on calcule la position (variable *position*) du dernier nombre premier inférieur au nombre à décomposer (variable *nombre*) ; il suffit alors d'une double boucle pour parcourir le tableau des nombres premiers et de ne retenir que ceux dont la somme est le *nombre* cherché.

La conjecture faible : « Tout naturel impair > 7 est la somme de 3 nombres premiers ».

La conjecture faible, encore nommée conjecture ternaire, a été démontrée en 2013 par un mathématicien péruvien, Harold HELFGOTT ; ce n'est donc plus une conjecture... la conjecture forte, quant à elle, reste encore une conjecture.

```
# goldbach3.rb
# -----------------
require 'prime'
nombre=277
# ici Eratosthene
position = 0
while prem[position]<= nombre
   position = position+1
end
compteur = 0
for i in 0..position
   for j in 0..position
      for k in 0..position
         if prem[i]+prem[j]+prem[k] == nombre and i<=j and j<=k
            compteur=compteur+1
            puts "#{compteur} : #{prem[i]} + #{prem[j]}+#{prem[k]}"
         end
      end
   end
end
puts "il y a #{compteur} triplet(s)"
```

Il y a (par exemple) 7 manières de décomposer 27 en somme de trois premiers :
$2+2+23, 3+5+19, 3+7+17, 3+11+13, 5+5+17, 5+11+11$, et $7+7+13$

5.7 Le crible de Matiiassevitch

Youri Matiiassevitch (1947), est un mathématicien russe ; le crible géométrique qui porte son nom et qui met en évidence les nombres premiers, est aussi dû à son compatriote Boris Stechkin (1881–1969).

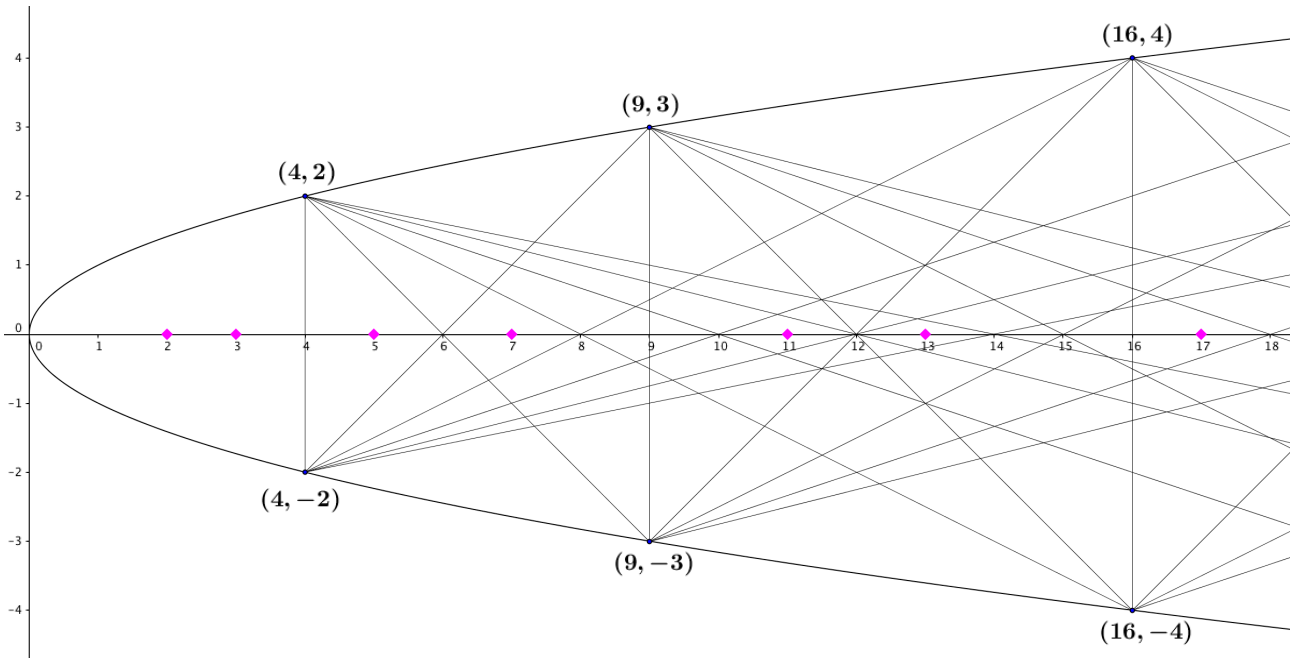
Il est difficile de ne pas le mentionner en parlant des nombres premiers, même si le sujet n'a pas de lien avec l'algorithmique. Le graphique est toutefois un bel exercice à réaliser avec *Géogébra*.

Soit la parabole d'équation $P := y^2 = x$ sur laquelle on choisit 2 points de coordonnées (a^2, a) et $(b^2, -b)$ avec $a, b \in \mathbb{N}_0 \setminus \{1\}$.
La droite passant par ces 2 points a pour équation $d := y - a = \frac{1}{a-b}(x - a^2)$; elle est parallèle à l'axe des ordonnées si $a = b$, mais surtout son intersection avec l'axe des abscisses vaut ab.

En reliant tous ces points par des droites, toutes les intersections avec l'axe des abscisses sont les produits de type ab, qui par définition ne sont pas des nombres premiers, puisque divisibles par a et par b.

Les abscisses par lesquelles ne passe aucune de ces droites sont des nombres premiers.

Et il y a encore beaucoup de choses à raconter !



Le crible de Matiassevitch

Références

[1] http://gecif.net.

[2] http://compoasso.free.fr.

[3] http://serge.mehl.free.fr.

[4] Corbalàn F., *Les nombres premiers, un long chemin vers l'infini.* RBA Colleccionables, 2011.

[5] Delahaye J.-P., *Merveilleux nombres premiers, voyage au coeur de l'arithmétique.* Belin - Pour la Science, 2000.

[6] Desbonnez J.-M., Algorithmique et langage Ruby, cuisiner les tableaux. *Losanges n°32*, pp. 53 – 56. 2016.

[7] Ribbenboim P., *The little book of bigger primes.* Second Edition. Springer, 2004.

Scilab, un logiciel de calcul numérique

Scilab (scientific laboratory) est un logiciel open source de calcul numérique et un langage de programmation basé sur le langage *Matlab* ; il est téléchargeable gratuitement à l'adresse `http://www.scilab.org` et est multi-plateformes (Linux, OsX, Windows).

Il a été créé en 1990 à l'*INRIA* (INstitut national de Recherche en Informatique et en Automatisme), institut de recherche français, et est édité par *Scilab Enterprises*.

Scilab contient des centaines de fonctions mathématiques, des fonctions graphiques 2D et 3D et un environnement de programmation. Ses domaines de prédilection sont les mathématiques, la simulation, la visualisation 2D et 3D, l'optimisation, les statistiques, le traitement de signal, etc.

Voilà pour les présentations.

Une fois la version de base installée, on peut y ajouter un module complémentaire appelé « module lycée ». Celui-ci contient des fonctions spécifiques à l'enseignement des mathématiques au lycée (équivalent du niveau secondaire supérieur en Belgique), dans les domaines de l'arithmétique, des probabilités, des statistiques et des ensembles. Pour cette installation, il faut une connexion internet, démarrer le logiciel *Scilab*, puis `Applications > Gestionnaire de Modules - ATOMS > Education > Module lycée > Installer`. Il faut ensuite quitter *Scilab* et le relancer pour activer le module.

Comme le langage *Ruby*, on peut utiliser *Scilab* soit en mode « console », dans lequel chaque commande est immédiatement exécutée, mais aussi (et surtout) en mode « éditeur » dans lequel on peut écrire des algorithmes, *Scilab* étant aussi un langage avec tous les outils de programmation traditionnels : alternatives, répétitives, gestion des variables simples et de type matrices (sa spécialité), ...

Aide et apprentissage

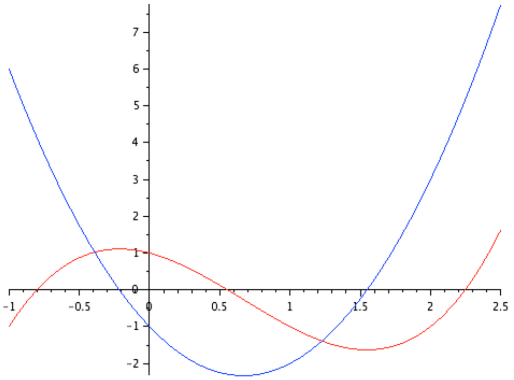
Une liste de diffusion est dédiée au monde de l'enseignement. Pour s'y inscrire, il suffit de compléter un formulaire à l'adresse `http://lists.scilab.org/mailman/listinfo/enseignement`, et tout message à `enseignement@lists.scilab.org` est diffusé à tous les membres.

On trouve aussi sur le net des cours et tutoriels très bien documentés, avec exercices et corrigés, notamment à l'adresse http://www.scilab.org/fr/community/education

Exemples

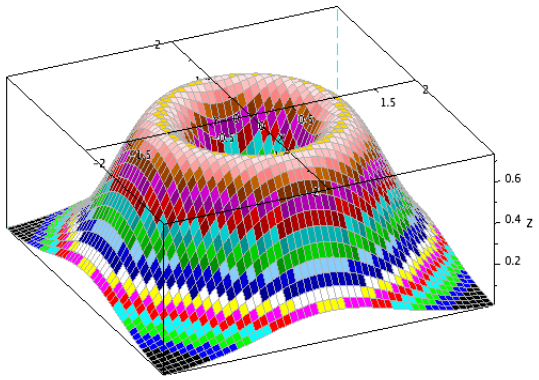
- Une fonction et sa dérivée

```
function y=f(x)
    y=(x^3-2*x^2-x+1)
endfunction
function y=g(x)
    y=(3*x^2-4*x-1)
endfunction
clf
x=linspace(-1,2.5,50);
plot(x,f,"r",x,g,"b")
```

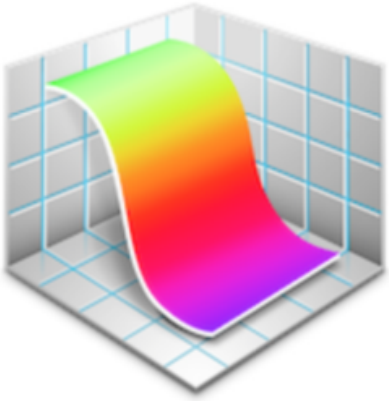


- Une surface 3D

```
function z=f(x,y)
    z=2*(x^2+y^2)*exp(-x^2-y^2);
endfunction
x=linspace(-2,2,50);
y=linspace(-2,2,50);
z=(feval(x,y,f));
clf;
surf(x,y,z)
```



Grapher, un logiciel de dessin 2D-3D

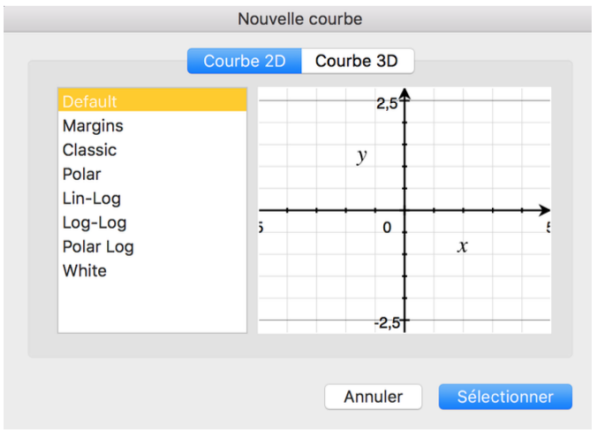


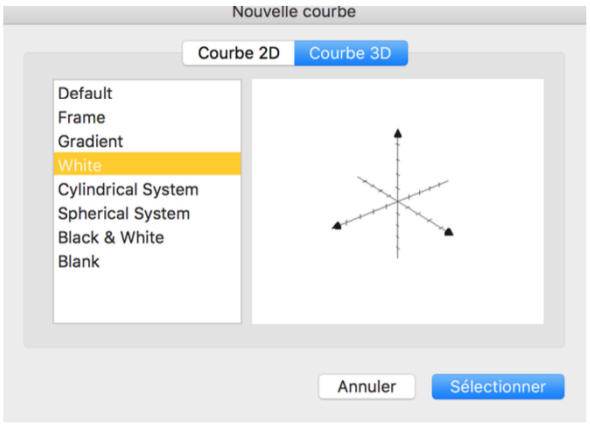
Si votre ordinateur est un *Mac* et/ou les cyber-classes de votre école sont équipées de matériel *Mac*, déjà vous avez de la chance, et ensuite ceci vous intéresse peut-être.

Dans le dossier *Applications*, il y a un dossier *Utilitaires* où se cache un petit bijou mathématique : l'application *Grapher.app*.

Elle permet, entre autres, de dessiner des graphiques en 2D ou 3D dans différents types de repères, et ce, *Mac* oblige, de manière très conviviale.

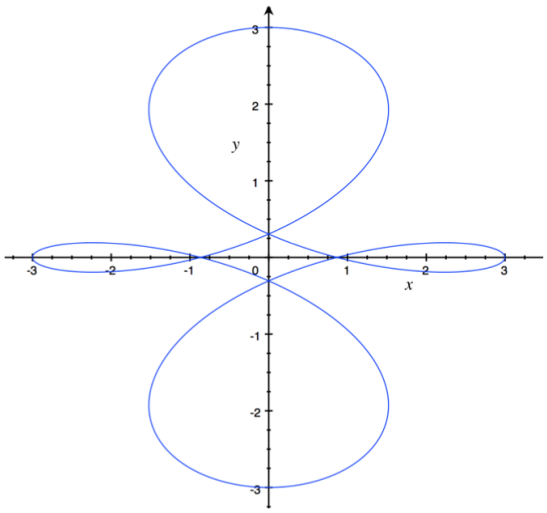
Dès le lancement, il faut choisir entre un graphique 2D ou 3D :



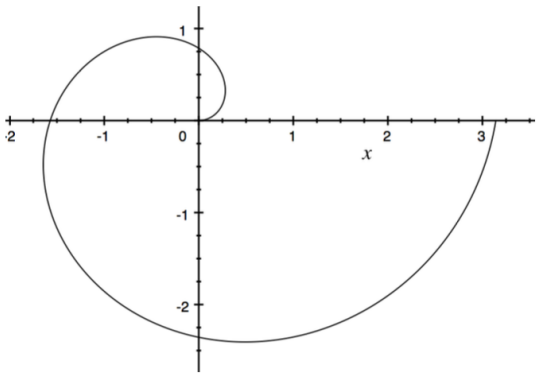


En 2D, il y en a pour tous les goûts et tous les niveaux, du secondaire à l'enseignement supérieur, d'une simple courbe $y = \sin x$ dont je vous prive d'illustration, à

$$\begin{cases} x = \cos t + 2\cos 3t \\ y = 2\sin t - \sin 3t \end{cases}$$



en passant par $r = \frac{1}{2}\theta$ (la spirale d'Archimède)



sans parler de la dérivation, l'intégration, la résolution d'équations différentielles, la liste est longue.

Les équations des trois exemples sont encodées telles qu'écrites ; on peut en effet encoder les courbes à partir d'équations paramétriques, cartésiennes ou polaires, le tout avec beaucoup de possibilités de mises en forme (couleurs, styles de traits, ...) et l'exportations des images sous différents types de fichiers graphiques.

Il est impossible en ces quelques pages de faire le tour de la bête.

En 3D les possibilités sont aussi nombreuses et variées, plus spectaculaires encore, et on peut, avec la souris, faire tourner le graphique dans l'espace, afin de l'observer sous toutes ses coutures.

Voici une simple illustration d'une séquence de cours (expérience personnelle [1]) en terminale, math 4h/sem, cours de géométrie analytique des plans et des droites dans l'espace. C'est un chapitre qui accroche difficilement les élèves de ce niveau (nombreux calculs techniques) ; le fait de dessiner les interprétations géométriques de ces calculs permet d'alléger un peu le repas...

L'exercice consiste à résoudre le système $\begin{cases} x + 2y + 3z = 14 \\ 2x - y + z = 3 \end{cases}$ et à en donner une interprétation géométrique.

Géométriquement parlant, ce sont deux équations cartésiennes de plans non parallèles, les solutions du système sont donc les points d'une droite. Techniquement, on peut utiliser, par exemple la méthode de Gauss pour éliminer des inconnues, et ainsi se ramener à des systèmes équivalents (d'autres plans, mais qui ont la même droite d'intersection). *Grapher* permettra

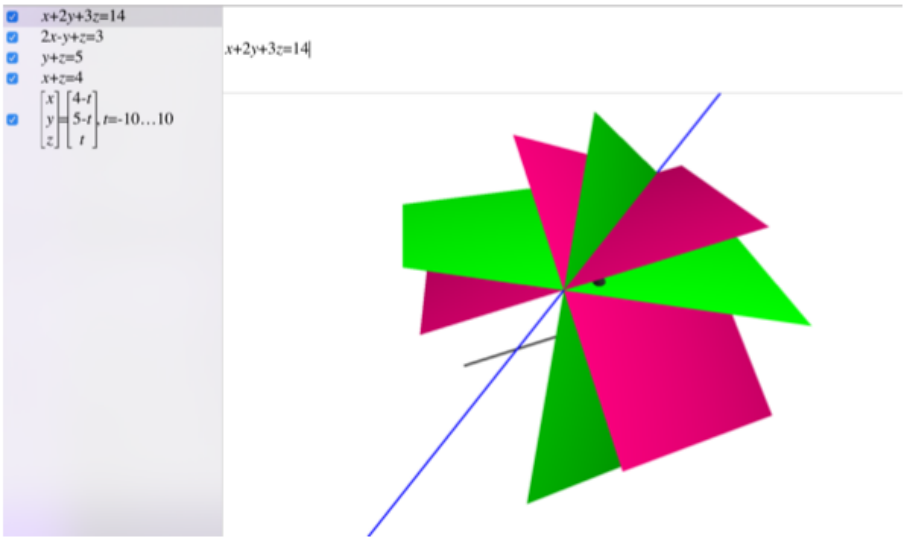
1. J'ai été bien inspiré de demander à ma direction du cyber-matériel *Mac*.

d'illustrer chaque étape (et donc aussi de vérifier les calculs) en encodant successivement les nouvelles équations obtenues, et finalement de vérifier le résultat final en encodant des équations paramétriques de la droite intersection.

Bien sûr, il y a eu des explications préalables quant aux équations cartésiennes d'un plan, aux équations paramétriques d'une droite, et la méthode de Gauss.

Voici les trois étapes du calcul (grâce vous est faite de tous les détails), et le graphique final, qui n'est pas simple à faire au tableau à l'aide d'une craie...

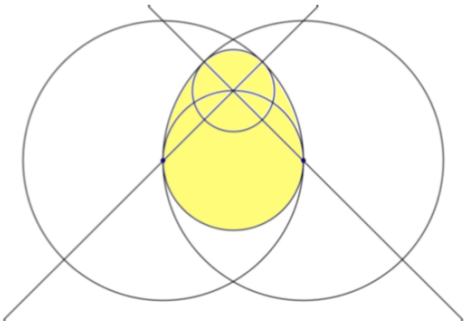
$$\begin{cases} x+2y+3z=14 \\ 2x-y+z=3 \end{cases} \Leftrightarrow \begin{cases} y+z=5 \\ x+z=4 \end{cases} \Leftrightarrow \begin{cases} x=4-t \\ y=5-t \\ z=t \end{cases}$$



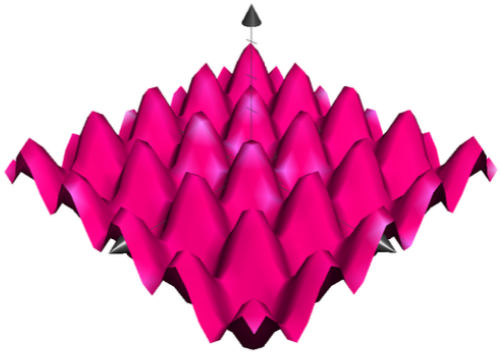
Le premier système est représenté par les 2 plans en vert ; le second par les 2 plans en magenta, et la droite en bleu.

Pour autant que le professeur maîtrise un peu le logiciel, il est facile d'en expliquer les rudiments ; en 50 minutes de cours, les élèves savent dessiner des plans et des droites à partir d'équations cartésiennes et paramétriques, de placer des points dans l'espace, et donc d'illustrer (et de vérifier) par exemple le calcul du point de percée d'une droite dans un plan (il suffit pour cet exercice de savoir représenter un plan, une droite et un point).

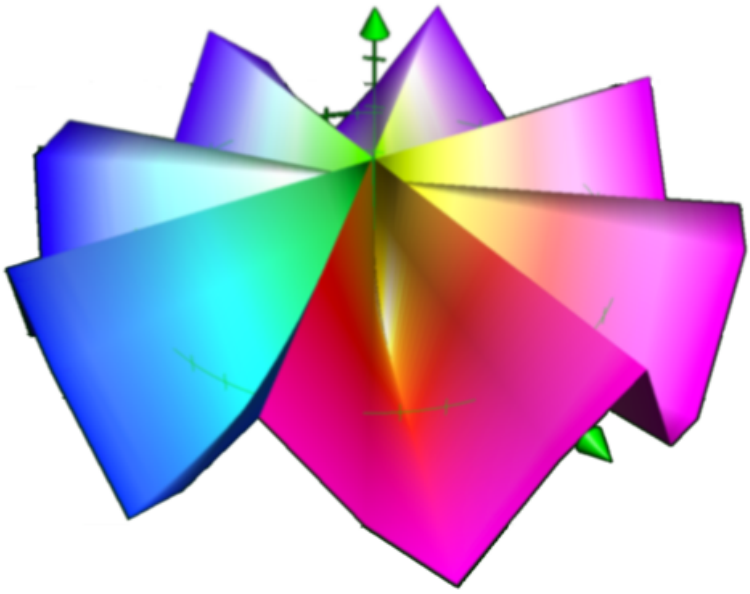
Et puisque Pâques n'est pas loin, on vous offre un oeuf (en 4 cercles et 2 droites), à construire par exemple avec *Géogébra*



et la boîte pour le mettre dedans, d'équation cartésienne $z = 0.8(\sin 3x + \sin 3y)$ à construire avec *Grapher*.



En dernier cadeau, le *cylindroïde* de PLÜCKER d'équation $z = 2\cos 8\theta$ en coordonnées cylindriques.



Il est également possible de créer des animations *Quicktime*, de traiter des séries discrètes, des champs scalaires et vectoriels, des inégalités booléennes, des fonctions définies par morceaux, des ensembles de points en liaison avec un tableur, etc.

Sur le site `macetprof.free.fr` on peut télécharger un manuel ; on trouve également de la documentation (en français) sur
`https://openclassrooms.com/courses/executer-des-fonctions-mathematiques-avec-grapher`
et sur `www.mac4ever.com/dossiers/74241_pour-bien-commencer-avec-grapher` et en anglais aussi sur bien d'autres sites, faire une recherche sur *grapher*.

La suite de Padovan, d'argent et de plastique

À mon papa, qui aimait voyager.

Résumé. *La suite de* Padovan *est une invitation au voyage dans le pays de l'analyse, via le calcul de limites, dans le pays de l'informatique et de l'arithmétique via les logiciels de calculs de type tableur, ainsi que Scilab et Ruby, et dans le pays de la géométrie via la Tortue-Logo, et une interface graphique de Ruby. Bonne route.*

Richard Padovan (1935 –) est un architecte anglais. C'est un autre architecte, Hans van der Laan (1904–1991), moine bénédictin des Pays-Bas, qui a donné le nom à la suite définie par récurrence de la manière suivante :

$$\begin{cases} u_1 = 1 \\ u_2 = 1 \\ u_3 = 1 \\ u_{n+1} = u_{n-1} + u_{n-2} \end{cases}$$

1 Première escale, le calcul arithmétique

Le premier réflexe est de calculer les premiers termes de la suite, ce qui peut se faire simplement avec un crayon et du papier : 1, 1, 1, 2, 2, 3, 4, 5, 7, 9, 12, 16, ...les additions ne sont pas compliquées...

```
// suite de Padovan

u(1)=1; u(2)=1; u(3)=1
afficher (u(1) ,u(2) ,u(3))
for n=3:50
    u(n+1)=u(n-1)+u(n-2)
    afficher (u(n+1))
end
```

Pour qui en veut plus, on peut utiliser un logiciel, comme par exemple *Scilab* [3], qui permet d'automatiser les calculs.
Ci-contre, l'algorithme qui permet de calculer les 50 premiers termes de la suite. On constate que la syntaxe est bien adaptée aux formules de récurrence.
// permet d'introduire une ligne de commentaires.

La suite semble anodine, mais intéressons-nous à la suite des $p_n = \frac{u_{n+1}}{u_n}$, un peu (beaucoup) comme le traitement de la suite de Fibonacci qui fait apparaître le nombre d'or Φ.

Pour varier les plaisirs, nous calculerons les termes de la suite des p_n avec un tableur ; on supposera que le lecteur en manipule le béhaba. Pour faire au plus simple :

- dans la colonne A, à partir de la cellule A1, les naturels 1, 2, 3, 4, …, indices n des termes des suites ;
- dans la colonne B, à partir de la cellule B1, les termes de la suite de PADOVAN : 1 en B1, 1 en B2, 1 en B3, la formule `=B1+B2` en B4, puis copier B4 (vers le bas) ;
- dans la colonne C, à partir de C1, les termes de la suite des p_n : la formule `=B2/B1` en C1, puis copier C1 (vers le bas).

n	u_n	p_n
1	1	1
…	…	…
6	3	1,3333333333333
7	4	1,25
8	5	1,4
9	7	1,2857142857143
10	9	1,3333333333333
…	…	…
50	696081	1,3247179566746
51	922111	1,3247179569488
52	1221537	1,324717957786
53	1618192	1,3247179568308
54	2143648	1,3247179574259
55	2839729	1,3247179572417
…	…	…
80	3208946545	1,3247179572448
81	4250949112	1,3247179572448
82	5631308624	1,3247179572448
83	7459895657	1,3247179572448
84	9882257736	1,3247179572448
85	13091204281	1,3247179572448
…	…	…

On constate très vite qu'il n'est pas nécessaire d'aller bien loin :

à partir de $n = 80$, et sans modifier les options d'affichage des décimales,

- les résultats de la colonne B (la suite des u_n) dépassent déjà 3 milliards ; sans grande surprise, $\lim_{n\to+\infty} u_n = +\infty$;
- les résultats de la colonne C (la suite des p_n), restent figés sur $1,3247179572448$, ce qui laisse pressentir que la suite des p_n semble converger…, $\lim_{n\to+\infty} p_n = 1,3247179572\ldots$?

Nous abordons le pays de l'analyse.

2 Deuxième escale, l'analyse

Commençons par transformer l'égalité $u_{n+1} = u_{n-1} + u_{n-2}$ en divisant les deux membres par u_n (aucun terme de la suite des u_n n'étant nul, les quotients sont valides) de manière à faire apparaître p_n, puis faisons en sorte de faire apparaître des quotients de la forme $\frac{\text{terme suivant}}{\text{terme}}$:

$$u_{n+1} = u_{n-1} + u_{n-2}$$
$$\frac{u_{n+1}}{u_n} = \frac{u_{n-1}}{u_n} + \frac{u_{n-2}}{u_n}$$
$$\frac{u_{n+1}}{u_n} = \frac{1}{\frac{u_n}{u_{n-1}}} + \frac{u_{n-2}}{u_n}$$
$$\frac{u_{n+1}}{u_n} = \frac{1}{\frac{u_n}{u_{n-1}}} + \frac{u_{n-2}}{u_{n-1}} \cdot \frac{u_{n-1}}{u_n}$$
$$\frac{u_{n+1}}{u_n} = \frac{1}{\frac{u_n}{u_{n-1}}} + \frac{1}{\frac{u_{n-1}}{u_{n-2}}} \cdot \frac{1}{\frac{u_n}{u_{n-1}}}$$

En supposant que

$$\Psi = \lim_{n\to+\infty} \frac{u_{n+1}}{u_n} = \lim_{n\to+\infty} \frac{\text{terme suivant}}{\text{terme}}$$

existe [1], et en passant à la limite dans la dernière égalité ci-contre, Ψ doit vérifier l'égalité

$$\Psi = \frac{1}{\Psi} + \frac{1}{\Psi} \cdot \frac{1}{\Psi}$$

que l'on ramène facilement à

$$\Psi^3 - \Psi - 1 = 0 \qquad (10.1)$$

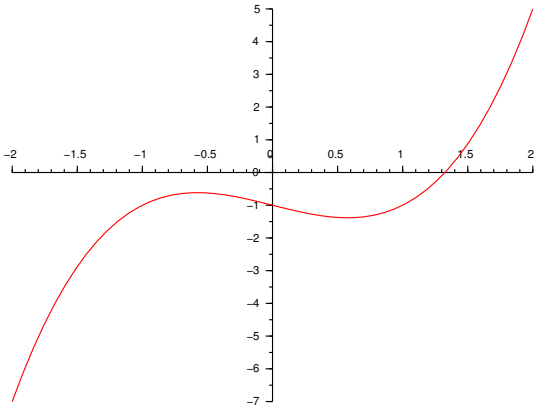
Cette équation du troisième degré admet une solution réelle, que l'on peut calculer, par exemple, avec la formule de Cardan [2] :

$$\Psi = \sqrt[3]{\frac{1}{2} - \frac{1}{2}\sqrt{\frac{23}{27}}} + \sqrt[3]{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{23}{27}}} = 1,3247179572\ldots$$

Dans [1], on a calculé une valeur approchée de Ψ avec la méthode de Newton-Raphson, algorithme en langage *Ruby*.

On peut aussi tracer le graphique de la fonction $f(x) = x^3 - x - 1$ avec, pourquoi pas, un algorithme en *Scilab* [3], qui permet de visualiser la racine dans l'intervalle $[1.3, 1.4]$:

```
function y=f(x)
    y=(x^3-x-1)
endfunction
clf
x=linspace(-2,2,50);
plot(x,f,"r")
```



1. C'est le cas, mais la démonstration n'est pas « immédiate ».
2. Gerolamo Cardano, 1501–1576

L'équation (10.1) peut aussi se ramener à :

$$\begin{aligned}\Psi^3 &= 1+\Psi\\ \Psi &= \sqrt[3]{1+\Psi}\\ &= \sqrt[3]{1+\sqrt[3]{1+\Psi}}\\ &= \sqrt[3]{1+\sqrt[3]{1+\sqrt[3]{1+\Psi}}}\\ &= \sqrt[3]{1+\sqrt[3]{1+\sqrt[3]{1+\sqrt[3]{1+\ldots}}}}\end{aligned}$$

Ce nombre Ψ a été baptisé nombre d'argent de par sa similitude avec le nombre d'or Φ [5], et aussi nombre plastique, au sens de la beauté architecturale, pour son utilisation en architecture. La transition est facile, ce qui nous amène au pays de la géométrie.

3 Troisième escale, la géométrie

L'idée est de dessiner une suite de triangles équilatéraux dont les longueurs des côtés sont les nombres de la suite de PADOVAN : la *spirale des triangles de* PADOVAN.

Pour agrémenter notre voyage, nous allons utiliser la *Tortue-Logo* [4] qui se fera un plaisir de « danser la spirale ».

```
pour padovan
  donne "a 1
  donne "b 1
  donne "c 1
  ecris :a
  ecris :b
  ecris :c
  repete 10
   [
    donne "d :a + :b
    ecris :d
    donne "a :b
    donne "b :c
    donne "c :d
   ]
fin
```

Pour dessiner des triangles équilatéraux dont les longueurs des côtés sont les nombres de la suite de PADOVAN, il faut d'abord dompter la *Tortue* pour calculer ces nombres. Ses mouvements n'étant pas aussi souples que ceux de *Scilab*, on utilisera la méthode des « chaises musicales » entre les variables (les trois dernières instructions).

donne "a 1 $\Leftrightarrow a \leftarrow 1$ (affectation, mettre 1 dans a)

écris :a $\Leftrightarrow$ affiche le contenu de la variable a

répète 10 [...] $\Leftrightarrow$ faire 10 fois les instructions entre les crochets

La procédure padovan ci-contre calcule et affiche les 13 premiers nombres de la suite de PADOVAN (les 3 premiers et les 10 suivants).

Pour dessiner la spirale des triangles, il faut tracer les 3 premiers de côtés respectifs a, b, et c, puis une suite de triangles de côtés d.

Dessiner un triangle équilatéral de côté x

La longueur du côté est passée en paramètre.
av :x ⇔ avance (et dessine un segment) de longueur égale au contenu de la variable x ;
td 120 ⇔ tourne à droite de 120°.

```
pour tri :x
  repete 3 [av :x td 120]
fin
```

L'unité de longueur pour la *Tortue-Logo* étant le pixel (ce n'est pas très grand), on peut multiplier la longueur x par un facteur d'échelle que nous appellerons… *echelle*.

La procédure tri, avec deux paramètres, devient donc

```
pour tri :x :echelle
  repete 3 [av :x * :echelle td 120]
fin
```

Les 3 triangles initiaux

Comme il est nécessaire d'initialiser les 3 premiers termes de la suite, il faut aussi dessiner les 3 premiers triangles de manière indépendante, adjacents, dans le sens horlogique. On sait que ces 3 triangles équilatéraux ont des côtés de longueur 1 (à multiplier par le facteur d'échelle) ; il faudra donc appliquer 3 fois la procédure `tri :x :echelle` (avec $x = 1$) en veillant à bien repositionner la *Tortue* après chaque tracé en vue de dessiner le triangle suivant. Cette procédure est nommée `initspirale :echelle`.

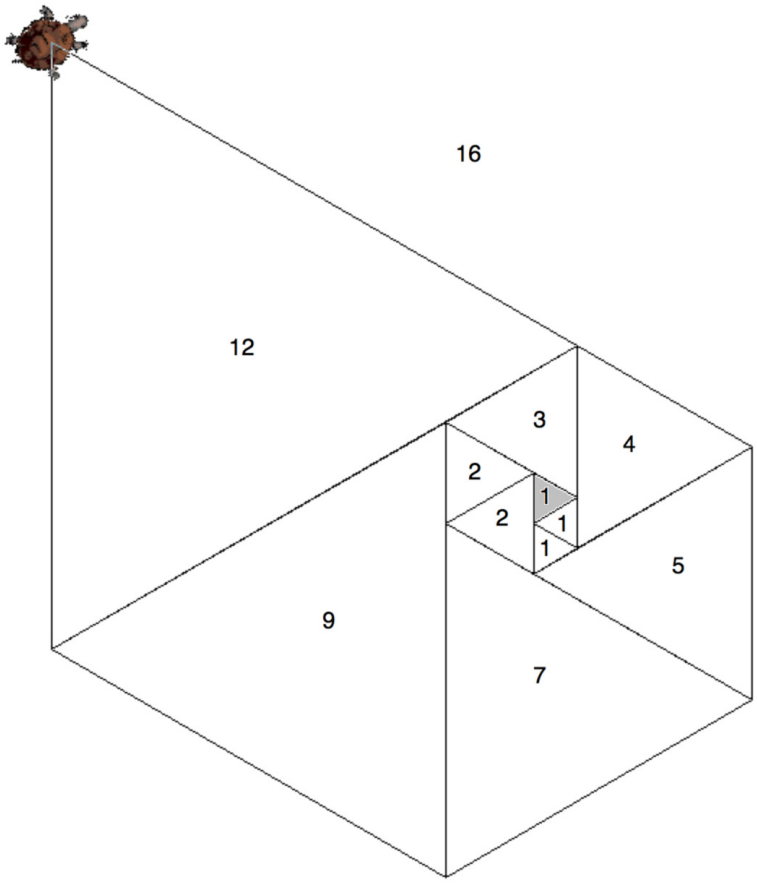
```
pour initspirale :echelle
  tri 1 :echelle
  td 60
  av 1 * :echelle
  td 120
  tri 1 :echelle
  av 1 * :echelle
  td 60
  tri 1 :echelle
  av 1 * :echelle
  td 60
fin
```

```
pour padovanspirale :nbtermes :echelle
  donne "a 1
  donne "b 1
  donne "c 1
  initspirale :echelle
  repete :nbtermes
   [
    donne "d :a + :b
    tri :d :echelle
    av :d * :echelle
    td 60
    donne "a :b
    donne "b :c
    donne "c :d
   ]
fin
```

La spirale complète

Il « suffit » de reprendre la procédure padovan qui calcule les termes (d) de la suite, et d'y remplacer l'instruction `écris :d` par l'instruction de dessin d'un triangle de côté d, à savoir `tri :d :echelle`, suivie de 2 instructions qui repositionnent la *Tortue* avant de dessiner le triangle suivant.
Il faut également fixer le nombre de termes de la suite que l'on désire calculer, en plus des 3 premiers (10 dans l'exemple) ; ce nombre est placé dans une variable (*nbtermes*), que l'on passe également en paramètre dans la procédure padovan, qui devient padovanspirale :nbtermes :echelle.

Ci-dessous la spirale obtenue par la commande `padovanspirale 8 35` ; les nombres ont été ajoutés par la suite, de même que l'arrière-plan du triangle de départ.
Après la procédure, la *Tortue* est en bonne position pour dessiner le triangle suivant, de côté 16.



4 Le saviez-vous ?

De la suite dans le porte-monnaie

Pour rester dans le thème des suites, de l'argent et des voyages, intéressons-nous à la suite formée des montants de nos pièces et billets en euro ; il y a 15 montants différents, que l'on peut noter, en langage suite, $u_0 = 0.01$, $u_1 = 0.02$, $u_2 = 0.05$, …, $u_{10} = 20$, $u_{11} = 50$, $u_{12} = 100$, $u_{13} = 200$, $u_{14} = 500$.

La somme des montants inférieurs à u_i est strictement inférieure à u_{i+1}, ce que l'on peut noter :

$$\sum_{k=0}^{i} u_i < u_{i+1} \quad (i < 14)$$

Par exemple, $\underbrace{u_0 + u_1 + u_2 + u_3 + u_4}_{0.48} < u_5 = 0.5$

Une suite ayant cette propriété est qualifiée de super-croissante. Elle est utilisée en cryptologie.

Cette propriété n'est par contre pas vérifiée pour le dollar américain. En effet, les différentes pièces et billets sont 0.01, 0.05, 0.10, 0.20, 0.25, 0.5, 1, 2, 5, 10, 20, 50 et 100. On a :

$$0.01 < 0,05$$
$$0.01 + 0.05 = 0.06 < 0.1$$
$$0.01 + 0.05 + 0.1 = 0.16 < 0,2$$
$$\text{mais } 0.01 + 0.05 + 0.1 + 0.2 = 0.36 > 0.25$$

La tentation est grande pour voyager dans d'autres pays :

la Suisse (franc) : 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 1000, 2000, 1000, suite super-croissante ;

la Grande-Bretagne (livre) : 0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, suite super-croissante ;

le Japon (yen) : 1, 5, 10, 50, 100, 500, 1000, 2000, 5000, 10000, suite super-croissante ;

la Chine (yuan) : 0.01, 0.02, 0.05, 0.5, 1, 2, 5, 10, 20, 50, 100, suite super-croissante.

Les États-Unis seraient-ils l'exception ? [6]

Références

[1] Desbonnez J.-M., Algorithmique élémentaire et Ruby. *Losanges*, 28, pp. 46–55. 2015.

[2] Desbonnez J.-M., Algorithmique et Ruby, Vol au-dessus d'un nid imaginaire. *Losanges*, 33, pp. 47—58. 2016.

[3] Desbonnez J.-M., Regards sur un logiciel de calcul numérique. *Losanges*, 35, 2016.

[4] Desbonnez J.-M., von Koch, la Tortue et Ruby, une rencontre du troisième type. *Losanges*, 34, pp. 49—54. 2016.

[5] Mawhin J., Le nombre d'or. *Losanges*, 06, pp. 11–23. 2009.

[6] Pasquet S., *Ainsi de suite*. 2016.

Des racines et des suites

Résumé. *Bien sûr, on a fait beaucoup de progrès : les touches* $\boxed{\sqrt{x}}$ *et* $\boxed{y^x}$ *de nos calculettes donnent de bons résultats... mais cela ne nous empêche pas de disserter un peu sur les racines, la distributivité, l'exponentielle, les suites, un peu de tableur et un soupçon d'algorithmique. Avec, dans le rôle de* $\sqrt{2} : \frac{19601}{13860}$*, dans celui de* $\sqrt{3} : \frac{70226}{40545}$*, dans celui de* $\sqrt{17} : \frac{9168}{2224}$*, dans celui de* $\sqrt[3]{5} : \frac{60422}{35335}$*, et enfin dans celui de* $\sqrt[3]{25} : \frac{619921}{212010}$*.*

Merci à Philippe Tilleuil pour l'idée d'abord, pour sa précieuse collaboration ensuite.

1 Racine carrée de 2

Commençons par des calculs algébriques élémentaires, dans lesquels on ne fait intervenir que le carré d'une différence, puis la distributivité :

$$\begin{aligned}
(\sqrt{2}-1)^1 &= \sqrt{2}-1 = -1+\sqrt{2}\\
(\sqrt{2}-1)^2 &= 3-2\sqrt{2}\\
(\sqrt{2}-1)^3 &= (\sqrt{2}-1)^2(\sqrt{2}-1) = -7+5\sqrt{2}\\
(\sqrt{2}-1)^4 &= (\sqrt{2}-1)^3(\sqrt{2}-1) = 17-12\sqrt{2}\\
(\sqrt{2}-1)^5 &= (\sqrt{2}-1)^4(\sqrt{2}-1) = -41+29\sqrt{2}\\
&\dots
\end{aligned} \tag{11.1}$$

Après ces quelques calculs, on voit que l'on peut écrire, en généralisant, et avec une arrière-pensée en terme de suite :

$$(\sqrt{2}-1)^n = A_n - B_n\sqrt{2} \tag{11.2}$$

L'exponentielle au secours de $\sqrt{2}$

Le plus grand entier strictement inférieur à $\sqrt{2}$ est 1, ce qui fait de l'expression $(\sqrt{2}-1)$ un nombre strictement inférieur à 1. On peut donc écrire $\lim_{n\to+\infty}(\sqrt{2}-1)^n = 0$.

Et par conséquent $\lim_{n\to+\infty}(A_n - B_n\sqrt{2}) = 0$.

En regardant la cinquième ligne de (11.1), par exemple, et en l'exprimant en terme de valeur approchée, on a :

$$-41+29\sqrt{2} \approx 0 \text{ d'où } \sqrt{2} \approx \frac{41}{29} \approx 1,41379310344828\dots$$

La valeur approchée de $\sqrt{2}$ est d'autant meilleure que n est grand ; on détaille à la page 104 la qualité de la précision de ce genre d'approximation.

À partir de l'écriture (11.2), on peut calculer aisément le terme suivant :

$$\begin{aligned}(\sqrt{2}-1)^n &= A_n - B_n\sqrt{2}\\ (\sqrt{2}-1)^{n+1} &= (\sqrt{2}-1)^n(\sqrt{2}-1)\\ &= (A_n - B_n\sqrt{2})(\sqrt{2}-1)\\ &= A_n\sqrt{2} - A_n - 2B_n + B_n\sqrt{2}\\ &= \underbrace{(-A_n - 2B_n)}_{A_{n+1}} - \underbrace{(-A_n - B_n)}_{B_{n+1}}\sqrt{2}\end{aligned}$$

et ainsi considérer les 3 suites

$$A_n := \begin{cases} A_1 = -1 \\ A_{n+1} = -A_n - 2B_n \end{cases}, \; B_n := \begin{cases} B_1 = -1 \\ B_{n+1} = -A_n - B_n \end{cases} \quad \text{et } R_n := \frac{A_n}{B_n}$$

dont on peut calculer les termes consécutifs avec un tableur :

n	A_n	B_n	$R_n = \frac{A_n}{B_n}$
1	-1	-1	1
2	3	2	1,5
3	-7	-5	1,4
4	17	12	1,4166666666667
5	-41	-29	1,4137931034483
6	99	70	1,4142857142857
7	-239	-169	1,414201183432
8	577	408	1,4142156862745
9	-1393	-985	1,4142131979696
10	3363	2378	1,4142136248949
11	-8119	-5741	1,4142135516461
12	19601	13860	1,4142135642136
13	-47321	-33461	1,4142135620573
14	114243	80782	1,4142135624273
15	-275807	-195025	1,4142135623638
16	665857	470832	1,4142135623747
17	-1607521	-1136689	1,4142135623728
18	3880899	2744210	1,4142135623732
19	-9369319	-6625109	1,4142135623731
20	22619537	15994428	1,4142135623731
...	...	...	...

ou avec un outil de calcul comme *Scilab* :

```
// racine2.sce
a(1)=-1
b(1)=-1
r(1)=a(1)/b(1)
disp(a(1))
disp(b(1))
disp(r(1))
for n=1:10
    a(n+1)=-a(n)-2*b(n)
    b(n+1)=-a(n)-b(n)
    r(n+1)=a(n+1)/b(n+1)
    disp(a(n+1))
    disp(b(n+1))
    disp(r(n+1))
end
```

avec vraisemblablement $\lim_{n\to+\infty} R_n = \sqrt{2}$.

En modifiant la définition des paramètres A_n et B_n, on peut aussi écrire l'expression (11.2) sous la forme

$$(\sqrt{2}-1)^n = (-1)^n(A_n - B_n\sqrt{2}) \tag{11.3}$$

ce qui nous amènerait à considérer les 3 suites

$$A_n := \begin{cases} A_1 = 1 \\ A_{n+1} = -A_n - 2B_n \end{cases}, \; B_n := \begin{cases} B_1 = 1 \\ B_{n+1} = -A_n - B_n \end{cases} \quad \text{et } R_n := \frac{A_n}{B_n}$$

qui engendrent les mêmes résultats R_n.

Héron d'Alexandrie est en embuscade...

La suite de HÉRON (qui converge vers $\sqrt{2}$) est définie par $\begin{cases} u_1 := 1 \\ u_{p+1} := \frac{1}{2}\left(u_p + \frac{2}{u_p}\right) \end{cases}$

Son interprétation géométrique est la suivante : $\sqrt{2}$ est la longueur du côté d'un carré dont l'aire vaut 2. La méthode de HÉRON consiste à démarrer d'un rectangle dont l'aire vaut 2 (2 × 1) et ensuite de remplacer une longueur de côté par la moyenne arithmétique des deux et l'autre par un nombre tel que le produit des deux longueurs (l'aire du rectangle) soit toujours égal à 2.

- Etape 0 : rectangle de côtés 1 et 2
- Etape 1 :
  → un côté devient moyenne arithmétique des deux : $\frac{1}{2}(1+2) = \frac{3}{2} = 1.5$
  → l'autre $= \frac{2}{\text{côté}} = \frac{2}{\frac{3}{2}} = \frac{4}{3} = 1.3333\ldots$
- Etape 2 :
  → un côté devient moyenne arithmétique des deux : $\frac{1}{2}\left(\frac{3}{2} + \frac{4}{3}\right) = \frac{17}{12} = 1.41666\ldots$
  → l'autre $= \frac{2}{\text{côté}} = \frac{2}{\frac{17}{12}} = \frac{24}{17} = 1.41176470588\ldots$
- Etape 3 :
  → un côté devient moyenne arithmétique des deux : $\frac{1}{2}\left(\frac{17}{12} + \frac{24}{17}\right) = \frac{577}{408} = 1.41421568627\ldots$
  → l'autre $= \frac{2}{\text{côté}} = \frac{2}{\frac{577}{408}} = \frac{816}{577} = 1.41421143847\ldots$
- et ainsi de suite ...

À l'étape 1, on retrouve le rapport $\frac{3}{2}$ correspondant à $n = 2$ dans le tableau précédent ; à l'étape 2, on retrouve le rapport $\frac{17}{12}$ correspondant à $n = 4$; à l'étape 3, on retrouve le rapport $\frac{577}{408}$ correspondant à $n = 8$... En comparant dans un tableau la méthode de HÉRON et la méthode qualifiée de « R_n », on a :

R_n	HÉRON
$n = 2$	étape 1
$n = 4$	étape 2
$n = 8$	étape 3
$n = 16$	étape 4
...	...

En effet, de (11.2) ou de (11.3), on peut écrire

$$(\sqrt{2}-1)^{2n} = (A_n - B_n\sqrt{2})^2 = A_n^2 + 2B_n^2 - 2A_nB_n\sqrt{2}$$

Comme $(\sqrt{2}-1)^{2n}$ peut aussi s'écrire sous la forme $A_{2n} - B_{2n}\sqrt{2}$, on en tire, en identifiant les coefficients correspondants,

$$\begin{cases} A_{2n} = A_n^2 + 2B_n^2 \\ B_{2n} = 2A_nB_n \end{cases}$$

d'où

$$R_{2n} = \frac{A_{2n}}{B_{2n}} = \frac{A_n^2 + 2B_n^2}{2A_nB_n} = \frac{1}{2}\left(\frac{A_n}{B_n} + 2\frac{B_n}{A_n}\right) = \frac{1}{2}\left(R_n + \frac{2}{R_n}\right) \tag{11.4}$$

on retrouve la formule de HÉRON.

2 Racine carrée de 3

On laissera au lecteur le soin de vérifier qu'en appliquant le même schéma de calcul que pour $\sqrt{2}$, on peut écrire, en posant

$$(2-\sqrt{3})^n = A_n - B_n\sqrt{3} \tag{11.5}$$

on obtient

$$(2-\sqrt{3})^{n+1} = \underbrace{(2A_n + 3B_n)}_{A_{n+1}} - \underbrace{(A_n + 2B_n)}_{B_{n+1}}\sqrt{3}$$

où 2 est le plus petit entier strictement supérieur à $\sqrt{3}$, ce qui fait de l'expression $(2-\sqrt{3})$ un nombre strictement inférieur à 1.

On peut dès lors considérer les 3 suites

$$A_n := \begin{cases} A_1 = 2 \\ A_{n+1} = 2A_n + 3B_n \end{cases}, \quad B_n := \begin{cases} B_1 = 1 \\ B_{n+1} = A_n + 2B_n \end{cases} \quad \text{et } R_n := \frac{A_n}{B_n},$$

dont les premiers termes sont calculés ci-après, et, vraisemblablement, $\lim_{n\to+\infty} R_n = \sqrt{3}$

n	A_n	B_n	$R_n = \frac{A_n}{B_n}$
1	2	1	2
2	-7	-4	1,75
3	26	15	1,7333333333333
4	-97	-56	1,7321428571429
5	362	209	1,732057416268
6	-1351	-780	1,7320512820513
7	5042	2911	1,7320508416352
8	-18817	-10864	1,7320508100147
9	70226	40545	1,7320508077445
10	-262087	-151316	1,7320508075815
11	978122	564719	1,7320508075698
12	-3650401	-2107560	1,732050807569
13	13623482	7865521	1,7320508075689
...	...	...	...

La suite de HÉRON qui converge vers $\sqrt{3}$ est définie par $\begin{cases} u_1 := 1 \\ u_{p+1} := \frac{1}{2}\left(u_p + \frac{3}{u_p}\right) \end{cases}$

On peut lui associer la même interprétation géométrique, en démarrant d'un rectangle dont l'aire vaut 3, et procéder au même schéma de calcul que (11.4) page 103, qui mène au résultat

$$R_{2n} = \frac{A_{2n}}{B_{2n}} = \ldots = \frac{1}{2}\left(R_n + \frac{3}{R_n}\right)$$

3 Une valeur approchée de bonne qualité

Développons l'exemple de $\sqrt{3}$, mais le principe est le même dans les autres cas.

Soit $\epsilon := 2 - \sqrt{3} < \frac{1}{2}$ l'erreur commise lors de la première approximation.

Par (11.5), on a $\epsilon^n = (2-\sqrt{3})^n = (A_n - B_n\sqrt{3})$, d'où

$$\begin{aligned} A_n - B_n\sqrt{3} &= \epsilon^n \\ \frac{A_n}{B_n} - \sqrt{3} &= \frac{\epsilon^n}{B_n} \\ \left|\frac{A_n}{B_n} - \sqrt{3}\right| = \frac{\epsilon^n}{B_n} &< \frac{\left(\frac{1}{2}\right)^n}{B_n} = \frac{1}{2^n \cdot B_n} \end{aligned}$$

Pour $n = 9$, on a $\left|\frac{A_9}{B_9} - \sqrt{3}\right| < \frac{1}{2^9 \cdot B_9} = \frac{1}{512 \cdot 40\,545} = \frac{1}{20\,759\,040} \approx 4 \cdot 10^{-8}$

soit 7 chiffres corrects après la virgule en approchant $\sqrt{3}$ par le rationnel $\frac{70\,226}{40\,545}$.

4 Et même si...

Pour les approximations de $\sqrt{2}$ et $\sqrt{3}$, on a utilisé l'entier qui leur était le plus proche. Et si... ?

Calculons (par exemple) une valeur approchée de $\sqrt{17}$ en développant $(\sqrt{17}-3)^n$ alors que 3 n'est pas le plus grand entier strictement inférieur à $\sqrt{17}$, et donc que $\lim_{n\to+\infty}(\sqrt{17}-3)^n = +\infty$.

En appliquant les schémas précédents, on a :

$$(\sqrt{17}-3)^{n+1} = \underbrace{(-3A_n - 17B_n)}_{A_{n+1}} - \underbrace{(-A_n - 3B_n)}_{B_{n+1}}\sqrt{17}$$

ce qui, en termes de tableur, donne les résultats suivants :

n	A_n	B_n	$R_n = \frac{A_n}{B_n}$
1	-3	-1	3
2	26	6	4,3333333333333
3	-180	-44	4,0909090909091
4	1288	312	4,1282051282051
5	-9168	-2224	4,1223021582734
...	...	...	...
15	-3083264409600	-747801460736	4,12310562561
16	21962418061312	5326668791808	4,1231056256189
17	-156440623644672	-37942424436736	4,1231056256175
18	1,1143430863585E+015	270267896954880	4,1231056256177
19	-7,9375835073086E+015	-1,9251467772232E+015	4,1231056256177
20	5,6540245734720E+016	1,3713023838978E+016	4,1231056256177
...	...	...	...

Pour information, la version 14 décimales du tableur OpenOffice Calc pour $\sqrt{17}$ est 4,12310562561766.

La méthode fonctionne encore, et même admirablement bien !

En fait, si $\alpha \in \mathbb{N}_0$, $D \in \mathbb{N}_0$ non carré parfait, et si $(\sqrt{D}-\alpha)^n = (-1)^n\left(A_n - B_n\sqrt{D}\right)$, alors $\lim_{n\to+\infty} \frac{A_n}{B_n} = \sqrt{D}$ peu importe l'ordre de grandeur de $(\sqrt{D}-\alpha)$, et en particulier même si $|\sqrt{D}-\alpha| > 1$!

5 Racine cubique de cinq : deux pour le prix d'une !

Toujours dans la même veine, semble-t-il inépuisable, utilisons le même schéma de calcul pour trouver une estimation de $\sqrt[3]{5}$.

$(2-\sqrt[3]{5}) < 1$, d'où $\lim_{n\to+\infty}(2-\sqrt[3]{5})^n = 0$. On a successivement, en appliquant une simple distributivité :

$$\begin{aligned}
(2-\sqrt[3]{5})^1 &= 2-\sqrt[3]{5}\\
(2-\sqrt[3]{5})^2 &= 4-4\sqrt[3]{5}+\sqrt[3]{25}\\
(2-\sqrt[3]{5})^3 &= 3-12\sqrt[3]{5}+6\sqrt[3]{25}\\
(2-\sqrt[3]{5})^4 &= -24-27\sqrt[3]{5}+24\sqrt[3]{25}\\
(2-\sqrt[3]{5})^5 &= -168-30\sqrt[3]{5}+75\sqrt[3]{25}\\
(2-\sqrt[3]{5})^6 &= -711+108\sqrt[3]{5}+180\sqrt[3]{25}\\
\dots &= \dots\\
(2-\sqrt[3]{5})^9 &= -9693+14256\sqrt[3]{5}-5022\sqrt[3]{25}\\
(2-\sqrt[3]{5})^{10} &= 5724+38205\sqrt[3]{5}-24300\sqrt[3]{25}\\
\dots &= \dots
\end{aligned} \tag{11.6}$$

De ces lignes successives, on peut exprimer

$$\begin{aligned}
(2-\sqrt[3]{5})^n &= A_n + B_n\sqrt[3]{5} + C_n\sqrt[3]{25}\\
(2-\sqrt[3]{5})^{n+1} &= \left(A_n + B_n\sqrt[3]{5} + C_n\sqrt[3]{25}\right)\cdot(2-\sqrt[3]{5})\\
&= \underbrace{(2A_n-5C_n)}_{A_{n+1}} + \underbrace{(-A_n+2B_n)}_{B_{n+1}}\sqrt[3]{5} + \underbrace{(-B_n+2C_n)}_{C_{n+1}}\sqrt[3]{25}
\end{aligned}$$

Si les calculs de (11.6) sont quelque peu longs, mais faciles, l'utilisation du tableur, une fois de plus, rend tout cela bien plus rapide :

n	A_n	B_n	C_n
1	2	-1	0
2	4	-4	1
3	3	-12	6
...	...	...	...
8	-5904	4176	-423
9	-9693	14256	-5022
10	5724	38205	-24300
...	...	...	...

mais pas encore de trace de $\sqrt[3]{5}$...

En regardant les puissances 9 et 10 (par exemple) dans (11.6) et sachant que $\lim_{n\to+\infty}(2-\sqrt[3]{5})^n =$

0, on peut écrire le système linéaire

$$\begin{cases} -9693 + 14256\sqrt[3]{5} - 5022\sqrt[3]{25} \approx 0 \\ 5724 + 38205\sqrt[3]{5} - 24300\sqrt[3]{25} \approx 0 \end{cases}$$

En posant $x = \sqrt[3]{5}$ et $y = \sqrt[3]{25}$ les deux « pseudo-inconnues », on a

$$\begin{cases} 14256x - 5022y \approx 9693 \\ 38205x - 24300y \approx -5724 \end{cases}$$

le système peut s'écrire

$$\begin{pmatrix} 14256 & -5022 \\ 38205 & -24300 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} \approx \begin{pmatrix} 9693 \\ -5724 \end{pmatrix}$$

et a pour solution

$$\begin{pmatrix} x \\ y \end{pmatrix} \approx \begin{pmatrix} 14256 & -5022 \\ 38205 & -24300 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 9693 \\ -5724 \end{pmatrix}$$

Il suffit de trois lignes de code en mode direct avec le logiciel *Scilab* pour résoudre l'équation matricielle :

```
A=[14256 -5022 ; 38205 -24300]
B=[9693 -5724]
S=inv(A)*B
```

et obtenir les solutions $x = \sqrt[3]{5} \approx 1.7099759445309$ et $y = \sqrt[3]{25} \approx 2.9240177350125$.

Remarque : dans le code *Scilab*, une matrice est définie entre crochets rectangulaires, les termes sont séparés par une [1] espace et les lignes par un point-virgule.

Si le système est résolu « manuellement » par une méthode de substitution ou d'élimination, on obtient les résultats $x \approx \dfrac{60422}{35335}$ et $y \approx \dfrac{619921}{212010}$.

Qualité de l'approximation

Comme pour $\sqrt{3}$, posons $\epsilon := 2 - \sqrt[3]{5} < \dfrac{3}{10}$ l'erreur commise lors de la première approximation.

Pour l'estimer en approchant $\sqrt[3]{5}$ par $\dfrac{60422}{35335}$, il faut résoudre le système

$$\begin{cases} -9693 + 14256\sqrt[3]{5} - 5022\sqrt[3]{25} = \epsilon^9 \\ 5724 + 38205\sqrt[3]{5} - 24300\sqrt[3]{25} = \epsilon^{10} \end{cases}$$

Bel exercice algébrique de simplifications, après avoir éliminé $\sqrt[3]{25}$ du système, il reste

$$\sqrt[3]{5} = \frac{162\epsilon^9(150 - 31\epsilon) + 264285828}{154555290}$$

1. En typographie, espace est *féminin*, ce n'est donc pas une faute de frappe.

qui peut s'écrire

$$\sqrt[3]{5} = \frac{60422}{35335} + \underbrace{\frac{\epsilon^9(150-31\epsilon)}{954045}}_{<2,902796\cdot 10^{-9}}$$

soit

$$0 < \sqrt[3]{5} - \frac{60422}{35335} < 2,902796 \cdot 10^{-9}$$

et, en guise de conclusion

$$\sqrt[3]{5} \approx 1,7099759466767\ldots$$
$$\frac{60422}{35335} \approx 1,7099759445309\ldots$$

Des outils du tableur : valeur cible et solveur

Résumé. *Le tableur est un outil de calcul extrêmement riche et puissant. Il regorge d'outils de calculs, du plus simple au plus complexe, dans de nombreux domaines faisant appel aux calculs. En voici deux, utilisables pour la résolution d'équations, de problèmes dits « de programmation linéaire », et de problèmes d'optimisation (entre autres) : les outils* valeur cible *et* solveur.

Pour cet article, je vais utiliser le tableur *Calc* de la suite bureautique *OpenOffice*, issue du monde des logiciels libres, et donc gratuits. Pour les utilisateurs du tableur *Excel*, les explications sont semblables, jusqu'à la version 2003 en tout cas, à quelques détails cosmétiques près ; en ce qui concerne les versions ultérieures, les outils existent encore, mais les commandes et menus sont sensiblement distincts.

Il faut toutefois être honnête : *Excel* étant sur le marché des tableurs depuis bien plus longtemps que *Calc*, certains de ses outils, et c'est (malheureusement, pour l'instant encore) le cas du *solveur*, sont plus sophistiqués. En effet, le *solveur* de *Calc* ne peut traiter que des problèmes linéaires. Encore un peu de patience.

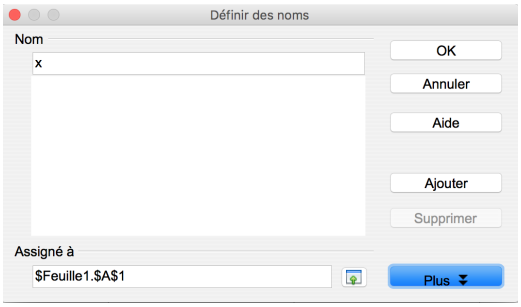
Je suppose également que le lecteur est familier avec les bases : cellule, adresse (relative et absolue), plage de cellules, sélection d'une cellule ou d'une plage, contenu d'une cellule, syntaxe d'une formule, mais nos besoins en cette matière seront assez limités.

1 Prérequis : nommer une cellule

Nommer une cellule ou une plage de cellules permet de rendre une formule plus « lisible ». Il est en effet plus facile d'écrire ou de lire : `=A1*tauxtva` que `=A1*$C$10` et `=somme(ventes)` que `=somme($A$1:$C$10)`

Il y a plusieurs manières de procéder, la plus classique est :

- Sélectionner la cellule ou la plage à nommer
- Insertion > Noms > Définir > taper un nom



Il y a des règles à respecter pour la syntaxe du nom : il ne peut pas contenir d'espace ni de caractères spéciaux ni de lettres accentuées ; il doit commencer par une lettre, ne peut pas être l'adresse d'une autre cellule, il doit être unique dans tout le classeur, etc.

Il n'y aura jamais de problème si on n'utilise que des lettres (non accentuées), des chiffres, et éventuellement le trait de soulignement.
Il vaut mieux choisir des noms courts mais suffisamment explicites : `ventes_juin_2016` est plus simple à encoder que `ventes_du_mois_de_juin_de_l_annee_2016` et plus facile à interpréter que `veju16`.

Il est important de savoir qu'un nom de cellule (ou de plage) a le statut d'adresse absolue, et qu'un nom est connu dans *toutes* les feuilles du classeur.

Il est impossible de modifier un nom ; si besoin est, il faut d'abord le supprimer, puis le créer comme souhaité. Pour tous les détails, voir l'aide du logiciel.

2 Valeur cible

L'utilisation « traditionnelle » du tableur consiste à calculer un (ou plusieurs) résultat(s) à partir de donnée(s). À cet effet, des cellules contiennent des données, et d'autres des formules.

Si la donnée est , alors le résultat est : donnée $\longrightarrow$ résultat

L'outil *valeur cible*, permet de calculer une donnée qui produira (si possible) un résultat souhaité.

Pour que le résultat soit , alors la donnée doit être : donnée $\longleftarrow$ résultat

2.1 Exemple 1

On veut résoudre l'équation $x^3 - x - 1 = 0$, ce qui revient à trouver le[1] zéro de la fonction $f(x) = x^3 - x - 1$. On présente le modèle de calcul suivant :

	A	B
1	x	3
2	f(x)	$= x^3 - x - 1$

Les cellules `A1` et `A2` contiennent du texte explicatif (sans intérêt pour les calculs, mais important pour leur lecture) ; la cellule `B1` contient une donnée initiale, cette cellule DOIT être nommée `x` pour que la formule dans la cellule `B2` ne génère pas un message d'erreur. Cette cellule affiche le résultat 23. Le fait de nommer `B1` en `x` n'est pas indispensable, mais cela permet d'encoder la formule de la cellule `B2` à la « sauce mathématique traditionnelle ».

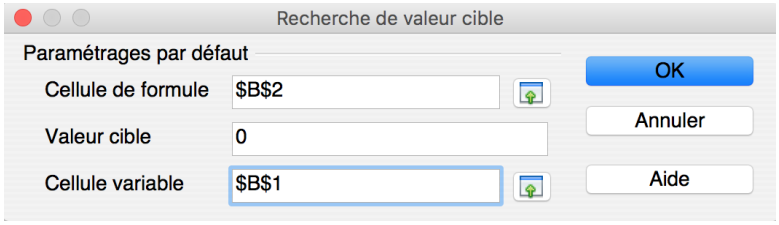
Pour résoudre l'équation $x^3 - x - 1 = 0$, il « suffirait » de remplacer la donnée « 3 » de la cellule `B1` jusqu'à l'obtention de 0 dans la cellule `B2`, on peut ainsi chercher longtemps

L'outil valeur cible

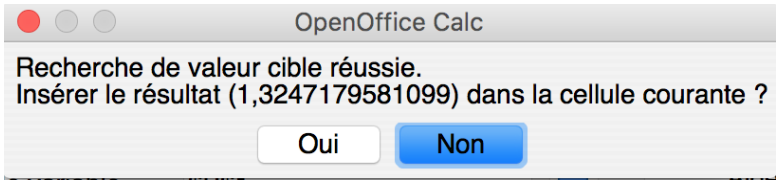
Après avoir préparé donnée et formule, la procédure est la suivante :

- sélectionner la cellule contenant la formule (pas indispensable, mais plus simple) ;
- Outils > Recherche de valeur cible et compléter la boîte de dialogue comme ci-après :

1. On sait qu'il est unique dans $\mathbb{R}$.



et on obtient le résultat suivant :



En choisissant Oui on obtient en B2 le résultat $3,6897 \cdot 10^{-9}$ ce qui indique que la racine obtenue est bien une valeur approchée, comme on pouvait s'y attendre.

Le résultat obtenu peut différer légèrement selon la valeur initiale (cellule B1).

2.2 Exemple 2

Résolution de l'équation $\sin x = \cos x$. Par rapport à l'exemple 1, cette équation-ci possède une infinité de solutions. La disposition des données est la même :

	A	B
1	x	3
2	f(x)	=SIN(x) – COS(x)

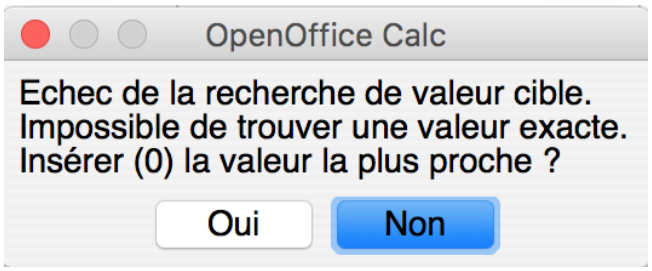
Dans ce cas, la solution trouvée par l'outil *valeur cible* dépendra de la valeur initiale dans la cellule B1.

B1= 3 → B1=3,926990645068 et B2= 0,0000002431305

B1= −2 → B1=−2,3561945035607 et B2= 0,0000000189057

2.3 Exemple 3

On peut toujours rêver, mais avec le tableur ça ne marche pas non plus : résolution de l'équation $x^2 + 1 = 0$.



3 Solveur

Le *solveur* est un outil d'optimisation ; si *valeur cible* ne permet de calculer qu'une seule cellule, le *solveur* permet, lui, d'en calculer plusieurs.

Un problème d'optimisation se caractérise par une valeur à minimiser ou à maximiser, et par plusieurs contraintes qui s'expriment sous forme d'égalités ou d'inégalités.

La valeur à optimiser et les contraintes doivent être entrées dans la feuille de calcul, une cellule pour chaque donnée variable.

3.1 Exemple 1

Un cas classique de « programmation linéaire » : douze couturières travaillent dans un atelier où l'on produit des chemises et des robes ; une couturière met 0.5 h pour produire une chemise et 1.5 h pour produire une robe ; le bénéfice est de 2,6 € par chemise et de 8,5 € par robe ; l'entreprise a évidemment pour objectif de réaliser le plus grand bénéfice possible, et on demande de calculer le nombre de chemises et de robes à produire pour atteindre cet objectif.

La production est soumise à certaines contraintes :

- la durée de travail hebdomadaire est de 35 h par semaine ;
- un client de longue date se fait livrer chaque semaine 20 chemises ; pour ne pas perdre le client, il faut absolument produire au minimum ces 20 chemises ;
- le stock de tissu ne permet pas de produire plus de 300 robes ;
- étant donné que l'on ne peut vendre que des chemises et des robes entières, le nombre de pièces produites doit donc être entier.

Disposition des données dans la feuille

L'utilisation d'un logiciel ne dispense pas de la réflexion, bien au contraire. Dans un premier temps il convient de placer chaque donnée et chaque élément calculable dans une cellule :

	A	B
1	nbre chemises	1
2	nbre robes	1
3	durée travail	=0,5*B1+1,5*B2
4	bénéfice	=2,6*B1+8,5*B2

Les cellules B1 et B2 contiennent 2 valeurs initiales quelconques (ce n'est pas obligatoire, puisque le *solveur* va les calculer, mais en attendant, elles produisent des résultats « rassurants » dans les cellules B3 et B4).

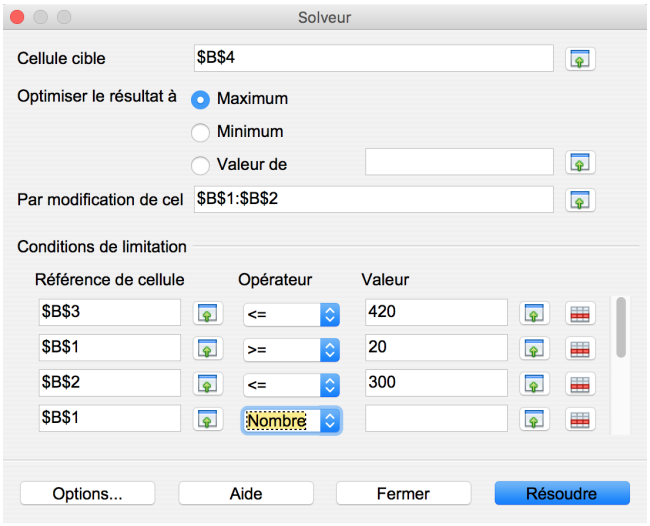
3.2 Préparation du solveur

Si c'est la première fois que vous compter utiliser le *solveur* sous Windows, il est possible qu'il n'apparaisse pas dans le menu *Outils* ; il faut alors l'activer via

- Outils > Macros complémentaires
- cocher la case Solveur

puis

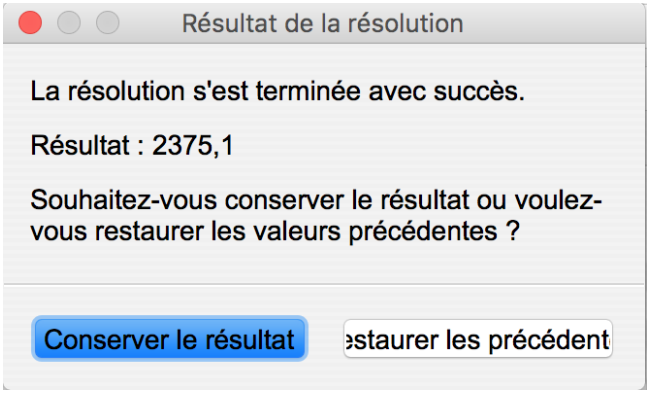
- Outils > Solveur
- compléter la boîte de dialogue comme ci-après :



La première contrainte porte sur la cellule B3 et exprime que la durée totale de travail ne peut pas excéder 420 h (12×35) ; la deuxième porte sur la cellule B1 et exprime qu'il faut produire au moins 20 chemises ; la troisième porte sur la cellule B2 et exprime qu'on ne peut pas produire plus de 300 robes ; les deux dernières portent également sur les cellules B1 et B2 et expriment que les vêtements sont « complets ».

Il n'est pas possible d'avoir une vue complète de la boîte de dialogue à l'écran, mais pour la quatrième contrainte, derrière le mot « nombre » il y a bien « nombre entier » et il y a une cinquième contrainte semblable pour la cellule $B2$.

En cliquant sur le bouton Résoudre on obtient le message suivant :



En cliquant sur le bouton Conserver le résultat le tableur assigne respectivement 21 et 273 aux cellules B1 et B2, correspondant à une production de 21 chemises et 273 robes, pour un bénéfice maximal de 2375,1 €.

3.3 Exemple 2

Un autre classique des applications des dérivées, l'optimisation de la surface d'une boîte cylindrique (avec couvercle et fond) de volume donné, à partir du rayon de la base et de la hauteur. Pour l'exemple, nous fixerons le volume à 800 cm^3, ce qui correspond à une grosse boîte de cassoulet, de haricots, de petits pois, selon les goûts de chacun. On demande de calculer la hauteur de la boîte et le rayon de la base pour que la surface soit minimale.

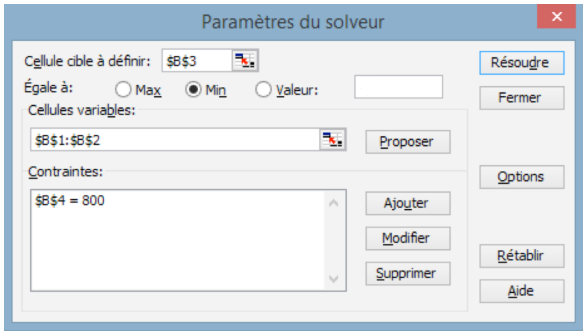
Le problème n'étant pas linéaire, il faudra utiliser un autre tableur que *OpenOffice Calc*, nous opterons pour *MicrosoftOffice Excel* (version 2003), exceptionnellement sous Windows.

Disposition des données dans la feuille

	A	B
1	rayon	1
2	hauteur	1
3	surface	=2*PI()*B1*(B2+B1)
4	volume	=PI()*B1*B1*B2

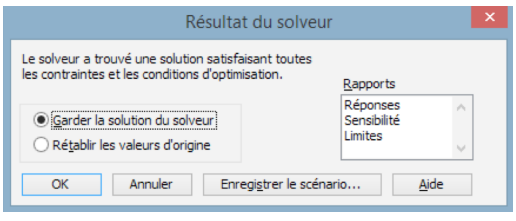
Ici non plus les deux valeurs initiales dans les cellules B1 et B2 n'ont aucune importance ; en B3, la surface de la boîte (un rectangle et deux cercles), $2\pi RH + 2\pi R^2$, et en B4, le volume, $\pi R^2 H$.

- Outils > Solveur
- compléter la boîte de dialogue comme ci-contre :



La boîte de dialogue est sensiblement différente qu'avec *OpenOffice Calc*, mais on y retrouve les mêmes rubriques.

En cliquant sur le bouton Résoudre on obtient le message suivant :



Avec l'option
• Garder la solution du solveur
on obtient les résultats suivants :

	A	B
1	rayon	5,0307997452698
2	hauteur	10,0615769685651
3	surface	477,061682486585
4	volume	800

Ce qui correspond approximativement à la réalité.

Graphiques animés : cycloïde, épicycloïde, hypocycloïde

Résumé. *Les barres de défilement font partie de notre univers informatique journalier ; on en voit quasi deux en permanence sur notre écran, une horizontale et une verticale, pour faire défiler le texte lorsque la fenêtre est trop petite. Dans l'environnement du tableur, une barre de défilement permet de faire varier rapidement le contenu d'une cellule. Si cette cellule contient un paramètre de position d'un objet dessiné dans le plan, alors la barre de défilement permet de faire varier la position de l'objet et créer ainsi une petite animation.*

Nous commencerons par visualiser le sinus et le cosinus d'un angle en faisant tourner un point sur un cercle, et dans la lancée, nous ferons tourner un cercle sur une droite, puis à l'extérieur et à l'intérieur d'un autre cercle.

C'est le tableur gratuit *OpenOffice Calc* qui est utilisé pour les manipulations. Les instructions sont très semblables pour les utilisateurs du tableur *Microsoft Excel* jusqu'à la version 2003 ; pour les versions plus récentes, l'outil existe, nous renvoyons le lecteur au support du logiciel.

1 Barre de défilement

1.1 Un peu de vocabulaire

La barre de défilement est un objet que l'on trouve dans la barre d'outils « Contrôles de formulaires », qu'il faudra afficher puisqu'elle n'est pas présente à l'écran par défaut.

Pour dessiner le point et le cercle, nous allons utiliser un graphique (diagramme, chez *OpenOffice*) de type XY ; comme son nom l'indique, il nous faudra calculer des abscisses et des ordonnées.

Dans ce type de graphique, on appelle *série* une plage de cellules d'ordonnées correspondant à une plage de cellules d'abscisses ; une plage peut aussi être réduite à une seule cellule ; une plage de cellules d'abscisses peut servir pour plusieurs séries.

Sur un graphique, on peut représenter simultanément plusieurs séries, et jouer sur les couleurs, styles de traits, etc.

Pour *activer* un graphique *en vue de modifier un de ses composants*, le plus simple est d'y faire un double-clic, en prenant soin de ne pas faire ce double-clic sur l'un des composants ; le graphique est alors bordé d'un trait gris avec les huit poignées de dimensionnement. Le « non-novice » peut aussi faire directement un double-clic sur le composant à modifier. Pour *désactiver* le graphique, il suffit de cliquer en dehors de celui-ci.

1.2 Un point fixe

On souhaite représenter un point P de coordonnée $(\cos\alpha, \sin\alpha)$, et rappelons au passage que le tableur ne parle qu'en radians, alors que le commun des mortels préfère le degré…

Disposition des données

	A	B
1	angle (deg)	32
2		
3	point P	
4	abscisse X	=COS(RADIANS(B1))
5	ordonnée Y	=SIN(RADIANS(B1))

La cellule `B1` contient une valeur d'angle quelconque (supposée en degrés), comprise entre 0 et 360. La cellule `A3` contient le nom que l'on donnera à la série. La fonction `RADIANS(...)` convertit en radians, comme son nom l'indique.

Création du graphique

Il faut commencer par mettre le pointeur souris dans une cellule « suffisamment éloignée » de la plage `A1:B5` pour s'assurer que le tableur ne prenne aucune initiative quant à la détection des données, puis

- Insertion > Diagramme
- un assistant se met à votre service…
- étape 1, choisir le type XY (dispersion), sous-type Points seuls
- étape 2, ne rien faire
- étape 3, clic sur Ajouter pour ajouter une série
- compléter les 3 plages demandées, à savoir Nom : `A3`, Valeurs X : `B4`, Valeurs Y : `B5` (selon la version du tableur, il est peut-être nécessaire d'utiliser l'adressage absolu des cellules : `$A$3`…, ou alors utiliser la sélection dans la feuille de calcul à l'aide du bouton situé à droite de la zone de texte)

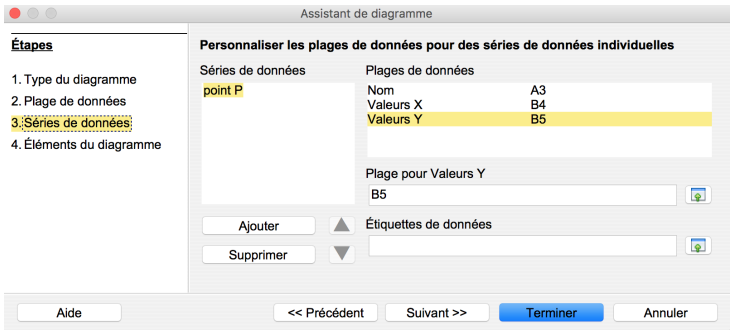


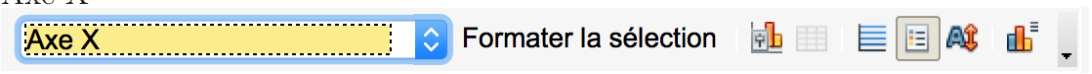
Fig. 2 *Ajout d'une série de données*

- étape 4, décocher l'affichage des grilles

- Terminer

À ce stade, on voit un point (carré bleu, mais on peut modifier la forme et la couleur) dans un système de coordonnées cartésiennes ; si on modifie la valeur de l'angle (cellule B1), c'est la graduation des axes qui varie, car par défaut elle est automatique et s'adapte aux données. Nous allons dès lors « fixer » la graduation :

- activer le graphique (double-clic…)
- la barre d'outils *Graphique* est affichée ; dérouler la liste des composants et y sélectionner Axe X



puis Formater la sélection

- dans l'onglet ÉCHELLE, décocher l'option 'Automatique', puis Minimum : -1 et Maximum : 1
- faire de même pour Axe Y

L'affichage des axes et de leur graduation étant inutile, on peut la supprimer :

- activer le graphique, puis clic-droit, et Insérer/supprimer des axes…
- décocher Axe X et Axe Y

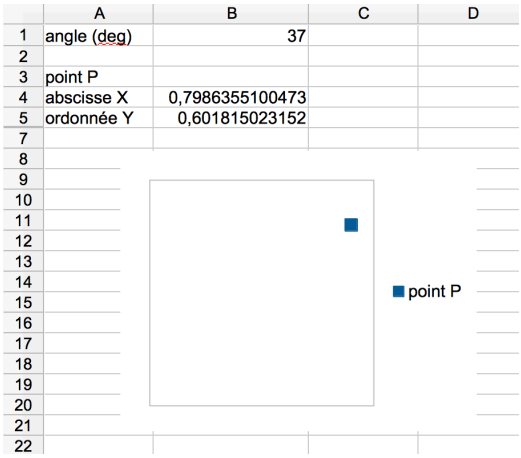


Fig. 3 *Un point P dans le plan*

Jusqu'ici rien de bien spectaculaire, mais on y arrive…

1.3 La barre de défilement

Création de la barre

Il faut commencer par afficher la barre d'outils adhoc :

- Affichage > Barre d'outils > Contrôles de formulaire

- Remarquer que l'icône  (Dés)activer le mode Ébauche est enfoncée ; c'est ce mode qui permet de dessiner les contrôles
- clic sur l'icône  Barre de défilement et dessiner une zone rectangulaire à proximité de la cellule B1 ; la voilà enfin, mais elle n'est pas encore opérationnelle :

Fig. 4 *Barre de défilement*

On pourra, si on le souhaite, modifier sa taille, sa position, sa couleur, son orientation, ... en modifiant ses propriétés :

Propriétés de la barre

- clic-droit > Contrôle...
  on peut traiter beaucoup d'aspects, seuls trois sont importants pour notre activité
- onglet Général
  Valeur de défilement min : 0
  Valeur de défilement max : 360
- onglet Données
  Cellule liée : B1

Activation de la barre

- Relâcher l'icône  (Dés)activer le mode Ébauche

Il n'y a plus qu'à déplacer le curseur de la barre, ou utiliser les boutons avant/arrière pour faire tourner le point P sur un cercle (de rayon 1).

Malheureusement, le graphique de type XY ne propose pas le mode orthonormé. Il sera donc peut-être nécessaire de modifier la taille du graphique « à l'oeil » pour donner l'impression d'une trajectoire circulaire.

2 Le cercle

Complétons le graphique en y ajoutant le cercle ; il suffira d'y ajouter une série créée à partir d'une plage d'abscisses et d'ordonnées définissant un cercle.

Calculs

Construisons un tableau de 60 couples $(\cos\alpha, \sin\alpha)$ avec $0 \leqslant \alpha \leqslant 2\pi$. On peut placer ce tableau n'importe où sur la feuille, à un endroit qui ne fait pas d'ombre au graphique, par exemple

	A	B	C
28	cercle		
29	α	$\cos\alpha$	$\sin\alpha$
30	0	=COS(A30)	=SIN(A30)
31	=A30+2*PI()/59	= COS(A31)	=SIN(A31)
...	...	...	...
89	=A88+2*PI()/59	=COS(B89)	SIN(B89)

Ajout de la série

- activer le graphique
- Format > Plages de données...
- onglet Séries de données
- clic sur Ajouter pour ajouter une série
- compléter les 3 plages demandées, à savoir Nom : `A38`, Valeurs X : `B30:B89`, Valeurs Y : `C30:C89`

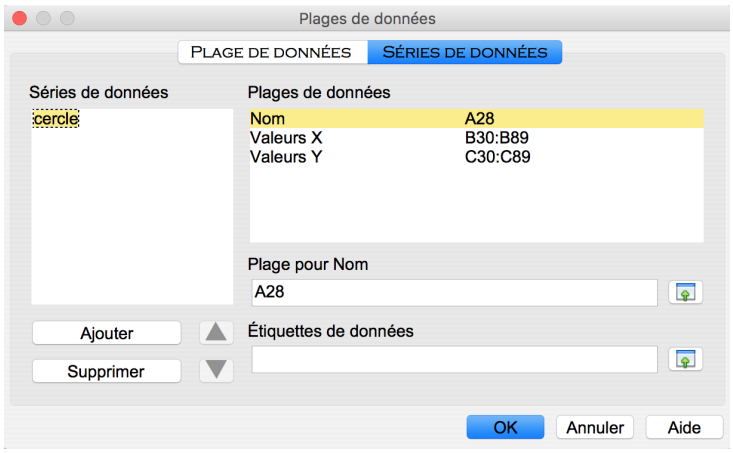


Fig. 5 *Ajout de la série 'cercle'*

- OK

On voit apparaître le cercle constitué de 60 carrés (bleus), ce qui n'est pas très esthétique. Il suffit de modifier les propriétés de la série 'cercle'

Propriétés de la série

- activer le graphique
- dans la barre d'outils 'Graphique', 

afficher la liste des composants, choisir Série de données 'cercle', puis Formater la sélection

- onglet Ligne, choisir un style, une couleur, une largeur, supprimer le symbole, ...

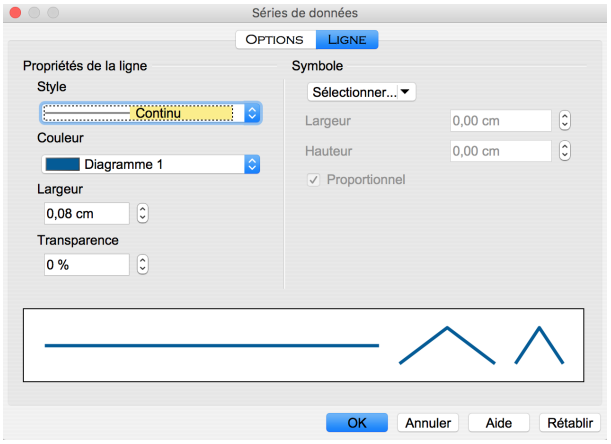


Fig. 6 *Propriétés de la série 'cercle'*

- OK

Résultat final

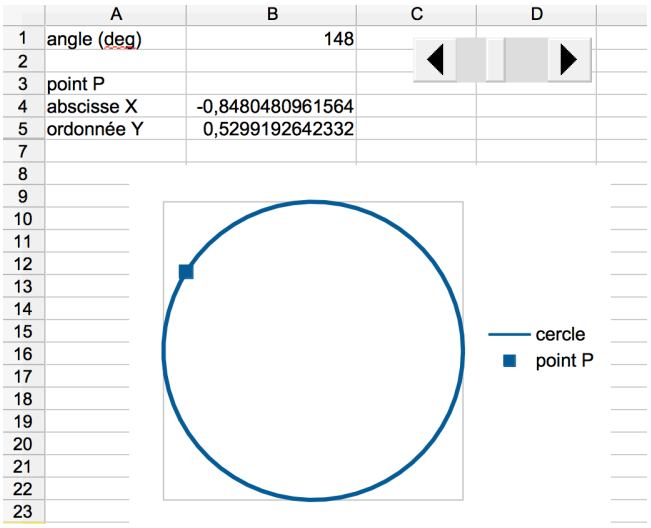


Fig. 7 *Résultat final*

Si on ne souhaite pas être encombré par tous les calculs, il suffit, une fois le graphique terminé, de mettre toutes les plages de calculs en blanc sur fond blanc, ou mieux, de masquer les lignes et colonnes correspondantes.

Pour épurer le graphique, on peut aussi supprimer la légende, ce qui va libérer un peu de place supplémentaire.

Sinus et cosinus

Les manipulations qui précèdent peuvent faire l'objet d'un cours d'informatique, voire de type « activités mathématiques complémentaires » (il y a un peu de trigonométrie), mais on peut aussi utiliser le « produit fini » dans un cours de mathématiques : on se propose de matérialiser le sinus et le cosinus d'un angle déterminé par un point P sur le cercle trigonométrique (nous noterons O son centre de coordonnées $(0,0)$).

Il nous faudra pour cela représenter trois séries supplémentaires sur notre graphique : un « segment sinus » déterminé par deux points de coordonnées respectives $(0,0)$ et $(0,\sin\alpha)$, un « segment cosinus » déterminé par deux points de coordonnées respectives $(0,0)$ et $(0,\cos\alpha)$, et le rayon OP du cercle déterminé par deux points de coordonnées respectives $(0,0)$ et $(\cos\alpha,\sin\alpha)$.

Voici les trois tableaux de calculs qui définissent ces trois séries (rappelons qu'une cellule doit contenir le nom de la cellule, et qu'il faut une plage d'abscisses et une plage d'ordonnées) :

	F	G	H
2	sinus	X	Y
3	origine	0	0
4	extrémité	0	=B5

	F	G	H
6	cosinus	X	Y
7	origine	0	0
8	extrémité	0	=B4

	F	G	H
10	OP	X	Y
11	origine	0	0
12	extrémité	=B4	=B5

Il faut ensuite ajouter ces trois séries au graphique existant ; pour rappel :

- activer le graphique
- Format > Plages de données...
- onglet Séries de données, Ajouter
- compléter les 3 plages demandées,
- OK

L'aspect des séries (couleur, épaisseur, motifs...) est automatique, et ne correspond pas nécessairement à notre goût ; pour modifier :

- activer le graphique
- dans la barre d'outils 'Graphique', afficher la liste des composants, choisir la série à modifier, puis Formater la sélection
- faire les modifications souhaitées
- OK

On peut aussi rétablir éventuellement l'affichage des axes, ou programmer plusieurs tours (modifier la valeur de défilement max dans les propriétés de la barre de défilement).
Et il n'y a plus qu'à jouer avec la barre de défilement...

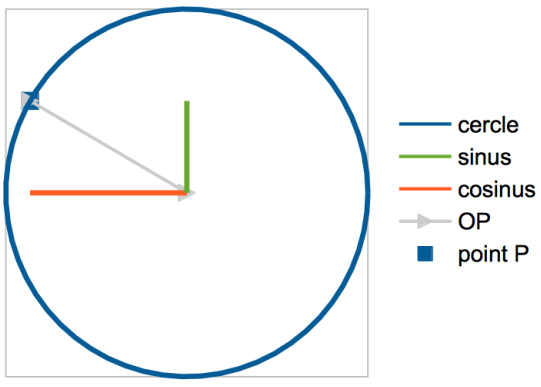


Fig. 8 *Sinus et cosinus*

3 La cycloïde

La cycloïde est la courbe engendrée par un point fixe sur un cercle qui roule sans glisser sur une droite.
C'est la courbe préférée des cyclos du dimanche matin ; fixez la valve d'une roue et imaginez la courbe décrite par celle-ci... et lorsque le cyclo se targue d'avoir roulé 60 km, la valve, elle, en a parcouru 76,39...
En espérant vous donner envie de sortir votre vélo... mais vous ne verrez jamais plus une valve comme avant !

3.1 La théorie mathématique, un peu

Dans tout bon livre d'analyse, on trouve les équations paramétriques de la cycloïde (le but ici est d'utiliser le produit fini, pas d'amener les éléments nécessaires à l'élaboration des équations) :

$$\begin{cases} x = a\beta - a\sin\beta \\ y = a - a\cos\beta \end{cases} \tag{13.1}$$

où a est le rayon du cercle et β l'angle déterminé par un point P fixe sur le cercle.

3.2 Les données pour le tableur

Une cellule (nommée `a`) contiendra le rayon du cercle ; une cellule (nommée `tours`) contiendra le nombre de cycles et une cellule (nommée `beta`) contiendra le paramètre angulaire qui variera avec la barre de défilement dans l'intervalle $[0, 2\pi\text{tours}]$.

Le graphique sera composé de trois séries :

- la cycloïde (courbe fixe) ;
- le cercle (courbe mobile) ;
- le point P (fixe sur le cercle et mobile sur la cycloïde).

Puisqu'il faut bien choisir, on représentera 101 points de la cycloïde, 101 points du cercle, et 101 valeurs du paramètre `beta`.

Attention : les valeurs de défilement de la barre ne pouvant être que des nombres *naturels*, pendant que la cellule liée (B4) variera de 0 à 100, la cellule nommée `beta`, le paramètre angulaire, variera de 0 à 2πtours ; elle doit donc contenir une formule qui transforme un naturel dans $[0, 100]$ (cellule B4) en un réel dans $[0, 2\pi\text{tours}]$.

	A	B
1	rayon cercle	3
2	nb cycles	2
3	beta	=2*pi()*tours*$B$4/100
4	cell. liée	

La barre de défilement

Une fois créée, il faut définir ses trois propriétés principales :
valeur de défilement minimum : 0 ; valeur de défilement maximum : 100 ; cellule liée : `B4`.

La cycloïde

Préparer le tableau

	A	B	C
19	cycloïde		
20	beta	X	Y
21	0	=a*A21-a*SIN(A21)	=a-a*COS(A21)
22	=A21+2*PI()*tours/100	=a*A22-a*SIN(A22)	=a-a*COS(A22)
...	...	...	...
121	=A120+2*PI()*tours/100	=a*A120-a*SIN(A120)	=a-a*COS(A120)

puis créer un diagramme avec une série ayant comme caractéristiques :
Plage pour nom : `A19` ; Valeurs X : `B21:B121` ; Valeurs Y : `C21:C121`.

Modifier éventuellement les propriétés de la série en ligne continue et couleur au choix.

Le point P

	A	B	C
9	P	X	Y
10		=a*beta-a*SIN(beta)	=a-a*COS(beta)

Ajouter au diagramme une série ayant comme caractéristiques :
Plage pour nom : `A9` ; Valeurs X : `B10` ; Valeurs Y : `C10`.

Modifier éventuellement les propriétés de la série en choisissant un symbole pour le point et une couleur.

Le cercle

Puisque la *roue du vélo* est sur la route, on commence par translater verticalement et vers le haut, d'une amplitude a (le rayon du cercle), le cercle de centre $(0,0)$ et de rayon a, ce qui nous ramène aux équations paramétriques

$$\begin{cases} x = a\cos\beta \\ y = a + a\sin\beta \end{cases} \tag{13.2}$$

Il n'est pas possible de faire tourner le cercle. L'illusion de rotation sera donnée par le point P fixe sur le cercle, mais mobile sur la cycloïde. À chaque rotation d'un angle β correspond une translation horizontale d'amplitude $a \cdot \beta$, ce qui nous ramène aux équations paramétriques

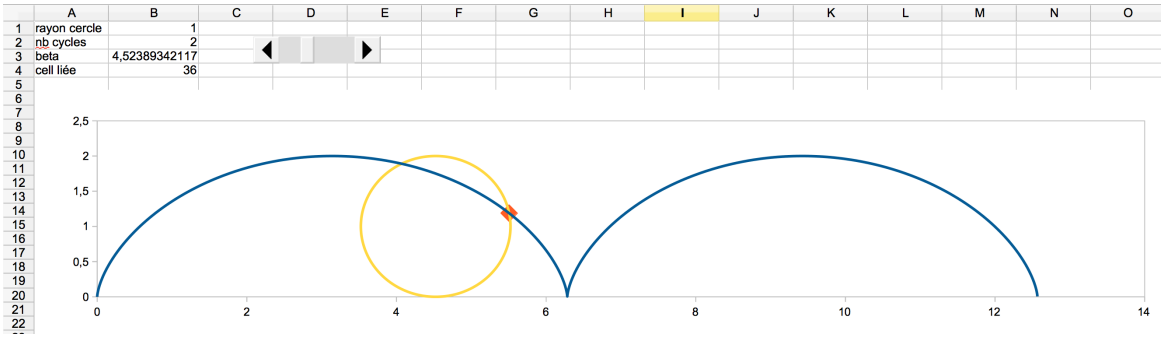
$$\begin{cases} x = a\cos\beta + a\beta \\ y = a + a\sin\beta \end{cases} \tag{13.3}$$

Les valeurs de β déjà utilisées pour la cycloïde (plage `A21:A121`) peuvent encore servir pour le cercle ; il faut simplement utiliser la cellule `B3` (nommée `beta`) pour le mouvement de translation horizontale $a \cdot \beta$.

	E	F
19	cercle mobile	
20	X	Y
21	=a*COS(A21)+a*beta	=a+a*SIN(A21)
...	...	...
121	=a*COS(A121)+a*beta	=a+a*SIN(A121)

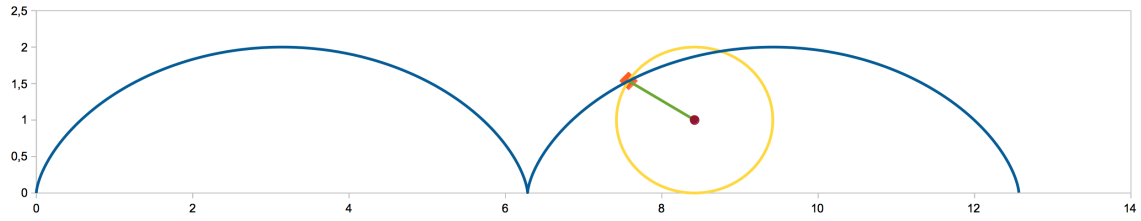
Ajouter au diagramme une série ayant comme caractéristiques :
Plage pour nom : `E19` ; Valeurs X : `E21:E121` ; Valeurs Y : `F21:F121`.

Rappelons que le tableur ne travaille pas dans un repère orthonormé, et qu'il faudra peut-être modifier les dimensions du diagramme pour donner l'illusion d'un cercle.



Exercice

Si vous avez bien suivi les explications, vous devriez être capable de dessiner le rayon OP du cercle mobile, où O est son centre et P le point fixe. Allez, au travail, et en silence !



On peut encore améliorer...

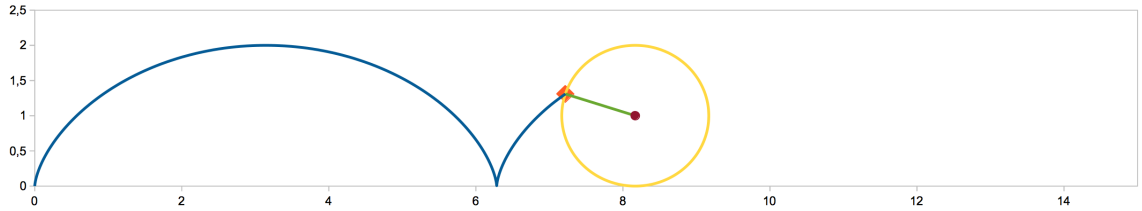
« En théorie », la cycloïde étant la trajectoire du point P, n'est « connue » qu'au fur et à mesure de l'avancement du cercle. Il faut donc apporter une petite modification aux formules de la plage `B21:C121` pour tenir compte de cette situation.

Pour l'explication, appelons *beta-fixe* une cellule de la plage `A21:A121` (puisqu'elle contient toutes les valeurs angulaires fixes — et donc prévues — de la trajectoire) et rappelons que la cellule `B3`, nommée *beta*, est la valeur angulaire variable liée à la barre de défilement.

Le principe est de ne dessiner un point de la cycloïde que si la valeur de *beta-fixe* est inférieure à la valeur de *beta* (mobile).

La formule en B21 devient donc $=SI(A21>beta;"";a*A21-a*sin(A21))$ et celle en C21 devient $=SI(A21>beta;"";a-a*cos(A21))$ à copier évidemment jusque'à la ligne 121.

Il n'y a rien à changer au graphique, puisqu'il est lié à la plage B21:C121, par contre il faudra fixer la graduation de l'axe des abscisses.



4 L'épicycloïde

L'épicycloïde est la courbe engendrée par un point fixe sur un cercle de rayon b qui roule sans glisser extérieurement sur un cercle de rayon a ($a \geqslant b$). C'est en quelque sorte une cycloïde enroulée autour d'un cercle.

4.1 La théorie mathématique, un peu

Dans le même bon livre d'analyse, on trouve les équations paramétriques de l'épicycloïde :

$$\begin{cases} x = (a+b)\cos\beta - b\cos\left(\frac{a+b}{b}\beta\right) \\ y = (a+b)\sin\beta - b\sin\left(\frac{a+b}{b}\beta\right) \end{cases} \tag{13.4}$$

où a est le rayon du cercle fixe, b le rayon du cercle mobile, et $\beta \in [0, 2\pi]$ (on ne fera qu'un seul tour, mais si on souhaite en faire plusieurs, faire comme pour la cycloïde, à savoir une cellule qui contiendra le nombre de tours souhaité).

Techniquement, il y a beaucoup de similitudes avec la cycloïde.

4.2 Les données pour le tableur

Une cellule (nommée a) contiendra le rayon du cercle fixe ; une cellule (nommée b) contiendra le rayon du cercle mobile et une cellule (nommée beta) contiendra le paramètre angulaire qui variera avec la barre de défilement dans l'intervalle $[0, 2\pi]$.

Le graphique sera composé de quatre séries :

- l'épicycloïde (courbe fixe) ;
- le cercle de rayon a (courbe fixe) ;
- le cercle de rayon b (courbe mobile) ;
- le point P (fixe sur le cercle de rayon b et mobile sur l'épicycloïde).

On représentera 101 points de l'épicycloïde, 101 points de chacun des cercles, et 101 valeurs du paramètre `beta`, afin de pouvoir récupérer les calculs de la cycloïde...

	A	B
1	rayon cercle fixe	1,5
2	rayon cercle mobile	0,5
3	beta	=2*PI()*$B$4/100
4	cell. liée	

La barre de défilement

La même que pour la cycloïde.

L'épicycloïde

	A	B	C
19	epicycloïde		
20	beta	X	Y
21	0	=(a+b)*COS(A21)- b*COS((a+b)/b*A21)	=(a+b)*SIN(A21)- b*SIN((a+b)/b*A21)
22	=A21+2*PI()/100	=(a+b)*COS(A22)- b*COS((a+b)/b*A22)	=(a+b)*SIN(A22)- b*SIN((a+b)/b*A22)
...	...	...	...
121	=A120+2*PI()/100	=(a+b)*COS(A121)- b*COS((a+b)/b*A121)	=(a+b)*SIN(A121)- b*SIN((a+b)/b*A121)

puis créer un diagramme avec une série ayant comme caractéristiques :
Plage pour nom : `A19` ; Valeurs X : `B21:B121` ; Valeurs Y : `C21:C121`.

Modifier éventuellement les propriétés de la série en ligne continue et couleur au choix.

Attention : il faut fixer l'échelle des axes X et Y avec une valeur supérieure à $a+2b$.

Le point P

	A	B	C
9	P	X	Y
10		=(a+b)*COS(beta)- b*COS((a+b)/b*beta)	=(a+b)*SIN(beta)- b*SIN((a+b)/b*beta)

Ajouter au diagramme une série ayant comme caractéristiques :
Plage pour nom : `A9` ; Valeurs X : `B10` ; Valeurs Y : `C10`.

Le cercle fixe, de rayon a

Le plus simple est d'utiliser un cercle de centre $(0,0)$, en utilisant 101 points de coordonnées $(a\cos\beta, a\sin\beta)$.

	E	F
19	cercle fixe	
20	X	Y
21	=a*COS(A21)	=a*SIN(A21)
...	...	...
121	=a*COS(A121)	=a*SIN(A121)

Ajouter au diagramme une série ayant comme caractéristiques :
Plage pour nom : `E19` ; Valeurs X : `E21:E121` ; Valeurs Y : `F21:F121`.

Le cercle mobile, de rayon b

Un cercle de centre $(0,0)$ et de rayon b, a pour équations paramétriques

$$\begin{cases} x = b\cos\beta \\ y = b\sin\beta \end{cases} \tag{13.5}$$

Il doit être tangent (à l'extérieur) au cercle fixe de rayon a ; il faut donc le translater d'une amplitude $(a+b)$ et l'effet de rotation sera donné par le paramètre de la cellule liée `beta`. Le cercle « mobile » a donc pour équations paramétriques

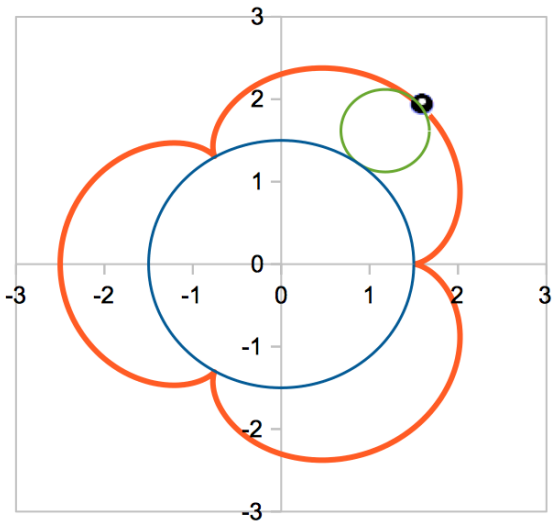
$$\begin{cases} x = b\cos\beta + (a+b)\cos beta \\ y = b\sin\beta + (a+b)\sin beta \end{cases} \tag{13.6}$$

Les valeurs de β déjà utilisées pour l'épicycloïde et le cercle fixe (plage `A21:A121`) peuvent encore servir pour le cercle mobile.

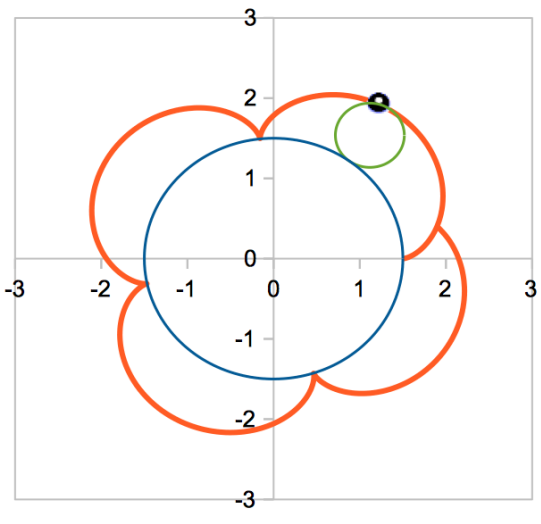
	H	I
19	cercle mobile	
20	X	Y
21	=b*COS(A21)+(a+b)*COS(beta)	=b*SIN(A21)+(a+b)*SIN(beta)
...	...	...
121	=b*COS(A121)+(a+b)*COS(beta)	=b*SIN(A121)+(a+b)*SIN(beta)

Ajouter au diagramme une série ayant comme caractéristiques :
Plage pour nom : `H19` ; Valeurs X : `H21:H121` ; Valeurs Y : `I21:I121`.

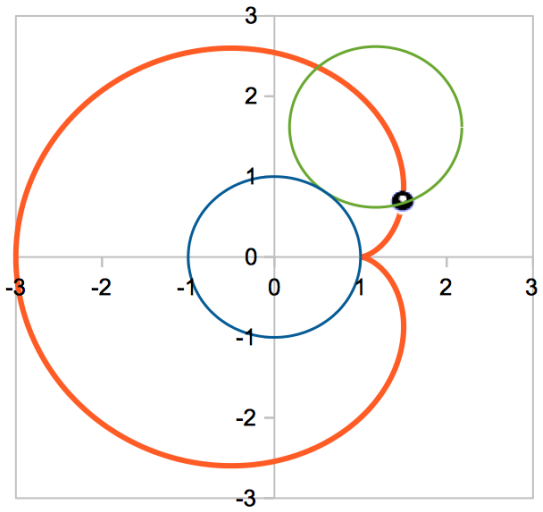
On peut maintenant profiter de notre travail en faisant varier les valeurs de a et de b ; voici un petit aperçu :



$a = 1,5$ et $b = 0,5$



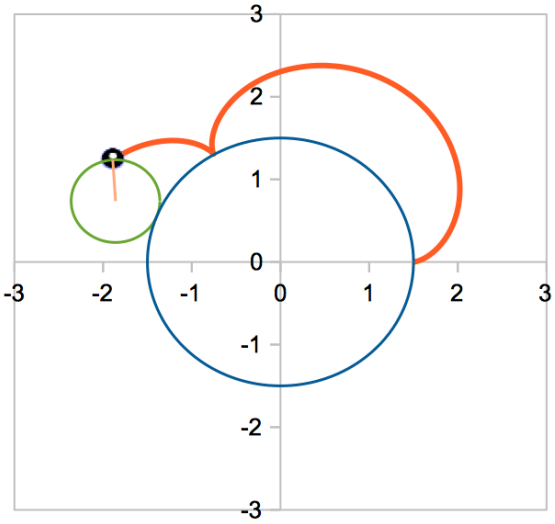
$a = 1,5$ et $b = 0,4$



$a = 1$ et $b = 1$

Exercice

Comme il n'est pas bon de laisser le lecteur sombrer dans l'ennui, on lui propose ici aussi de représenter le rayon du cercle mobile, de même que le tracé de l'épicycloïde en fonction de la variation du paramètre `beta`.



$a = 1,5$ et $b = 0,5$

5 L'hypocycloïde

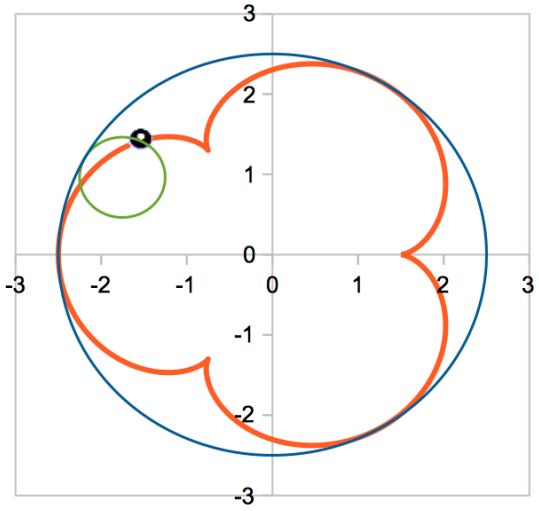
C'est la courbe engendrée par un point fixe sur un cercle de rayon b qui roule sans glisser intérieurement à un cercle de rayon a $(a > b)$.

On peut reprendre quasiment le classeur dans lequel on a traité l'épicycloïde, il suffit de changer quelques signes dans les équations, et le tour est joué. On se contentera donc de donner les équations de l'hypocycloïde, ainsi que celles du cercle mobile, et, évidemment, le résultat final.

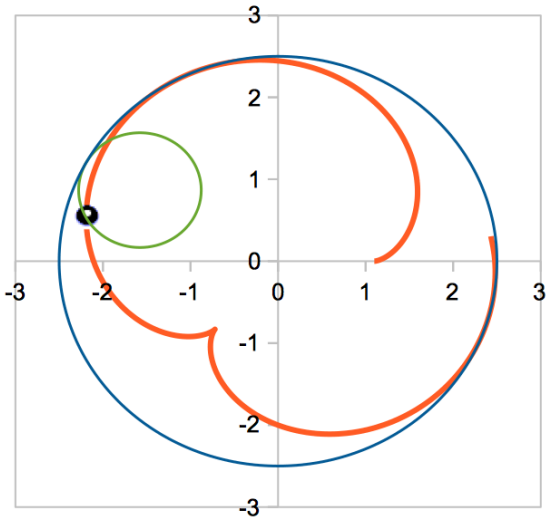
Equations de l'hypocycloïde et du cercle mobile

$$\begin{cases} x = (a-b)\cos\beta - b\cos\left(\frac{a-b}{b}\beta\right) \\ y = (a-b)\sin\beta - b\sin\left(\frac{a-b}{b}\beta\right) \end{cases} \tag{13.7}$$

$$\begin{cases} x = b\cos\beta + (a-b)\cos beta \\ y = b\sin\beta + (a-b)\sin beta \end{cases} \tag{13.8}$$

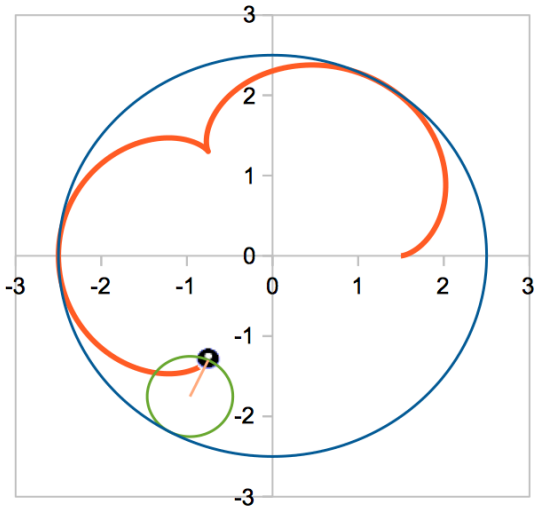


$a = 2,5$ et $b = 0,5$

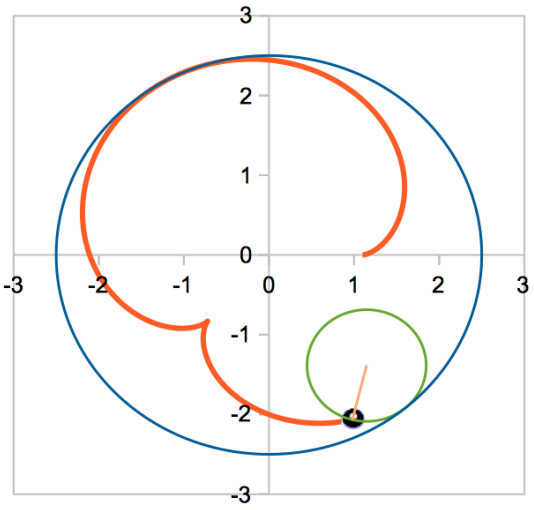


$a = 2,5$ et $b = 0,7$

L'exercice devenu traditionnel...



$a = 2,5$ et $b = 0,5$



$a = 2,5$ et $b = 0,7$

Mathématiques pour rire et pour pleurer : le smiley

Résumé. *Qui ne connaît pas ces petites figurines, les smileys, évoquant la joie, l'indifférence ou la tristesse, que l'on voit souvent accompagner nos messages électroniques, voire nos commentaires de résultats d'interrogations. Avec un peu d'imagination, leurs composants ne sont que points, cercles, segments et parabole, à la portée donc d'un élève du secondaire, dès la quatrième. Voici un exercice qui sort des sentiers battus, avec le logiciel* Géogébra.

Le smiley basique qui vous est proposé est composé de trois cercles (la tête et les deux yeux), quatre points (les deux pupilles et les deux narines), deux segments (le nez) et un arc de parabole (la bouche), des objets mathématiques relativement simples.

Comme la bouche sera animée, on utilisera un objet de type « curseur » (même principe qu'une barre de défilement) pour un coefficient variable.

Il a bien fallu choisir des dimensions pour développer l'exemple (rayons des cercles, positions des yeux et des pupilles, de la bouche, ...) ; le lecteur adaptera en fonction de sa conception artistique.

On supposera le lecteur familier avec les commandes de base du logiciel : encoder une équation dans la zone de saisie, modifier les propriétés d'un objet, etc.

1 La tête et les deux yeux

Il suffit de connaître l'équation cartésienne d'un cercle centré en $C(x_C, y_C)$ et de rayon R :

$$(x - x_C)^2 + (y - y_C)^2 = R^2$$

Nous avons choisi

- pour la tête : $x^2 + y^2 = 4$ centrée en $(0,0)$ et de rayon 2 ;
- pour l'œil gauche : $(x+1)^2 + (y-1)^2 = 0.2$ centré en $(-1,1)$ et de rayon $\sqrt{0.2}$;
- pour l'œil droit : $(x-1)^2 + (y-1)^2 = 0.2$ centré en $(1,1)$ et de rayon $\sqrt{0.2}$.

2 Les deux pupilles (fixes dans un premier temps)

- pupille gauche : un point de coordonnée $(-1,1)$;

- pupille droite : un point de coordonnée $(1,1)$.

Les pupilles sont au centre des yeux, le lecteur choisira leur couleur, leur forme et leur taille...

3 Le nez et les narines

Pas de consignes particulières ni de formules mathématiques ; on s'est contenté de dessiner deux segments passant par trois points, dont un est invisible, et les deux autres servant de narines (là aussi, choix de la taille...).

4 La bouche animée

La bouche est un arc de parabole dont on limitera l'équation à

$$y = kx^2 + b$$

où k est responsable de la concavité (le sens déterminera le sentient de joie ou de tristesse), et le coefficient b est responsable d'une translation verticale.

On fera varier le coefficient k à l'aide d'un curseur (que l'on nommera... k). Le choix est un peu aléatoire pour les premiers essais. Seules contraintes logiques à priori : l'intervalle de variation de k doit être centré en 0 (et 0 doit pouvoir être atteint), et l'arc de parabole doit être inclus au cercle représentant la tête.

Nous avons choisi k dans l'intervalle $[-0.5, 0.5]$ avec un incrément de 0.1.

Quant au coefficient b, il doit être négatif, pour que la bouche soit sous le centre de la tête ; nous avons choisi $b = -0.8$. La bouche aura donc pour équation

$$y = kx^2 - 0.8$$

Lorsque k prendra la valeur 0, la bouche sera une droite horizontale d'équation $y = -0.8$, ce qui reflètera le sentiment d'indifférence.

Limitation du domaine

Le graphique de $y = kx^2 - 0.8$ déborde du cercle d'équation $x^2 + y^2 = 4$ (un bel exercice serait de calculer les coordonnées des points d'intersection en fonction du paramètre k...) ; on peut en limiter le domaine en utilisant la fonction `Si[condition, instruction]`. On limitera le domaine à l'intervalle $[-1.2, 1.2]$.

Les exemples fournis dans l'aide du logiciel permettent rapidement de trouver la syntaxe d'appartenance à un intervalle ; on prendra donc comme définition de l'arc de parabole

$$\text{Si}[-1.2 < x < 1.2, kx^2 - 0.8]$$

pour que celui-ci reste confiné dans le cercle d'équation $x^2 + y^2 = 4$.

5 Mouvement des pupilles

Profitons des variations du paramètre k pour animer les pupilles. La pupille gauche a pour abscisse -1 et on souhaite la faire varier horizontalement sur toute la largeur de l'œil, soit dans l'intervalle $[-1-\sqrt{0.2}, -1+\sqrt{0.2}]$ que l'on va arrondir, pour simplifier, à $[-1.44, -0.56]$.

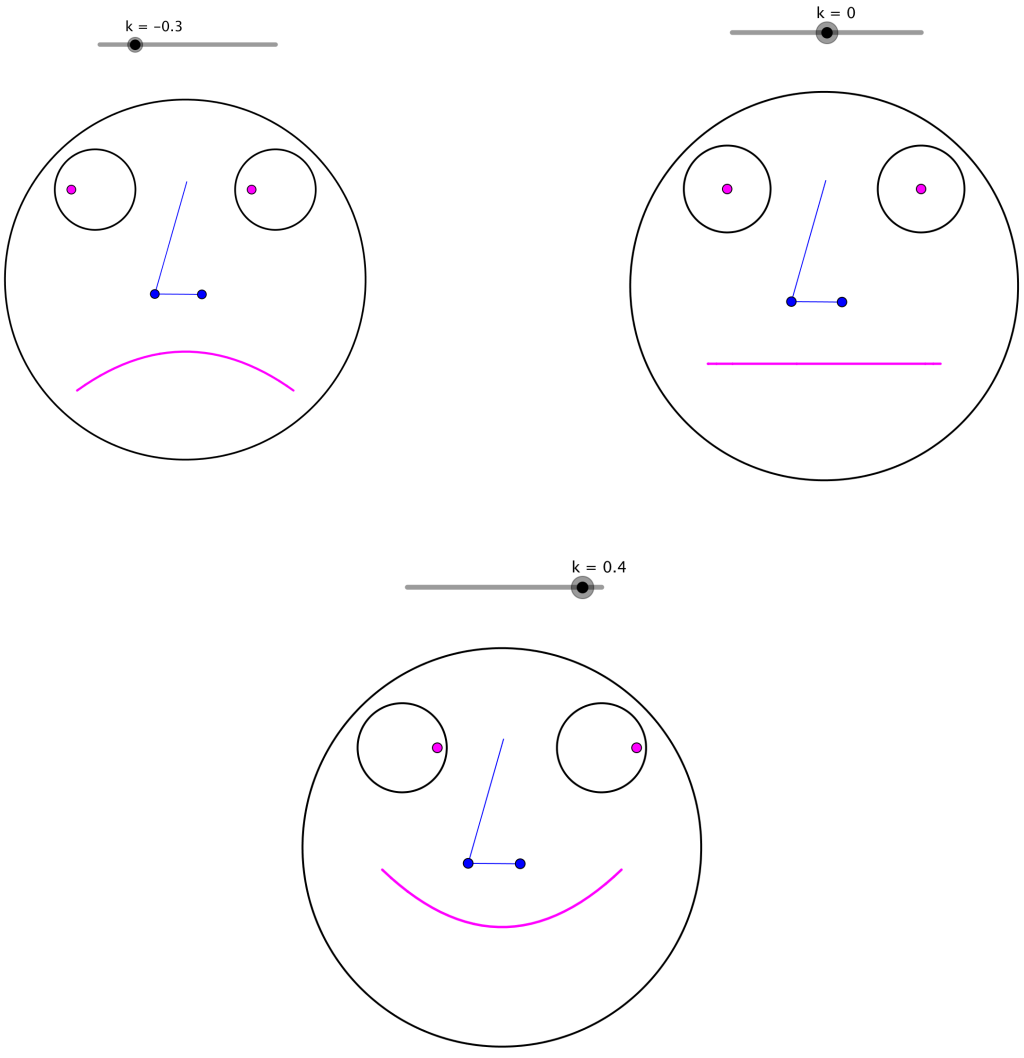
Il nous faut donc trouver une formule qui va transformer une valeur de k de l'intervalle $[-0.5, 0.5]$ en une valeur dans l'intervalle $[-1.44, -0.56]$. Pour faire simple, restons dans le linéaire, et calculons l'application $f(x) = Mx + P$ telle que $f([-0.5, 0.5]) = [-1.44, -0.56]$.

Il suffit de résoudre le système $\begin{cases} -0.5M + P = -1.44 \\ 0.5M + P = -0.56 \end{cases}$ qui a pour solutions $M = 0.88$ et $P = -1$.

On prendra donc comme nouvelle coordonnée de la pupille gauche le couple $(0.88k - 1, 1)$.

De même pour la pupille droite, on doit faire varier son abscisse dans l'intervalle $[1-\sqrt{0.2}, 1+\sqrt{0.2}]$, que l'on va arrondir à $[0.56, 1.44]$.

En résolvant le système $\begin{cases} -0.5M + P = 0.56 \\ 0.5M + P = 1.44 \end{cases}$ on obtient les solutions $M = 0.88$ et $P = 1$, ce qui permet de définir la nouvelle coordonnée de la pupille droite, le couple $(0.88k + 1, 1)$.



Le tableur et les tables de simulation à une et deux entrées

Résumé. *Le principe d'une table de simulation (appelée aussi table de prévision) est de reproduire dans un tableau un modèle de calcul en faisant varier un ou deux paramètres de ce modèle, et cela sans devoir recopier des formules. Pour expliquer l'outil, nous utiliserons une formule simple de calcul financier, et nous terminerons par deux petites applications, l'une qui peut aider numériquement au calcul d'une limite dans des cas de fonctions simples du niveau du secondaire, et l'autre permettant de dessiner le graphique d'une fonction à deux variables.*

Nous utiliserons le tableur *Calc* de la suite gratuite *OpenOffice* (version 4.4.1, 2014) pour expliquer le principe d'une table de simulation (il n'y a aucune différence avec *LibreOffice*) ; l'outil existe aussi dans *Excel* avec quelques différences de vocabulaire et de mise en place des données. En fin d'article, on en donnera la version *Excel*, version 2016.

1 Le modèle de départ

J'emprunte un montant de 23 000 € remboursable en 36 mensualités et au taux d'intérêt annuel de 2,5% ; il faut calculer le montant d'une mensualité. C'est un beau problème d'algèbre financière.

Le tableur propose la fonction `VPM` qui permet de faire ce calcul ; elle possède trois arguments obligatoires (et deux arguments optionnels dont nous ne tenons pas compte ici) : le montant emprunté, la durée de l'emprunt et le taux de l'emprunt. La syntaxe générale est `VPM(taux;durée;montant)`.

	A	B
1	montant	23000
2	taux (annuel)	2,50%
3	durée (mois)	36
4	mensualité	=VPM(-1+(B2+1)^(1/12);B3;B1)

Dans une formule financière, il faut uniformiser les unités de temps. Le remboursement étant une mensualité, il faut utiliser un taux mensuel (on a utilisé le taux équivalent $i_m = \sqrt[12]{1+i_a} - 1$).

Le résultat dans la cellule `B4` est `-663,53` ; c'est un nombre négatif affiché en rouge, correspondant à la mise en forme par défaut d'un *débit*[1] dans une fonction financière.

1. Le montant de la mensualité sera débité de votre compte en banque (malheureusement…)

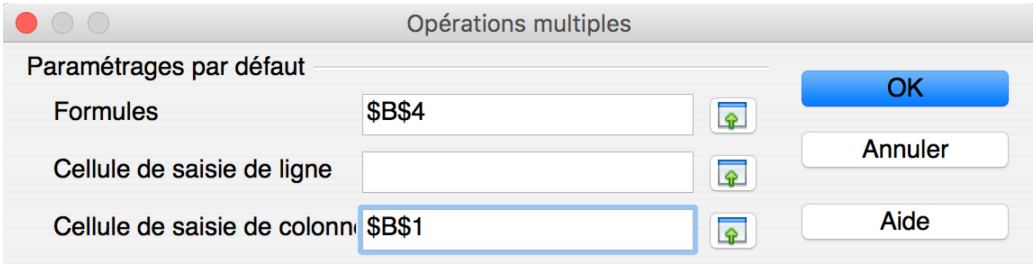
2 Table de simulation à une entrée

On souhaite simuler le calcul de la mensualité pour différents montants empruntés [2]. On va donc prévoir dans un tableau (vertical, c'est important de le faire remarquer) les différents montants que l'on souhaite tester, par exemple :

	A	B
6	montant	mensualité
7	20000	
8	21000	
9	22000	
10	23000	
11	24000	
12	25000	

Le principe de calcul de la table de simulation consiste à « dire au tableur » de prendre successivement toutes les valeurs de la plage `A7:A12`, de les placer dans la cellule `B1`, et de recopier le résultat de la cellule `B4` dans la plage `B7:B12`. On y va :

- sélectionner la plage `A7:B12` (attention, pas les titres de colonnes) ;
- `Données > Opérations multiples...` et compléter la boîte de dialogue comme ci-dessous.



Le **référencement absolu ($B$1, $B$4)** est important ; on peut soit taper les adresses dans les zones correspondantes, soit utiliser l'icône à l'extrémité de ces zones.

Le tableau de données étant « vertical », on utilise la zone « Cellule de saisie de colonne ». On aurait pu créer le tableau de manière horizontale (sur deux lignes, mais c'est moins pratique), et alors utiliser la zone « Cellule de saisie de ligne ». Voici les résultats après arrondi à l'unité :

	A	B
6	montant	mensualité
7	20000	−577
8	21000	−606
9	22000	−635
10	23000	−664
11	24000	−692
12	25000	−721

On peut modifier à volonté les données de la plage `A7:A12`.
Il n'a pas été nécessaire de recopier de formule. Si on regarde le contenu de la cellule `B7`, on trouve la formule `=OPERATIONS.MULTIPLES(B4;B1;A7)` que l'on peut alors copier si on ajoute des données à la plage `A7:A12`.
On peut créer deux autres tableaux semblables où l'on fait varier successivement le taux et la durée.

Dans la petite application (dans un autre registre) qui suivra, on va créer quatre tables de type $(x, f(x))$ où l'on fera varier des valeurs de x successivement de plus en plus petites, de plus en plus grandes, de plus en plus proches d'un réel a tout en lui étant inférieures, et de plus en plus proches d'un réel a tout en lui étant supérieures. On devine facilement pourquoi.

2. On hésite encore sur le modèle de voiture…

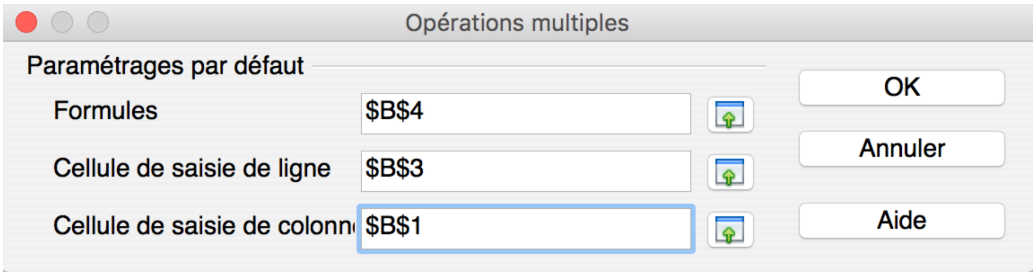
3 Table de simulation à deux entrées

Toujours avec le même modèle de départ, on souhaite faire varier simultanément le montant emprunté et la durée de remboursement. On construit donc un tableau semblable à celui-ci :

Les montants empruntés forment une colonne et les durées de remboursement forment une ligne.

	D	E	F	G	H
6		12	18	24	30
7	20000				
8	21000				
9	22000				
10	23000				
11	24000				
12	25000				

- sélectionner la plage `D6:H12` (la plage doit inclure toutes les données) ;
- `Données > Opérations multiples...` et compléter la boîte de dialogue comme ci-dessous : (en ligne, `$B$3`, la durée et en colonne, `$B$1`, les montants).



	D	E	F	G	H
6		12	18	24	30
7	20000	−1689	−1133	−855	−688
8	21000	−1774	−1190	−898	−723
9	22000	−1858	−1246	−940	−757
10	23000	−1942	−1303	−983	−791
11	24000	−2027	−1360	−1026	−828
12	25000	−2111	−1416	−1069	−860

On peut modifier à volonté les données des plages `D7:D12` et `E6:H6`. Si on ajoute des données à ces plages, il faut prolonger les formules.
Le contenu de la cellule `E7` est
`=OPERATIONS.MULTIPLES(B4;B1;D7;B3;E6)`

Une application immédiate de cet outil est de construire un tableau de valeurs d'une fonction à deux variables, tableau qui peut déboucher sur le graphique de cette fonction.

4 Une aide à l'évaluation numérique d'un calcul de limite

Le modèle de départ est très simple, puisqu'on ne construira que des tableaux de type $(x, f(x))$. Pour exprimer une fonction en termes de x (en `B1`), il est nécessaire qu'une cellule (`B2`) soit nommée `x`. On a besoin en plus d'une cellule (`B3`), nommée `a`, pour le calcul de $\lim_{x\to a} f(x)$. Rappelons que le nom d'une cellule permet d'utiliser celui-ci en lieu et place de son adresse ; un nom a le statut d'une adresse absolue ; pour le créer, `Insertion > Nom > Définir...`

	A	B
1	f(x)	$=(x^2-4)/(x-2)$
2	x	6,4
3	a	2

On a choisi une fonction simpliste $f(x) = \frac{x^2-4}{x-2}$ pour mettre en place notre exemple. La seule contrainte est d'écrire l'expression analytique de $f(x)$ en respectant la syntaxe du tableur.

La cellule `B2` peut contenir un réel quelconque, on peut même masquer les cellules `A2` et `B2`.

Le contenu de la cellule B3 peut en théorie être un réel quelconque, mais les cas intéressants sont évidemment les points critiques pour $f(x)$, à savoir 2 en ce qui nous concerne.

L'intérêt de ce que l'on va mettre en place est qu'il ne faudra modifier que les cellules B1 et B3 pour changer d'exercice. De plus, on a une vue d'ensemble sur les quatre tableaux de valeurs qui permettront d'avoir une idée de solution pour les calculs de $\lim_{x\to a} f(x)$ et $\lim_{x\to\pm\infty} f(x)$.

Mise en place des données (les valeurs de x)

Pour les plages D5:D10 et G5:G10, on a choisi des multiples de 10, mais ce n'est évidemment pas une obligation. On peut aussi choisir des nombres bien plus grands (petits), selon la fonction.

Pour la plage D14:D19, des formules de type =a-0,1, =a-0,01, ... (en D14, on peut aussi écrire la formule =a-1/D5 et recopier vers le bas).

Pour la plage G14:G19, des formules de type =a+0,1, =a+0,01, ... (en G14, on peut aussi écrire la formule =a+1/D5 et recopier vers le bas).

Les plages peuvent être prolongées vers le bas, mais tôt au tard il y aura des arrondis de calculs, qu'il faudra bien évidemment expliquer aux élèves. Dans certaines situations, il faudra aussi afficher plus de décimales dans les colonnes $f(x)$.

Les titres et les couleurs peuvent agrémenter la présentation, mais ce n'est pas ici notre préoccupation principale.

Calcul des tables

À faire quatre fois, mais c'est pour la bonne cause. La formule est en $B$1, et la cellule de saisie de colonne est $B$2. Voici le résultat :

	A	B	C	D	E	F	G	H
1	f(x)	11						
2	x	9						
3	a	2		en moins l'infini			en plus l'infini	
4				x	f(x)		x	f(x)
5				100	102		-100	-98
6				1000	1002		-1000	-998
7				10000	10002		-10000	-9998
8				100000	100002		-100000	-99998
9				1000000	1000002		-1000000	-999998
10				10000000	10000002		-10000000	-9999998
11								
12				à gauche de a			à droite de a	
13				x	f(x)		x	f(x)
14				1,9	3,9		2,1	4,1
15				1,99	3,99		2,01	4,01
16				1,999	3,999		2,001	4,001
17				1,9999	3,9999		2,0001	4,0001
18				1,99999	3,99999		2,00001	4,00001
19				1,999999	3,999999		2,000001	4,000001
20								

Les problèmes de domaine

Le tableur ne calcule (malheureusement) pas le domaine d'une fonction, mais s'il rencontre un calcul impossible, il affiche un message d'erreur qui dépend du calcul (et du tableur) : « Err :502 » pour la racine paire d'un nombre négatif (#NOMBRE ! chez *Excel*), « #DIV/0 ! » pour une division par zéro.

Si on n'y prend pas garde, on risque d'avoir l'un de ces messages dans une ou plusieurs colonnes $f(x)$ des tableaux de valeurs. Les élèves sont évidemment interpellés (et c'est bien), car souvent (en tout cas les miens, en cinquième math 4h/sem [3]) ils ne pensent pas à calculer le domaine avant de calculer les limites.

On peut remplacer le message d'erreur « brutal » par un message plus doux du genre « hors domaine », voire pas de message du tout. C'est la fonction ESTERREUR qui permet de le faire, à taper dans la cellule B1 : `=SI(ESTERREUR(fx);"hors domaine";fx)` où fx représente l'expression analytique de la fonction.

Ci-dessous le résultat pour la fonction $f(x) = \frac{\sqrt{x}}{x+1}$. La cellule B1 contient la formule `=SI(ESTERREUR(RACINE(x)/(x+1));"hors domaine";RACINE(x)/(x+1))` et B3 contient -1.

	A	B	C	D	E	F	G	H
1	f(x)	0,3						
2	x	9						
3	a	-1		en moins l'infini			en plus l'infini	
4				x	f(x)		x	f(x)
5				100	0,099009901		-100	hors domaine
6				1000	0,031591185		-1000	hors domaine
7				10000	0,009999		-10000	hors domaine
8				100000	0,003162246		-100000	hors domaine
9				1000000	0,000999999		-1000000	hors domaine
10				10000000	0,000316228		-10000000	hors domaine
11								
12				à gauche de a			à droite de a	
13				x	f(x)		x	f(x)
14				-1,1	hors domaine		-0,9	hors domaine
15				-1,01	hors domaine		-0,99	hors domaine
16				-1,001	hors domaine		-0,999	hors domaine
17				-1,0001	hors domaine		-0,9999	hors domaine
18				-1,00001	hors domaine		-0,99999	hors domaine
19				-1,000001	hors domaine		-0,999999	hors domaine
20								

Graphique d'une fonction à deux variables

Après vérification, le tableur *Calc* ne propose pas (dans sa version actuelle) de réaliser un graphique d'une fonction à deux variables. Nous allons donc utiliser le tableur *Excel* pour cette application.

En toute honnêteté, le logiciel *GeoGebra* permet de réaliser de tels graphiques de manière bien plus souple. Donc, *just for fun* !

3. Et ce n'est pas faute de le leur dire…

5 Avec Microsoft Excel

Table à une entrée

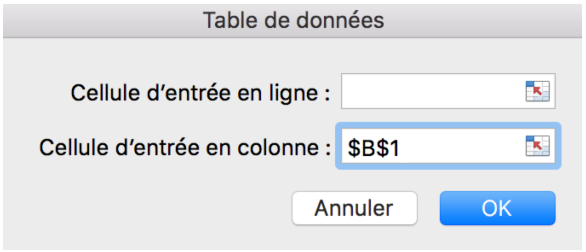
Comme pour *Calc*, le modèle de départ pour faire un calcul reste identique (plage `A1:B4`).

	A	B
6		=B4
7	20000	
8	21000	
9	22000	
10	23000	
11	24000	
12	25000	

La table de simulation doit avoir la structure comme ci-contre. La différence principale avec *Calc* est que la formule doit faire partie de la table. Si des titres sont nécessaires, il faut les placer en dehors de la table, à la ligne 5 par exemple (les titres ne font jamais partie de la table à calculer).

- sélectionner la plage `A6:B12` ;
- `Données > Analyse de scénarios > Table de données...`

La suite est identique à *Calc*, compléter la zone de la cellule d'entrée en colonne, à savoir `$B$1`. Dans les cellules calculées (plage `B7:B12`) on trouve par contre la formule `=TABLE(;B1)`. C'est une formule dite « matricielle » à cause des accolades qui l'encadrent.



Les cellules de la plage `B7:B12` sont protégées (on ne peut ni les modifier, ni les supprimer individuellement). Si on veut ajouter ou supprimer des lignes, il faut redéfinir une table avec les nouvelles données.

Par contre, contrairement à *Calc*, il est possible, pour les mêmes données, de calculer d'autres formules, par exemple en `C6`, `D6`, ...

Pour une raison esthétique, il est souhaitable de masquer la cellule `B6`, par exemple en choisissant une couleur de texte identique à la couleur de fond.

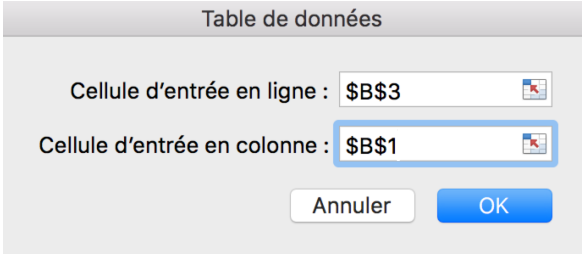
Table à deux entrées

	D	E	F	G	H
6	=B4	12	18	24	30
7	20000				
8	21000				
9	22000				
10	23000				
11	24000				
12	25000				

Les montants empruntés forment une colonne et les durées de remboursement forment une ligne.

- sélectionner la plage `D6:H12` (la plage doit inclure toutes les données et la formule du coin supérieur gauche) ;
- `Données > Table de données...` et compléter la boîte de dialogue comme ci-dessous : (en ligne, `$B$3`, la durée et en colonne, `$B$1`, les montants).

Dans les cellules calculées (plage B7:B12) on trouve la formule {=TABLE(B3;B1)}. Ces cellules sont aussi protégées, et en cas d'ajout ou de suppression de données, il faut redéfinir une nouvelle table.



Graphique d'une fonction à deux variables

Pour l'exemple, on a choisi un classique, à savoir $f(x,y) = x^2 + y^2$ dont le graphique est un paraboloïde. Pour d'autres fonctions, il suffit de modifier le contenu de la cellule B1, en respectant la syntaxe du tableur.

	A	B
1	f(x,y)	=x*x+y*y
2	x	
3	y	

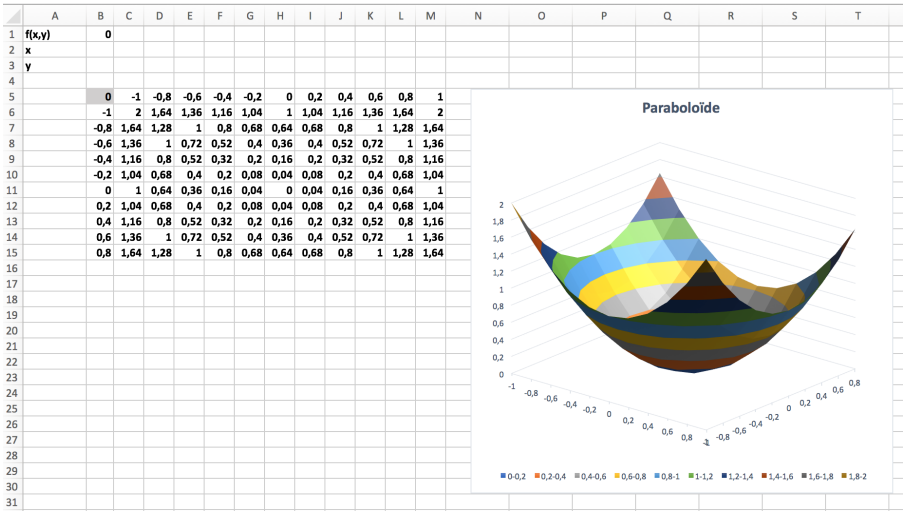
Le modèle de départ est très simple. Les cellules B2 et B3 doivent être nommées respectivement x et y pour que la formule en B1 puisse s'exprimer en x et en y. Il n'est même pas nécessaire de leur donner un contenu, et la plage A2:B3 peut être masquée.

	=B1	x_1	x_2	...	x_n
	y_1				
	y_2				
	...				
	y_n				

Calculer des valeurs d'abscisses et d'ordonnées en fonction des besoins, puis appliquer la procédure de calcul de table comme décrite précédemment.

Pour créer le graphique :

- sélectionner le tableau ;
- Insérer > Graphique > Surface et ... amusez-vous !



Le tableur OpenOffice Calc et le générateur aléatoire

Résumé. *Qui, comme moi, a connu un prof de math faire lancer une pièce de monnaie par un élève, pendant au moins deux heures (de cinquante minutes) de cours, pour introduire le chapitre consacré aux probabilités, sera certainement intéressé par ce qui suit. Il y a moyen de faire mieux, plus vite, et d'agrémenter par un graphique.*

C'est encore le tableur *Calc* de la suite bureautique *OpenOffice*, issue du monde des logiciels libres (et donc gratuits) qui s'y colle. Pour les amateurs d'*Excel*, tout est transposable...

Une petite mise au point avant de débuter : on devrait plutôt qualifier de *pseudo aléatoires* les nombres qui sont engendrés par les logiciels ; ils sont en fait *calculés* par l'ordinateur (ou nos calculettes) avec des moyens sophistiqués. Par la suite, le préfixe *pseudo* sera sous-entendu sur le papier, mais doit bien rester présent dans notre esprit.

1 Préliminaire : mode de calcul

Par défaut, le tableur recalcule automatiquement toutes les cellules du classeur (et il y en a beaucoup !) dès que le contenu de l'une d'entre elles est modifié. On trouve cette option dans le menu
Outils > Contenu des cellules

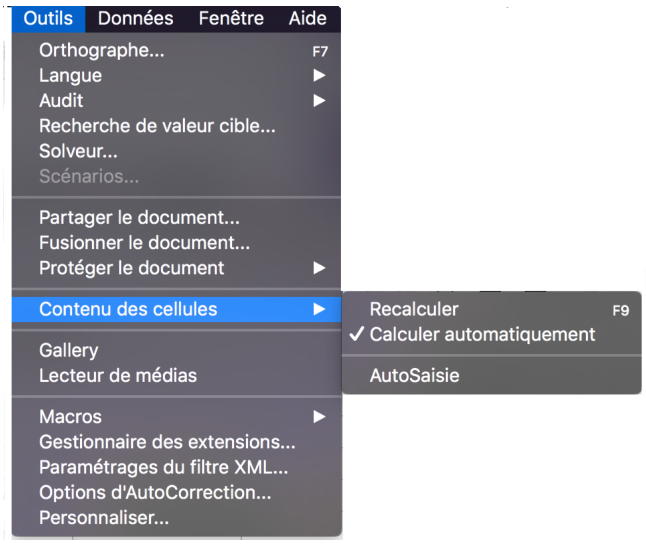


Fig. 9 *Calcul automatique du contenu des cellules*

Évidemment, les calculs sont rapides, et pour beaucoup d'applications « simples », l'utilisateur

n'y voit que du feu. Il en va tout autrement lorsqu'un classeur contient beaucoup de formules, dans beaucoup de feuilles, et qu'il y a beaucoup de données à encoder (je pense par exemple à un cas vécu où il fallait encoder les résultats de l'année pour tous les élèves de l'école dans toutes les branches, pour les trois trimestres, ...) : après chaque nouveau résultat encodé il fallait attendre que tout le classeur soit recalculé...Abominable !

Dans ce genre de situation, il convient de décocher cette option, et on voit sur l'image qui précède qu'il faut alors presser la touche F9 pour calculer le classeur (calculer toutes les cellules qui contiennent une formule).

C'est ce que l'on fera pour simuler un *pile ou face* avec une pièce de monnaie à chaque pression sur cette touche F9.

2 La fonction `alea()`

La fonction `alea()` engendre un nombre réel aléatoire dans l'intervalle $[0,1[$. Si une cellule contient cette fonction dans une formule, son contenu est recalculé dès qu'il y a une modification ailleurs dans le classeur, et par conséquent un autre réel est engendré (en tout cas tant que le calcul des cellules est automatique).

Si on souhaite engendrer un autre nombre qu'un réel dans l'intervalle $[0,1[$, il suffit de manipuler comme suit la fonction :

`k*alea()`	réel dans $[0,k[$
`ent(k*alea())`	entier dans $[0,k-1]$
`1+ent(k*alea())`	entier dans $[1,k]$
`k+ent((p-k+1)*alea())`	entier dans $[k,p]$ avec $p>k$
`1+k*alea()`	réel dans $[1,k+1]$
`k+(p-k)*alea()`	réel dans $[k,p]$

Noter qu'il y a aussi la fonction `alea.entre.bornes(min;max)` qui engendre un nombre entier dans l'intervalle $[min,max]$, mais le contenu de la cellule reste figé (n'est pas recalculé lorsque le contenu d'une autre cellule est modifié).

3 Pile ou Face, telle est la question

L'idée est de simuler plusieurs jets d'une pièce de monnaie (chaque lancer correspondra à un appui sur la touche F9) en comptant le nombre de fois que *pile* et *face* vont apparaître, et à partir de ces résultats, calculer les fréquences correspondantes ; et pourquoi pas, représenter la situation sur un graphique.

On supposera qu'un nombre aléatoire inférieur à 0.5 correspond à *pile* et, on l'aura compris, un nombre aléatoire supérieur à 0.5 correspond à *face*.

Présentation des calculs

	A	B	C
1	=ALEA()		fréquence
2	pile	=SI(A1<0,5;pile+1;pile)	=B2/(B2+B3)
3	face	=SI(A1>=0,5;face+1;face)	=B3/(B2+B3)

La cellule A1 contient la formule qui va engendrer un réel aléatoire dans l'intervalle [0,1[; les cellules B2 et B3 sont respectivement nommées `pile` et `face`, ce qui rend les formules plus limpides.

Référence circulaire

Les formules en B2 et B3 vont engendrer un message d'erreur ! En effet, par défaut, le tableur interdit, dans une cellule, une référence à elle-même, ce qui est appelé *référence circulaire*. Qu'à cela ne tienne, on va modifier une option de calcul qui autorisera les *références circulaires* : pour le tableur *Calc*,
OpenOffice > Préférences...> OpenOffice Calc > Calculer > Cocher l'option 'Itérations'

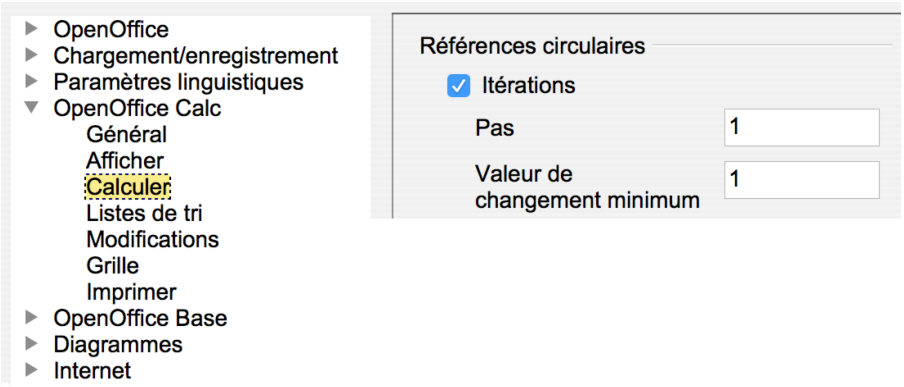


Fig. 10 *Références circulaires*

Un bug ?

En théorie, l'exercice est quasi terminé, mais à l'heure [1] où ces quelques lignes sont écrites, il semble qu'il y ait une anomalie dans les calculs (de OpenOffice Calc) [2], puisque l'on constate qu'à chaque pression de la touche F9 les compteurs sont incrémentés de deux unités ; je n'ai pas d'explication...mais il y a moyen de contourner le problème : plutôt que d'ajouter 1 à chaque compteur, il suffit d'ajouter 0,5...

	A	B	C
1	=ALEA()		fréquence
2	pile	=SI(A1<0,5;pile+0,5;pile)	=B2/(B2+B3)
3	face	=SI(A1>=0,5;face+0,5;face)	=B3/(B2+B3)

Après avoir désactivé le calcul automatique des cellules, et en maintenant enfoncée la touche F9 on simule de nombreux jets d'une pièce, et on voit les fréquences (futures probabilités) osciller autour de 0,5.

1. 11:31:34
2. Avec le tableur *Microsoft Excel*, pas de problème !

Mais comment remettre les compteurs à zéro ?

En modifiant la formule des cellules B2 et B3 on constate que les compteurs respectifs sont remis à zéro. Mais ce n'est qu'un « one shot »… Offrons-nous un interrupteur.

	A	B
5	interrupteur	0

On supposera que la cellule B5 contiendra 0 (off) ou 1 (on) ; on applique alors le raisonnement suivant pour les contenus des cellules B2 et B3

en B2 (pile) :

```
Si B5=1
    Alors Si A1<0,5
            Alors pile+0,5
            Sinon pile
    Sinon 0
```

en B3 (face) :

```
Si B5=1
    Alors Si A1>=0,5
            Alors face+0,5
            Sinon face
    Sinon 0
```

Et les formules respectives en B2 et B3 deviennent respectivement :

`=SI(B5=1;SI(A1<0,5;pile+0,5;pile+0);0)` et `=SI(B5=1;SI(A1>=0,5;face+0,5;face+0);0)`

Pour faire varier le contenu de la cellule B5 de 0 à 1, on peut aussi se payer le luxe d'une petite barre de défilement.

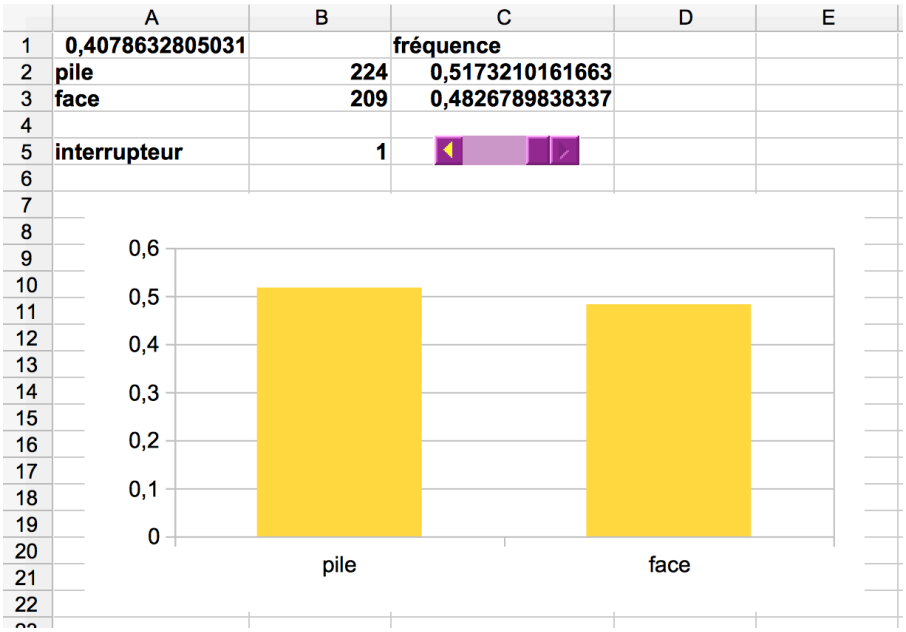


Fig. 11 *Le produit fini*

4 Les prolongations

Une fois acquis le principe du *pile ou face* d'une pièce de monnaie, il est aisé de simuler le jet d'un dé à n faces ; il suffit de découper l'intervalle $[0,1[$ en n sous-intervalles et de traiter les n tests. Les formules sont un peu plus longues.

5 Les statistiques de la fonction `alea()`

N'ayant pas d'informations sur le procédé de calcul des nombres pseudo aléatoires, nous ne pouvons que faire des tests sur la répartition de ces nombres sur l'intervalle [0,1[. De toute manière, nous n'avons pas le choix, il faut prendre la fonction `alea()` telle qu'elle est.

Nous [3] avons testé 10 000 nombres engendrés par la formule `=ALEA()` dans la plage `A1:E2000` et calculé le tableau ci-après, ainsi que les traditionnels moyenne, variance, écart-type et médiane.

	G	H	I
1	inférieur à	eff. cumulés	fréq. cumulées
2	0,1	`=NB.SI($A$1:$E$2000;"<" & G2)`	`=H2/10000`
3	0,2	`=NB.SI($A$1:$E$2000;"<" & G3)`	`=H3/10000`
...	...	...	...
11	1	`=NB.SI($A$1:$E$2000;"<" & G11)`	`=H11/10000`

Dans la cellule `H2`, la fonction `NB.SI(plage;condition)` compte le nombre de cellules vérifiant une condition donnée dans une plage donnée. Étrangement, la condition doit être précisée entre guillemets ; l'opérateur de concaténation (`&`) permet de construire les conditions `"<0,1"`, `"<0,2"`, ..., `"<1"`.

Dans la plage `L1:L4` on a utilisé les formules respectives `=MOYENNE(A1:E2000)`, `=VAR(A1:E2000)`, `=RACINE(L2)` et `=MEDIANE(A1:E2000)`.

G	H	I	J	K	L
inférieur à	eff. cumulés	fréq. cumulées		Moyenne	0,5021936214927
0,1	983	0,0983		Variance	0,0840280653534
0,2	2028	0,2028		Écart-type	0,2898759482148
0,3	3023	0,3023		Médiane	0,5052847354673
0,4	4004	0,4004			
0,5	4953	0,4953			
0,6	5942	0,5942			
0,7	6944	0,6944			
0,8	7938	0,7938			
0,9	8998	0,8998			
1	10000	1			

Fig. 12 *Statistiques de la fonction ALEA()*

Les résultats obtenus changent évidemment à chaque appui sur la touche F9. Ce tableau de résultats est donc une photographie à un instant t.

On peut aussi faire le graphique des fréquences cumulées sur lequel on a inséré la droite de régression dont l'équation $f(x)$ (au même instant t) est affichée.

3. Je.

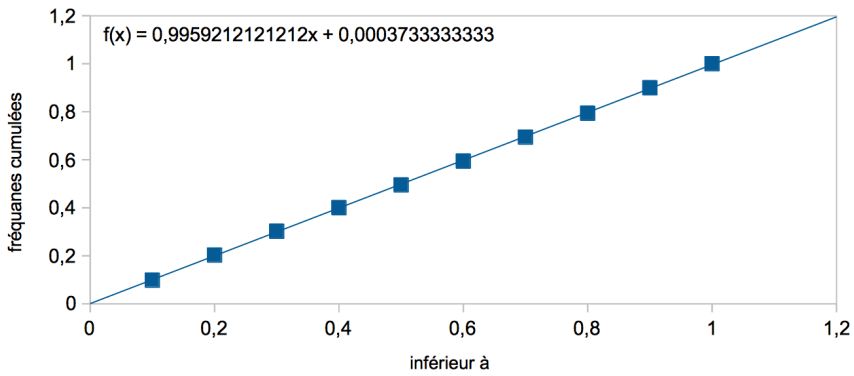


Fig. 13 *Graphique des fréquences cumulées*

Un calendrier perpétuel

Résumé. *À partir du mois de décembre nous sommes chaque année à l'affût du nouveau calendrier, et les occasions ne manquent pas de s'en procurer : les pompiers, le pharmacien, l'assureur, le politicien local, le restaurant chinois, etc. Voici comment, avec le tableur, créer un calendrier perpétuel, pour lequel il suffit de choisir l'année (entre 1900 et 9999). Dans un premier temps on va construire une vue globale de l'année, on peut utiliser les mêmes outils pour faire une feuille par mois, élargir les cellules, insérer des images, etc.*

Pour rentabiliser l'investissement chez Microsoft, j'utiliserai le tableur *Excel 2016* ; les fonctions utilisées sont absolument identiques pour *OpenOffice Calc*, excepté les commandes de mises en forme conditionnelles.

1 Date et mise en forme de date

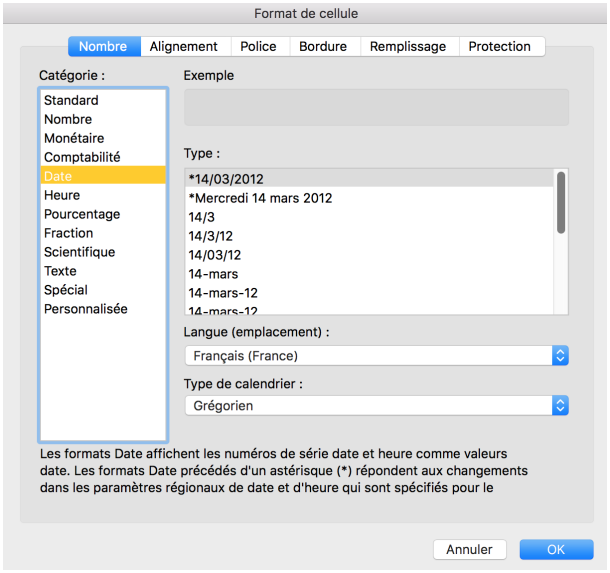
Pour le tableur, une date est un nombre ; ce nombre représente le nombre de jours écoulés depuis le 1er janvier 1900. Ce principe permet de comprendre que pour connaître le nombre de jours écoulés entre deux dates, il suffit de soustraire les deux cellules contenant ces dates. Il va de soi que le tableur tient compte des mois à 28, 29, 30 et 31 jours ainsi que des années bissextiles !

Le passage de date à nombre ou de nombre à date n'est qu'une question de mise en forme. Exemples :

- En tapant `25/9/57` dans une cellule, vous tapez un nombre avec une mise en forme de date ; pour connaître ce nombre, il suffit de mettre la cellule au format nombre standard (`Mise en forme > Cellule > Standard`), et vous obtiendrez `21088`. Il y a donc 21 088 jours écoulée entre le 01/01/1900 et le 25/09/1957. Remarquer qu'une date dans une cellule est justifiée à droite, comme les nombres.

- En tapant le nombre 25 678 dans une cellule, et en lui appliquant un format de date (`Mise en forme > Cellule > Date`), on accède à différents choix d'affichage ; si l'un d'eux convient, le choisir.

  Attention, ne pas se fier à la date proposée (14 mars 2012) ; la date correspondant à 25 678 est bien le 20 avril 1970.



- Il est aussi possible de choisir une mise en forme `Personnalisée`, où dans la zone `Type` il suffit de taper un code correspondant au format souhaité. Différents formats sont proposés pour différents contextes ; voici ceux que nous utiliserons pour notre calendrier :
  - `j` donne le numéro du jour en 1 caractère, par exemple `3`
  - `jj` donne le numéro du jour en 2 caractères, par exemple `03`
  - `jjj` donne les 3 premiers caractères du nom du jour, par exemple `Lun`
  - `jjjj` donne le nom complet du jour, par exemple `Lundi`
  - `m` donne le numéro du mois en 1 caractère, par exemple `3`
  - `mm` donne le numéro du mois en 2 caractères, par exemple `03`
  - `mmm` donne les 3 premiers caractères du nom du mois, par exemple `Jan`
  - `mmmm` donne le nom complet du mois, par exemple `Janvier`
  - `aa` donne l'année en 2 caractères, par exemple `18`
  - `aaaa` donne l'année complète en 4 caractères, par exemple `2018`

  Pour obtenir `Mer 25/09/1957`, il faut taper le code `jjj jj/mm/aaaa`

2 Les mois et les jours

Tout le calendrier doit être calculé uniquement à partir de l'année ; il peut être intéressant que la cellule `B1` soit nommé `an` par exemple. Voici donc la première chose à mettre en place :

	A	B
1	Année	2018

Pour automatiser le calcul des jours de chaque mois, on aura besoin de la série des 12 numéros de mois, ligne que l'on pourra masquer si on la trouve gênante. On aura aussi besoin des douze noms des mois, que l'on peut engendrer facilement en copiant vers la droite la cellule contenant le mot `Janvier` :

	A	B	…	L
2	1	2	…	12
3	Janvier	Février	…	Décembre

Dans la cellule A4, on calcule le 1er janvier à l'aide de la fonction DATE(année;mois;jour) : =DATE(an;A2;1). Cette formule doit être copiée jusqu'en L4 pour avoir ainsi les premiers de chaque mois.

Dans la cellule A5, le deuxième jour du mois de janvier ; sachant qu'une date est un nombre, il suffit d'ajouter 1 à la date précédente, soit =A4+1. Cette formule doit être copiée jusqu'en A31 puis jusqu'en L31, puisque tous les mois ont au moins 28 jours.

On pourrait traiter *manuellement* les derniers jours de chaque mois (à l'exception de février), à savoir continuer la copie précédente suffisamment bas pour couvrir les 29, 30 et 31 éventuels. Mais on peut faire mieux avec trois formules copiables pour chaque mois.

En A32, on fait le raisonnement suivant : si en ajoutant 1 à A31 on reste dans le même mois, alors ajouter 1 à A31, sinon rien. La fonction MOIS(date) renvoie le numéro du mois. En A32 on a donc
=SI(MOIS(A31+1)=A2;A31+1;"") ; cette formule est copiable jusqu'en L32. Les résultats sont des nombres qui ne sont pas au format date (pas grave, on s'en occupe après) et remarquer que la cellule B32 est vide, l'année 2018 n'étant pas bissextile, il n'y a pas de 29 février.

En A33, on fait un raisonnement semblable : si en ajoutant 2 à A31…, ce qui donne
=SI(MOIS(A31+2)=A2;A31+2;"") formule copiable jusqu'en L33.

Et en A34, toujours le même raisonnement, mais en ajoutant 3…
=SI(MOIS(A31+3)=A2;A31+3;"") formule copiable jusqu'en L34.

3 Les mises en forme des jours spéciaux

À partir d'ici, les goûts et les couleurs…

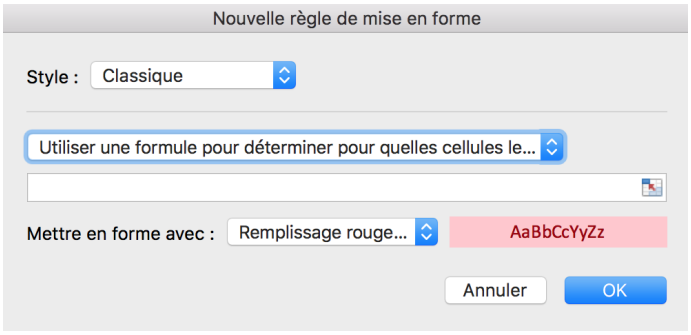
- Le format des jours : dans chaque colonne on connait le mois et l'année, on se contentera donc du nom et du numéro du jour. On donnera par exemple à toute la plage A4:L34 la mise en forme personnalisée jjj jj

- Les week-end : ce sont des jours importants pour les enseignants (pas seulement d'ailleurs) et on se propose de choisir un fond de couleur rouge pour les faire ressortir.

  La fonction JOURSEM(date) renvoie par défaut 7 pour un samedi et 1 pour un dimanche (on peut modifier cela mais c'est sans importance ici). Pour tester si une date est un jour de week-end, il suffit de tester l'expression OU(joursem(date)=7;joursem(date)=1). On aura également besoin de la mise en forme conditionnelle des cellules, afin que celle-ci ne reste pas figée dans une cellule (les samedis et dimanches ne seront pas toujours au même endroit).

  Astuce de survie : lorsqu'on sélectionne une plage de cellules, la cellule à partir de laquelle on a commencé la sélection n'a pas de couleur d'arrière-plan ; elle a un statut spécial, que l'on pourrait qualifier de « représentant de la sélection ». Cette notion sera indispensable pour la suite.

- Sélectionner la plage A4:L34 à partir de la cellule A4, qui a donc le statut de représentante ;
- Mise en forme conditionnelle > Nouvelle règle
  et faire les choix comme ci-dessous :



- Dans la zone prévue à cet effet, taper la formule =OU(joursem(A4)=7;joursem(A4)=1)
  (la formule doit être une expression booléenne)
- OK

- Pour les profs superstitieux, les vendredis 13 : le 13 du mois est à la ligne 16 (plage A16:L16) ; il suffit donc d'y tester si JOURSEM(date) vaut 6. Faire donc comme ci-dessus en utilisant la formule =joursem(A16)=6 et en choisissant une autre mise en forme.

- Il y a interro de math tous les premiers lundis du mois, il faut également faire ressortir ces dates parmi les autres.

  On aura besoin de l'alternative multiple, la fonction CHOISIR(expression;cas1;cas2;...). expression doit être un entier dans [1,254] ; si expression vaut 1, c'est cas1 qui est engendré, etc.

  JOURSEM(date) est un entier dans [0,6] et le lundi a pour valeur 2, d'où 1+JOURSEM(date) est un entier dans [1,7] et le lundi a pour valeur 3.

  Les premiers lundis de chaque mois sont nécessairement dans la plage A4:L10 (la première semaine) ; c'est donc cette plage qu'il faut sélectionner pour la mise en forme conditionnelle. Et la formule conditionnelle sera =CHOISIR(1+JOURSEM(A4);1;2;3;4;5;6;7)=3.

  Il reste à choisir une mise en forme distincte des deux précédentes, ce qui ne devrait pas être trop compliqué.

Si vous êtes à la retraite et que votre club de joueurs de cartes se réunit tous les deuxièmes jeudis du mois, c'est un bel exercice.

On peut évidemment modifier le format du calendrier, largeur des colonnes, hauteur des lignes, un mois par feuille, etc. selon les goûts, les couleurs et les besoins.

En avant-première, le calendrier de l'année 2052.

Le 25 septembre est un mercredi, c'est le jour de mon anniversaire… chouette, il n'y a pas cours cet après-midi !

	A	B	C	D	E	F	G	H	I	J	K	L
1	année	**2052**										
2	1	2	3	4	5	6	7	8	9	10	11	12
3	Janvier	Février	Mars	Avril	Mai	Juin	Juillet	Août	Septembre	Octobre	Novembre	Décembre
4	Lun 01	Jeu 01	Ven 01	Lun 01	Mer 01	Sam 01	Lun 01	Jeu 01	Dim 01	Mar 01	Ven 01	Dim 01
5	Mar 02	Ven 02	Sam 02	Mar 02	Jeu 02	Dim 02	Mar 02	Ven 02	Lun 02	Mer 02	Sam 02	Lun 02
6	Mer 03	Sam 03	Dim 03	Mer 03	Ven 03	Lun 03	Mer 03	Sam 03	Mar 03	Jeu 03	Dim 03	Mar 03
7	Jeu 04	Dim 04	Lun 04	Jeu 04	Sam 04	Mar 04	Jeu 04	Dim 04	Mer 04	Ven 04	Lun 04	Mer 04
8	Ven 05	Lun 05	Mar 05	Ven 05	Dim 05	Mer 05	Ven 05	Lun 05	Jeu 05	Sam 05	Mar 05	Jeu 05
9	Sam 06	Mar 06	Mer 06	Sam 06	Lun 06	Jeu 06	Sam 06	Mar 06	Ven 06	Dim 06	Mer 06	Ven 06
10	Dim 07	Mer 07	Jeu 07	Dim 07	Mar 07	Ven 07	Dim 07	Mer 07	Sam 07	Lun 07	Jeu 07	Sam 07
11	Lun 08	Jeu 08	Ven 08	Lun 08	Mer 08	Sam 08	Lun 08	Jeu 08	Dim 08	Mar 08	Ven 08	Dim 08
12	Mar 09	Ven 09	Sam 09	Mar 09	Jeu 09	Dim 09	Mar 09	Ven 09	Lun 09	Mer 09	Sam 09	Lun 09
13	Mer 10	Sam 10	Dim 10	Mer 10	Ven 10	Lun 10	Mer 10	Sam 10	Mar 10	Jeu 10	Dim 10	Mar 10
14	Jeu 11	Dim 11	Lun 11	Jeu 11	Sam 11	Mar 11	Jeu 11	Dim 11	Mer 11	Ven 11	Lun 11	Mer 11
15	Ven 12	Lun 12	Mar 12	Ven 12	Dim 12	Mer 12	Ven 12	Lun 12	Jeu 12	Sam 12	Mar 12	Jeu 12
16	Sam 13	Mar 13	Mer 13	Sam 13	Lun 13	Jeu 13	Sam 13	Mar 13	Ven 13	Dim 13	Mer 13	Ven 13
17	Dim 14	Mer 14	Jeu 14	Dim 14	Mar 14	Ven 14	Dim 14	Mer 14	Sam 14	Lun 14	Jeu 14	Sam 14
18	Lun 15	Jeu 15	Ven 15	Lun 15	Mer 15	Sam 15	Lun 15	Jeu 15	Dim 15	Mar 15	Ven 15	Dim 15
19	Mar 16	Ven 16	Sam 16	Mar 16	Jeu 16	Dim 16	Mar 16	Ven 16	Lun 16	Mer 16	Sam 16	Lun 16
20	Mer 17	Sam 17	Dim 17	Mer 17	Ven 17	Lun 17	Mer 17	Sam 17	Mar 17	Jeu 17	Dim 17	Mar 17
21	Jeu 18	Dim 18	Lun 18	Jeu 18	Sam 18	Mar 18	Jeu 18	Dim 18	Mer 18	Ven 18	Lun 18	Mer 18
22	Ven 19	Lun 19	Mar 19	Ven 19	Dim 19	Mer 19	Ven 19	Lun 19	Jeu 19	Sam 19	Mar 19	Jeu 19
23	Sam 20	Mar 20	Mer 20	Sam 20	Lun 20	Jeu 20	Sam 20	Mar 20	Ven 20	Dim 20	Mer 20	Ven 20
24	Dim 21	Mer 21	Jeu 21	Dim 21	Mar 21	Ven 21	Dim 21	Mer 21	Sam 21	Lun 21	Jeu 21	Sam 21
25	Lun 22	Jeu 22	Ven 22	Lun 22	Mer 22	Sam 22	Lun 22	Jeu 22	Dim 22	Mar 22	Ven 22	Dim 22
26	Mar 23	Ven 23	Sam 23	Mar 23	Jeu 23	Dim 23	Mar 23	Ven 23	Lun 23	Mer 23	Sam 23	Lun 23
27	Mer 24	Sam 24	Dim 24	Mer 24	Ven 24	Lun 24	Mer 24	Sam 24	Mar 24	Jeu 24	Dim 24	Mar 24
28	Jeu 25	Dim 25	Lun 25	Jeu 25	Sam 25	Mar 25	Jeu 25	Dim 25	Mer 25	Ven 25	Lun 25	Mer 25
29	Ven 26	Lun 26	Mar 26	Ven 26	Dim 26	Mer 26	Ven 26	Lun 26	Jeu 26	Sam 26	Mar 26	Jeu 26
30	Sam 27	Mar 27	Mer 27	Sam 27	Lun 27	Jeu 27	Sam 27	Mar 27	Ven 27	Dim 27	Mer 27	Ven 27
31	Dim 28	Mer 28	Jeu 28	Dim 28	Mar 28	Ven 28	Dim 28	Mer 28	Sam 28	Lun 28	Jeu 28	Sam 28
32	Lun 29	Jeu 29	Ven 29	Lun 29	Mer 29	Sam 29	Lun 29	Jeu 29	Dim 29	Mar 29	Ven 29	Dim 29
33	Mar 30		Sam 30	Mar 30	Jeu 30	Dim 30	Mar 30	Ven 30	Lun 30	Mer 30	Sam 30	Lun 30
34	Mer 31		Dim 31		Ven 31		Mer 31	Sam 31		Jeu 31		Mar 31

Tableau d'amortissement

Résumé. *Les crédits à la consommation sont des opérations qui sont devenues banales, que ce soit pour acheter le super ordinateur dernier cri, ou un bien immobilier, et que l'on soit un particulier ou une entreprise. Si le montant périodique à rembourser est un élément fondamental de la transaction, il y a parfois des considérations fiscales à prendre en compte si le capital peut être amorti. D'où l'utilité du tableau d'amortissement.*

Rappelons à toutes fins utiles qu'un crédit est un engagement, qu'il faut vérifier ses capacités de remboursement, et qu'il faut être majeur [1].

Que ce soit avec le tableur *Microsoft Excel* ou *OpenOffice Calc*, les fonctions utilisées et les formules sont absolument identiques.

1 Un peu de mathématiques pour commencer

1.1 Valeur future de n versements identiques équidistants à taux constant

Par exemple, verser tous les mois un montant identique sur un compte d'épargne.

Notons t le taux **périodique**, n le nombre de versements, v le montant d'un versement, et v_i la valeur acquise après i versements. On a, successivement :

$$\begin{aligned}
v_1 &= v \\
v_2 &= v(1+t)+v \\
v_3 &= [v(1+t)+v](1+t)+v \\
&= v(1+t)^2+v(1+t)+v \\
&\cdots \\
v_n &= v(1+t)^{n-1}+v(1+t)^{n-2}+\cdots+v(1+t)+v
\end{aligned}$$

On reconnaît là une somme de termes d'une suite géométrique de raison $(1+t)$ et de premier terme v, ce qui permet d'écrire

$$\boxed{v_n = v\frac{(1+t)^n-1}{t}} \tag{18.1}$$

1. Ça, c'est pour ma responsabilité !

1.2 Valeur actuelle d'une suite de n versements futurs

Calcul de v_0 à partir de v_n.

La formule dite des intérêts composés permet d'écrire $v_n = v_0(1+t)^n$ pour le placement d'un capital initial v_0 pendant n périodes au taux périodique t. On en tire

$$\boxed{v_0 = \frac{v_n}{(1+t)^n}} \tag{18.2}$$

Par (18.1), on peut donc écrire

$$\begin{aligned} v_0 &= v\frac{(1+t)^n - 1}{t(1+t)^n} \\ &= v\left(\frac{(1+t)^n}{t(1+t)^n} - \frac{1}{t(1+t)^n}\right) \\ &= v\left(\frac{1}{t} - \frac{1}{t(1+t)^n}\right) \end{aligned}$$

Et donc

$$\boxed{v_0 = v\frac{1-(1+t)^{-n}}{t}} \tag{18.3}$$

1.3 Prêt à tempérament

Supposons que j'emprunte **maintenant** un capital (initial) v_0 qui sera remboursé en n versements identiques à un taux périodique constant t.

Le prêteur ne dispose donc plus de ce capital v_0 mais pourrait placer chaque versement pendant n périodes et obtenir ainsi un capital final v_n. Il faut donc que les n remboursements identiques v aient comme valeur future v_n. Par la formule (18.3) on peut calculer v (le montant du remboursement périodique) en fonction de v_0 (le capital emprunté) :

$$\boxed{v = v_0\frac{t}{1-(1+t)^{-n}}} \tag{18.4}$$

1.4 Taux annuel - taux mensuel

Le TAEG (taux annuel effectif global) est un taux d'intérêt qui intègre, en situation de remboursement d'un crédit, l'intérêt bancaire, les frais de dossier, d'assurance, etc.

Comme son nom l'indique, il est exprimé en taux annuel, mais dans la majorité des cas, les remboursements sont des mensualités. Il est donc nécessaire de calculer le taux mensuel correspondant.

Notons $TAEG$ le taux annuel, t le taux mensuel et C_i le capital à la période i, et comparons les deux tableaux :

Période (année)	Capital
0	C_0
1	$C_1 = C_0(1 + TAEG)$

Période (mois)	Capital
0	C_0
1	$C_1 = C_0(1 + t)$
2	$C_2 = C_0(1 + t)^2$
…	…
12	$C_{12} = C_0(1 + t)^{12}$

Comme dans nos régions une année est composée de douze mois, on peut égaler C_1 du premier tableau avec C_{12} du second, et après simplification par $C_0 \neq 0$, il reste

$$(1 + t)^{12} = 1 + TAEG$$
$$(1 + t) = \sqrt[12]{1 + TAEG}$$

Et donc

$$\boxed{t = -1 + \sqrt[12]{1 + TAEG}} \tag{18.5}$$

Ce taux est appelé *taux actuariel*. Si l'on doit prendre en compte un taux actuariel trimestriel ou semestriel, il convient de remplacer ci-dessus 12 respectivement par 4 ou par 2.

2 Le tableur est pratique pour les calculs

2.1 Tableau d'amortissement

Lorsqu'on emprunte un capital, il faut évidemment le rembourser avec des intérêts en sus. La somme des deux montants est partagée en n mensualités [(2)] identiques calculées par la formule (18.4).

Chaque mensualité est donc composée d'une part de capital et d'une part d'intérêts. Dans la première mensualité, la part d'intérêts est maximale, puisque calculée sur le montant total emprunté. Dans la seconde mensualité, la part d'intérêts est moindre, puisque calculée sur le montant emprunté diminué de la part de capital déjà remboursée lors de la première mensualité ; et en conséquence, la part de capital est plus élevée. Et ainsi de suite. On rembourse donc de moins en moins d'intérêts et de plus en plus de capital.

Avec un tableur, il est aisé de calculer le tableau (dit d'amortissement) dans lequel on trouve, pour chaque période, la part d'intérêts et la part de capital (appelée amortissement), ainsi que le capital restant dû.

Un tableau d'amortissement comporte autant de lignes qu'il y a de périodes de remboursement. Pour les explications, ce n'est pas le montant emprunté ni sa durée qui importe. Pour éviter un tableau trop long, imaginons un crédit fictif de 2000 € remboursable en une année avec un TAEG de 2,70 %. Ce sont les premières données à mettre en place dans la feuille de calculs.

	A	B
1	Montant	2000
2	Durée (mois)	12
3	TAEG	2,70 %
4	Taux mensuel (t)	
5	Mensualité	

2. Ou annuités, ou …

Utilisées uniquement en mode d'adressage absolu dans la suite des calculs, les cellules B1, B2, B4 et B5 seront nommées respectivement `montant`, `mois`, `tm` et `mens`.

La cellule B4 est calculée avec la formule (18.5) ; la cellule B5 quant à elle peut être calculée soit avec la formule (18.4), soit avec la fonction financière `VPM(taux;durée;montant)`qui fournit un résultat négatif (car débit) ; on choisira la formule `–VPM(tm;mois;montant)` avec les noms de cellules qui ont été choisis.

Dans un premier temps, nous construirons la table d'amortissement à partir du raisonnement ad'hoc, puis, dans un second temps, nous reproduirons certains calculs au moyen de deux fonctions financières dédiées.

La figure ci-après montre le tableau terminé ; les titres de colonnes ne nécessitent pas d'explications particulières.

	A	B	C	D	E	F	G	H
1	Montant	2000						
2	Durée (mois)	12						
3	TAEG	2,70%						
4	Taux mensuel	0,002222627						
5	Mensualité	169,08 €						
6							INTPER()	PRINCPER()
7	Période	Mensualité	Intérêts	Amortissement (capital)	Capital restant dû		Intérêts	Amortissement
8	0				2000			
9	1	169,0843134	4,445254589	164,6390588	1835,360941		(4,45 €)	(164,64 €)
10	2	169,0843134	4,079323323	165,0049901	1670,355951		(4,08 €)	(165,00 €)
11	3	169,0843134	3,712578728	165,3717347	1504,984216		(3,71 €)	(165,37 €)
12	4	169,0843134	3,345018997	165,7392944	1339,244922		(3,35 €)	(165,74 €)
13	5	169,0843134	2,976642317	166,1076711	1173,137251		(2,98 €)	(166,11 €)
14	6	169,0843134	2,607446874	166,4768666	1006,660384		(2,61 €)	(166,48 €)
15	7	169,0843134	2,237430846	166,8468826	839,8135017		(2,24 €)	(166,85 €)
16	8	169,0843134	1,866592411	167,217721	672,5957806		(1,87 €)	(167,22 €)
17	9	169,0843134	1,49492974	167,5893837	505,006397		(1,49 €)	(167,59 €)
18	10	169,0843134	1,122441002	167,9618724	337,0445245		(1,12 €)	(167,96 €)
19	11	169,0843134	0,74912436	168,3351891	168,7093355		(0,75 €)	(168,34 €)
20	12	169,0843134	0,374977974	168,7093355	0		(0,37 €)	(168,71 €)
21	Total	2029,011761	29,01176116	2000				
22								

Dans notre situation il y a 12 périodes (1 à 12). On ajoutera la période fictive 0 pour initialiser la série de calculs des soldes restant dûs, mais ce n'est pas choquant. Les périodes doivent être des nombres naturels (pour les deux fonctions financières qui seront utilisées par la suite) ; s'il s'avère nécessaire de les coupler à une date par exemple, il faut ajouter une colonne supplémentaire.

- E8 : le montant emprunté (`=montant`) ;
- plage B9:B20 : les mensualités, toutes identiques (`=mens`) ;
- C9 : les intérêts de la première période, calculés sur la capital restant dû pour cette période, (`=E8*tm`), formule copiable jusqu'en C20 ;
- D9 : la partie capital de la première période, différence entre la mensualité et les intérêts (`=B9–C9`), formule copiable jusqu'en D20 ;
- E9 : capital restant dû après le premier versement, différence entre le capital restant dû précédent et le capital amorti de cette période (`=E8–D9`), formule copiable jusqu'en E20.

Une fois compris le principe des calculs, on peut utiliser deux fonctions permettant de calculer directement, pour une période donnée (c'est pour ces fonctions que la période doit être un entier), la partie intérêts et la partie capital ; il s'agit respectivement des fonctions `INTPER(taux;période;durée;montant)` et `PRINCPER(taux;période;durée;montant)`.

- `G9` : (`=INTPER(tm;A9;mois;montant)`), copiable jusqu'en `G20` ;
- `H9` : (`=PRINCPER(tm;A9;mois;montant)`), copiable jusqu'en `H20`.

Nous avons ici laissé le format d'affichage par défaut, à savoir un nombre rouge entre parenthèses signifiant un nombre négatif (débit), un arrondi à deux décimales et le suffixe €. Si cela est gênant, il suffit de faire précéder les deux fonctions par le signe –.

Remarques

Pour une durée plus longue, il suffit de prolonger le tableau avec le nombre de périodes approprié.

Si les remboursements sont annuels, semestriels, ou autre, ce sont les mêmes formules, il faut simplement veiller à utiliser le taux périodique correct.

Pour un autre montant et/ou un autre taux, il suffit de modifier respectivement les cellules `B1` et/ou `B3`.

L'utilisation d'un taux mensuel **proportionnel** $\left(\frac{TAEG}{12}\right)$ en lieu et place de la formule (18.5) donne les résultats suivants :

	A	B	C	D	E	F	G	H
1	**Montant**	**2000**						
2	**Durée (mois)**	**12**						
3	**TAEG**	**2,70%**						
4	**Taux mensuel**	**0,00225**						
5	**Mensualité**	**169,11 €**						
6							**INTPER()**	**PRINCPER()**
7	**Période**	**Mensualité**	**Intérêts**	**Amortissement (capital)**	**Capital restant dû**		**Intérêts**	**Amortissement**
8	**0**				**2000**			
9	**1**	**169,1142099**	**4,5**	**164,6142099**	**1835,38579**		**(4,50 €)**	**(164,61 €)**
10	**2**	**169,1142099**	**4,129618028**	**164,9845919**	**1670,401198**		**(4,13 €)**	**(164,98 €)**
11	**3**	**169,1142099**	**3,758402696**	**165,3558072**	**1505,045391**		**(3,76 €)**	**(165,36 €)**
12	**4**	**169,1142099**	**3,38635213**	**165,7278578**	**1339,317533**		**(3,39 €)**	**(165,73 €)**
13	**5**	**169,1142099**	**3,013464449**	**166,1007455**	**1173,216788**		**(3,01 €)**	**(166,10 €)**
14	**6**	**169,1142099**	**2,639737772**	**166,4744722**	**1006,742315**		**(2,64 €)**	**(166,47 €)**
15	**7**	**169,1142099**	**2,26517021**	**166,8490397**	**839,8932757**		**(2,27 €)**	**(166,85 €)**
16	**8**	**169,1142099**	**1,88975987**	**167,2244501**	**672,6688257**		**(1,89 €)**	**(167,22 €)**
17	**9**	**169,1142099**	**1,513504858**	**167,6007051**	**505,0681206**		**(1,51 €)**	**(167,60 €)**
18	**10**	**169,1142099**	**1,136403271**	**167,9778067**	**337,0903139**		**(1,14 €)**	**(167,98 €)**
19	**11**	**169,1142099**	**0,758453206**	**168,3557567**	**168,7345572**		**(0,76 €)**	**(168,36 €)**
20	**12**	**169,1142099**	**0,379652754**	**168,7345572**	**-7,95808E-13**		**(0,38 €)**	**(168,73 €)**
21	**Total**	**2029,370519**	**29,37051924**	**2000**				
22								

Et pour terminer, un graphique où sont représentés simultanément la part d'intérêts et la part de capital remboursées mensuellement pour le même capital emprunté de 2000 € pendant une année, avec un TAEG de 15%.

Note technique : si on choisit un taux trop petit, il devient difficile de distinguer la part d'intérêts dont la proportion devient trop petite par rapport à celle du capital. Voilà pour le choix de la valeur 15%.

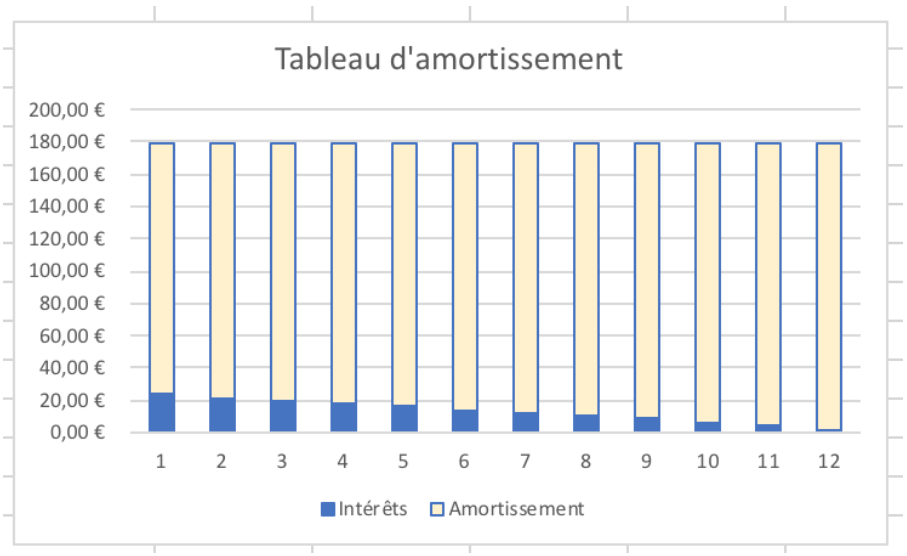


Tableau d'amortissement
200,00 €
180,00 €
160,00 €
140,00 €
120,00 €
100,00 €
80,00 €
60,00 €
40,00 €
20,00 €
0,00 €
1
2
3
4
5
6
7
8
9
10
11
12
Intérêts
Amortissement

Stéganographie, une partie de cache-cache avec des 0 et des 1

Résumé. *Une petite recherche dans notre navigateur préféré, et on trouve rapidement le renseignement cherché : la* stéganographie *est l'art de dissimuler un message dans un autre message, celui-ci pouvant être un texte, une image, un son. Cela se passe toujours en deux étapes : le codage, puis évidemment le décodage. L'informatique est d'un grand secours pour les nombreux calculs, en ce qui concerne les fichiers de type image ou son, d'autant plus que ceux-ci sont codés dans un langage que nous n'utilisons pas quotidiennement, le langage binaire.*

On expliquera dans cet article **une** méthode parmi d'autres, la méthode LSB —Least Significant Bits[1]—, que l'on appliquera à des images simplissimes afin de ne pas entrer dans de la programmation sophistiquée. On touchera donc au calcul binaire pour la méthode, au langage *Ruby* et au tableur *Microsoft Excel*[2] pour les calculs, et à LaTeX pour les images d'application, qui ne seront composées chacune que de quatre carrés, que l'on pourrait assimiler à quatre pixels. Le passage de 4 pixels à quelques millions de pixels n'est qu'une question de temps…

1 Langage binaire

Le langage binaire ne comporte que deux « mots » de vocabulaire, $\boxed{0}$ et $\boxed{1}$, appelés *bit* en informatique.

L'écriture décimale d'un nombre (en base 10) consiste à le décomposer en somme de puissances de 10 et à noter successivement les coefficients de ces puissances ; ainsi 5437 signifie $5\times10^3+4\times10^2+3\times10^1+7\times10^0$. Si on travaille avec plusieurs bases simultanément, on peut écrire $(5437)_{10}$ pour éviter les confusions.

Avec le même raisonnement, l'écriture binaire d'un nombre (en base 2) consiste à le décomposer en somme de puissances de 2 et à noter successivement les coefficients de ces puissances ; ainsi 11010 signifie $1\times2^4+1\times2^3+0\times2^2+1\times2^1+0\times2^0$, ce qui vaut 26, en base 10.

On a donc $(26)_{10}=(11010)_2$.

En pratique, une information numérique est souvent (en tout cas dans le contexte qui nous intéresse) codée sur 8 bits, ce qu'on appelle un *byte* en anglais, un *octet* en français. Ainsi, le plus petit nombre codé sur un octet vaut $(00000000)_2=(0)_{10}$ et le plus grand vaut $(11111111)_2=(255)_{10}$.

Dans un octet, par exemple $\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}\boxed{0}\boxed{1}$, les bits à gauche sont appelés *bits*

1. Bits de moindre importance
2. Toutes les fonctions nécessaires n'existent pas (encore) dans *OpenOffice Calc*.

de poids fort et ceux à droite sont appelés *bits de poids faible*. En effet, les bits à gauche sont les coefficients des plus fortes puissances de 2 ($2^7, 2^6, 2^5$) tandis que les bits à droite sont les coefficients des plus faibles puissances de 2 ($2^0, 2^1, 2^2$). Notons qu'il n'y a pas de loi qui fixe la frontière entre les bits de poids fort et les bits de poids faible, mais la qualité des résultats que nous obtiendrons vont dépendre de cette frontière fictive.

2 La partie de cache-cache

2.1 Cacher un nombre dans un autre nombre

Que voilà un truc bizarre ! Sans compter qu'il faudra le retrouver… Cachons par exemple 100 dans 20.

Soit $a = (100)_{10}$ | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | et $b = (20)_{10}$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

Construisons c qui ressemble à b et qui contient une partie de a :

$c = (22)_{10}$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |

Algorithme : les 4 bits de poids fort de c sont les 4 bits de poids fort de b (ça c'est pour la ressemblance avec b) et les 4 bits de poids faible de c sont les 4 bits de poids fort de a (ça c'est pour les gênes de a). (3)

Comme on a conservé les bits de poids fort de b, l'écart maximal entre c et b est | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | c'est-à-dire $(15)_{10}$. C'est beaucoup, diront certains, mais ces nombres seront en fait des proportions de couleur allant de 0 à 255 (tiens donc) et un écart de 15 ne représente que 5,88%, ce qui sera quasiment invisible à l'oeil humain.

2.2 Et on calcule comment ?

Il nous faut des opérateurs qui calculent avec des nombres binaires ; certaines calculatrices le permettent, mais aussi la plupart des langages de programmation, et le tableur *Microsoft Excel*.

En ce qui concerne les langages de programmation, nous avons l'habitude du langage *Ruby* [1], mais les opérations binaires sont assez universelles.

- Le décalage à droite de n bits (**»n**) : supprime les n bits de poids faibles puis insère n bits 0 de poids fort.

  Dans notre exemple, **a»4** va transformer a en | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

- L'opération « et » appelée « masque » et notée **&** : c'est l'opération logique de conjonction (1 et 1 vaut 1, et 0 dans les 3 autres cas). Elle opère « bit à bit ». N'importe quel bit face à 0 annule le bit, et n'importe quel bit face à 1 garde le bit : 1 et 0 vaut 0, 0 et 0 vaut 0, 1 et 1 vaut 1, 0 et 1 vaut 0.

  On voudrait annuler les 4 bits de poids faible de b, sans modifier les 4 bits de poids fort : on choisira donc le masque | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |.

3. On a choisi ici une « découpe » 4 bits forts et 4 bits faibles, on aurait aussi pu choisir 3×5, ou 5×3, c'est un compromis à faire entre la qualité du camouflage et la qualité de la restitution.

D'où : $b = (20)_{10}$ | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | **&** | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | $(240)_{10}$ **=**

0	0	0	1	0	0	0	0

Il reste à ajouter ce résultat à celui obtenu avec **a»4** et on obtient $c = (22)_{10}$

0	0	0	1	0	1	1	0

En résumé, avec le mode direct de *Ruby*, il suffit de taper l'instruction `(100 » 4) + (20 & 240)` ; les parenthèses sont obligatoires pour la priorité des opérations.

Avec le tableur (*Excel*) ce sera sans doute plus simple, et surtout on pourra visualiser simultanément les résultats de calculs pour plusieurs valeurs de a et de b. Voici donc les fonctions nécessaires à notre projet.

- `BITDECALD(nombre;n)` : décale à droite n bits de nombre ;
- `BITET(nombre1;nombre2)` : exécute le ET logique bit à bit entre les deux nombres ;
- `DECBIN(nombre;n)` : convertit un nombre décimal en binaire en affichant n bits.

On construit donc le tableau suivant :

	A	B	C
1	a	b	c
2	100	20	=BITDECALD(A2;4) + BITET(B2;240)
3	=DECBIN(A2;8)	=DECBIN(B2;8)	=DECBIN(C2;8)

qui donne les résultats suivants :

	A	B	C
1	a	b	c
2	100	20	22
3	01100100	00010100	00010110

La ligne 3 sert plutôt de visualisation et de vérification.

Remarque importante : lorsqu'il s'agira de lire un fichier image composé de plusieurs millions de pixels, et que pour chaque pixel il faudra masquer 3 ou 4 nombres, il est clair qu'il sera indispensable d'abandonner le tableur et passer par un langage de programmation.

2.3 Un peu de théorie de couleurs et une application concrète

Une image est composée de pixels, et chaque pixel est caractérisé par une couleur. Il existe plusieurs méthodes pour coder une couleur, selon le support.

Sur un écran, c'est souvent la méthode RGB (Red, Green, Blue) [4], un triplet représentant un mélange savant de ces trois couleurs de base ; chaque nombre de ce triplet est compris entre 0 et 255.

4. RVB en français, rouge, vert, bleu.

Sur papier, chez l'imprimeur, c'est la méthode CMYK (Cyan, Magenta, Yellow, Black) [5], appelée *quadrichromie* ; même principe que RGB, mais avec quatre coefficients. L'imprimeur utilise successivement quatre plaques de couleurs, et le document passe donc quatre fois dans la machine. *Losanges* est imprimé en bichromie, où seules les plaques magenta et black sont utilisées ; les couleurs de chaque pixel d'une image sont donc de la forme $(0,x,0,y)$, ce qui a été respecté dans les images qui suivent.

Soit une image a que l'on voudrait dissimuler dans une image b ; pour simplifier, on s'est limité à quatre carrés de couleur par image ; à côté de chaque image, un tableau contenant les codes de couleurs utilisés pour chaque carré.

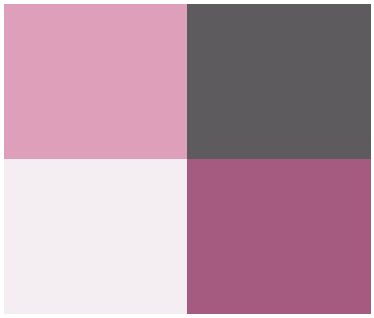


Fig. 14 *Image a*

(0,100,0,27)	(0,10,0,197)
(0,10,0,12)	(0,150,0,97)

Fig. 15 *Codes image a*

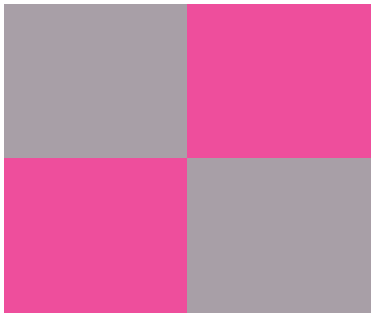


Fig. 16 *Image b*

(0,20,0,100)	(0,215,0,0)
(0,215,0,0)	(0,20,0,100)

Fig. 17 *Codes image b*

On va créer une image c en appliquant la méthode décrite précédemment à chaque composante des couleurs. Pour le carré supérieur gauche, par exemple, il faudra cacher 100 dans 20, ce qui a déjà été fait, et dont le résultat est 22, et cacher 27 dans 100, dont le résultat est 97. Le carré supérieur gauche de l'image c aura donc comme code couleur le quadruplet (0,22,0,97). Voici le tableau Excel qui calcule toutes les transformations nécessaires :

	A	B	C	D
1	a	b	c	
2	100	20	22	coin sup. gauche
3	27	100	97	
4	10	215	208	coin sup. droit
5	197	0	12	
6	10	215	208	coin inf. gauche
7	12	0	0	
8	150	20	25	coin inf. droit
9	97	100	102	

5. CMJN en français, cyan, magenta, jaune, noir.

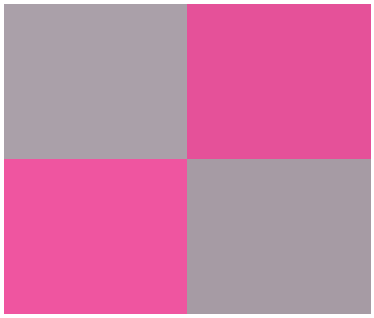


Fig. 18 *Image c*

(0,22,0,97)	(0,208,0,12)
(0,208,0,0)	(0,25,0,102)

Fig. 19 *Codes image c*

Et on n'y voit que du feu…

2.4 Retrouver le nombre caché

Il faut évidemment pouvoir extraire l'image a de l'image c.

Nous avons caché 100 dans 20, ce qui a donné 22 ; à partir de 22, il faut pouvoir (espérer) retrouver 100.

À partir de $c = (22)_{10}$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | on va construire… a'.

- Il faut déjà récupérer les 4 bits de poids faible de c qui étaient les 4 bits de poids fort de a : | 0 | 1 | 1 | 0 | | | | |.

  On commence par annuler les 4 bits de poids fort avec un masque « et » qui conserve les bits de poids faible : $c = (22)_{10}$ | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | ET | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | = | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

  Et ensuite appliquer au résultat un décalage à gauche de 4 bits, avec effet de mettre des 0 dans les bits de poids fort, ce qui donnera comme résultat | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

  La valeur décimale du masque vaut 15.

  L'instruction *Ruby* correspondante est donc `(c & 15) « 4`

- Les 4 bits de poids faible de a n'ont jamais été sauvegardés, mais ils sont compris entre $(0000)_2$ et $(1111)_2$, soit entre 0 et 15 ; on peut prendre n'importe lequel de ces nombres, on a opté ici pour la moyenne (8), qu'il suffit d'ajouter au résultat précédent.

En résumé, avec le mode direct de *Ruby*, il faut taper l'instruction `((c & 15) « 4)+8` qui a pour valeur 104. On n'a pas retrouvé 100, mais presque…

Avec le tableur *Excel*, on a besoin de la fonction `BITDECALG(nombre;n)` qui décale à gauche d'un nombre de n bits.

On construit donc le tableau suivant :

	A	B
1	c	a'
2	22	=BITDECALG(BITET(A2;15);4)+8

qui donne le résultat annoncé :

	A	B
1	c	a'
2	22	104

On peut donc appliquer la méthode à tous les codes de couleur intervenant dans l'image c pour retrouver une image… a'.

	A	B	C	D
1	c	a'	a original	
2	22	104	100	coin sup. gauche
3	97	24	27	
4	208	8	10	coin sup. droit
5	12	200	197	
6	208	8	10	coin inf. gauche
7	0	8	12	
8	25	152	150	coin inf. droit
9	102	104	97	

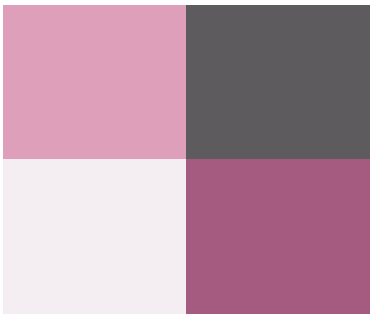


Fig. 20 *Image a*

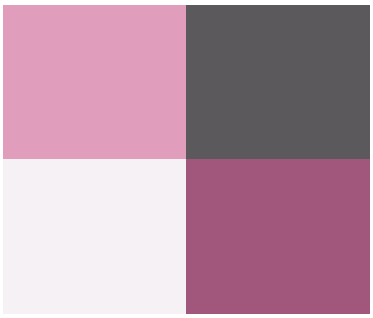


Fig. 21 *Image a'*

(0,104,0,24)	(0,8,0,200)
(0,8,0,8)	(0,152,0,104)

Fig. 22 *Codes image a'*

3 Annexe : code LaTeX

- Pour définir une couleur CMYK, il faut utiliser l'instruction `\definecolor{nom_coul}{cmyk}{x,y,z,t}` où x, y, z et t représentent respectivement les proportions de cyan, magenta, jaune et noir. Mais ces nombres doivent être compris entre 0 et 1, et non entre 0 et 255. Votre serviteur a donc dû faire lui-même les divisons par 255. Ainsi, ce qui devait être par exemple (0,104,0,24) a été encodé `\definecolor{nom_coul}{cmyk}{0,0.596,0,0.407}` (3 décimales suffisent largement).
- Pour colorer l'arrière-plan d'une cellule, on utilise l'instruction `\cellcolor{nom_coul}`, disponible si le package `colortbl` a été spécifié dans le préambule : `\usepackage{colortbl}`

- Voici le code pour réaliser la figure 7 :

```
\renewcommand{\arraystretch}{4}
\begin{tabular}{p{2cm}p{2cm}}
    \cellcolor{coulap11} & \cellcolor{coulap12} \\
    \cellcolor{coulap21} & \cellcolor{coulap22} \\
\end{tabular}
\end{verbatim}

\texttt{arraystretch\{4\}} permet d'augmenter la hauteur d'une ligne standard d'u

Les quatre couleurs ont été définies dans le préambule~:

\begin{verbatimtab}[4]
\definecolor{coulap11}{cmyk}{0, 0.407, 0, 0.094}
\definecolor{coulap12}{cmyk}{0, 0.031, 0, 0.784}
\definecolor{coulap21}{cmyk}{0, 0.031, 0, 0.031}
\definecolor{coulap22}{cmyk}{0, 0.596, 0, 0.407}
```

4 De la démo à la réalité

L'exemple décrit dans ce qui précède n'a d'intérêt que la compréhension du principe de dissimulation puis de reconstruction de la couleur d'un pixel d'une image. Et il est sans intérêt du tout sur une image déjà imprimée.

La sténographie est utile (si on a besoin de camoufler des choses, évidemment) sur des fichiers informatiques qui sont échangés. Pour traiter de tels fichiers, il faut écrire des programmes capables de lire un fichier image, d'en extraire chaque pixel, de traiter chaque pixel (par exemple par la méthode expliquée précédemment) et de reconstituer une nouvelle image ; voilà pour le camouflage. Il faut ensuite le programme pour l'extraction de l'image masquée.

Ce genre de programmation n'est plus tout-à-fait élémentaire et dépasse le cadre de cet article.

La méthode LSB convient bien à des images au format .BMP ; la conversion d'une image d'un format dans un autre (par exemple de .BMP vers .JPG) va annihiler l'éventuelle image camouflée, suite aux algorithmes de compression.

Références

[1] Desbonnez J.-M., Algorithmique élémentaire et Ruby. *Losanges*, 28, pp. 46 – 55, Mars 2015.

[2] Signac L., *Divertissements mathématiques et informatiques*. MiniMax, 2011.

Le tableur Excel et les macro-instructions (1)

Résumé. *Une macro-instruction (« macro » pour les intimes) est un ensemble d'instructions qui permet, d'une part, d'automatiser des tâches répétitives, mais aussi de créer des outils qui permettent à un utilisateur « non initié » d'exploiter un logiciel spécifique (ici le tableur Excel) avec un minimum de connaissances (par exemple cliquer sur un bouton, choisir dans un menu, remplir une boîte de dialogue), et même aussi créer ses propres fonctions. Voici le nécessaire pour débuter et donner envie d'approfondir si affinité.*

Avertissement(s)

Le pré-requis est d'avoir déjà manipulé un tableur (ses formules, barres d'outils, menus, etc.). Il s'agira aussi d'apprendre un langage de programmation avec ses structures associées : alternatives, répétitives, et les syntaxes qui lui sont propres. Comme on dit dans les offres d'emplois, *une expérience en algorithmique et en programmation est un plus...*

Tout choix est un renoncement... les macros sont intimement liées à un environnement ; le choix s'est porté sur Microsoft Excel 2016 pour Mac. Les instructions sont quasi identiques pour Windows, même pour des versions antérieures d'Excel.

Sous Excel, le langage de programmation des macros s'appelle **VBA** (Visual Basic for Applications) ; avec OpenOffice, la possibilité de créer des macros existe aussi ; le langage s'appelle **OpenOffice Basic** mais est, à mon sens, beaucoup plus lourd à manipuler.

Pour la petite histoire, Microsoft avait supprimé VBA dans la version Excel 2008, mais l'a rétabli dans la version suivante suite à de nombreuses levées de bouclier.

Quant à la sacro sainte compatibilité entre ce qui se fît, se fait et se fera, elle reste une histoire dont seule l'informatique a le secret.

Il ne s'agira pas de faire un cours complet et complexe sur le langage VBA ; il existe bon nombre de littératures écrites et virtuelles qui traitent de ce sujet. Restons pédagogique...

1 Utile pour la suite, mais aussi pour la culture personnelle

1.1 Référencement par défaut, dit « A1 »

Par défaut, les colonnes d'une feuille de calculs sont numérotées A, B, C, ... tandis que les lignes sont numérotées 1, 2, 3, ... Les adresses des cellules sont de la forme A1, B4, ... MAIS lorsqu'on est en B4 et que l'on y écrit la formule `=A1+1`, celle-ci s'interprète (est interprétée par Excel) non pas comme « ajouter 1 au contenu de A1 », mais « ajouter 1 au contenu de la cellule située 1 colonne à gauche et 4 lignes plus bas que B4 ». Preuve en est que lorsque l'on copie le contenu de la cellule B4 en B5, elle devient `=A2+1`. L'interprétation relative est conservée, mais pas l'adresse A1. L'écriture `A1` est une adresse relative.

Si A1 doit rester A1 en cas de recopie verticale ou horizontale, il faut écrire $A$1 ; si A doit rester A, il faut écrire $A1, et si 1 doit rester 1, il faut écrire A$1. Ces différentes écritures sont appelées respectivement adresse absolue, adresse absolue en ligne et relative en colonne, adresse absolue en colonne et relative en ligne.

Une fois que l'on a compris le système, à savoir placer un symbole `$` devant l'élément ligne ou colonne à bloquer en cas de recopie, on n'y pense plus. Mais tôt ou tard, il faut expliquer à l'apprenant pourquoi la copie horizontale du mot Losanges reste Losanges, tandis que celle de la formule `=A2+1` devient `=B2+1`.

1.2 Référencement « Row Column »

Ce type de référencement [1] consiste à numéroter les colonnes 1, 2, 3, ... comme les lignes ; pour l'activer :

- `Préférences... > Général` ;
- cocher l'option « Utiliser le style de référence R1C1 ».

Sur PC sous Windows :

- `Outils > Options...` ;
- onglet `général` > cocher « Style de référence L1C1 ».

Si on active ce mode de référencement lorsqu'un classeur est ouvert, toutes les adresses de cellules dans les formules sont automatiquement « traduites » dans le nouveau mode.

	A	B	C	D	E
1	5				
2	0				
3	-3				
4		=A1+1	=$A$2+1	=A$3+1	=$A3+1
5		=A2+1	=$A$2+1	=A$3+1	=$A4+1
6		=A3+1	=$A$2+1	=A$3+1	=$A5+1
7					

Fig. 23 *Référencement A1*

	1	2	3	4	5
1	5				
2	0				
3	-3				
4		=L(-3)C(-1)+1	=L2C1+1	=L3C(-3)+1	=L(-1)C1+1
5		=L(-3)C(-1)+1	=L2C1+1	=L3C(-3)+1	=L(-1)C1+1
6		=L(-3)C(-1)+1	=L2C1+1	=L3C(-3)+1	=L(-1)C1+1

Fig. 24 *Référencement R1C1*

1. Indisponible actuellement avec OpenOffice.

Dans les figures ci-dessus, les formules originales sont dans les cellules de la ligne 4 et ont été copiées vers le bas jusqu'à la ligne 6.

- `$A$2` se traduit par `L2C1`, cellule située à l'intersection de la 2e ligne et de la 1re colonne ; c'est de l'adressage absolu ; remarquer que l'adresse commence par l'indice de ligne suivi de l'indice de colonne ;
- en `B4`, on fait référence à `A1` qui est située 3 lignes au-dessus et 1 colonne à gauche, ce qui se traduit par `L(-3)C(-1)` ; c'est de l'adressage relatif ; l'ampleur et le sens du pseudo-déplacement est indiqué entre parenthèses ; le sens positif est vers la droite en horizontal et vers le bas en vertical ;
- les détracteurs diront qu'il est plus compliqué de taper `L(-3)C(-1)` que `A1`... mais en mode composition de formule, on peut aussi cliquer sur la cellule dont on a besoin... et on copie plus de formules que l'on en tape...
- les accrocs répliqueront que les formules copiées sont identiques aux originales, que la numérotation est la même pour les lignes que pour les colonnes, et qui pourrait dire rapidement quelle est la lettre correspondant à la 1564e colonne ? Mais chacun fait son lit comme il se couche !

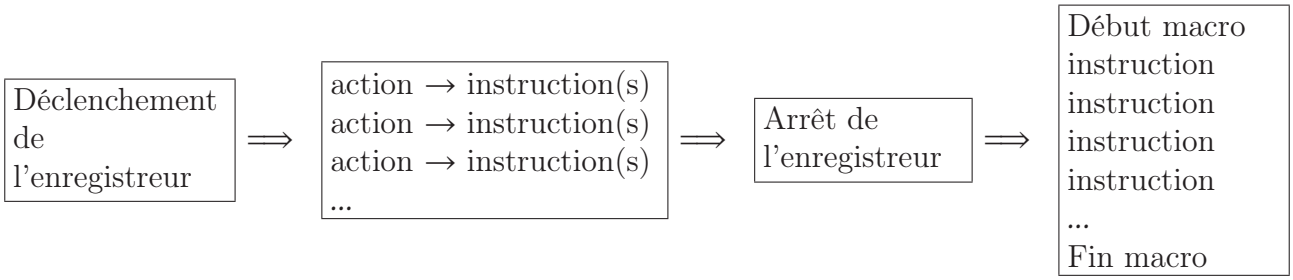
Que l'on se rassure, la création et l'utilisation de macros n'obligent nullement à travailler en mode `R1C1`, mais dans certaines syntaxes d'instruction on retrouvera `R1C1`... c'est maintenant expliqué.

2 L'enregistreur de macros

C'est un utilitaire qui permet de traduire en instructions VBA toute frappe au clavier, y compris les choix dans les menus et les barres d'outils, et la complétion de boîtes de dialogue. Il ne permet donc de créer que des macros composées de séquences d'instructions simples, mais surtout d'avoir un premier contact avec le langage VBA, ce qui n'est pas négligeable dans une situation d'apprentissage.

Nous allons créer une petite macro simple à l'aide de l'enregistreur, expliquer où elle est enregistrée, comment y accéder pour l'éditer, ainsi que les différentes manières de l'exécuter. Ceci est important pour la gestion des macros, aussi complexes soient-elles. Par la suite, on n'aura plus qu'à s'occuper des structures de programmation et des instructions plus complexes. C'est parti...

Attention : l'enregistreur enregistre [2] toutes les activités du clavier, y compris les erreurs de frappe... il est donc parfois utile de faire une répétition avant de commencer...



2. Ceci est une affirmation extrêmement forte !

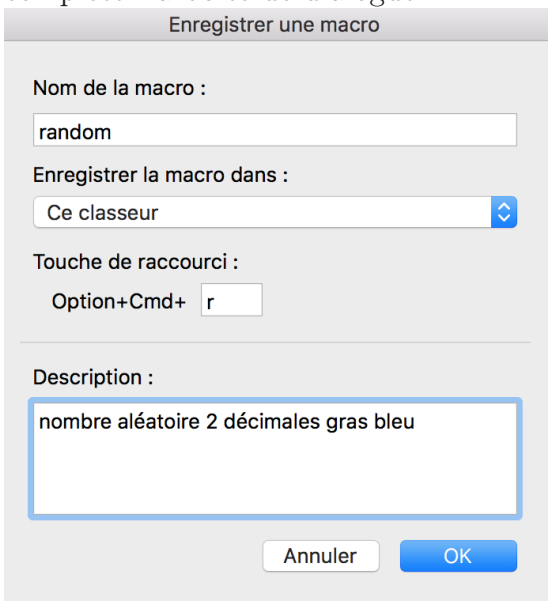
2.1 Une première application simple

On souhaite automatiser le calcul d'un nombre aléatoire dans la cellule active, arrondir ce nombre à 2 décimales, le mettre en gras et en bleu.

Pour rappel, la formule qui calcule un nombre aléatoire est `=ALEA()` ; une fois la formule tapée dans la cellule, il faut la valider en pressant `ENTER` (et c'est la cellule en-dessous qui devient active) ; puis il faut activer la cellule contenant la formule, et on a accès aux icônes de mises en forme, gras, couleur de police (standard) et réduction des décimales (autant de clics que nécessaires). Comme expliqué ci-avant, il est peut-être judicieux de faire un essai avant d'enclencher l'enregistreur.

2.1.1. Enregistrer la macro

- commencer par activer une cellule vide (afin de voir le résultat)
- (déclenchement de l'enregistreur) `Outils > Macro > Nouvelle macro...`
- compléter la boîte de dialogue



Il vaut mieux donner un nom plus parlant que macro1 ; par défaut, la macro est dans le classeur actif (ou peut demander un autre classeur ou un classeur de macros personnelles) ; on peut éventuellement choisir une lettre pour associer un raccourci clavier à l'exécution de la macro (cette lettre est toujours combinée à d'autres touches fixées, qui peuvent différer de Mac à Pc) ; on verra un peu plus loin (dans le texte de la macro) que la combinaison Option+Cmd sera remplacée par CTRL (je n'ai malheureusement pas d'explication...) ; enfin une éventuelle description qui peut parfois être utile.

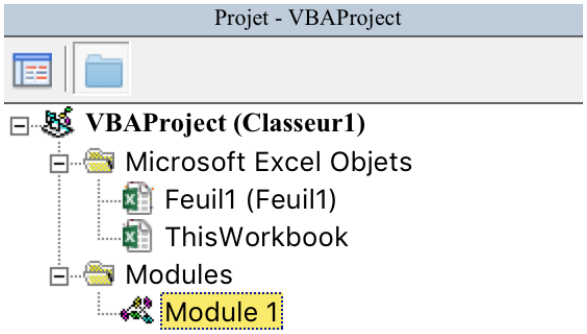
- (fin de l'enregistrement) `Outils > Macro > Arrêter l'enregistrement`

Dans les versions précédentes d'Excel, dès que l'enregistrement a démarré, un bouton (rouge) d'arrêt d'enregistrement est présent à l'écran.

Remarque : Si on ouvre le menu « Développeur », on a accès à une icône « Enregistrer la macro » qui fait la même chose que ci-dessus. Dès que l'enregistrement a démarré, l'icône « Enregistrer la macro » est remplacée par l'icône « Arrêter l'enregistrement ».

2.1.2. Voir la macro

- `Outils > Macro > Visual Basic Editor` ou icône « Visual Basic » dans la barre d'outils du menu développeur
- s'ouvre alors une nouvelle fenêtre « Microsoft Visual Basic » avec à gauche une structure arborescente appelée « VBAProject ».



En ouvrant la branche « Modules » on voit apparaître une feuille Module 1 dans laquelle on trouve les instructions de la macro. Toutes les macros vont s'écrire dans ces feuilles de type « Module » ; on peut en ajouter via le menu `Insertion > Module`.

Voici les instructions qui ont été enregistrées :

```
Sub random()
'
' random Macro
' nombre aléatoire 2 décimales gras bleu
'
' Touche de raccourci du clavier: Ctrl+r
'
    ActiveCell.FormulaR1C1 = "=RAND()"
    Range("G8").Select
    Selection.Font.Bold = True
    With Selection.Font
        .Color = -4165632
        .TintAndShade = 0
    End With
    Selection.NumberFormat = "0.0000000"
    Selection.NumberFormat = "0.000000"
    Selection.NumberFormat = "0.00000"
    Selection.NumberFormat = "0.0000"
    Selection.NumberFormat = "0.000"
    Selection.NumberFormat = "0.00"
End Sub
```

2.1.3. Première exploration des instructions

- les instructions d'une macro sont encadrées par la structure `Sub ... End Sub` ;
- le caractère « apostrophe » sert à introduire une ligne de commentaire (non exécutable) ; on y retrouve le nom de la macro, la description, et le raccourci clavier qui servira à l'exécution ;

- `ActiveCell.FormulaR1C1` est l'instruction qui permet d'écrire une formule dans la cellule active ;
- `Range("adresse").Select` est l'instruction qui permet de sélectionner une cellule dont on précise l'adresse. Lorsque l'on exécutera la macro, on verra que cette instruction devra être supprimée (ou commentée), puisqu'elle impose que le nombre aléatoire soit toujours dans la cellule `G8` (ou l'adresse d'une autre cellule, celle qui a été activée lors du déclenchement de l'enregistreur) ;
- les instructions de mises en forme commencent par `Sélection.` suivi du nom de la mise en forme, et de la mise en forme proprement dite ; on y apprend avec grand plaisir que la couleur bleue standard porte le numéro `-4165632` ; si on souhaite connaître le code d'une autre couleur, il suffit ... d'enregistrer une nouvelle macro !
- tous les clics sur l'icône de réduction de décimales ont été enregistrés ; on peut ne garder que le dernier, à savoir celui qui correspond à 2 décimales ;
- enfin, on peut rassembler dans une même structure `With ... End With` toutes les instructions qui commencent par le(s) même(s) mot-clé(s), avec « mise en évidence » de celui-ci (ceux-ci).

Après « épuration » (non indispensable) des instructions, il reste ceci :

```
Sub random()
' random Macro
' nombre aléatoire 2 décimales gras bleu
' Touche de raccourci du clavier: Ctrl+r
'
    ActiveCell.FormulaR1C1 = "=RAND()"
    With Selection
            .Font.Bold = True
            .Font.Color = -4165632
            .NumberFormat = "0.00"
    End With
End Sub
```

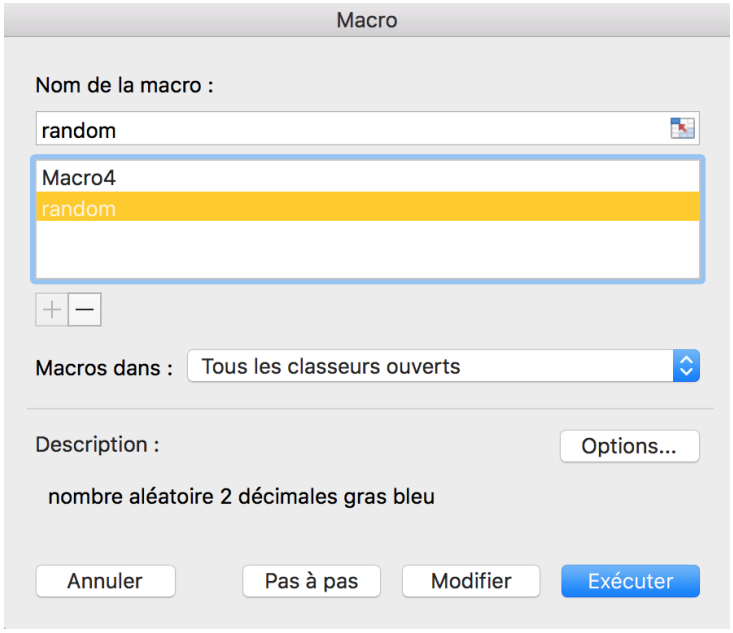
2.1.4. Exécuter la (une) macro

1 Par raccourci clavier

Le raccourci clavier est utile s'il y a peu de macros ; l'utilisateur doit en effet retenir les lettres associées à chaque macro... Attention, car certains raccourcis clavier sont déjà utilisés par le système d'exploitation (CTRL C, CTRL V, etc.), mais on en est prévenu lors du choix de la lettre.
Dans notre cas, clic dans une cellule vide, puis CTRL R.

2 Par son nom

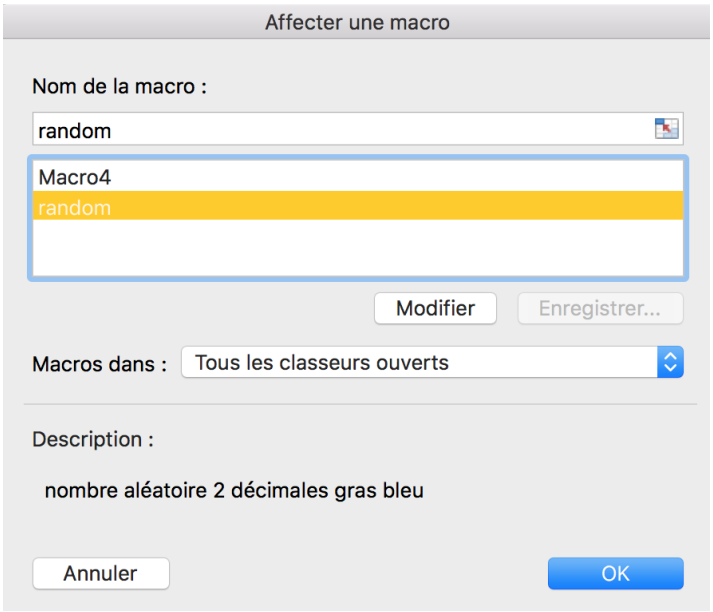
- `Outils > Macro > Macros... > choix de la macro > Exécuter` ; ou bien
- barre d'outils `Développeur` > icône 'Macro' ;

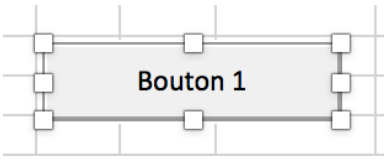


3 Par un bouton auquel on a assigné la macro

C'est la méthode la plus élégante.

- barre d'outils `Développeur` > icône 'Bouton'
- dessiner une petite zone rectangulaire dans la feuille
- choix de la macro à affecter > OK





On peut déplacer le bouton, modifier sa taille, son texte, etc.

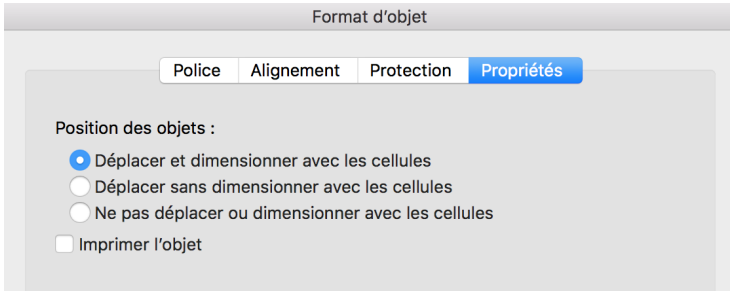
→ clic droit sur le bouton et choix dans le menu contextuel

Dans les versions plus anciennes d'Excel, l'icône permettant de créer un bouton est dans une barre d'outils appelée `Formulaires`, barre d'outils qu'il faut d'abord afficher à l'écran, évidemment ; la suite de la procédure est quasiment identique.

Propriétés du bouton

Par défaut, la taille et la position du bouton dépend de la taille et de la position de la cellule à proximité de laquelle il a été créé, et le bouton n'est pas imprimable. Pour modifier l'une ou l'autre de ces propriétés,

→ clic droit sur le bouton > `Format de contrôle...` > onglet 'Propriétés'



2.1.5. Macros toujours disponibles

Dans l'exemple qui précède, nous avons enregistré la macro dans « ce classeur », et par conséquent elle n'est disponible que dans ce classeur...

Si l'on souhaite avoir toujours sous la main certaines ou toutes les macros, il faut les enregistrer dans « le classeur de macros personnelles » ; celui-ci est créé automatiquement dès la première macro qui lui est destinée ; il s'appelle « personal.xlsb » et est ouvert automatiquement au démarrage d'Excel. Après une recherche via le Finder, j'ai trouvé le dit classeur dans
Library/Group Containers/UBF8T346G9.Office/Contenu utilisateur/Démarrage/Excel

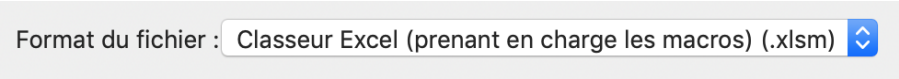
Suite au prochain numéro, et maintenant il ne reste plus qu'à s'entraîner !

Le tableur Excel et les macro-instructions (2)

Résumé. *Dans cette deuxième partie, nous aborderons la notion de variable, les instructions d'entrée-sortie (dialogues avec les cellules et l'utilisateur) et la création de fonctions personnalisées. Nous terminerons par une application simple et pratique : la gestion d'un journal de classe.*

1 Remarque importante

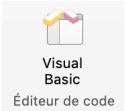
Lorsqu'un classeur va contenir des macros, il convient de l'enregistrer avec un format particulier : `Fichier > Enregistrer sous...` puis choisir comme ci-dessous.



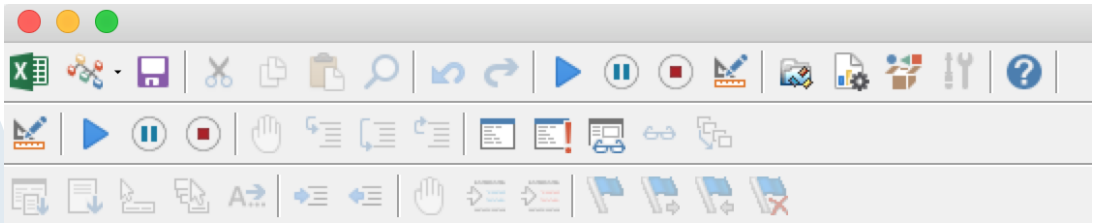
2 L'environnement VBA

Pour écrire du code *Visual Basic*, il faut être dans un environnement adapté. Pour y accéder,

- Clic sur le menu « développeur » pour afficher les outils qui seront nécessaires.
- La première icône « Visual basic - Éditeur de code » fait basculer dans cet environnement :



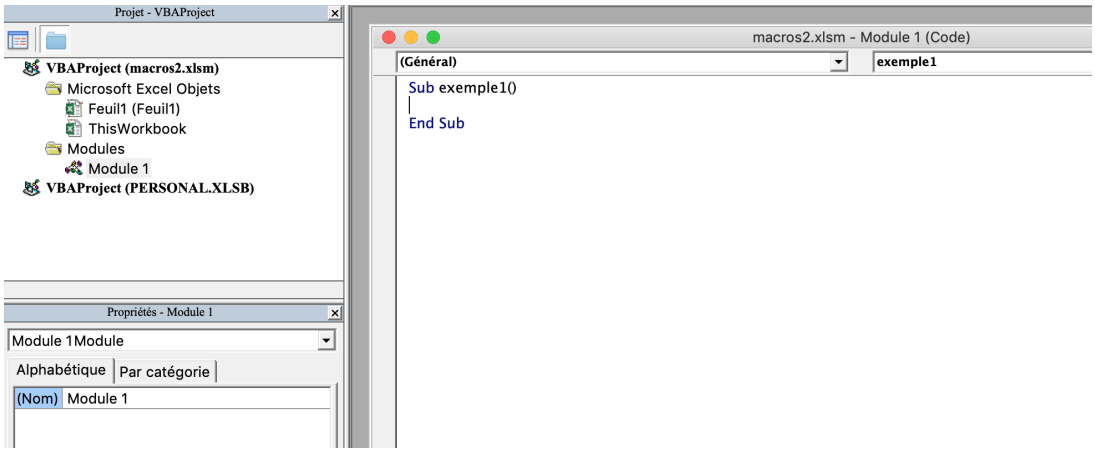
- En haut de la feuille « Microsoft Visual Basic » on trouve des barres d'outils :



- À gauche, une première fenêtre « Projet - VBA Project » où apparaîtront les feuilles de code, et sous cette fenêtre, une autre où l'on retrouve les propriétés [1] de l'élément sélectionné au-dessus (on ne s'en préoccupe pas pour l'instant).

- Les macros sont écrites dans des feuilles de type « Module » : `Insertion > Module`

- Une macro est encadrée par une structure `Sub ... End Sub` ; dans la feuille « Module 1 » qui vient d'être créée, on crée une macro en tapant l'instruction `sub exemple1` suivie de Enter. L'éditeur complète automatiquement l'instruction en ajoutant un couple de parenthèses, en ajustant la casse, et en clôturant la structure :



On peut mettre plusieurs macros dans le même module, les unes à la suite des autres.

3 Les variables

3.1 Nom, type, déclaration et affectation

Incontournables dans un langage de programmation, les variables servent à stocker une valeur pour un usage ultérieur (affichage écran, affectation à une cellule, etc.).

Une variable possède

- Un **nom** : il doit commencer par une lettre, ne peut pas contenir de point, ni d'espace, ni de caractères accentués, ni de caractères spéciaux (se contenter de lettres et de chiffres éventuels est une excellente idée) ni être un mot-clé du langage Visual Basic... (conseil d'ami : toujours taper un nom de variable en lettres minuscules, et si une majuscule apparaît, c'est qu'il s'agit d'un mot-clé).

- Un **type** : il est déterminé par le genre de contenu (texte, nombre entier, nombre réel, date, ...).

1. VBA est un langage de programmation qualifié de « orienté objet » ; y sont traitées des classes contenant des objets qui ont des propriétés et auxquels on applique des méthodes... il y a beaucoup de littérature à ce sujet.

- En début de macro, il est de bonne convenance de **déclarer** les variables qui seront utilisées (la place en mémoire sera réservée et de taille optimale, le programmeur aura une vision globale de ses besoins, et le code sera protégé en cas d'erreur de frappe — il sera impossible d'utiliser la variable `NON` si on a déclaré `NOM` —) ; la déclaration n'est pas obligatoire, mais tellement importante ; on peut la rendre obligatoire par l'instruction `Option Explicit` en tête de module, avant le texte de la première macro.

La déclaration explicite d'une variable (méthode la plus claire) se fait avec l'instruction

```
Dim nom_de_variable As type_de_variable
```

L'affectation d'une donnée à une variable se fait avec l'instruction

```
nom_de_variable = donnée
```

Dans le tableau qui suit, on décrit les principaux types de variable, ainsi que la plage de valeurs qu'elles peuvent contenir.

Type	Plages de valeurs
Byte	$0 \leqslant$ entier $\leqslant 255$
Boolean	true ou false ou vrai ou faux
Integer	$-32\,768 \leqslant$ entier $\leqslant 32\,767$
Long	$-2\,147\,483\,648 \leqslant$ entier $\leqslant 2\,147\,483\,647$
Single	$-3.402\,823E38 \leqslant$ réel $\leqslant -1.401\,298E-45$ pour les négatifs $1.401\,298E-45 \leqslant$ réel $\leqslant 3.402\,823E38$ pour les positifs
Double	$-1.797\,693\,134\,862\,31E308 \leqslant$ réel $\leqslant -4.940\,656\,458\,412\,47E-324$ pour les négatifs $4.940\,656\,458\,412\,47E-324 \leqslant$ réel $\leqslant 1.797\,693\,134\,862\,32E308$ pour les positifs
Date	$1/10/100 \leqslant$ date $\leqslant 31/12/9999$ $00{:}00{:}00 \leqslant$ heure $\leqslant 23{:}59{:}59$
String	chaîne de 0 à environ 2.10^9 caractère(s)
Variant	tout type

3.2 La boîte à message dans sa plus simple expression

Avant de donner quelques exemples de déclaration et d'affectation, il nous faut une instruction permettant d'afficher une donnée à l'écran. Voici la version sans option qui suffit pour l'instant ; la version complète sera traitée plus loin.

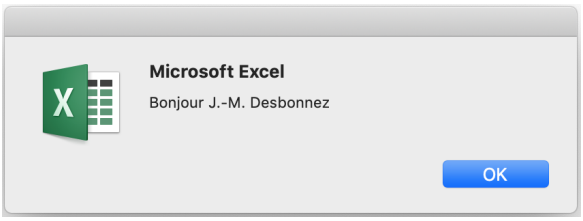
```
Msgbox nom_de_variable
Msgbox "texte du message"
```

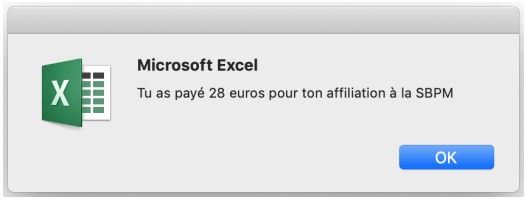
3.3 Exemples

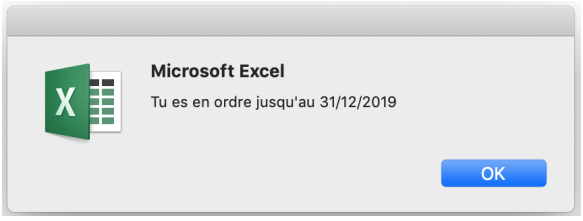
```
Option Explicit

Sub exemple1()
    MsgBox "Merci de lire cet article consacré aux macros avec Excel"

    Dim nom As String
    Dim prix As Byte
    Dim date_limite As Date

    nom = "J.-M. Desbonnez"
    prix = 28
    date_limite = #12/31/2019#

    MsgBox "Bonjour " & nom
    MsgBox "Tu as payé " & prix & " euros pour ton affiliation à la SBPM"
    MsgBox "Tu es en ordre jusqu'au " & date_limite
End Sub
```

Une chaîne de caractères est délimitée par des guillemets (c'est classique dans beaucoup de langages de programmation) ; une date est délimitée par le caractère # et le caractère & sert à concaténer deux chaînes de caractères.

Pour tester (exécuter) une macro dans laquelle se trouve le pointeur, on peut utiliser le bouton « Exécuter/Sub/UserForm » ▶ de la barre d'outils. Voici les quatre résultats correspondant aux quatre instructions **MsgBox** :







On verra par la suite comment compléter l'instruction `MsgBox` avec des options permettant de mettre un titre, et d'y incorporer d'autres boutons comme Annuler, Oui, Non, mais il faudra d'abord parler de la **structure alternative** pour traiter le résultat renvoyé par ces boutons.

4 Instructions de sélection

Dans l'usage du tableur, on manipule des classeurs composés de feuilles elles-mêmes composées de cellules. Toute activité sur ces objets doit être précédée d'une sélection de ceux-ci : clic sur un

onglet de feuille, clic sur une cellule, sélection d'une plage de cellules ; voici quelques sélections courantes :

- Sélectionner **une feuille**
  Clic sur l'onglet *Feuil1* : `Sheets("Feuil1").Select`

- Sélectionner **une cellule** (en mode absolu)
  Clic sur la cellule "E3" : `Range("E3").Select` ou `Cells(3,5).Select`
  Les deux instructions sont valables que l'on soit en mode d'affichage *A1* ou en mode d'affichage *R1C1* ; la seconde est toutefois plus adaptée aux calculs sur les adresses des cellules :

```
Dim i,j As Integer
i = 3
j = 5
Cells(i+1,j-2).Select
```

  Pour la suite, nous allons donc opter pour la seconde version, et, par voie de conséquence, rester logique en utilisant le mode d'affichage *R1C1*. Cela ne devrait pas poser de problème à un scientifique, puisque ce mode correspond aux coordonnées cartésiennes inversées.

- Sélectionner **une plage** (en mode absolu)
  Sélectionner la plage L2C3:L3C5 : `Range(Cells(2,3),Cells(3,5)).Select`
  Rappel : lorsqu'une plage est sélectionnée, c'est la cellule du coin supérieur gauche qui est *active*, donc ici la cellule L2C3.

- Sélectionner **une cellule** (en mode relatif)
  Sélectionner la cellule située 2 lignes plus bas et 4 colonnes à droite de la *cellule active* :
  `ActiveCell.Offset(2,4).Select`

- Sélectionner **une plage** (en mode relatif)
  Sélectionner une plage rectangulaire de 3 lignes et 5 colonnes à partir la cellule située 2 lignes plus bas et 4 colonnes à droite de la *cellule active* :
  `ActiveCell.Offset(2,4).Range(Cells(1,1), Cells(3,5)).Select`
  où `Range(Cells(1,1), Cells(3,5))` est à comprendre comme une plage **de la taille de** la plage allant de la cellule L1C1 à la cellule L3C5, c'est à dire 3 lignes et 5 colonnes.

5 Les instructions d'entrée-sortie

Ce sont les instructions qui permettent d'écrire et de lire dans une cellule, et de « communiquer » avec l'utilisateur.

5.1 Écrire une donnée dans une cellule active

```
ActiveCell.Formula = donnée
```

```
Dim nombre As Single
nombre = 3.14
ActiveCell.Formula = nombre
ActiveCell.Offset(1, 0).Select
ActiveCell.Formula = 33
ActiveCell.Offset(1, 0).Select
ActiveCell.Formula = "Pythagore"
ActiveCell.Offset(1, 0).Select
ActiveCell.Formula = #9/25/1957#
```

5.2 Écrire une donnée dans une cellule non active

```
adresse.Formula = donnée
```

```
Cells(5, 4).Formula = 33
```

5.3 Écrire une formule dans une cellule active

```
ActiveCell.FormulaR1C1 = "formule en mode LigneColonne"
```

```
ActiveCell.FormulaR1C1 = "=average(r[-10]c:r[-1]c)"
```

Une pause s'impose !
Cette formule calcule la moyenne des 10 cellules au-dessus de la cellule active. Après exécution de cette instruction, on trouve dans la cellule active la formule `=MOYENNE(L(-10)C:L(-1)C)` qui est celle que l'on devrait taper normalement dans la feuille (de calculs). Le mode *macro* nécessite donc le passage à la langue de Shakespeare, *average* pour moyenne, R pour *Row*, C pour *Column* —mais cà c'est chou vert et vert chou—, et pour couronner le tout, les parenthèses deviennent des crochets rectangulaires. Et l'enregistreur de macro traduit bien le français en anglais. Heureusement.

5.4 Écrire une formule dans une cellule non active

```
adresse.FormulaR1C1 = "formule en mode LigneColonne"
```

```
Cells(11,1).FormulaR1C1 = "=sum(r[-10]c:r[-1]c)"
Cells(15,3).FormulaR1C1 = "=pi()"
```

5.5 Lire le contenu d'une cellule

La lecture du contenu d'une cellule va de pair avec son affectation à une variable. Une variable déclarée *Integer* et recevant un contenu de type *String* provoquera un message d'erreur et l'arrêt immédiat de la macro. Si on est certain du type de contenu de la cellule, on déclare une variable de ce type ; en cas d'incertitude, il faut la déclarer de type *Variant*. Selon que la cellule soit active ou non,

```
variable = ActiveCell
variable = Cells(...,...)
```

Si la cellule contient une formule, c'est le **résultat** qui est retourné. Si l'on souhaite récupérer la formule, il faut utiliser

```
variable = ActiveCell.Formula
variable = Cells(...,...).Formula
```

avec, au passage, une traduction en anglais.

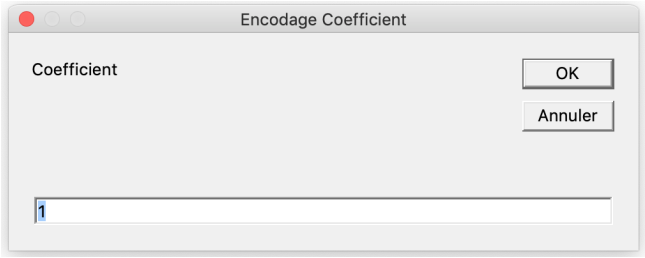
5.6 Saisie interactive

La saisie interactive consiste à affecter à une cellule ou à une variable un contenu encodé par l'utilisateur via une boîte de dialogue *simple*. Dans un futur article, on verra comment gérer une boîte de dialogue complexe contenant à la fois des zones de texte, des listes déroulantes, des boutons de choix exclusifs, etc.

```
variable = InputBox("message","titre",valeur par défaut, ...)
cells(...,...) = InputBox("message","titre",valeur par défaut, ...)
ActiveCell = InputBox("message","titre",valeur par défaut, ...)
```

"message" est la question posée à l'utilisateur (il doit évidemment savoir ce qu'on lui demande) ; c'est le seul argument obligatoire. Il y en a six autres, la syntaxe complète apparaît à l'écran lors de la frappe de l'instruction. Deux boutons traditionnels OK et Annuler sont automatiquement présents dans la boîte. Comme on peut s'y attendre, OK valide l'encodage, et Annuler renvoie une chaîne vide.

```
Cells(14, 1) = InputBox("Coefficient","Encodage Coefficient", 1)
```



5.7 La boîte à message

Nous l'avons déjà rencontrée dans sa plus simple expression (page 173). En voici une version plus fournie en options. L'une des options consiste à choisir quels boutons seront présents dans la boîte : OK, Annuler, Oui, Non, et pas mal d'autres possibilités qu'il serait trop long de détailler ici. Pour les curieux, toutes ces options apparaissent dans une info-bulle lors de la frappe de l'instruction ; il faut prendre du temps pour les tester.

Le clic sur l'un des boutons **renvoie une valeur numérique**, qui est affectée à une variable, et que l'on doit tester pour décider de la suite à prendre. Impossible donc de retarder plus longtemps l'utilisation d'une structure de programmation classique : l'*alternative simple*.

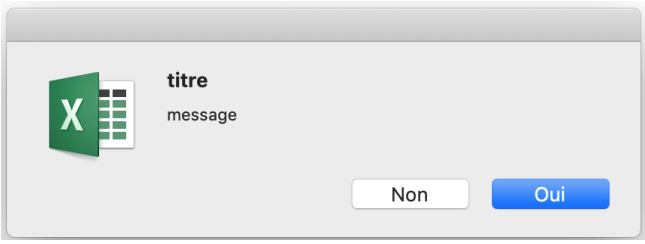
La syntaxe générale est

```
variable = MsgBox("message",type de boîte,"titre",...)
```

Comme *Type de boîte*, il y a entre autres *vbOkCancel*, *vbOkOnly*, *vbYesNo*, *vbYesNoCancel*, etc. les noms sont suffisamment explicites.

```
Option Explicit
Sub message()

Dim retour As Byte
retour = MsgBox("message", vbYesNo, "titre")
' MsgBox retour
If retour = 6 Then
    MsgBox ("vous avez choisi OUI")
Else
    If retour = 7 Then
        MsgBox ("vous avez choisi NON")
    End If
End If

End Sub
```

L'affichage du contenu de la variable *retour* permet de constater que Oui renvoie la valeur 6 et Non renvoie la valeur 7. On s'est contenté dans l'exemple d'afficher le choix de l'utilisateur. On peut aussi dans chacun des cas appeler une autre macro, ce que nous ne manquerons pas de faire dans des exemples plus sophistiqués, mais il nous faut d'abord passer par d'autres structures de programmation.



6 Création de fonctions personnalisées

À toute la panoplie des fonctions d'Excel, il est possible d'ajouter ses propres fonctions.

Une fonction est une macro dont les instructions sont encadrées par une structure

`Function(...) … End Function`

Entre les parenthèses, on déclare les différents *arguments* (éventuels) nécessaires à la fonction pour produire un résultat. Le résultat retourné doit être affecté à une variable dont le nom est celui de la fonction.

Voici une fonction appelée *hypotenuse* qui calcule la longueur de l'hypoténuse d'un triangle rectangle dont on donne la longueur des deux côtés de l'angle droit. Les deux longueurs sont supposées être des nombres positifs, et dans le cas contraire, c'est un message d'erreur qui sera retourné.

```
Option Explicit
Function hypotenuse(x As Single, y As Single)
    If (x <= 0 Or y <= 0) Then
        hypotenuse = "Erreur"
    Else
        hypotenuse = Sqr(x * x + y * y)
    End If
End Function
```

Après création de cette macro, la fonction *hypotenuse* peut être utilisée dans des instructions du type

```
=hypotenuse(3;7)
=hypotenuse(LC[-2];LC[-1])
=hypotenuse(A2;B2)
```

À priori, une fonction n'est disponible que dans le classeur dans lequel elle a été créée. Il est toutefois possible de la rendre utilisable dans n'importe quel classeur, via une procédure d'enregistrement (du classeur dans lequel on a créé la (les) fonction(s)) en tant que *Complément d'Excel*. Voici la procédure à suivre :

- Fichier > Enregistrer sous…
  Format de fichier : Complément Excel (.xlam)

  Choisir un dossier, et donner un nom explicite, par exemple `MesFonctions`

- Outils > Compléments Excel… > parcourir…
- Sélectionner `MesFonctions.xlam` > Ouvrir > OK

Un exemple de fonction sans argument

On connaît la fonction `PI()` ; voici la fonction `e()` :

```
Function e()
    e = 2.71828182845905
End Function
```

7 Une macro peut en appeler une autre

Méthode classique en programmation : « il faut diviser pour mieux régner ».

Il est en effet plus simple de décomposer un problème complexe en plusieurs problèmes plus simples, ce qui permet par exemple à plusieurs personnes de traiter chacune une partie.

L'instruction d'appel d'une macro est simplement... son nom.

```
Sub macro_principale()
    encoder_donnees
    calculer
    imprimer
End Sub

Sub encoder_donnees()
    ...
End Sub

Sub calculer()
    ...
End Sub

Sub imprimer()
    ...
End Sub
```

8 Domaine de validité et durée de vie des variables

Il y a trois domaines de validité pour les variables :

1. la macro ou la fonction (variables *locales*) ;
2. le module (variables *globales*) ;
3. l'application (variables *publiques*).

La durée de vie est la période pendant laquelle la variable conserve sa valeur.

Variables locales

Elles sont déclarées au niveau de la macro ou de la fonction au moyen de l'instruction `Dim` et n'ont de durée de vie que dans cette macro ou fonction (lorsque la macro se termine —`End Sub`— la variable est « oubliée »).

Variables globales

Elles sont déclarées dans le module, AVANT toute macro ou fonction, également au moyen de l'instruction `Dim`. Elles sont par conséquent accessibles dans toutes les macros ou fonctions du module, et les contenu peut y être modifié. Elles ont comme durée de vie toutes les macros ou fonction d'un module, et sont « oubliées » lors de l'exécution d'une macro ou fonction d'un autre module.

```
Option Explicit

Sub principale()
   Dim x As Integer
   x = 10
   ajouter
   MsgBox ("x= " & x)
End Sub

Sub ajouter()
    Dim x As Integer
    x = x + 1
End Sub
```

```
Option Explicit
Dim x As Integer

Sub principale()
   x = 10
   ajouter
   MsgBox ("x= " & x)
End Sub

Sub ajouter()
    x = x + 1
End Sub
```

Dans la situation de gauche, le contenu de x n'aura pas changé et vaudra toujours 10 ; dans celle de droite, x vaudra 11.

Variables publiques

Un programme peut parfois être réparti sur plusieurs feuilles de type *Module*. Si variables doivent être « partagées » par des macros de plusieurs modules, il faut les déclarer au moyen de l'instruction

```
Public nom_variable As Type_variable
```

Remarque

Les macros et fonctions sont publiques par défaut, ce qui explique que lors de l'affectation d'un bouton à une macro, on accède à toutes les macros de tous les modules. Les macros peuvent par conséquent être utilisées dans d'autres modules sans qu'il soit nécessaire de spécifier le mot *Public*.

Par contre, si l'on souhaite créer une macro dont la durée de vie se limite au seul module dans lequel elle a été créée, il faut la déclarer par l'instruction

```
Private sub nom_macro()
```

9 Application : gestion d'un journal de classe

Lorsque j'étais encore en activité (d'enseignement), on devait tenir à jour un « journal de classe », un cahier où il fallait renseigner, pour chaque classe et à chaque date de cours, le sujet de la leçon. En cas d'inspection, c'était un document justificatif important à montrer, mais aussi un document de travail personnel permettant de se rappeler des notions vues, des devoirs donnés, de voir l'évolution, bref, tout ce qui peut concerner un cours.

Voici venu le temps des nouvelles technologies ; renseignements pris, il semblerait que beaucoup d'écoles optent pour une application « plateforme numérique » qui gère aussi le journal de classe des enseignants... mais on fera quand-même l'exercice !

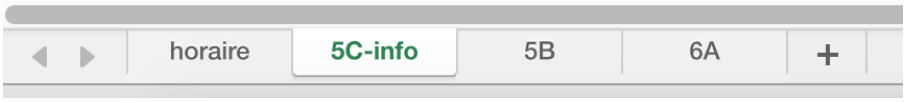
L'idée est la suivante : dans la première feuille du classeur que nous nommerons `horaire`, on reproduit l'horaire hebdomadaire dans lequel sont renseignés les jours, les heures, ainsi que les

intitulés de cours et/ou de classes. En cliquant sur un bouton, on voudrait activer la feuille correspondant à la classe ou au cours, avec, ou non, l'insertion automatique de la date du jour au bon endroit, et activation de la bonne cellule pour y écrire le sujet du cours.

	Lundi	Mardi	Mercredi	Jeudi	Vendredi
08:30 - 09:10		6A			
09:10 - 10:00		6A		5C-info	
10:20 - 11:10			5B		
11:10 - 12:00					
13:30 - 14:20					
...					

Dans l'application, seuls les contenus des cellules avec les cours et/ou les classes sont importants : ils ne peuvent pas contenir d'espace ; c'est la seule restriction. Peu importe l'endroit où se trouve le tableau, peu importe les mises en forme, bordures, couleurs, police, textes des jours, des heures.

Il faudra également une feuille pour chaque cours/classe différent, en l'occurrence pour l'exemple ci-dessus, une feuille `6A`, une feuille `5B`, et une feuille `5C-info`. Il est impératif que le nom de la feuille soit absolument identique au texte du cours/classe dans la cellule de l'horaire. Par facilité, ces « préparatifs » peuvent être faits manuellement, au préalable.



On s'est limité au minimum nécessaire pour le contenu de chaque feuille ; l'essentiel est que l'on se souvienne de l'adresse de la cellule qui contiendra la première date, à savoir ici `L3C1`.

	1	2
1	*5C-info*	
2	*Date*	*Sujet*
3		
4		
5		

Toutefois, pour les mordus de l'automatisation, on détaillera la mise au point d'une macro pour créer une feuille correspondant au contenu de la cellule active de l'horaire, et en y inscrivant le minimum comme le montre la figure ci-avant.

9.1 Les macros « lire » et « horaire »

La macro « lire » sélectionne la feuille correspondant au cours/classe de la cellule active dans l'horaire. On lui affectera un bouton lire dans la feuille « horaire »[3]. Il suffira donc de sélectionner une cellule dans l'horaire puis de cliquer sur ce bouton. La cellule active dans l'horaire ne peut pas être vide, d'où le test.
La fonction `IsEmpty(...)` est une fonction booléenne d'information permettant de savoir si un contenu est vide.
La macro « horaire » sélectionne la feuille du même nom. On lui affectera un bouton horaire dans chaque feuille du journal de classe.

```
Option Explicit

Sub lire()
    Dim nom_feuille As String
     If IsEmpty(ActiveCell) Then
        MsgBox ("Erreur cellule vide")
    Else
        nom_feuille = ActiveCell
        Sheets(nom_feuille).Select
    End If
End Sub

Sub horaire()
    Sheets("horaire").Select
End Sub
```

9.2 La macro « ecrire »

La macro « ecrire » sélectionne la feuille correspondant au cours/classe de la cellule active dans l'horaire. Elle y recherche la première cellule libre (vide) dans la colonne 1 pour y écrire la date du jour, et sélectionne la cellule correspondant de la colonne 2 pour permettre l'encodage du sujet du cours.
On lui affectera un bouton écrire dans la feuille « horaire ».

```
Sub ecrire()
    Dim k As Byte
    lire
    k = 3
    While Not (IsEmpty(Cells(k, 1)))
        k = k + 1
    Wend
    Cells(k, 1) = Date
    Cells(k, 2).Select
End Sub
```

Il a été indispensable d'utiliser une structure répétitive du type `Tant que condition faire...` qui se traduit en VBA par `While condition ... Wend`.

La troisième partie sera consacrée aux structures de programmation.

9.3 La macro « creer_feuille »

La macro « creer_feuille » crée une feuille du journal de classe en lui donnant pour nom le texte de la cellule active de l'horaire. Elle y écrit le contenu de trois cellules, avec une mise en forme minimale de ces cellules. On peut bien évidemment lui affecter un bouton dans la feuille « horaire ».

Pour les instructions de création de feuille et de mises en forme, il suffit d'utiliser l'enregistreur de macros [3] pour en connaître la syntaxe.

```
Sub creer_feuille()
   Dim nom_feuille As String
    If IsEmpty(ActiveCell) Then
        MsgBox ("Erreur cellule vide")
    Else
        nom_feuille = ActiveCell
        Sheets.Add After:=ActiveSheet
        ActiveSheet.Name = nom_feuille
        Cells(1, 1) = nom_feuille
        Cells(2, 1) = "Date"
        Cells(2, 2) = "Sujet"
        Cells(1, 2).ColumnWidth = 80
        Range(Cells(1, 1), Cells(2, 2)).Select
        With Selection.Font
            .HorizontalAlignment = xlCenter
            .Name = "Calibri"
            .FontStyle = "Gras"
            .Size = 14
            .Color = 255
        End With
        Cells(3, 1).Select
        Sheets("horaire").Select
    End If
End Sub
```

Références

[1] Desbonnez J.-M., Le tableur Excel et les macros-instructions (1). *Losanges*, 43, pp. 56–63., 2018.

Le tableur Excel et les macro-instructions (3)

Résumé. *Cette troisième partie est consacrée aux structures de programmation (alternatives et répétitives), classiques dans tout langage moderne. On les a déjà rencontrées longuement dans des articles consacrés au langage* Ruby *(Losanges 28 à 33), et VBA n'y échappe pas. Impossible de s'en passer, et certaines ont déjà été utilisées discrètement dans les exemples de [3] et [4]. Voici une liste plus complète et des exemples variés.*

1 Les structures alternatives

1.1 Alternatives simples

Elles sont de deux formes, selon la présence ou non de l'option *sinon*.

Exemple 1

Si la taille de police de la cellule active est 12 pt, la mettre en 16 pt.

```
Si condition alors
    instruction(s)
Fin
```

```
Sub taille()
    If ActiveCell.Font.Size = 12 Then
        ActiveCell.Font.Size = 16
    End If
End Sub
```

Seules les polices de taille 12 seront transformées en 16.

Exemple 2

Si la taille de police de la cellule active est 12 pt, la mettre en 16 pt, sinon en 10 pt.

```
Si condition alors
    instruction(s)
Sinon
    instruction(s)
Fin
```

```
Sub taille()
    If ActiveCell.Font.Size = 12 Then
        ActiveCell.Font.Size = 16
    Else
        ActiveCell.Font.Size = 10
    End If
End Sub
```

Les polices de taille 12 seront transformées en 16 et toutes les autres en 10.

Exemple 3

Un peu plus mathématique, avec deux alternatives imbriquées : les cellules `A1`, `B1` *et* `C1` *contiennent respectivement les trois coefficients* `a`, `b` *et* `c` *d'une équation du second degré* $ax^2+bx+x=0$. *La cellule* `D1` *contiendra le discriminant,* `E1` *et* `F1` *les solutions réelles, si toutefois elles existent.*

```
Option Explicit
Sub degre2()
   Dim a, b, c, delta, As Single
   a = Cells(1, 1) : b = Cells(1, 2) : c = Cells(1, 3)
   delta = b * b - 4 * a * c
   Cells(1, 4) = delta
   If delta > 0 Then
        Cells(1, 5) = (-b + Sqr(delta)) / (2 * a)
        Cells(1, 6) = (-b - Sqr(delta)) / (2 * a)
    Else
        If delta = 0 Then
            Cells(1, 5) = -b / (2 * a)
        Else
            Cells(1, 5) = "pas de solution réelle"
        End If
    End If
End Sub
```

L'indentation [1] des instructions n'est pas obligatoire, mais rend le code plus lisible. On peut mettre plusieurs instructions sur une même ligne en les séparant par un double-point. Ci-dessous une autre version qui utilise le mot-clé *elseif* (`sinon si`).

```
Option Explicit
Sub degre2()
   Dim a, b, c, delta, As Single
   a = Cells(1, 1): b = Cells(1, 2): c = Cells(1, 3)
   delta = b * b - 4 * a * c
   Cells(1, 4) = delta
   If delta > 0 Then
            Cells(1, 5) = (-b + Sqr(delta)) / (2 * a)
            Cells(1, 6) = (-b - Sqr(delta)) / (2 * a)
    ElseIf delta = 0 Then
            Cells(1, 5) = -b / (2 * a)
    Else
            Cells(1, 5) = "pas de solution réelle"
    End If
End Sub
```

1. Retrait par rapport à la marge de gauche.

Les structures répétitives permettront de traiter facilement plusieurs lignes de triplets de coefficients et de résoudre ainsi en un seul clic (celui sur le bouton associé à la macro) autant d'équations du second degré... qu'il y a d'élèves dans la classe.

1.2 Alternative multiple

Lorsqu'il y a plus de trois cas à envisager, l'utilisation d'alternatives simples imbriquées devient lourde. L'alternative multiple permet de clarifier et d'alléger le code.

```
Selon que expression
    cas condition_1
        instruction(s)1
    cas condition_2
        instruction(s)2
    ...
    cas condition_n
        instruction(s)n
    [autres cas]
        autre(s) instruction(s)
Fin
```

Les conditions sont traitées dans l'ordre de leur écriture : si la première n'est pas vérifiée, on passe à la deuxième, et ainsi de suite.

Dès qu'une condition est vérifiée, les instructions associées sont exécutées et on sort de la structure.

Si aucune condition n'est vérifiée, ce sont les instructions spécifiées dans l'option `[autres cas]` qui sont exécutées.

L'alternative multiple peut en fait remplacer toutes les autres formes d'alternatives ; son exécution est même un peu plus rapide, ce qui peut être une raison de la préférer pour ceux qui courent derrière les millisecondes...

Exemple 1

On suppose que la cellule A1 contienne un nombre dans l'intervalle $[0,20]$ correspondant au résultat d'une interrogation. La cellule A2 devra contenir une appréciation calculée selon la tableau suivant :

Résultat	Appréciation
$\in [0,6[$	Nettement insuffisant
$\in [6,10[$	Insuffisant
$\in [10,12[$	Satisfaisant
$\in [12,16[$	Bien
$\in [16,18[$	Très bien
$\in [18,20]$	Excellent

Dans le dernier cas on mettra l'arrière-plan de la cellule en vert.

```
Option Explicit
Sub appreciation()
Dim cote As Single
cote = Cells(1, 1)
Select Case cote
    Case Is < 6
        Cells(1, 2) = "Nettement insuffisant"
     Case Is < 10
        Cells(1, 2) = "Insuffisant"
    Case Is < 12
        Cells(1, 2) = "Satisfaisant"
    Case Is < 16
        Cells(1, 2) = "Bien"
    Case Is < 18
        Cells(1, 2) = "Très bien"
    Case Else
        Cells(1, 2) = "Excellent"
        Cells(1, 2).Interior.Colorindex=10
End Select
End Sub
```

Même remarque qu'à la page 187 si l'on doit traiter les résultats de tous les élèves d'une classe...

Remarques

Il existe aussi d'autres types de tests sur les nombres. Après une instruction `Select Case x`,

Pour traduire	On écrit
si $x = 1$	`Case 1`
si $x = 1$ ou $x = 3$ ou $x = 5$	`Case 1,3,5`
si $x \in [10, 20]$	`Case 10 To 20`

Exemple 2 : rien que pour le plaisir...

L'instruction Excel `=ent(6*alea())+1` calcule un nombre naturel dans l'intervalle [1,6] et permet de simuler ainsi le jet d'un dé. Voici une fonction personnalisée qui va engendrer `un, deux, trois, quatre, cinq, six` en lieu et place de `1, 2, 3, 4, 5, 6`.

On a vu dans [3] que l'on peut utiliser l'enregistreur de macros pour traduire des instructions en VBA. Pour l'instruction ci-dessus on obtient `=int(6*rand())+1`.

Mais... cette instruction n'est pas reconnue dans la macro. Sur le site [3] (c'est le premier que j'ai trouvé) on trouve une liste des instructions VBA, et on y apprend qu'il faut utiliser `rnd()` pour le calcul d'un nombre aléatoire.

```
Option Explicit
Function de() As String
    Select Case Int(6 * Rnd()) + 1
        Case 1
            de = "un"
        Case 2
            de = "deux"
        Case 3
            de = "trois"
        Case 4
            de = "quatre"
        Case 5
            de = "cinq"
        Case Else
            de = "six"
    End Select
End Function
```

Le lecteur trouvera dans [4] la manière de créer une fonction personnalisée. Pour utiliser celle que l'on vient de créer, il suffit de taper dans une cellule l'instruction `=de()` ; elle est copiable dans d'autres cellules.

Il est vrai que c'est d'une utilité assez limitée, mais on avait prévenu, rien que pour le plaisir... et la création d'une fonction personnalisée avec une alternative multiple.

Des tests pour le traitement des chaînes de caractères sont aussi disponibles. Le lecteur intéressé ne manquera pas de se renseigner... sur le net par exemple.

2 Les structures répétitives

Comme leur nom l'indique, ces structures permettent d'exécuter *plusieurs fois* une ou plusieurs instructions. On peut les classer en plusieurs catégories, selon que

- le programmeur connaît à l'avance le nombre d'exécutions nécessaires (répétitive automatique) ;
- le programmeur ne le connaît pas mais il faut au moins une exécution (répétitive à test d'arrêt final) ;

- le programmeur ne le connaît pas mais il ne faut pas nécessairement une exécution (répétitive à test d'arrêt initial).

Le puriste dira que toutes peuvent se ramener à la dernière catégorie, mais on peut aussi profiter des structures disponibles pour rendre le code plus clair.

LA règle d'or : une répétitive DOIT pouvoir s'arrêter, et donc les instructions qui seront répétées doivent en contenir au moins une qui peut rendre VRAIE la condition d'arrêt (excepté pour la répétitive automatique, où le *pas* doit permettre à la *valeur initiale* d'atteindre la *valeur finale*).

2.1 Répétitive automatique

```
Pour compteur de début à fin [par pas de ...]
    instruction(s)
Fin
```

Lorsque le pas est omis, il vaut 1.
Les fantaisies suivantes sont permises :
de 1 à 15 par 0.5
de 10 à 5 par −1
de 17 à 34

Exemple

Reprenons l'exemple de la page 186 et résolvons 19 (2) équations du second degré ; on supposera que les trois coefficients respectifs occupent les trois premières colonnes des dix-neuf premières lignes. On fera donc varier un compteur de lignes de 1 à 19, tout naturellement.

2. C'est le nombre du jour.

```
Option Explicit
Public k
Dim k As Integer

Sub equations19() ' macro principale
    For k = 1 To 19
        degre2
    Next
End Sub

Sub degre2() ' macro appelée 19 fois
   Dim a, b, c, delta, As Single
   a = Cells(k, 1): b = Cells(k, 2): c = Cells(k, 3)
   delta = b * b - 4 * a * c
   Cells(k, 4) = delta
   If delta > 0 Then
        Cells(k, 5) = (-b + Sqr(delta)) / (2 * a)
        Cells(k, 6) = (-b - Sqr(delta)) / (2 * a)
    ElseIf delta = 0 Then
        Cells(k, 5) = -b / (2 * a)
    Else
        Cells(k, 5) = "pas de solution réelle"
    End If
End Sub
```

Pour illustrer l'appel d'un sous-programme ([4]), on a remplacé dans la macro *degre2* (qui traite une seule équation) l'indice ligne de l'adresse des cellules (1) par le compteur (k).

La variable k a été déclarée *publique* car elle doit être connue dans les deux macros.

Cas particulier

```
Pour chaque objet dans Collection
    instruction(s)
Fin
```

Exemple 1 (des cellules dans une plage) : *ajouter 10% à toutes les cellules d'une plage présélectionnée, à condition que la cellule ne soit pas vide (sinon contenu considéré comme nul) et que la cellule contienne un nombre (sinon message d'erreur, on ne fait pas de calcul avec du texte).*

```
Option Explicit
Sub dixpourcent()
Dim cellule As Range
    For Each cellule In Selection
        If IsNumeric(cellule) And Not (IsEmpty(cellule)) Then
            cellule = cellule * 1.1
        End If
    Next
End Sub
```

La macro fonctionne aussi si les cellules de la sélection ne sont pas contigües.

Si la macro doit s'appliquer toujours à une même plage, on peut utiliser une instruction comme `For Each cellule In Range(Cells(1, 1), Cells(3, 5))`.

Exercice : *on a une série de nombres correspondant à des résultats d'interrogation. Écrire une macro qui met en rouge les cotes inférieures à leur moyenne.*

Exemple 2 (des feuilles dans un classeur) : *écrire un nom dans le coin supérieur gauche de toutes les feuilles du classeur, à l'exception de la feuille* `Feuil2`.

```
Option Explicit
Sub nom_dans_feuilles()
    Dim feuille As Worksheet
    For Each feuille In Worksheets
        If feuille.Name <> "Feuil2" Then
            feuille.Cells(1, 1) = "Jean-Marc Desbonnez"
        End If
    Next
End Sub
```

On peut aussi spécifier les feuilles à traiter (s'il n'y en a pas trop) au moyen d'une instruction du type `For Each feuille In Sheets(Array("truc", "machin", "chouette"))`.

2.2 Répétitive à test d'arrêt final

```
Répéter
    instruction(s)
Jusqu'à ce que condition
```

ou

```
Répéter
    instruction(s)
Tant que condition
```

Avec ce type de répétitive, on est certain que les instructions seront exécutées au moins une fois puisque le test d'arrêt a lieu après la première exécution.

Exemple 1 : validation de donnée (un nombre dans un intervalle fixé)

L'utilisateur doit encoder un nombre dans l'intervalle $[0,20]$... *et rien d'autre. Un message d'erreur sera affiché si l'encodage ne correspond pas à ce qui est demandé.*

Il faudra tester si ce qui est encodé est bien un nombre d'une part (fonction `IsNumeric(...)`), et compris dans le bon intervalle d'autre part ; tout le reste est rejeté.

```
Option Explicit
Sub validation_nombre()
    Dim x As Variant
    Do
        x = InputBox("nombre entre 0 et 20")
        If Not (IsNumeric(x) And x >= 0 And x <= 20) Then
            MsgBox ("erreur d'encodage")
        End If
    Loop Until IsNumeric(x) And x >= 0 And x <= 20
End Sub
```

La variable *x* doit être déclarée comme `variant` (type quelconque) car on ne sait pas à l'avance ce que va encoder l'utilisateur.

On peut aussi utiliser l'autre variante de cette répétitive, qui nécessite de prendre la négation du test d'arrêt :

```
Option Explicit
Sub validation_nombre()
    Dim x As Variant
    Do
        x = InputBox("nombre entre 0 et 20")
        If Not (IsNumeric(x) And x >= 0 And x <= 20) Then
            MsgBox ("erreur d'encodage")
        End If
    Loop While Not( IsNumeric(x) And x >= 0 And x <= 20)
End Sub
```

Chacun fait son lit comme il se couche...

Exemple 2 : validation de donnée (chaîne de caractères)

L'utilisateur doit encoder O (pour oui) ou N (pour non) et rien d'autre. Il faudra accepter la version minuscule et la version majuscule.

La fonction `Left(x,1)` prélève le premier caractère à gauche de la variable *x* (au cas ou l'utilisateur encode un mot de plus d'une lettre) ; la fonction `UCase(...)` convertit en majuscule (*Upper case*), ce qui permet de ne pas devoir tester un encodage en minuscule (on gagne deux comparaisons).

```
Option Explicit
Sub ouinon()
    Dim x As Variant
    Do
        x = InputBox("O (oui) ou N (non)")
        x = UCase(Left(x, 1))
        If Not (x = "O" Or x = "N") Then
            MsgBox ("erreur d'encodage")
        End If
    Loop Until (x = "O" Or x = "N")
    If x="O" then
        'traitement du oui
    Else
        'traitement du non
    End If
End Sub
```

2.3 Répétitive à test d'arrêt initial

```
Tant que condition
    instruction(s)
Fin
```

Avec ce type de répétitive, si la condition d'arrêt est vérifiée dès le début, les instructions ne sont pas exécutées.

Exemple : la suite de Syracuse [3]

La dite suite est définie comme suit : $u_0 \in \mathbb{N}^*$ et $\begin{cases} u_{n+1} = \frac{u_n}{2} \text{ si } u_n \text{ est pair} \\ u_{n+1} = 3u_n + 1 \text{ si } u_n \text{ est impair} \end{cases}$

Après avoir atteint le nombre 1, les valeurs 4, 2, 1 se répètent indéfiniment (conjecture).

En ce qui nous concerne, nous allons engendrer cette suite dans des cellules consécutives jusqu'à obtenir la première valeur 1. Mais si $u_0 = 1$, c'est terminé avant de commencer. On supposera que la cellule A1 contienne u_0, naturel non nul.

```
Option Explicit
Sub syracuse()
    Dim ligne, terme, suivant As Integer
    ligne = 1
    terme = Cells(1, 1)
    Do While terme <> 1
        If terme Mod 2 = 0 Then
            suivant = terme / 2
        Else
            suivant = 3 * terme + 1
        End If
        ligne = ligne + 1
        Cells(ligne, 1) = suivant
        terme = Cells(ligne, 1)
    Loop
End Sub
```

La fonction Mod 2 calcule le reste de la division modulo 2 ; si ce reste est nul, c'est que le nombre est pair.

On associe souvent deux nombres à cette suite :

- la **durée de vol** : le plus petit indice n tel que $u_n = 1$ (en ce qui nous concerne, la dernière valeur de l'indice *ligne*) ;
- l'**altitude** : la valeur maximale de la suite.

Pour le calcul du maximum d'une suite, on suppose que le premier terme est le maximum, et dès qu'un terme le dépasse, c'est lui qui devient le maximum.

3. Ou suite de Collatz ou suite d'Ulam ou suite $3x+1$.

```
Option Explicit
Sub syracuse2()
    Dim ligne, terme, suivant, altitude As Integer
    ligne = 1
    terme = Cells(1, 1)
    altitude = terme
    Do While terme <> 1
        If terme Mod 2 = 0 Then
            suivant = terme / 2
        Else
            suivant = 3 * terme + 1
        End If
        ligne = ligne + 1
        Cells(ligne, 1) = suivant
        terme = Cells(ligne, 1)
        If terme > altitude Then
            altitude = terme
        End If
    Loop
    Cells(1, 2) = "durée vol": Cells(1, 3) = ligne
    Cells(2, 2) = "altitude": Cells(2, 3) = altitude
End Sub
```

Attention

En vertu de la règle d'or qui précise qu'une répétitive doit s'arrêter, il est indispensable que la cellule `A1` contienne bien un nombre naturel non nul. Elle ne peut même pas être vide !

Par précaution, il vaut donc mieux démarrer la macro en mettant un tel nombre (par exemple 1) dans la cellule `A1`. À bon entendeur, salut !

Tant qu'à faire, on peut également en profiter pour effacer le contenu de la première colonne si on a l'intention de faire plusieurs essais.

On commencera donc la macro par les instructions

```
Columns("A:A").Select
Selection.Clear
Cells(1,1)=1
```

Dans [4] à la page 56, il y a un autre exemple d'une répétitive à test d'arrêt initial, qui permet de trouver la première cellule vide dans une colonne.

On y utilise une autre syntaxe (très voisine) de ce type de répétitive : `While condition ... Wend`, parfaitement équivalente.

3 Complément : mise en forme conditionnelle

Lorsqu'on applique une mise en forme à une cellule (par exemple texte en rouge si le contenu est strictement inférieur à 10), la suppression (au moyen de la touche *Delete*) ou la modification du contenu ne supprime pas la mise en forme de celle-ci.

Par conséquent, si un 9 (rouge) devient 19 après correction d'une erreur d'encodage par exemple, le 19 reste en rouge.

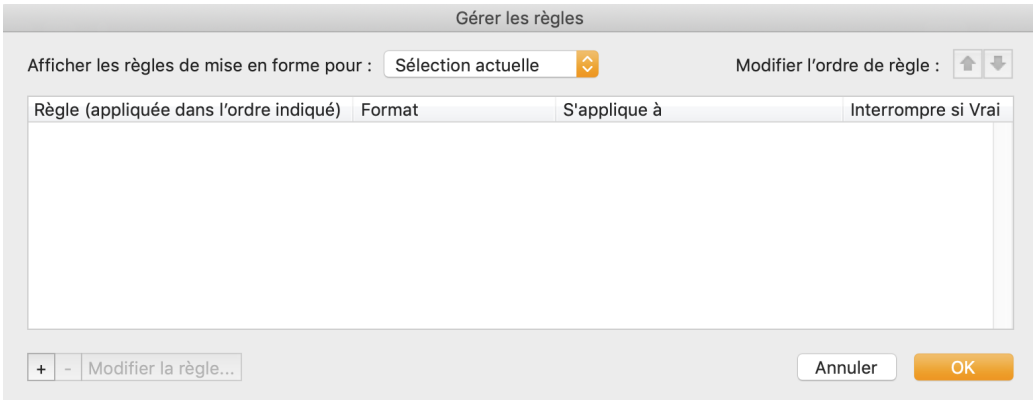
Pour supprimer la mise en forme d'une ou plusieurs cellules, il faut, après sélection, passer par le menu `Edition > Effacer > Formats...`

La mise en forme conditionnelle permet de pallier à cet inconvénient, puisque la mise en forme conditionnelle est attachée au contenu de la cellule et non à l'adresse de celle-ci.

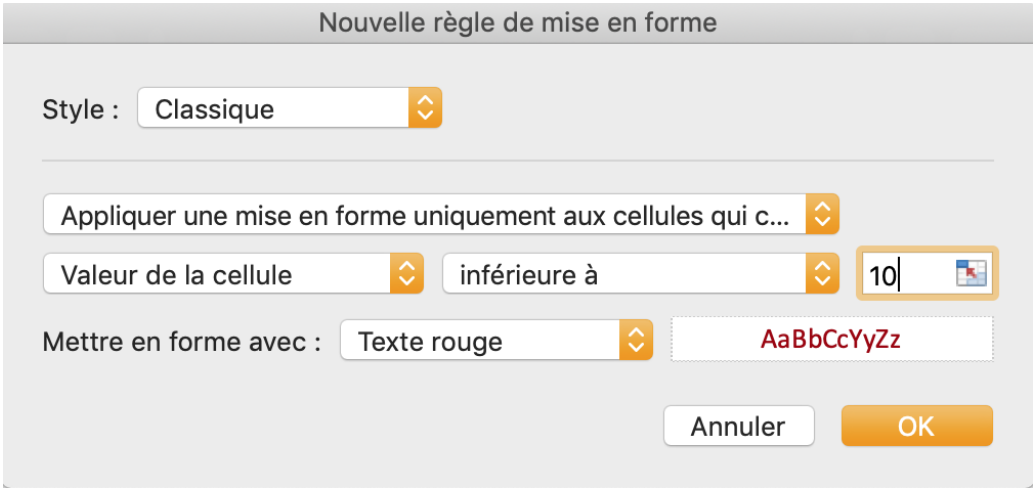
La mise en forme conditionnelle s'est considérablement *sophistiquée* dans les nouvelles versions d'Excel ; je ne peux malheureusement pas dire à partir de laquelle, puisque je suis passé directement de la version 2000 à la version 2016. Il n'y a que les... qui ne changent pas d'avis.

Le lecteur intéressé prendra le temps de parcourir et d'essayer les multiples possibilités de l'outil, en fonction de ses besoins. Nous allons nous limiter ici à un problème simple d'enseignant, à savoir mettre en rouge [4] les cotes strictement inférieures à 10/20 et en vert celles supérieures à 16.

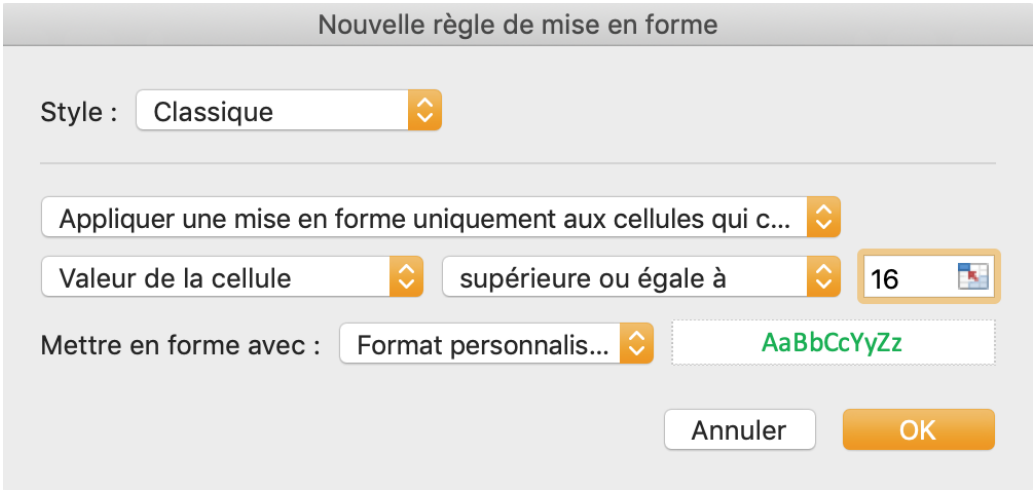
Nous supposerons, pour l'exemple, que la plage `A1:A20` contienne les noms des élèves, et la plage `B1:B20` des cotes comprises dans l'intervalle $[0,20]$. Il faut commencer par sélectionner cette plage `B1:B20`, puis
`Mise en forme > Mise en forme conditionnelle...` ce qui affichera une boîte de dialogue *Gérer les règles* (vide pour l'instant).



Deux fois de suite, Clic sur + pour définir une nouvelle mise en forme conditionnelle, et remplir comme suit la boîte de dialogue (il y a beaucoup de choix dans les listes déroulantes) :

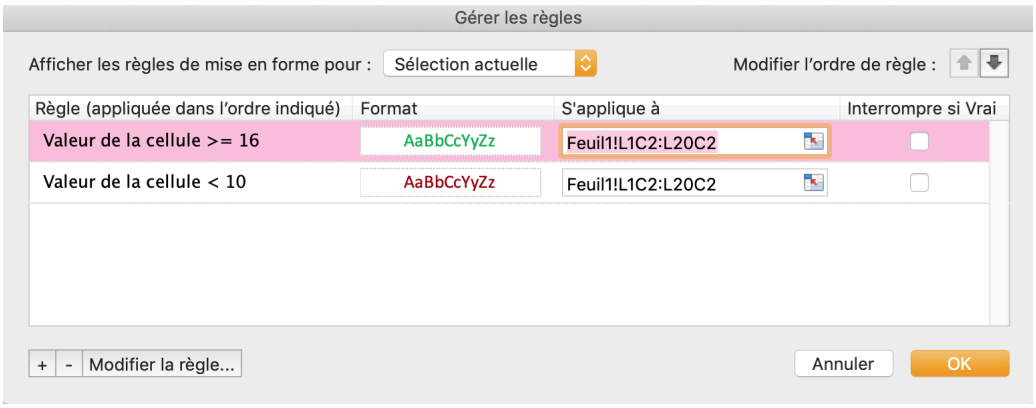


4. On peut évidemment choisir un autre type de mise en forme pour ne pas traumatiser les élèves.



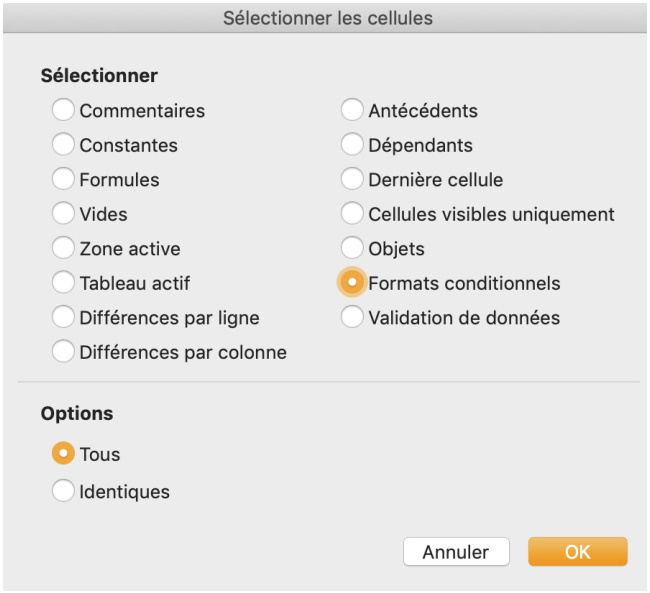
Le choix des formats personnalisés donne accès à la traditionnelle boîte de dialogue où l'on peut déterminer les formats des nombres, de la police, de la bordure, du remplissage.

Dans les versions précédentes d'Excel, on était limité à 3 règles de mise en forme, et seul le style correspondant au « style classique » était disponible. Étant donné l'aspect de la boîte de dialogue *Gérer les règles* ci-après, cette limite peut être vraisemblablement dépassée.



Toute modification d'une note entraîne automatiquement une mise en forme telle que définie dans les règles.

On peut être amené à définir des mises en forme conditionnelles pour beaucoup de plages de cellules d'une feuille. Si on souhaite apporter des modifications à certaines règles, il peut être difficile de se souvenir des adresses des plages, puisqu'il n'y a aucune indication physique dans les plages traitées.
Microsoft fait parfois bien les choses, et il est possible de retrouver toutes les plages traitées :
`Edition > Rechercher > Atteindre... > Cellules...`



Références

[1] Desbonnez J.-M., Le tableur Excel et les macros-instructions (1). *Losanges*, 43, pp. 56–63, 2018.

[2] Desbonnez J.-M., Le tableur Excel et les macros-instructions (2). *Losanges*, 44, pp. 45–58, 2019.

[3] https://excel-malin.com (Consulté le 27/03/19).

Le tableur Excel et les contrôles de formulaires

Résumé. *Cet article devait être consacré à la création et à la gestion des boîtes de dialogue... mais celles-ci ne sont pas (encore) implémentées dans la version 2016 pour Mac. Par contre, nous avons accès, dans l'onglet « développeur » aux différents « objets »(appelés* contrôles de formulaires*) que l'on retrouve dans une boîte de dialogue. Ces objets peuvent être insérés directement dans la feuille de travail, et permettent de rendre un peu plus conviviale la saisie de données dans une cellule.*

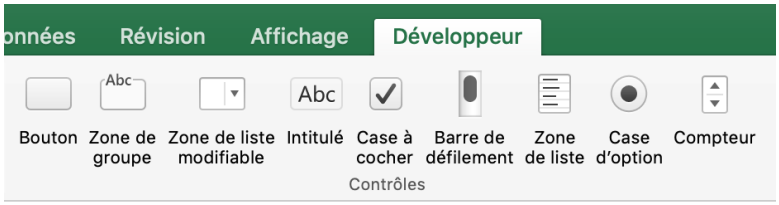


Fig. 25 *Les contrôles*

1 Principes

La première étape consiste à choisir un objet dans la liste, et à le dessiner dans la feuille à l'aide de la souris. La seconde consiste à le paramétrer.

2 Le bouton

Ce contrôle permet de démarrer l'exécution d'une macro. Il a déjà été traité dans un article précédent [3]. Une fois dessiné, le bouton contient le texte `Bouton 1` (si c'est le premier), et est immédiatement attaché à une macro nommée `Bouton1_cliquer` qui n'existe pas encore. Le contenu de cette macro peut être soit directement enregistré, soit encodé au clavier, soit copié d'une autre macro existante (copier-coller des instructions). On peut modifier le nom de la macro associée, le texte sur le bouton, la police, la couleur...

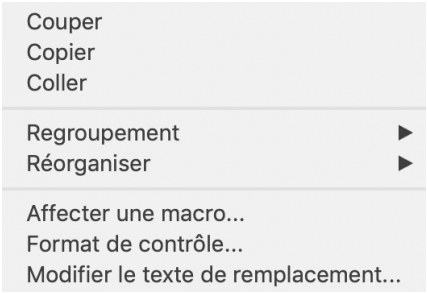
3 L'intitulé

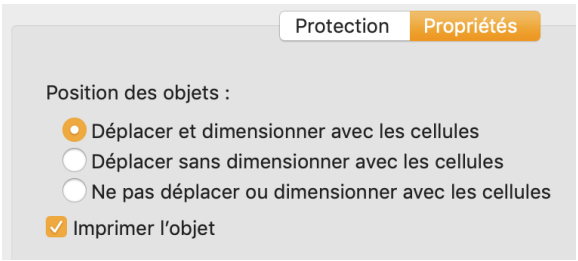
Ce contrôle permet simplement d'afficher une zone de texte dans la feuille, surtout à proximité d'un autre contrôle pour donner quelques explications à l'utilisateur.

- choisir le contrôle et le dessiner dans la feuille (le texte par défaut est `Etiquette 1`)
- clic sur le texte `Etiquette 1` pour le modifier

On peut le déplacer et modifier sa taille à l'aide des poignées traditionnelles. Le paramétrage des contrôles dépend du type de contrôle, mais c'est toujours via le menu contextuel :

- clic droit sur le contrôle
- Format de contrôle...

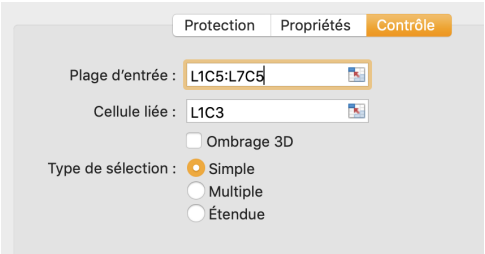


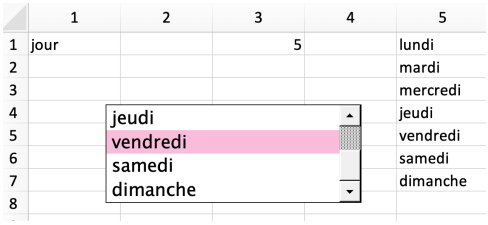


Les options sont ici suffisamment explicites et ne nécessitent pas de commentaires particuliers, si ce n'est que les dimensions des cellules varient en cas de modification des largeurs de colonnes ou de hauteurs de lignes.

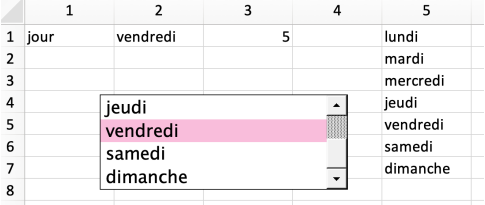
4 La zone de liste

La zone de liste permet de choisir un futur contenu de cellule dans une liste pré-définie (en réalité, la cellule va contenir le numéro d'ordre du choix dans la liste). La liste prédéfinie est le contenu d'une plage de cellules, appelée *plage d'entrée* dont il faudra préciser l'adresse, et la cellule qui recevra le choix est appelée *cellule liée*, dont il faudra aussi préciser l'adresse.





On souhaite en fait que la cellule `L1C2` contienne **le nom du jour**... On peut (on doit) dans ce cas y utiliser la fonction `INDEX(...)` qui permet de retrouver un élément dans une matrice connaissant la position dans la matrice : `=INDEX(matrice; n°ligne; n°colonne)`, ce qui, dans notre situation, devient `=INDEX(L1C5:L7C5;L1C3;1)`



En pratique, la cellule `L1C3` et la plage `L1C5:L7C5`, qui sont des cellules de travail, peuvent trouver leur place dans une autre feuille du classeur afin de ne pas encombrer la feuille principale.

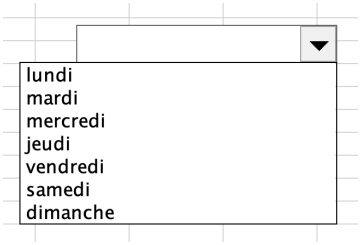
Quant aux types de sélection « Multiple » et « Étendue », je ne comprends pas leur utilité ni le fonctionnement... je suis preneur si quelqu'un a une explication.

Remarque

Si une feuille est nommée par exemple `work` et contient les données nécessaires aux contrôles (ou à d'autres fins d'ailleurs), les adresses ont la forme `work!L1C5:L7C5` pour reprendre l'exemple précédent.

5 Zone de liste modifiable

Ce contrôle a la même fonction que le précédent, seule la présentation diffère un peu, il n'y a pas les options « Types de sélection » et il faut aussi préciser l'adresse d'une plage d'entrée et l'adresse d'une cellule liée. La même plage d'entrée peut servir pour plusieurs contrôles.



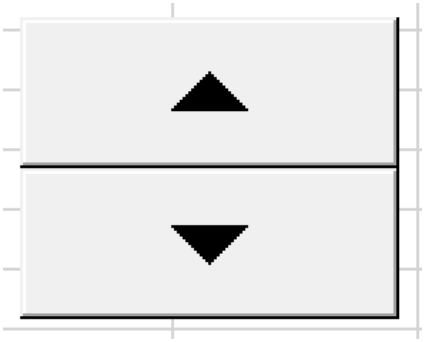
Avant choix

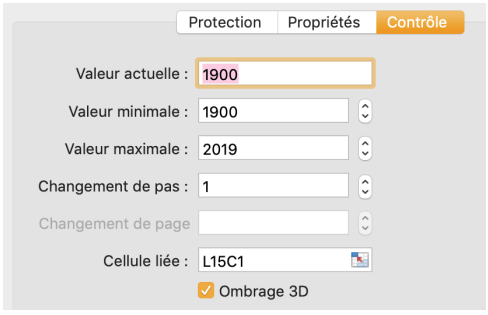
Après choix

Ce contrôle s'appelle aussi « zone de liste déroulante », ce qui à mon sens semble un vocabulaire plus adéquoit.

6 Le compteur

Le compteur permet d'augmenter ou de diminuer le contenu d'une cellule (*la cellule liée*) d'un nombre entier **naturel** compris entre 0 et 30 000 (*changement de pas*) qu'il faut préciser. Il est formé uniquement de deux triangles, l'un pour augmenter, l'autre pour diminuer.





Il faut également préciser la valeur minimale et la valeur maximale de la cellule liée. L'exemple ci-dessus permettrait d'encoder une année de naissance, par exemple.

7 Interlude : protection des cellules

Lorsque le contenu d'une cellule est géré par un contrôle, c'est très convivial, mais rien n'empêche l'utilisateur de modifier lui-même le contenu de la cellule en y tapant n'importe quoi, ce qui peut engendrer des erreurs ailleurs dans la feuille.

Il est donc important de protéger le contenu d'une cellule afin d'y empêcher toute intrusion externe. Voici comment.

Par défaut, toutes les cellules d'une feuille sont verrouillées, mais le verrouillage est inactif tant que la feuille n'est pas protégée (c'est le cas par défaut). On peut donc modifier le contenu de toutes les cellules. Si on protège la feuille, on n'a plus accès à aucune cellule... ce qui est **très** handicapant !

L'idée est donc de ne verrouiller **que** les cellules auxquelles l'utilisateur ne doit pas avoir accès, les cellules liées par exemple, puis de protéger la feuille. Comme il y aura plus de cellules non verrouillées que de cellules verrouillées, on commencera par déverrouiller toutes les cellules de la feuille :

- Sélectionner toutes les cellules (clic dans le coin supérieur gauche)
- Mise en forme > Cellule... > onglet 'Protection' > décocher l'option 'Verrouillé'

On sélectionne ensuite les cellules que l'on veut protéger, puis

- Mise en forme > Cellule... > onglet 'Protection' > cocher l'option 'Verrouillé'

Et enfin on protège la feuille :

- Outils > Protection > Protéger la feuille (avec mot de passe éventuel)

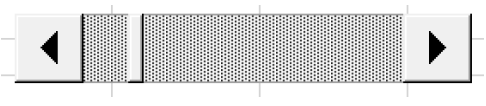
Tout ce travail se réalise évidemment lorsque la feuille est terminée et que l'on ne doit plus y apporter de modifications.

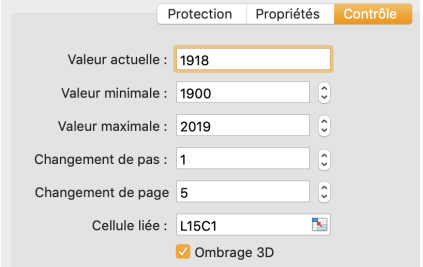
8 La barre de défilement

Ce contrôle a été largement exploité dans [1], avec le tableur d'OpenOffice. C'est le même principe avec Excel.

La barre de défilement est en fait un « compteur allongé » avec l'option supplémentaire *Changement de page* qui précise le pas en cas de clic dans la « cage de l'ascenseur », entre les triangles et l'« ascenseur ».

En manipulant l'« ascenseur » avec la souris, on obtient une variation rapide du contenu de la cellule liée, ce qui permet, dans certaines situations, d'animer des graphiques (voir [1]).

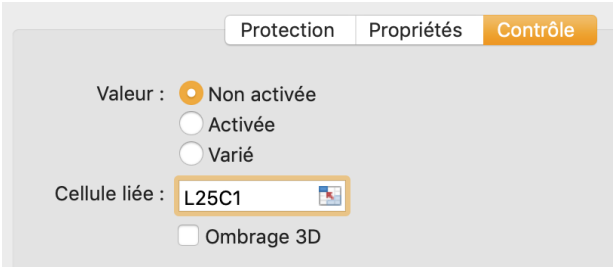




9 La case à cocher

Ce contrôle engendre les valeurs booléennes VRAI (si case activée) et FAUX (si case non activée) dans la cellule liée (qui doit être une cellule de travail). Cette cellule de travail peut alors être utilisée dans des formules du type `=si(L25C1;…;…)`.

Pour modifier le texte qui accompagne la case à cocher, il suffit de cliquer sur le texte par défaut (même principe que pour l'intitulé).

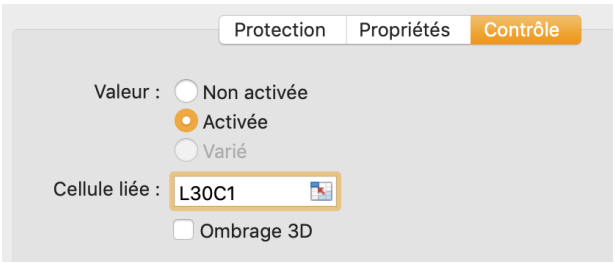


Si l'option *Varié* est sélectionnée, la case est grisée et engendre le message d'erreur `#N/A` ce qui force l'utilisateur à faire un choix. Dès qu'un choix est fait, c'est une des deux autres valeurs qui est sélectionnée.

10 La case d'option

Utilisée **seule**, la case d'option se comporte presque comme la case à cocher, sauf qu'elle engendre 0 ou 1 dans la cellule liée, mais une fois activée, on ne peut plus la désactiver, d'où le nom d'« option ».

En réalité, la case d'option trouve sa place dans une *zone de groupe* où l'on en place au moins deux, et dans une *zone de groupe*, les options s'excluent mutuellement.

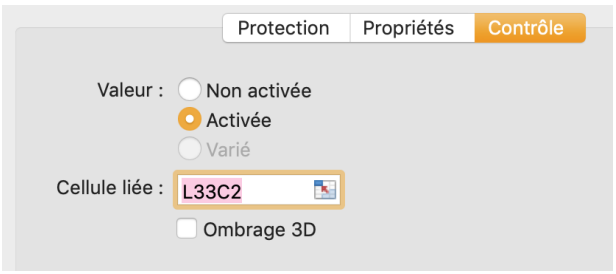


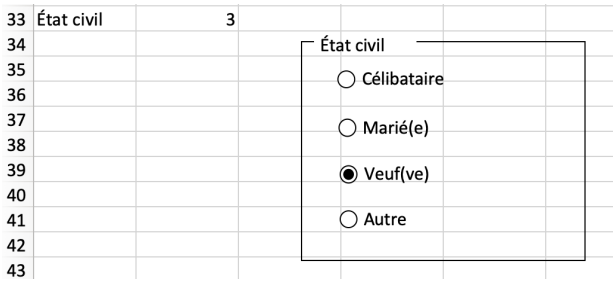
11 La zone de groupe

Ce contrôle est utile pour encadrer plusieurs cases d'option afin qu'elles s'excluent mutuellement. On commence par dessiner le contrôle ; il n'a pas d'options particulière, si ce n'est *Ombrage 3D*.

On y inclut ensuite au moins deux cases d'option, l'ordre de création est important. On définit la cellule liée de l'une des cases d'option (peu importe laquelle, ce sera la même pour toutes).

La cellule liée va contenir le numéro d'ordre de création de la cade d'option choisie.





Le traitement du contenu de la cellule liée peut se faire, par exemple, avec la fonction CHOISIR() :

```
=CHOISIR(L33C2;"célibataire";"marié(e)";"veuf(ve)";"autre")
```

Références

[1] Desbonnez J.-M., Tableur et barre de défilement, graphiques animés : cycloïde, épicycloïde, hypocycloïde. *Losanges*, 38, pp. 47–59, 2017.

[2] Desbonnez J.-M., Tableur Excel et macro-instructions. *Losanges*, 43, pp. 56–63, 2018.

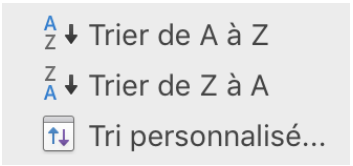
Le tableur Excel et les fonctions de recherche dans les tableaux

Résumé. *Dans cet article, nous décrirons quelques fonctions qui permettent de faire des recherches d'éléments dans un tableau (ou une matrice) à partir d'une clé de recherche ou d'une adresse ; il s'agit des fonctions* RECHERCHEV(), RECHERCHEH(), INDEX() *et* EQUIV(). *Ces fonctions existaient déjà dans les versions précédentes d'Excel, que ce soit sous Windows ou sous MacOs.*

1 Prérequis : tri d'un tableau

Certaines de ces fonctions nécessitent que le tableau soit préalablement trié (classé en ordre numérique, alphabétique ou chronologique, croissant ou décroissant, selon les besoins). Voici comment faire.

Bien que l'on puisse annuler la (les) dernière(s) commande(s), à condition de ne pas enregistrer le classeur, il vaut mieux éviter une mauvaise manœuvre, car le tri mal fait d'un tableau peut entraîner le mélange de toutes les données. Alors un conseil d'ami : éviter d'utiliser les icônes A → Z ou Z → A, et préférer Tri personnalisé... dans le menu ci-contre.



Il est toujours préférable de sélectionner le tableau (lignes de titres comprises) *avant* de lancer le tri.

Supposons que l'on souhaite trier le tableau suivant (les numéros de lignes et de colonne n'ont pas d'importance ici) :

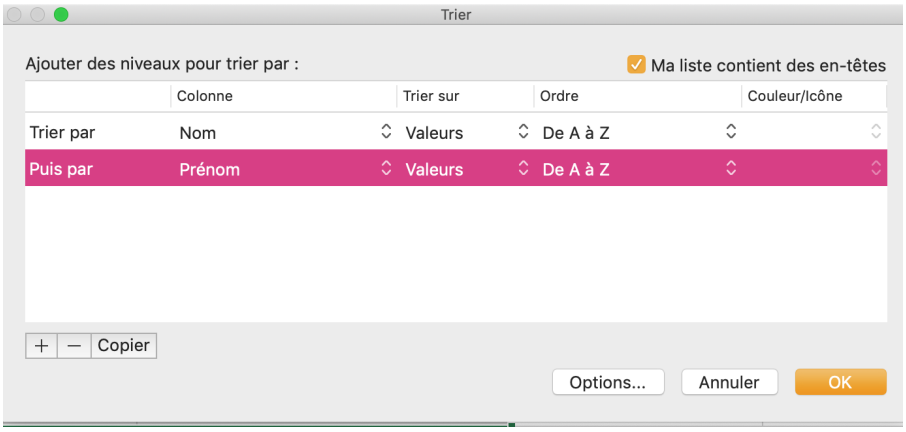
Nom	Prénom	Adresse
Népérien	Logarithme	10 rue Courte
Linéaire	Algèbre	12 rue du Gaz
Régulier	Triangle	100 avenue Bleue
Régulier	Ennéagone	1 rue Verte

Ce tableau ne contient que du texte ; toute personne, plutôt francophone, comprend que la première ligne est une ligne de titre. Ce n'est pas le cas d'Excel, (sauf si les intitulés sont en gras) et il faudra le lui signaler...

Deux personnes ont le même nom (c'est voulu). Si on fait un tri en choisissant uniquement le nom comme première *clé de tri*, l'ordre initial (des deux noms qui se suivent) sera conservé, et

« Triangle » sera avant « Ennéagone ». Pour avoir un tri supplémentaire en cas d'égalité de la première clé, il faudra en ajouter une seconde — le prénom — (et ainsi de suite si nécessaire, toute clé supplémentaire n'intervenant qu'en cas d'égalité de la précédente).

- Sélectionner le tableau, *ligne de titre comprise* > Tri personnalisé...
- Cocher l'option 'Ma liste contient des en-têtes' (Excel va alors nommer les colonnes avec le texte de la première ligne, et ne pas les inclure dans les données)
- Compléter la première clé de tri (liste déroulante de la zone 'Colonne')
- Clic sur l'icône [+] en bas à gauche pour ajouter une seconde clé de tri

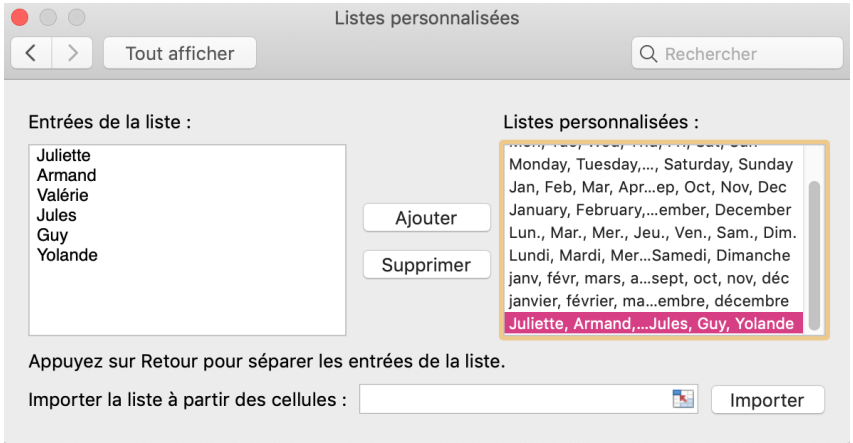


Exception : tri de listes personnalisées

Certains types de listes ne peuvent pas se trier « classiquement » en ordre alphabétique, comme par exemple les noms de jours, les noms de mois, les noms des membres d'un conseil d'administration en fonction de leur grade, etc.

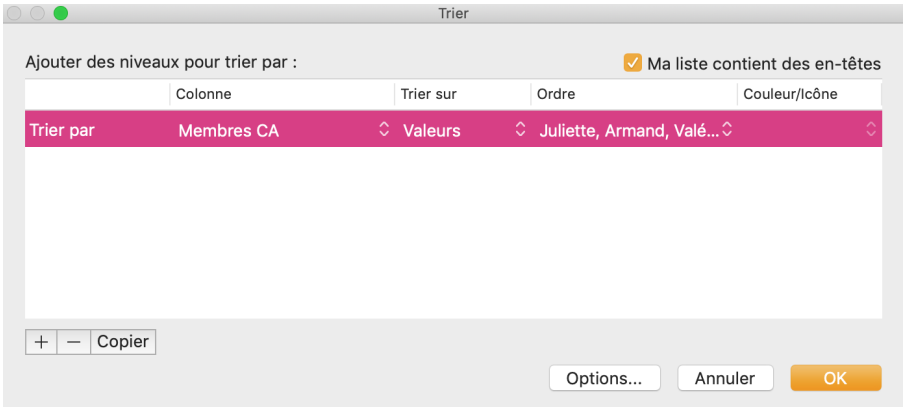
Si on utilise ce type de listes, il faut d'abord les définir :

- Excel > Préférences... > Listes personnalisées



- Les listes des jours et des mois sont pré-encodées, en français et en anglais, sous différentes formes ; on peut ajouter ses propres listes particulières, et supprimer celles qui ne sont plus utiles.

Pour trier un tableau dont la clé de tri fait partie d'une liste personnalisée, il faut le préciser dans la liste déroulante de la zone 'Ordre' (la liste des listes personnalisées va s'afficher et il suffit de choisir celle que l'on souhaite utiliser) :



2 Les fonctions RECHERCHEV() et RECHERCHEH()

Ces fonctions permettent de rechercher un élément donné (la *clé de recherche*) :

- dans la première colonne d'un tableau, et de renvoyer n'importe quelle donnée située sur la même ligne que l'élément cherché (fonction RECHERCHEV()) ;
- dans la première ligne d'un tableau, et de renvoyer n'importe quelle donnée située sur la même colonne que l'élément cherché (fonction RECHERCHEH()).

La clé de recherche est souvent un *code* permettant de reconnaître les données, voire même souvent construit à partir des données elles-mêmes.

Les deux fonctions ont exactement la même syntaxe, seule la *disposition du tableau* diffère dans le cadre de leur utilisation.

RECHERCHEV()
pour extraire des données d'une table
à orientation verticale

Clé	Donnée 1	Donnée 2	Donnée 3	...
Clé 1				
Clé 2				
Clé 3				
...				

RECHERCHEH()
pour extraire des données d'une table
à orientation horizontale

Clé	Clé 1	Clé 2	Clé 3	...
Donnée 1				
Donnée 2				
Donnée 3				
...				

Les tables à orientation verticale étant plus souvent utilisées, nous continuerons avec un exemple de ce type.

Dans le tableau ci-dessous [1], une recherche sur la clé *Code* permettra d'extraire les données *Libellé* (colonne 2), *Prix unitaire* (colonne 3) et *Taux tva* (colonne 4).

Code	Libellé	Prix unitaire	Taux tva
fra	fraises	5,4	0,05
fram	framboises	4,5	0,08
pomgol	pommes golden	2,5	0,06
pomgra	pommes granny smith	3,75	0,06
prumir	prunes mirabelle	5,7	0,07
prurc	prunes reine claude	4,15	0,07

Selon les arguments de la fonction de tri, le tableau DOIT être trié en ORDRE CROISSANT sur le code clé (première colonne physique), ce qui est fait.

Les formules seront plus lisibles si le tableau des données est *nommé* (lignes de titre NON COMPRISES) ; de plus, le nom du tableau se comportera comme une *adresse absolue* en cas de copie de formule, ce qui est plus logique.

Nous supposerons que le tableau de données a été nommé *fruits*.

Syntaxe de la fonction

RECHERCHEV(clé_cherchée ; nom_table ; numéro_colonne ; valeur_proche)

Premier cas : l'argument *valeur_proche* est omis ou VRAI

La fonction RECHERCHEV() se positionne sur la ligne du plus grand élément de la première colonne dont la valeur est *inférieure ou égale* à la clé cherchée, puis renvoie la donnée de la colonne dont le numéro est précisé par l'argument *numéro_colonne*.

Formule	Valeur renvoyée
=RECHERCHEV("pomgra" ; fruits ; 3)	3,75
=RECHERCHEV("prumir" ; fruits ; 4 ; VRAI)	0,07
=RECHERCHEV("mir" ; fruits ; 2 ; VRAI)	framboises
=RECHERCHEV("a" ; fruits ; 2 ; VRAI)	#N/A
=RECHERCHEV("z" ; fruits ; 2 ; VRAI)	prunes reine claude

Commentaires utiles

- Lignes 1 et 2 : lorsque la clé cherchée existe, facile à comprendre.
- Ligne 3 : la clé cherchée n'existe pas, mais il existe une clé qui la précède (d'accord, c'est un exemple un peu farfelu avec du texte, mais un autre exemple numérique, plus intéressant, suivra).
- Ligne 4 : la clé cherchée n'existe pas, et il n'existe pas de clé qui la précède, d'où le message d'erreur ; un exemple suivra pour voir comment traiter ce message d'erreur.

1. Les taux de TVA sont fantaisistes, les prix unitaires sont des prix au kg.

- Ligne 5 : idem que ligne 3.

Second cas : l'argument *valeur_proche* est présent et égal à FAUX
La table PEUT mais NE DOIT PAS être triée en ordre croissant sur la première colonne.

La fonction RECHERCHEV() se positionne sur la ligne dont l'élément de la première colonne est *exactement égal* à la clé cherchée (il n'est pas tenu compte de la casse), puis renvoie la donnée de la colonne dont le numéro est précisé par l'argument *numéro_colonne*.

Formule	Valeur renvoyée
=RECHERCHEV("pomgra" ; fruits ; 3 ; FAUX)	3,75
=RECHERCHEV("pongra" ; fruits ; 3 ; FAUX)	#N/A

Application 1

Recherche de la plus grande valeur inférieure ou égale à un élément d'une table (rappel : celle-ci doit être triée en ordre croissant des *quantités*). La table a été nommée *remise*.

	1	2	3	4	5	6
1		quantités	% remise			
2		0	0%			
3		4	5%			
4		24	10%			
5		49	15%			
6		99	25%			
7		499	50%			
8						
9		article	quantité	prix unit.	% remise	prix net
10		pommes	20	2,00 €	5,00%	38,00 €
11		poires	49	2,50 €	15,00%	104,13 €
12		fraises	130	3,00 €	25,00%	292,50 €
13		oranges	53	1,20 €	15,00%	54,06 €

Les quantités et prix unitaires sont donnés. La remise est calculée à partir d'une recherche du taux. Le prix net est calculé.

Formule en L10C5 : **=RECHERCHEV(LC(-2) ; remise ; 2 ; VRAI)**

Cette formule est copiable vers le bas ; on laisse au lecteur le soin de trouver la formule en L10C6 [(2)] ; elle doit être copiable vers le bas.

Application 2

Recherche d'une valeur exactement égale à un élément d'une table (rappel : celle-ci ne doit pas nécessairement être triée en ordre croissant, mais c'est plus joli...). La table a ici été nommée *tva*.

2. Si celui-ci est fin mathématicien, cela ne devrait pas être trop difficile.

	1	2	3	4	5	6	7
1		code tva	taux tva				
2		0	0%				
3		1	6%				
4		2	19,50%				
5		3	25%				
6							
7		article	quantité	prix unit.	code tva	taux tva	prix net
8		pommes	20	2,00 €	2	19,50%	
9		poires	49	2,50 €	3	25,00%	
10		fraises	130	3,00 €	1	6,00%	
11		oranges	53	1,20 €	0	0,00%	

Les quantités, prix unitaires et code tva sont donnés. Le taux de tva est calculé à partir d'une recherche. Le prix net est calculé (par le lecteur).

Formule en L8C6 : **=RECHERCHEV(LC(-1) ; tva ; 2 ; FAUX)** copiable vers le bas.

Application 3

Le risque dans une recherche est que l'élément cherché n'existe pas. Excel affiche alors le message d'erreur #N/A [3] qui n'est pas très parlant pour l'utilisateur.

On peut remplacer ce message d'erreur par un autre, plus parlant, en utilisant une fonction dite d'*information* **ESTNA()** dans une fonction alternative **SI()**.

La table des données a été nommée *carnet* ; le code est donné, le numéro de téléphone est à rechercher.

	1	2	3	4	5	6
1						
2		code	nom	adresse	localité	téléphone
3		dup	Dupont	12 rue du gaz	7780 Comines	033.67.89.94
4		dur	Durand	34 avenue Sleeckx	1030 Schaerbeek	022.61.12.34
5		vab	VanTruc	181 chaussée de Nivelles	1472 Vieux Genappe	011.55.66.88
6						
7			code	téléphone		
8			vab	011.55.66.88		
9			vba	non trouvé		
10						

À partir du code, il faut tester si la fonction de recherche va engendrer le message d'erreur `#N/A` ; si c'est le cas, afficher un autre message d'erreur (ou une autre commande), et dans le cas contraire appliquer la fonction de recherche.

Formule en L8C4 :
=SI(ESTNA(RECHERCHEV(LC(-1) ;carnet ;5 ;FAUX)) ;"non trouvé" ;RECHERCHEV(LC(-1) ; carnet ;5 ;FAUX))

Pour information...

Il existe d'autres fonctions d'information (elles retournent toujours un résultat booléen, on les retrouve donc souvent dans le premier argument de la fonction alternative SI()). En voici la liste exhaustive :

EST.IMPAIR(), EST.PAIR(), ESTERR(), ESTERREUR(), ESTFORMULE(), ESTLOGIQUE(), ESTNA(), ESTNONTEXTE(), ESTNUM(), ESTREF(), ESTTEXTE(), ESTVIDE().

Leur nom est souvent suffisamment explicite ; le lecteur qui s'y intéresse consultera l'aide du logiciel.

3. C'est en fait l'abréviation de *Non Available* (Non Disponible).

Les limites des fonctions RECHERCHEV() et RECHERCHEH()

Une première contrainte est que la recherche doit se faire dans la première colonne (resp. ligne) du tableau des données, et donc que la valeur de retour doit impérativement être située à droite (resp. en dessous).

Si la structure du tableau de données doit évoluer, par exemple après insertion de colonne(s) à gauche (resp. de ligne(s) au-dessus), il sera impossible d'utiliser la fonction RECHERCHEV() (resp. RECHERCHEH()) dont le troisième argument serait -1, -2, etc.

Une seconde contrainte est que le troisième argument de la fonction de recherche (le numéro de colonne ou de ligne de retour) est généralement (quasi toujours) un *nombre fixé* (c'est-à-dire n'est pas une adresse de cellule).

Pour les mêmes raisons qu'expliquées ci-dessus, en cas d'insertion de lignes et/ou de colonnes, ce nombre fixé risque de ne plus représenter la position de l'élément cherché.

Et enfin une dernière, dès qu'on veut trier le tableau selon une autre clé, on perd tous les résultats de RECHERCHEV().

Ces « limites » n'en sont pas si le tableau des données est figé et organisé de telle manière que les clés de recherche occupent la première ligne ou la première colonne du tableau.

Si ce n'est pas le cas, on résout le problème en utilisant une combinaison des fonctions INDEX() et EQUIV(), ce qui constitue une excellente mise en bouche pour la suite...

3 La fonction INDEX()

3.1 Pré-requis : formules matricielles

Remarque préliminaire : on peut remplacer ci-après le terme *tableau* par *plage* ou *matrice*, dans tous les cas il s'agit d'un ensemble rectangulaire de cellules contiguës.

Une formule matricielle utilise tous les éléments d'un (ou plusieurs) tableau(x) pour calculer un résultat unique ou un autre tableau, sans qu'il soit nécessaire de recopier des formules.

Certaines fonctions ont le « statut » de formule matricielle, car elles ont des tableaux comme arguments. C'est le cas par exemple de la fonction SOMMEPROD(tableau1 ; tableau2 ; ...) qui calcule la somme des produits (terme par terme équivalent) des éléments de tableau1 par ceux de tableau2... (tous les tableaux intervenant dans la formule doivent avoir la même dimension, au sens matriciel du terme). De même pour les fonctions PRODUITMAT(), DETERMAT() et INVERSEMAT() liées directement au calcul matriciel.

Il n'y a par contre pas de fonction pour calculer par exemple les sommes ou produits terme par terme. Il faut dans ce cas obligatoirement copier des formules, certes simples, ... sauf si on utilise une *formule matricielle*.

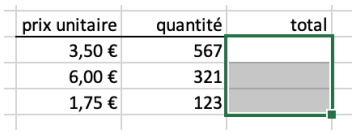
Pour encoder une formule matricielle, il faut toujours commencer par sélectionner le tableau qui va contenir les résultats.

Lorsqu'on encode une formule matricielle, il faut la valider par la combinaison des trois touches CTRL, SHIFT et ENTER. La formule sera alors encadrée par une paire d'accolades, qu'on ne peut pas taper soi-même pour éviter la validation à trois touches. Comme d'habitude, les tableaux peuvent être nommés, ce qui rend les formules plus lisibles.

Une formule matricielle est protégée contre la modification et la suppression. Pour modifier une formule matricielle, il est obligatoire de valider par CTRL, SHIFT et ENTER ; pour supprimer une formule matricielle, il faut sélectionner tout le tableau des résultats avant d'opérer la suppression.

Dans les exemples qui suivent, les résultats des calculs n'ont pas été affichés ; le lecteur s'en chargera à titre d'exercice.

Exemple 1



Les deux plages *prix unitaire* et *quantité* ont été respectivement nommées *pu* et *qu* (titres non compris).

- Sélectionner la plage des résultats
- Taper la formule `=pu*qu` et valider par CTRL SHIFT ENTER

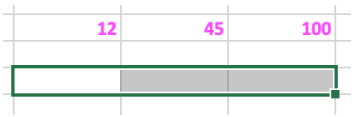
Les trois cellules résultats contiennent la formule matricielle `{=pu*qu}`

Une formule matricielle est appliquée individuellement à toutes les cellules de la (des) plage(s).

Exemple 2

On demande d'ajouter 5% aux éléments d'un tableau à condition qu'ils soient supérieurs à 50. La plage des trois cellules de données a été nommée *plage*[4]

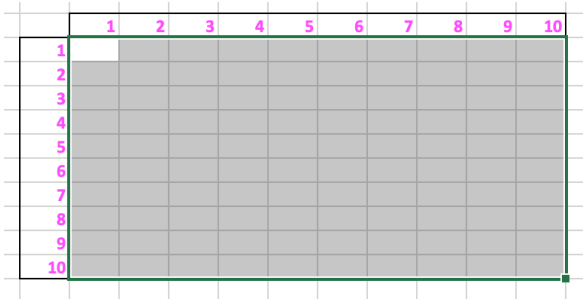
- Sélectionner la plage des résultats
- Taper la formule `=si(plage >= 50;plage*1,05;plage)` et valider par CTRL SHIFT ENTER



Les trois cellules résultats contiennent la formule matricielle
`{=si(plage >= 50;plage*1,05;plage)}`

Exemple 3

Une table de multiplication.



Les deux plages de données (une ligne et une colonne) ont été respectivement nommées *li* et *co*.

- Sélectionner la plage des résultats
- Taper la formule `=li*co` et valider par CTRL SHIFT ENTER

Les cent cellules résultats contiennent la formule matricielle `{=li*co}`

4. Oui, bon...

Remarque : Pour utiliser une formule matricielle, il faut commencer par sélectionner la plage des résultats. C'est une opération facile, avec la souris ou le trackpad, lorsque la plage est entièrement visible à l'écran ; il en est tout autrement si la plage de résultats est formée de centaines de colonnes et de milliers de lignes. Dans ce cas, il vaut mieux faire la sélection via le menu
`Edition > Rechercher > Atteindre... > compléter la zone 'Référence'`

3.2 INDEX()

La fonction INDEX() renvoie la valeur de la cellule située à l'intersection d'une ligne et d'une colonne d'un tableau.

Syntaxe : **INDEX(tableau ; ligne ; colonne)**

1	2	3	12
4	5	6	13
7	7	9	14

Dans le tableau ci-contre, nommé... *tableau*, la formule `=INDEX(tableau;2;3)` donnera le nombre 6 comme résultat.
La formule `=INDEX(tableau;2;5)` va engendrer le message d'erreur `#REF!` car l'argument colonne dépasse la taille du tableau.

Cas particuliers

- Si le tableau est réduit à une ligne (resp. une colonne), l'argument *ligne* (resp. *colonne*) est facultatif.

  La syntaxe est alors réduite à **INDEX(tableau ;; colonne)** (resp. **INDEX(tableau ; ligne ;)**)

- Si le tableau comporte plusieurs lignes et plusieurs colonnes, et que seul l'argument *ligne* (resp. *colonne*) est utilisé, la fonction retourne la matrice de la totalité de la ligne (resp. colonne), MAIS doit être validée comme une formule matricielle, avec CTRL SHIFT ENTER, et il faut préalablement sélectionner une plage de résultats de la taille d'une ligne (resp. colonne) du tableau des données.

  Pour le tableau ci-dessus, la formule matricielle `{=INDEX(tableau;3;)}` va retourner la troisième ligne du tableau.

4 La fonction EQUIV()

4.1 Syntaxe et remarques

La fonction EQUIV() recherche un élément dans un tableau (de type ligne ou colonne) et renvoie sa *position* dans le tableau.

Syntaxe : **EQUIV(élément_cherché ; tableau ; précision)**

L'argument *précision* peut prendre trois valeurs :

- -1 pour une recherche de la plus proche valeur inférieure à l'élément cherché ;
- 0 pour une recherche de la valeur exacte ;

- 1 pour une recherche de la plus proche valeur supérieure à l'élément cherché.

Dans la majorité des cas, on fait une recherche exacte, et on choisit donc une *précision* égale à 0.

La *précision* est un argument facultatif, mais attention, sa valeur par défaut est 1. Il vaut donc mieux toujours la préciser.

Attention

- Si l'argument *précision* vaut 1, il faut que les valeurs dans le tableau des données (rappel : ligne ou colonne) soient classées en ordre croissant.
- Si l'argument *précision* vaut -1, il faut que les valeurs dans le tableau des données soient classées en ordre décroissant.
- Si l'argument *précision* vaut 0 (recherche exacte), les valeurs peuvent être dans un ordre quelconque.

Remarques

En cas de recherche exacte,

- si la valeur cherchée n'est pas trouvée, la fonction renvoie le message d'erreur `#N/A`, et il peut être traité comme expliqué pour les fonctions RECHERCHEV() et RECHERCHEH() ;
- si la valeur cherchée est une chaîne de caractères, on peut utiliser le *caractère joker* « * » qui remplace n'importe quel groupe de caractères (par exemple "abc*" pour n'importe quel mot commençant par "abc") ;
- s'il y a des clés de recherche identiques (en principe, il faut s'arranger pour que ce ne soit jamais le cas), c'est la première trouvée qui est prise en compte.

En combinaison avec la fonction INDEX(), on a un outil puissant qui permet de rechercher quasiment n'importe quoi dans n'importe quel tableau.

4.2 Application

Souvenirs d'un horairiste retraité...

Situation : Avec sa dotation horaire, une école organise des cours dans différentes options qui sont regroupées dans différentes classes. Toutes les options comportent un cours de français (tronc commun), certaines un cours de mathématique 4 pér./sem, d'autres un cours de mathématique 6 pér./sem, d'autres encore un cours de latin, etc. Le tout est synthétisé dans un tableau comme celui ci-après (restreint pour la bonne cause) ; il y a un tableau par niveau, mais c'est sans importance pour la suite. Une astérisque dans une cellule signifie que le cours est organisé dans l'option.

	1	2	3	4	5	6	7
1	cours	code	option1	option2	option3	option4	option5
2	Français 5h	fr5	*	*	*	*	*
3	Mathématique 4h	ma4	*	*			*
4	Mathématique 6h	ma6			*	*	
5	Sciences 3h	sc3	*	*			*
6	Sciences 5h	sc5				*	
7	Sciences 7h	sc7			*		
8	Latin 4h	la4			*		*

Le tableau `L1C1:L8C7` a été nommé *grille_horaire*, la ligne `L1C3:L1C7` a été nommée *options* et la colonne `L2C2:L2C8` a été nommée *codcours*.

Vous aurez compris aisément qu'on recherchera si à l'intersection d'une ligne de cours et d'une colonne d'option il y a une astérisque.

À partir de ces données, il faut compléter le tableau suivant (il est dans une autre feuille, mais les noms de plages sont connus dans tout le classeur) :

	1	2	3	4	5	6
1		classe A		classe B		classe C
2		option1	option3	option4	option2	option5
3	nb élèves	5	12	7	15	14
4	ma4					
5	ma6					
6	la4					
7	sc7					
8	fr5					
9						

La direction a composé les classes avec les options, le nombre d'élèves par option est connu ; les codes de branches de la colonne 1 sont encodés au fur et à mesure. Les horairistes étant des gens précis et attentifs, respectent à la lettre les noms des options et des codes de cours, et on peut donc faire des recherches exactes.

Il faut connaître, par cours, le nombre d'élèves qui suivent ce cours dans chaque option (et donc classe), afin de pouvoir décider ou non d'un dédoublement (voir plus) éventuel.

Il faut construire une formule en `L4C2` qui soit copiable dans tout le tableau. Décortiquons-là.

`=EQUIV(L2C;options;0)+2` donne comme résultat **3** (il faut ajouter 2 car la ligne des options ne commence qu'en colonne 3) ; l'option *option1* est bien en première position dans la ligne *options*, et donc en troisième colonne du tableau *grille_horaire* (`L2C` signifie ligne 2, même colonne).

`=EQUIV(LC1;codcours;0)+1` donne comme résultat **3** (il faut ajouter 1 car la colonne des codes de cours ne commence qu'en ligne 2) ; le cours *ma4* est bien en seconde position dans la colonne *codcours*, et donc en troisième ligne du tableau *grille_horaire* (`LC1` signifie même ligne, colonne 1).

Il reste à vérifier si la cellule de coordonnée (3,3) du tableau *grille_horaire* contient une astérisque. Si c'est le cas, la cellule `L4C2` contiendra le nombre d'élèves (cellule `L3C`), sinon rien.

```
=SI(INDEX(grille_horaire;3;3)="*";L3C;"")
```

En plaçant les deux premières fonctions dans la troisième, cela donne :

```
=SI(INDEX(grille_horaire;EQUIV(LC1;codcours;0)+1;EQUIV(L2C;options;0)+2)="*";L3C;"")
```

Cette formule est copiable dans tout le tableau ; en ajoutant une colonne *total*, on voit immédiatement les cours à dédoubler (en l'occurence celui de mathématique 4 pér./sem et celui de français). (5)

	1	2	3	4	5	6	7
1		classe A		classe B		classe C	
2		option1	option3	option4	option2	option5	
3	nb élèves	5	12	7	15	14	total
4	ma4	5			15	14	34
5	ma6		12	7			19
6	la4		12			14	26
7	sc7		12				12
8	fr5	5	12	7	15	14	53
9							

Il suffit, pour ces deux cours, d'ajouter deux lignes, et d'effacer le contenu des cellules devenues inutiles. L'ajout d'une colonne *prof* permet de gérer le nombre de professeurs nécessaires par branche.

	1	2	3	4	5	6	7	8
1		classe A		classe B		classe C		
2		option1	option3	option4	option2	option5		
3	nb élèves	5	12	7	15	14	total	prof
4	ma4				15		15	Jules
5	ma6		12	7			19	
6	la4		12			14	26	
7	sc7		12				12	
8	fr5				15	14	29	Luc
9	fr5	5	12	7			24	Lucie
10	ma4	5				14	19	Louise

En gros, c'est ça l'idée.

5. Le nombre d'élèves n'est pas toujours le seul élément pris en compte pour dédoubler un cours...

Le tableur Excel et la trigonométrie de l'horloge (analogique)

Résumé. *Voici une petite application avec le tableur qui permettra de se rappeler de la définition analytique d'un cercle ainsi que de la trigonométrie qui calcule la position de la petite et de la grande aiguille d'une horloge (analogique). Le but est de faire tourner les aiguilles à l'aide d'une barre de défilement [1][2][3], et de construire ainsi une horloge qui donne... votre heure préférée.*

Le graphique (de type XY) sera composé de quatre séries (le cadran, les heures, la petite aiguille, la grande aiguille) composées chacune d'un ensemble de points dont on calculera la coordonnée (abscisse, ordonnée), ce qui nécessitera un peu de trigonométrie élémentaire.

Pour le cadran nous calculerons soixante et un points, douze pour les heures, et deux pour chacune des aiguilles (origine et extrémité).

Rappelons que le tableur ne s'exprime qu'en *radians*.

1 Le cadran, cercle de rayon 1

On pourrait choisir un rayon supérieur à 1, mais inutile d'alourdir les formules (mais rien n'empêche de les expliquer). Les points sur un cercle ont une coordonnée du type $(\cos\alpha, \sin\alpha)$ avec $\alpha \in [0, 2\pi]$.

Pour dessiner une courbe, il faut un nombre « suffisant » de points afin que le lissage opéré par le tableur soit efficace. Pour dessiner une courbe fermée, il faut que le dernier point de la série ait la même coordonnée que le premier.

	1	2	3
18	cadran		
19	α	$\cos\alpha$	$\sin\alpha$
20	0	`=COS(LC(-1))`	`=SIN(LC(-2))`
21	`=L(-1)C+2*PI()/60`	⋮	⋮
22	⋮		
...	...	...	...
80	`=L(-1)C+2*PI()/60`	`=COS(LC(-1))`	`=SIN(LC(-2))`

Explications

Les lignes 18 et 19 sont des lignes de titre ; la cellule `L18C1` servira de nom pour la série.

`L(-1)C` signifie « 1 ligne au-dessus, même colonne »
`LC(-1)` signifie « même ligne, 1 colonne à gauche »
`LC(-2)` signifie « même ligne, 2 colonnes à gauche »

Le symbole ⋮ signifie que les formules sont copiables vers le bas.

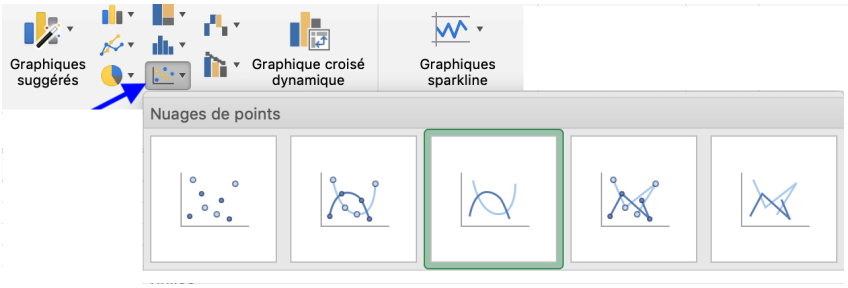
En mode d'affichage « `LC` », les formules copiées restent identiques, contrairement au mode d'affichage « `A1` » où les adresses de cellules sont ajustées.

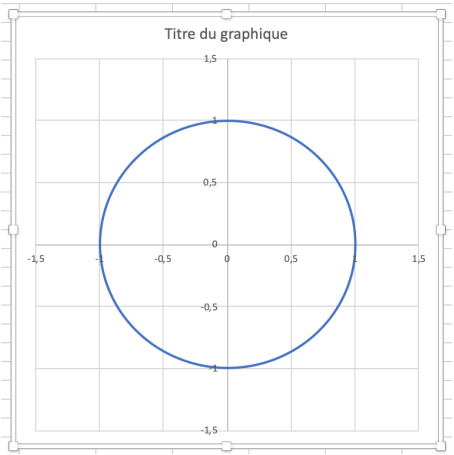
2 Le dessin du cadran

Il y a plusieurs manières de démarrer l'insertion d'un graphique. Le nôtre est composé de quatre séries ; trois d'entre elles (le cadran et les deux aiguilles) auront l'aspect de traits continus (un cercle et deux segments), et la quatrième (les heures) sera réduite à des points.

Il faudra donc superposer deux types différents de graphiques XY. Le type « trait continu » ne demandera pas beaucoup de modifications pour trois des quatre séries. C'est parti.

- Sélectionner la plage `L20C2:L80C3`
- Insertion > Graphiques (comme pointé par la flèche ci-dessous)
- choisir le sous-type « ligne continue »

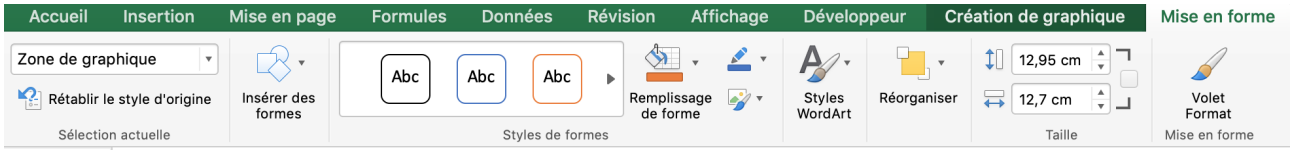




Le tableur ne connaît pas le système orthonormé et le cadran est plutôt ovale.
Faire au mieux avec les poignées de dimensionnement...

Nous avons maintenant à notre disposition un bandeau « Création de graphique »

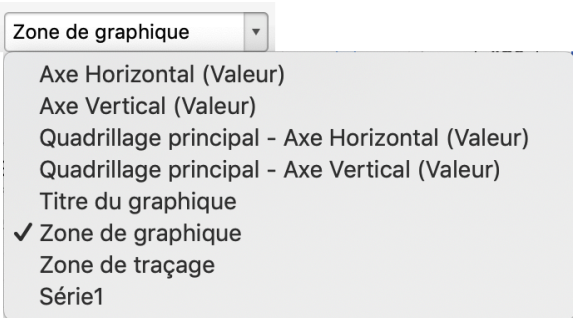
et un autre, « Mise en forme » qui, comme son nom l'indique, permet d'apporter des mises en forme.



L'étape suivante, esthétique, consiste à enlever les éléments inutiles pour une horloge, à savoir le titre, le quadrillage horizontal et vertical ainsi que les axes.

Pour sélectionner un composant du graphique, on peut soit cliquer dessus (précisément), soit dérouler la liste « Zone de graphique » que l'on trouve en haut à gauche du bandeau *Mise en forme*. Une fois un composant séléctionné, il suffit de presser Delete ou la touche équivalente de votre clavier pour supprimer l'élément.

Remarquer que le cadran porte le nom *Série1*, puisque le graphique a été réalisé directement à partir de la sélection de la plage des données. On aura la possibilité de modifier ce nom générique lorsque l'on va ajouter la série des heures.



Après l'opération, il ne devrait rester qu'un cercle.

3 les heures

	5	6	7	8
19	heures	α	$\cos\alpha$	$\sin\alpha$
20	1	`=PI()/2-LC(-1)*2*PI()/12`	`=COS(LC(-1))`	`=SIN(LC(-2))`
21	2	⋮	⋮	⋮
...	...	...	...	...
31	12	...	...	...

Il y a 12 heures sur le cadran [1] et l'amplitude angulaire entre deux heures vaut donc $\frac{2\pi}{12}$ (le fait de ne pas simplifier rend peut-être le raisonnement plus simple pour les élèves).

Ici plusieurs raisonnements sont possibles, en voici un. Midi correspond, sur le cercle trigonométrique, à l'angle $\frac{\pi}{2}$. Pour les heures suivantes (1, 2, 3, ...) il faut donc ôter de $\frac{\pi}{2}$ autant de fois $\frac{2\pi}{12}$ qu'il y a d'unités dans le nombre représentant l'heure.

On pourrait aussi partir de $\frac{\pi}{2}$ (heure 12 ou midi) et retirer à chaque heure suivante $\frac{2\pi}{12}$ de l'angle précédent.

En procédant de la sorte, on travaille dans le sens « horloger » ; normal, notre esprit est concentré sur une horloge. À partir de 3h, les angles deviennent donc négatifs.

1. Souvent, bien qu'il existe des cadrans de 24 heures avec une seule aiguille.

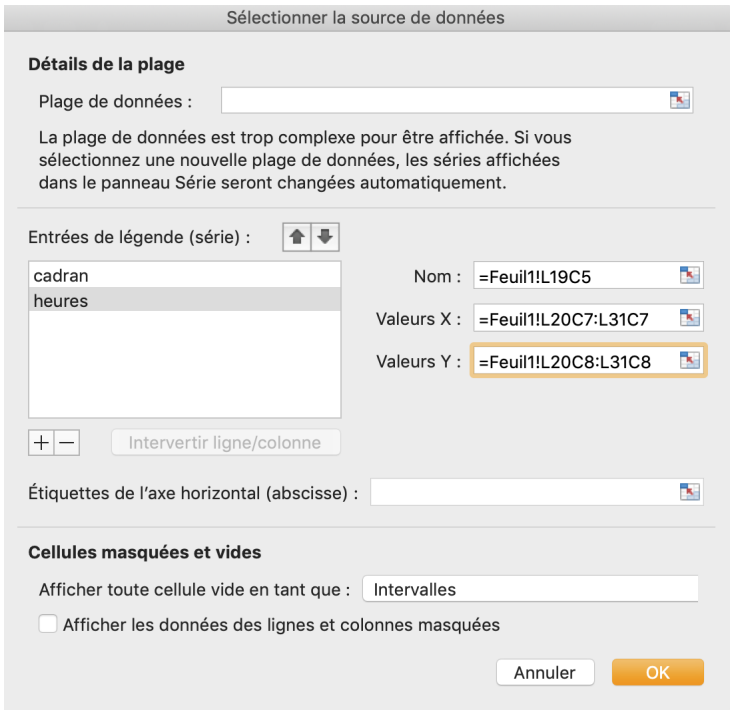
On aurait pu travailler dans le sens « anti-horloger », plus mathématique, en prenant 3h comme heure initiale, et en ajoutant $\frac{2\pi}{12}$ à chaque angle précédent. La colonne des heures contiendrait alors la série 3, 2, 1, 12, 11,..., 4.

Tout cela est sans importance pour le graphique, puisque seule la plage `L20C7:L32C8` sera utilisée, et l'ordre des points est sans importance pour un graphique de type XY.

4 Le dessin des heures

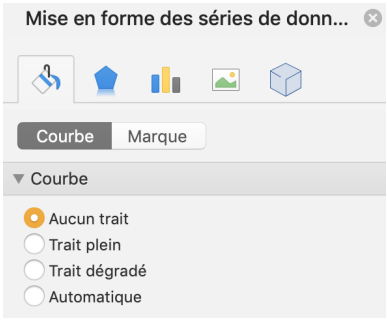
On ne peut pas procéder comme pour le cadran, sinon il y aurait création d'un second graphique. On va *ajouter* la série des heures au graphique existant.

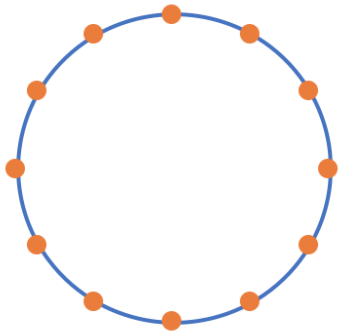
- Sélectionner le graphique (1 clic dessus)
- Onglet 'Création de graphique', icône 'Sélectionner des données' (en profiter pour modifier le nom de la *Série1* en indiquant l'adresse `L18C1` dans la zone 'Nom')
- Clic sur [+] pour ajouter une série : le nom est dans la cellule `L19C5`, les valeurs X dans la plage `L20C7:L31C7` et les valeurs Y dans la plage `L20C8:L31C8`



Le résultat est à peu près un cercle. Normal, le graphique de base était en trait continu et les douze points sont sur le même cercle que le cadran ; le tableur a lissé... il faut modifier le format de la série.

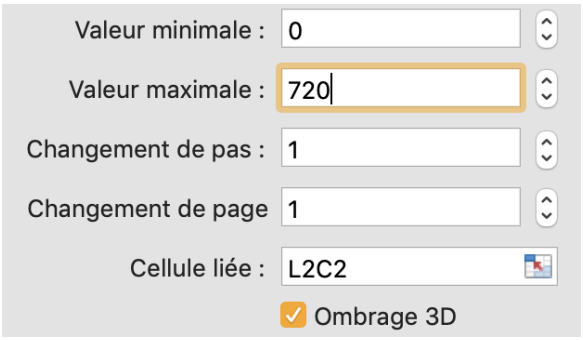
Via le bandeau *Mise en forme*, on sélectionne la *Série "heures"* dans la liste des composants du graphique, puis l'icône *Volet format*, à droite, affiche toute une panoplie d'outils de transformations. Voici les choix qui ont été faits, et le résultat obtenu :

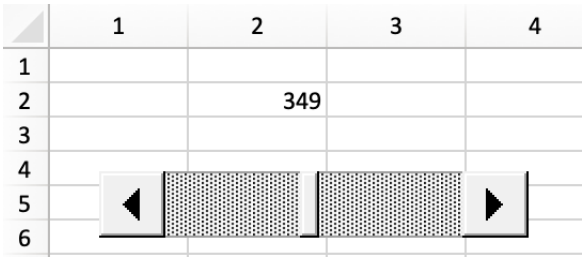




5 De quoi animer les aiguilles

On souhaite animer un tour complet du cadran, soit douze heures, avec une précision à la minute. Il faut donc un compteur de 0 à 720. Pour assurer la fluidité du mouvement, on mettra en place une *barre de défilement* liée à une cellule que l'on nommera *min*. Le lecteur trouvera la procédure à suivre dans [3].





6 Position des aiguilles à une heure donnée

Nous sommes ici dans l'obligation de travailler dans le sens horloger, ce qui explique la dernière ligne du tableau ci-dessous. Le raisonnement est une simple règle de trois avec les angles parcourus. Pour la clarté du raisonnement, aucune simplification n'a été opérée.

temps	α parcouru petite aiguille	α parcouru grande aiguille
1 heure	$\frac{2\pi}{12}$	2π
1 minute	$\frac{2\pi}{12\cdot 60}$	$\frac{2\pi}{60}$
min minute(s)	$\frac{2\pi\cdot min}{12\cdot 60}$	$\frac{2\pi\cdot min}{60}$
heure à min minute(s)	$\frac{\pi}{2} - \frac{2\pi\cdot min}{12\cdot 60}$	$\frac{\pi}{2} - \frac{2\pi\cdot min}{60}$

7 Le dessin des aiguilles

Le cadran ayant un rayon de valeur 1, nous décidons de fixer la longueur de la grande aiguille à 0.9 et celle de la petite aiguille à 0.6.

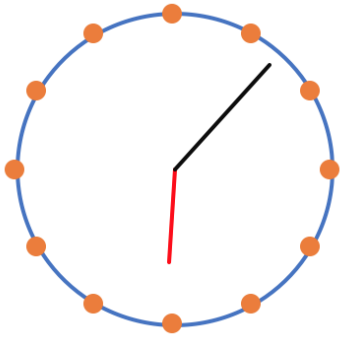
Les aiguilles étant des segments de droite, il suffit de deux points pour les dessiner. Le premier est le centre du cadran, soit le point de coordonnées (0,0). Le second est le point de coordonnées $(\cos\alpha, \sin\alpha)$ avec α dans la dernière ligne du tableau précédent.

Il nous faut donc construire les deux séries suivantes :

	10	11
19	petite aiguille	
20	0	0
21	=0,6*COS(PI()/2-(2*PI()*min)/720)	=0,6*SIN(PI()/2-(2*PI()*min)/720)

	10	11
24	grande aiguille	
25	0	0
26	=0,9*COS(PI()/2-(2*PI()*min)/60)	=0,9*SIN(PI()/2-(2*PI()*min)/60)

Il reste ensuite à ajouter les deux séries au graphique, même marche à suivre que pour les heures.



8 Digitalisation de l'horloge

Restons modernes...

Pour calculer l'heure, on divise par 60 le nombre de minutes, et on prend la partie entière du résultat (fonction ENT()).
Pour calculer les minutes, on prend le reste de la division entière par 60 du nombre total de minutes (fonction MOD()).

Il reste à concaténer les deux résultats en les séparant par un double-point. Voici la formule :
`=ENT(min/60)&":"&MOD(min;60)`

Avec la position des aiguilles dans le graphique qui précède, on obtient comme résultat `6:7`, ce qui n'est pas très joli. Il faudra donc s'appliquer un peu plus et traiter le nombre de digits de l'affichage. Si le résultat du calcul est strictement inférieur à 10, il faut le faire précéder d'un 0. Facile à dire. Nous allons séparer les deux calculs dans deux cellules nommées respectivement *heures* et *minutes*.

Dans la cellule *heures* : `=SI(ENT(min/60)<10;"0"&ENT(min/60);ENT(min/60))`.

Dans la cellule *minutes* : `=SI(MOD(min;60)<10;"0"&MOD(min;60);MOD(min;60))`.

Et il reste à concaténer les deux résultats, comme précédemment : `=heures&":"&minutes` ce qui donne comme résultat 06:07. C'est mieux, mais peut mieux faire encore.

9 AM/PM

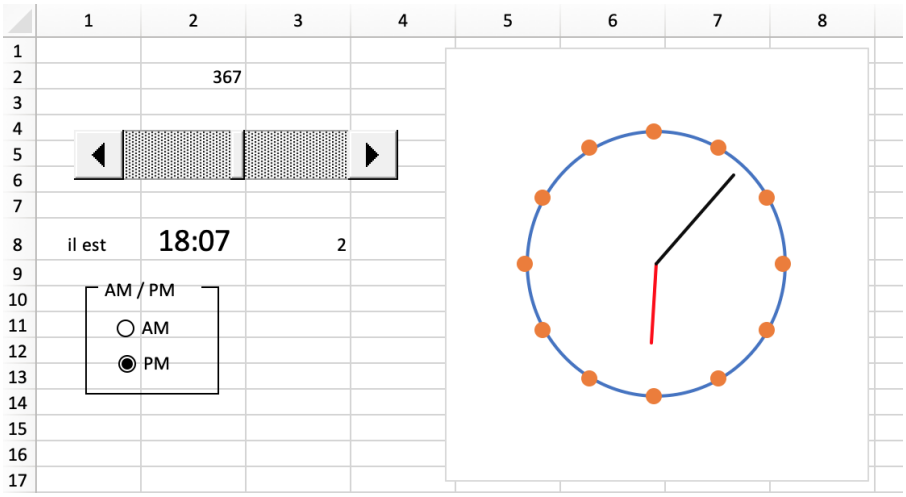
Il faudrait pouvoir choisir entre un affichage 12h ou 24h. Une *zone de groupe* avec deux *cases d'option* [3] fera l'affaire.

On associe la première case d'option créée à AM et la seconde à PM ; la cellule `L8C3` est nommée *ampm*. Elle contiendra 1 si c'est l'option AM qui est choisie et 2 si c'est l'option PM.

Dans la formule de la cellule *heures*, il suffit de tester si *ampm* vaut 2, auquel cas on ajoute 12. Et dans ce cas, le résultat ne peut plus être strictement inférieur à 10.

La cellule *heures* contient maintenant la formule :
`SI(ampm=2;ENT(min/60)+12;SI(ENT(min/60)<10;"0"&ENT(min/60);ENT(min/60)))`.

Voici le résultat final :



Références

[1] Desbonnez J.-M., Tableur et barre de défilement, graphiques animés : cycloïde, épicycloïde, hypocycloïde. *Losanges*, 38, pp. 47–59, 2017.

[2] Desbonnez J.-M., Mathématiques et smiley. *Losanges*, 39, pp. 51–53, 2017.

[3] Desbonnez J.-M., Tableur Excel et contrôles de formulaires. *Losanges*, 46, pp. 57–62, 2019.

Le problème de Flavius Josèphe

Petite histoire ou grande Histoire... Joseph BEN MATTATHIAS (devenu Flavius JOSEPH après avoir obtenu la citoyenneté romaine, puis orthographié en français Flavius JOSÈPHE au XVII^{e} siècle) est un historien juif du I^{er} siècle.

Lors de la prise de la garnison juive de la forteresse de *Jotapata* (actuelle *Yodfat*) par l'empereur romain VESPASIEN en 67, il est pris au piège dans une grotte avec quarante de ses compagnons.

Ceux-ci refusent de se rendre aux Romains et décident d'un suicide collectif. Légende ou non, un tirage au sort est choisi pour fixer l'ordre dans lequel ils se donneront mutuellement la mort.

Tous se placent en cercle, une personne est désignée comme numéro 1, puis 2, 3, ... jusqu'à 41. La personne située toutes les 3 positions doit être sacrifiée par celle située juste après elle, et ainsi de suite jusqu'à ce qu'il ne reste plus qu'un seul survivant, censé se suicider.

Josèphe se plaça en 31^{e} position...

1 Solution manuelle

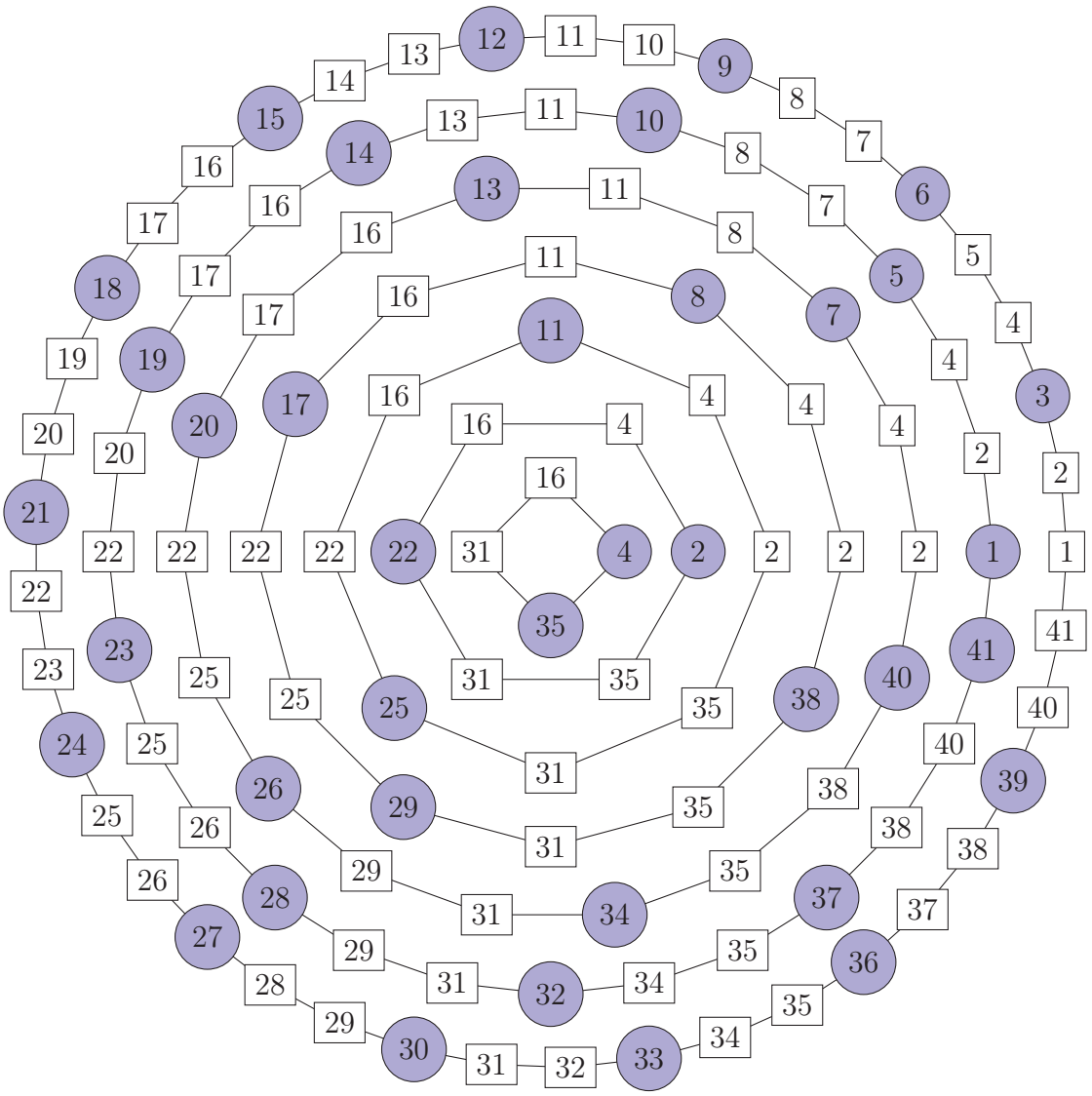
Dans la figure qui suit, on a visualisé le macabre scénario. Sur le cercle extérieur sont représentées les 41 personnes initiales, et sur fond coloré, celles qui sont sacrifiées.

On trouve ensuite dans le premier cercle intérieur les 28 survivants du « premier tour », et sur fond magenta, ceux qui n'arriveront pas au troisième tour, et ainsi de suite.

On passe ainsi de 41 survivants à 28, puis 18, 12, 8, 6, 4, puis il n'en reste plus que 2, Josèphe et son pote... on peut supposer qu'après une courte délibération, ils décident de ne pas s'entretuer, et de rester vivants pour porter le drapeau blanc.

Cette manière de solutionner le problème [(1)] est certes relativement simple, mais lourde à mettre en œuvre (et je passe sous silence le temps à réaliser le dessin, contenant un carré, un hexagone, un octogone, un dodécagone, un octakaidecagone, un icosikaioctagone et un eenenveertigone).

1. Je me rends compte que j'ai oublié de poser le problème : où se placer dans la suite pour rester en vie ? Et s'il y avait plus de 41 personnes ? Et si le sacrifice se fait de 4 en 4 ?... Et si... une petite pensée pour Bernard.



2 Solution informatique

Le fait d'avoir d'abord traité le problème « manuellement » va nous rendre service, comme c'est souvent le cas. Le problème n'est autre qu'un problème de *liste* et de *position dans une liste.*

2.1 Première version en Python

La liste de départ, que nous nommerons *soldats* aura la forme `soldats[1, 2, 3, 4,..., 41]`. La plupart des langages modernes indexent à zéro le premier élément d'une liste. On a donc `soldats[0]=1`, `soldats[1]=2`, ..., `soldats[40]=41`.

L'instruction *Python* qui permet de créer cette liste est `soldats=list(range(1,42))` ; le dernier élément de la fonction `range` n'est pas compris dans la liste engendrée ; la différence entre les deux paramètres donne le nombre d'éléments engendrés.

L'instruction `len(soldats)` calcule la longueur de la liste, à savoir le nombre d'éléments qu'elle contient.

L'instruction `del soldats[position]` supprime de la liste l'élément d'index `position`, et de

ce fait diminue de 1 sa longueur.

Il « suffit » maintenant de supprimer un à un les éléments de la liste tant que sa longueur dépasse strictement 1. Lorsqu'il ne reste plus qu'un seul élément, il suffit de l'imprimer. Dans ce cas, les instructions `print (soldats)` ou `print (soldats[0])` sont équivalentes.

Pour supprimer le bon élément, il faut gérer un compteur que l'on a appelé `position`. Il est initialisé à zéro (premier index de la liste), et pour son calcul, il faut (si on souhaite généraliser plus tard le processus) faire intervenir le fait que les éliminations s'opèrent toutes les 3 positions.

L'instruction `position=position+3-1` (on retire 1 car l'indexation commence à 0, et on fait apparaître 3) permet d'atteindre `soltats[2]`, l'élément 3. Mais, une fois que cet élement aura été supprimé, il y aura un décalage vers la gauche et la liste `soldats` vaudra [1, 2, 4, 5, 6, ..., 41] de longueur 40.

L'élément suivant à supprimer sera 6, qui sera en position 4, c'est-à-dire `soldats[4]` ; il reste alors [1, 2, 4, 5, 7, 8, ..., 41] de longueur 39.

Le suivant, c'est l'élément 9, qui sera en position 6, etc.

Donc, en ajoutant $3-1$ (c'est-à-dire 2) à la variable `position`, on s'en sort bien... MAIS...

Si on dresse la liste des couples « position–longueur », on a successivement 2–40, 4–39, 6–38, 8–37, 10–36, 12–35, 14–34, 16–33, 18–32, 20–31, 22–30, 24–29, 26–28, ce qui correspond au « premier tour » ; 13 éléments ont été supprimés.

Il faut maintenant entamer le « deuxième tour ». L'élément suivant à supprimer est 1, et il est le premier de la liste, donc en position 0. La suite des couples « position-longueur » doit être poursuivie par 0–27, 2–26, 4–25, 6–24, 8–23, 10–22, 12–21, 14–20, 16–19, 18–18. Fin du « deuxième tour » ; 10 éléments ont été supprimés.

Viennent ensuite les séries 2–17, 4–16, 6–15, 8–14, 10–13, 12–12, puis 2–11, 4–10, 6–9, 8–8, puis 2–7, 4–6, puis 0–5, 2–4, puis 0–3, 2–2, et enfin 0–1.

La variable `position` ne peut pas dépasser la longueur de la liste (restante) ; il faut donc se repositionner au début de la liste dès que l'on dépasse le dernier élément.

L'opération arithmétique qui « remet les compteurs à zéro » est **modulo**. Dans la plupart des langages de programmation elle se traduit par le symbole %,
d'où l'instruction `position=(position+3-1)%len(soldats)`.

Voici le script *Python* ; son exécution produit le résultat 31 [(2)].

```
# josephe.py
soldats=list(range(1,42))
position=0
while len(soldats)>1:
    position=(position+3-1)%len(soldats)
    del soldats[position]
print (soldats)
```

Et si... il y avait 1000 personnes au départ, et que la boucherie s'opère par pas de 5, je ne fais pas de dessin, mais il suffit d'adapter le script :

2. Dans ce script, Joseph tue son pote.

```
# josephe.py
soldats=list(range(1,1001))
position=0
while len(soldats)>1:
    position=(position+5-1)%len(soldats)
    del soldats[position]
print (soldats)
```

Et le gagnant est... 763.

Généralisation

Pour ne pas devoir modifier le script pour chaque essai que l'on souhaite exécuter, on peut définir une *fonction* avec les deux paramètres qui peuvent varier, à savoir le nombre de personnes à traiter et le pas. Voici :

```
# josephe2.py

def josephe(nb_personnes,pas):
    soldats=list(range(1,nb_personnes+1))
    position=0
    while len(soldats)>1:
        position=(position+pas-1)%len(soldats)
        del soldats[position]
    return (soldats)

print (josephe(1000,5))
```

Et on peut encore aller plus loin : si on accorde la grâce à plus d'une personne ? Il suffit d'ajouter un troisième paramètre à la fonction :

```
# josephe3.py

def josephe(nb_personnes,pas,survivants):
    soldats=list(range(1,nb_personnes+1))
    position=0
    while len(soldats)>survivants:
        position=(position+pas-1)%len(soldats)
        del soldats[position]
    return (soldats)

print (josephe(997,7,4))
```

Et les gagnants sont... 82, 276, 383 et 526.

2.2 Deuxième version en Ruby

Le changement de langage de programmation n'est rien d'autre qu'une traduction de la syntaxe. Les explications qui ont été données pour *Python* sont évidemment les mêmes pour *Ruby*.

Les instructions de manipulations de listes (tableaux) ont été longuement détaillées dans [1] et [2]. Voici donc le script version *Ruby* :

```
# josephe.rb

soldats=[]
for i in 1..41
    soldats.push(i)
end
position=0
while soldats.size > 1
    position=(position+3-1)%soldats.size
    soldats.delete_at(position)
end
puts soldats
```

2.3 Troisième version en Excel

Il est vrai qu'à priori le traitement de ce genre de problème n'est pas la spécialité d'un tableur, mais... soyons un peu fous !

Si nécessaire, le lecteur est invité à se rafraîchir un peu la mémoire avec les articles traitant de la programmation en *VBA* du tableur *Excel* dans [3], [4] et [5].

Commençons par la version de base, 41 personnes, élimination 3 par 3.

La première idée qui vient à l'esprit est d'écrire les entiers 1 à 41 dans la plage `L1C1:L41C1`. Voici une manière de faire :

```
Option Explicit

Sub init()
    Dim k As Integer
    For k =1 to 41
        Cells(k, 1) = k
    Next
End Sub
```

Il est sage d'associer un bouton à cette macro (voir [3]), en tout cas tant que l'on est en phase d'essais.

Ensuite, il faut éliminer [3]. On peut, pour ce faire, supprimer le contenu des cellules `L3C1`, `L6C1`, `L9C1`, etc. ce qui correspond aux nombres 3, 6, 9, ... 39. C'est facile pour le « premier tour », mais cela laisse des « trous » dans la liste, et c'est mission impossible de gérer les adresses pour les suppressions suivantes.

3. Et dans ce cas-ci, boire ne suffit pas !

Les cellules vides doivent être comblées en faisant « remonter » le contenu des cellules qui suivent... et il y a une opération qui fait cela tout naturellement, c'est la suppression d'une ligne. Si on supprime la ligne 3, qui ne contient rien, la 4 devient 3, et ainsi de suite, mais on ne voit rien ! Si la ligne 3 contient des données, celles-ci disparaissent, la ligne 4 devient 3, et c'est le contenu de la ligne 4 qui remplace le contenu de la ligne 3. Et le tour est joué, on pourra quasi récupérer l'algorithme *Python* et *Ruby*, et faire la traduction en *VBA*.

```
Option Explicit
Sub josephe()
    Dim reste, position As Integer
    reste = 41
    position = 0
    Do While reste > 1
        position = (position + 3 - 1) Mod reste
        Cells(position + 1, 1).Select
        Selection.EntireRow.Delete
        reste = reste - 1
    Loop
End Sub
```

La variable `reste` joue le rôle de longueur de la liste ; elle est initialisée à 41 au départ, et est diminuée de 1 unité à chaque suppression. Le processus de suppression va s'arrêter lorsqu'elle sera égale à 1, et comme toutes les données auront disparu, sauf une, celle-ci se retrouvera dans la cellule `L1C1`, ce qui va correspondre au numéro du dernier survivant.

Pour supprimer une ligne, il faut qu'une cellule quelconque de cette ligne soit sélectionnée. On prend celle qui se trouve dans la première colonne, celle qui contient le nombre à supprimer.

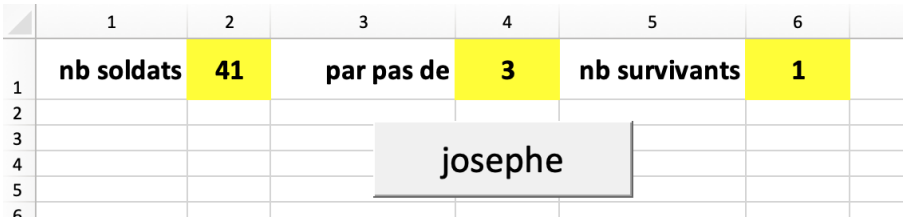
Finalement la plus grande différence avec les autres langages, c'est qu'en *Python* et *Ruby*, le premier élément de la liste est en position 0, tandis que dans une feuille de calcul, la première ligne porte le numéro... 1 ; d'où l'instruction `Cells(position+1,1).Select`.

Après exécution de la macro, sans surprise, la cellule `L1C1` contient le nombre 31.

Généralisation

Il faut que l'on puisse choisir de manière assez souple les 3 données du problème (le nombre initial de personnes, la manière de faire les éliminations et le nombre de survivants souhaités). La figure ci-après propose un modèle de présentation, les 3 données étant respectivement dans les cellules `L1C2`, `L1C4` et `L1C6`.

Comme des lignes vont être supprimées, il est vivement conseillé de travailler avec deux feuilles : la première, nommée `josephe`, contenant la présentation (pérenne) ci-après, et la seconde, que nous avons appelée `calculs`, qui contiendra la liste (que nous ne verrons pas) des nombres 1 à ..., et en fin de calcul les numéros des survivants.



```
Option Explicit
Public reste As Integer
'------
Sub josephe()
    Dim position, pas, survivants As Integer
    Sheets("josephe").Select
    reste = Cells(1, 2)
    pas = Cells(1, 4)
    survivants = Cells(1, 6)
    position = 0
    Sheets("calculs").Select

    init (reste)

    Do While reste > survivants
        position = (position + pas - 1) Mod reste
        Cells(position + 1, 1).Select
        Selection.EntireRow.Delete
        reste = reste - 1
    Loop
End Sub
' -------
Sub init(x)
    Dim k As Integer
    For k = 1 To x
        Cells(k, 1) = k
    Next
End Sub
```

La macro `init` est appelée par la macro `josephe` (et donc il n'y a plus qu'un seul bouton qui est nécessaire) pour afficher la liste des nombres. Elle doit posséder maintenant 1 paramètre, le nombre d'éléments de la liste. La variable `reste` doit être déclarée *publique* car utilisée dans les deux procédures (`josephe` et `init`).

Il faut toutefois noter que la version *Excel* est plus lente que les versions *Python* et *Ruby* (quasi instantanées), la suppression des lignes étant assez chronophage.

3 Le problème « 15–15 »

Dans [6], on trouve une recherche historique assez poussée à propos du problème de Josèphe, et aussi d'un autre problème, très lié, mentionné dans un ouvrage de Tartaglia au XVI^{e} siècle et repris par un certain Édouard Lucas dans un ouvrage — *Artithmétique amusante* — publié en 1895, le problème appelé « 15–15 ».

Il y plusieurs versions, dont une faisant intervenir des caractères religieux, mais pour ne froisser personne, disons que sur un bateau il y a 15 *Shadocks* et 15 *Gibis*[4]. Pour éviter le naufrage, le capitaine décide de jeter par dessus bord la moitié des passagers, en désignant un « volontaire »

4. Série télévisée en 208 épisodes de 3 minutes, diffusée entre 1968 et 1973 sur l'ORTF, ancienne chaîne de télévision française.

toutes les 9 positions. Les *Shadocks* étant les seuls à savoir ramer [5], comment doivent-ils être disposés pour n'être que les seuls survivants ?

Ce n'est rien d'autre que le problème de Josèphe, dont la solution est donnée par la fonction `josephe(30,9,15)` : les *Shadocks* doivent occuper les positions 1, 2, 3, 4, 10, 11, 13, 14, 15, 17, 20, 21, 25, 28 et 29.

Des moyens mnémotechniques ont été inventés pour mémoriser ces positions.

Voici les intéressés...

Références

[1] Desbonnez J.-M., Algorithmique et langage Ruby. Les tableaux (1). *Losanges*, 31, pp. 45–54, Décembre 2015.

[2] Desbonnez J.-M., Algorithmique et langage Ruby. Les tableaux (2). *Losanges*, 32, pp. 53–58, Mars 2016.

[3] Desbonnez J.-M., Tableur Excel et macro-instructions (1). *Losanges*, 43, pp. 56–63, Décembre 2018.

[4] Desbonnez J.-M., Tableur Excel et macro-instructions (2). *Losanges*, 44, pp. 45–57, Mars 2019.

[5] Desbonnez J.-M., Tableur Excel et macro-instructions (3). *Losanges*, 45, pp. 48–59, Juin 2019.

[6] http://www.bibnum.education.fr/sites/default/files/flavius-analyse-1.pdf.

5. D'habitude, ils pompent...

LaTeX, la classe beamer

Résumé. *La classe* BEAMER *est une bibliothèque d'instructions (au même titre que les classes* ARTICLE, BOOK, *etc.) qui permet de créer et de gérer ce que l'on appelle une* présentation, *support informatique à un exposé, un cours, une conférence, projeté sur un écran sous la forme de* diapositives *(*slides*) successives. Voici de quoi démarrer ; si affinité, on trouve beaucoup d'info sur le net !*

1 Pourquoi, avec quoi ?

À la question « pourquoi utiliser une présentation ? » il y a beaucoup de réponses, toutes aussi pertinentes les unes que les autres :

- il n'y a pas de tableau ;
- je suis allergique à la craie ;
- je dois présenter des images, des vidéos, des dessins complexes ;
- c'est à la mode ;
- la navigation entre les diapositives est plus souple que la recherche du 17e transparent ;
- ...

À la question « avec quoi créer une présentation ? » il y a plusieurs logiciels ; parmi les plus connus, citons :

- **PowerPoint** : il fait partie de la suite bureautique *Microsoft Office*, c'est sans doute le plus ancien sur le marché ; il y a une version pour Windows et une version pour MacOs, plus ou moins compatibles...
- **Impress** : c'est le pendant de *PowerPoint* de la suite bureautique libre *Open Office*, disponible pour Windows, MacOs et Linux.
- **Keynote** : il fait partie de la suite bureautique *iWork* et fonctionne uniquement sous MacOs.

Ces logiciels, tous du même crû, ont leurs avantages et inconvénients.

Citons parmi les avantages, leur prise en main rapide, les mises en page simples, des effets de transition plus ou moins spectaculaires entre les diapositives, ...

Parmi les inconvénients, il y a la nécessité d'avoir sur l'ordinateur du lieu de présentation le « logiciel mère » qui a servi à la créer, la gestion manuelle de la navigation entre les diapositives (pas de gestion automatique des hyper liens internes), le manque de moyens pour créer des *formules* de type mathématique lorsque le sujet est scientifique, pas de création automatique d'une table des matières, pas de barre de navigation qui contient la structure logique de la présentation, ...

Le lecteur futé aura évidemment compris que tous ces inconvénients sont les points forts de la classe *beamer* sous LaTeX. Le fichier produit est un *.pdf*, format universel, et les scientifiques n'utilisent pas d'effets spectaculaires de transition, ni de multiples polices de caractères.

La messe est dite.

Bon, c'est vrai, il faut quelques notions de LaTeX... ceci s'adresse donc à celles et ceux qui les possèdent, qui ne connaissent pas la classe *beamer*, et qui souhaitent évidemment la découvrir.

Beamer est donc une classe LaTeX servant à produire des documents de type « présentation ». Elle a été proposée par Till Tantau en 2003 et développée ensuite par de nombreux auteurs.

En plus des instructions qui lui sont propres, on peut y utiliser presque toutes les instructions LaTeX, en particulier *section* et *subsection* pour la structure de la présentation, ainsi que les environnements *itemize* et *enumerate* pour les listes, structures très utilisées dans ce type de document.

Le résultat est un fichier au format *pdf*. Le logiciel *Adobe Acrobat Reader* sert de « visionneuse », particulièrement si l'on se sert d'un petit « boîtier-pointeur » pour gérer à distance le déroulement des diapositives.

2 Premiers pas, une diapositive de titre

Thème *PaloAlto*

Thème *AnnArbor*

2.1 Le code

Comme dans tout document LaTeX, il y a un préambule et le corps du document. Voici un préambule minimum et la création d'une diapositive de titre. Les lignes sont numérotées pour faciliter les commentaires (les numéros de ligne ne font pas partie du code LaTeX).

```
1  \documentclass[aspectratio=1610]{beamer}
2  \usepackage[utf8]{inputenc}
3  \usepackage[T1]{fontenc}
4  \usepackage[french]{babel}
5
6  % choix du thème global
7  %\usetheme{PaloAlto}
8  \usetheme{AnnArbor}
9
10 \hypersetup{pdfpagemode=FullScreen}
11
12 \title[Beamer]{\LaTeX \, et la classe \textit{BEAMER}}
13 \subtitle{pour des présentations de haute qualité}
14 \author[J.-M. Desbonnez]{Jean-Marc \\  DESBONNEZ}
15 \institute[SBPMef]{\textit{SBPMef \\Société Belge des Professeurs
16 de Mathématique\\d'expression française}}
17 \date{Mars 2020}
18 \logo{\includegraphics[scale=0.15]{sbpm.png}}
19
20 \begin{document}
21
22 % diapositive de titre
23 \begin{frame}
24         \maketitle
25 \end{frame}
26
27 \end{document}
```

2.2 Commentaires et explications

1 Par défaut, la dimension d'une diapositive est 128mm×96mm, ce qui correspond à *ratio=43*, c'est-à-dire 4 : 3. La taille la plus grande est 160mm×100mm, ce qui correspond à *ratio=1610*, c'est-à-dire 16 : 10. Il y a d'autres possibilités, voir [3].

2 à 4 Trois packages traditionnels pour le type d'encodage et la langue ; il suffit d'ajouter ce qui est nécessaire en fonction du contenu. *Beamer* charge automatiquement les packages geometry, graphicx, xcolor, pgf, hyperref, amssymb, amsmath, amsfonts, amsthm, translator et enumerate.

6 à 8 Le thème général régit, entre autres, l'harmonie des couleurs, l'aspect des titres, la présence ou non d'une barre de navigation, d'un sommaire, etc. Si on est tatillon, on peut évidemment tout modifier... en appliquant les bonnes instructions. Il suffit donc de modifier une seule ligne pour donner à la présentation un aspect totalement différent. C'est une histoire de goût et de couleurs, donc on ne discute pas.

Voici les noms de quelques thèmes et leur particularité :

- sans barre de navigation : Pittburgh, Rochester, Bergen, Boadilla, Madrid, AnnArbor, CambridgeUS
- avec barre de navigation : Montpellier, Antibes, JuanLesPins
- avec sommaire latéral : Göttingen, Hannover, Marburg, Berkeley, PaloAlto

- avec mini cadre de navigation : Singapore, Szeged, Berlin, Dresden, Frankfurt
- avec sommaire des sections et sous-sections : Malmoe, Copenhagen, Luebeck, Warsaw

Voir [2] et [3] pour plus de détails.

Les deux copies d'écran qui suivent montrent ce qui résulte du code qui précède, l'une avec le thème *PaloAlto*, l'autre avec le thème *AnnArbor*.

10 Cette instruction force le passage automatique en mode plein écran dès l'ouverture de la présentation. La compilation du code qui précède produit un fichier au format *.pdf* qui DOIT être ouvert avec le logiciel *Adobe Acrobat Reader* (gratuit) pour rendre actifs les symboles de navigation et permettre aussi la navigation entre les diapositives avec un dispositif présent sur les *pointeurs laser* modernes.

12 à 17 Les instructions habituelles pour préparer les champs d'une page de titre ; [en option, la version « courte »]

18 L'instruction qui permet d'afficher un logo sur toutes les diapositives. La position du logo dépend du thème choisi.

23 à 25 Le contenu d'une diapositive est encadré par la structure `\begin{frame} ... \end{frame}`. On verra par la suite qu'il y a des options possibles, non nécessaires pour la diapositive de titre. L'instruction `\maketitle` met en forme les champs définis dans les lignes 12 à 18, comme dans tout document LaTeX.

Comme le montrent les images qui suivent, tout est centré, aussi bien verticalement qu'horizontalement. On ne s'occupe de rien.

Les symboles de navigation, affichés discrètement, sont automatiques ; on peut personnaliser la barre (sera traité plus loin).

Pour la suite, nous continuerons avec l'un de ces deux thèmes. Au lecteur de tester les autres, la curiosité n'est parfois pas un vilain défaut.

Pour la petite histoire [4]...

Palo Alto est une ville californienne, rendue célèbre par le *Palo Alto Research Center*, fondation où sont mises au point de nombreuses innovations informatiques. Un certain Steve Jobs y a élu domicile.

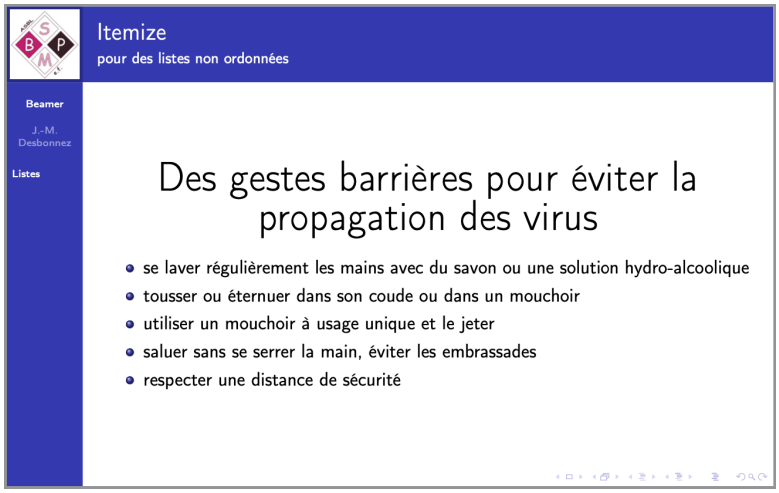
Ann Arbor est une ville du Michigan où l'économie est également centrée sur les hautes technologies.

3 Itemize et enumerate

Les listes sont des objets couramment utilisés dans les présentations. Beamer permet l'utilisation des trois types de listes standards sous LaTeX : itemize, enumerate, et description (ne sera pas traité ici), avec la même syntaxe.

Dans les codes qui suivent, les lignes sont numérotées à partir de 1 pour la facilité, mais elles doivent bien être intégrées entre le `\begin{document}` et le `\end{document}`.

3.1 Itemize




```
1   \section{Listes}
2   % slide itemize
3   \begin{frame}
4     \frametitle{Itemize}
5     \framesubtitle{pour des listes non ordonnées}
6     \begin{center}
7       {\Huge Des gestes barrières pour éviter la propagation des virus}
8     \end{center}
9     \begin{itemize}
10      \item se laver régulièrement les mains avec du savon
11      ou une solution hydro-alcoolique
12      \item tousser ou éternuer dans son coude ou dans un mouchoir
13      \item utiliser un mouchoir à usage unique et le jeter
14      \item saluer sans se serrer la main, éviter les embrassades
15      \item respecter une distance de sécurité
16    \end{itemize}
17  \end{frame}
```

1 Pour donner une structure logique à notre présentation. Voir dans l'image ci-après le plan de la structure.

4 et 5 Les instructions sont suffisamment explicites.

6 à 8 Par défaut, le contenu d'une diapositive (sauf celle de titre) est aligné à gauche. Si on souhaite le centrer, il faut le demander. Par contre, la globalité du contenu est toujours centré verticalement.

Trois niveaux d'items sont possibles. La forme et la couleur des puces dépend du thème choisi, mais sont modifiables. On peut obtenir des cercles, des triangles, des carrés ou des sphères, au niveau choisi, au moyen des instructions suivantes :

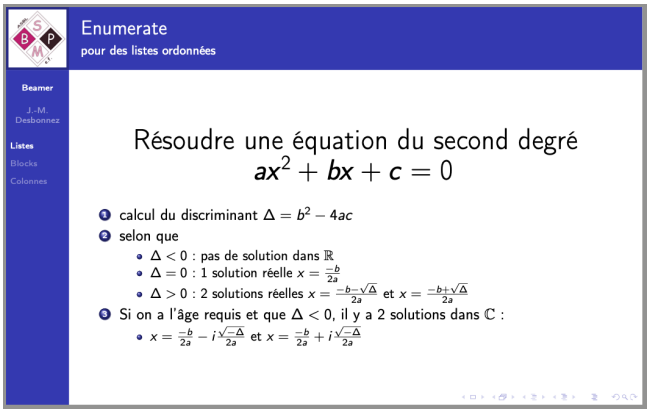
- `\setbeamertemplate{itemize item}[triangle]`
- `\setbeamertemplate{itemize subitem}[circle]`
- `\setbeamertemplate{itemize subsubitem}[ball]`

- `\setbeamertemplate{...}[square]`

On peut en modifier la taille : `\setbeamertemplate{itemize item}{size=\huge}` par exemple. La couleur est traitée plus loin.

Ces instructions sont à placer soit dans le préambule pour avoir un effet dans toute la présentation, soit localement, mais il faut en mettre la portée entre accolades.

3.2 Enumerate




```
1  % slide enumerate
2  \begin{frame}
3    \frametitle{Enumerate}
4    \framesubtitle{pour des listes ordonnées}
5    \begin{center}
6      {\huge Résoudre une équation du second degré \\
7      $ax^2+bx+c=0$}
8     \end{center}
9    \bigskip
10
11   \begin{enumerate}
12     \item calcul du discriminant $\Delta=b^2-4ac$
13     \item selon que
14       \begin{itemize}
15        \item $\Delta < 0$~: pas de solution dans $\mathbb{R}$
16        \item $\Delta = 0$~: 1 solution réelle $x=\frac{-b}{2a}$
17        \item $\Delta > 0$~: 2 solutions réelles $x=\frac{-b -\sqrt{\Delta}}{2a}$
18         et $x=\frac{-b + \sqrt{\Delta}}{2a}$
19       \end{itemize}
20     \item Si on a l'âge requis et que $\Delta < 0$,
21         il y a 2 solutions dans $\mathbb{C}$~:
22     \begin{itemize}
23       \item $x=\frac{-b}{2a}-i\frac{\sqrt{-\Delta}}{2a}$
24       et $x=\frac{-b}{2a}+i\frac{\sqrt{-\Delta}}{2a}$
25     \end{itemize}
26   \end{enumerate}
27 \end{frame}
```

4 Les bloc(k)s

Un bloc (*block* dans l'instruction) est une structure qui permet de mettre un contenu en évidence. Il est constitué d'un titre éventuel (*title*) et d'un corps (*body*). Le rendu d'un bloc dépend du thème choisi (mais on peut tout modifier).

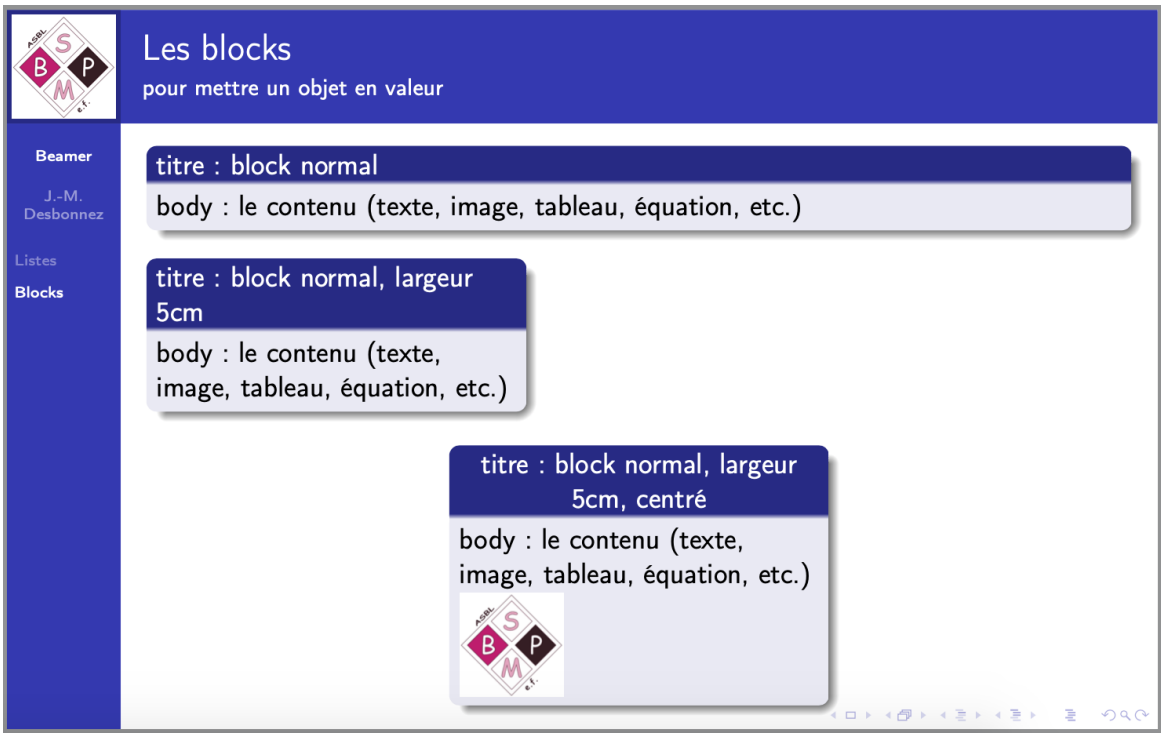
Par défaut, la largeur d'un bloc est la largeur de la diapositive ; pour une largeur au choix, il suffit de la placer dans une minipage de largeur souhaitée, ou dans une colonne (voir plus loin).

Dans un bloc, le titre et le corps sont par défaut alignés à gauche. Si on souhaite un alignement centré, il faut le demander.

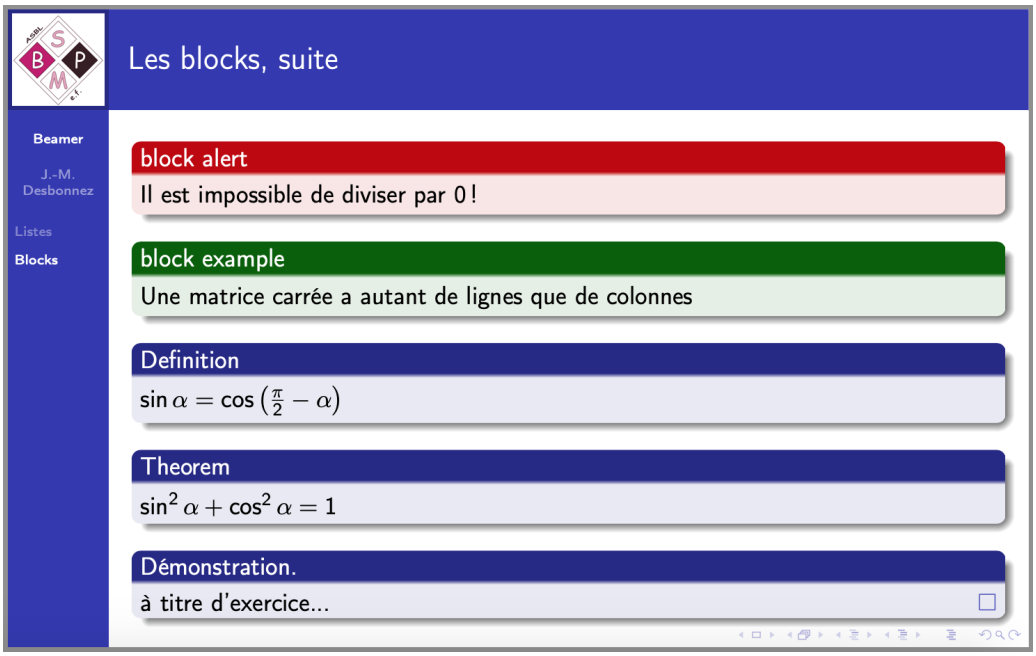
Il y a plusieurs environnements prédéfinis : *block*, *alertblock*, *exampleblock*, mais aussi des environnement prédéfinis qui ont à peu près la même fonction :

- `\begin{definition} ... \end{definition}`
- `\begin{theorem} ... \end{theorem}`
- `\begin{proof} ... \end{proof}`

Dans certains thèmes, il n'y a pas de cadre dessiné.



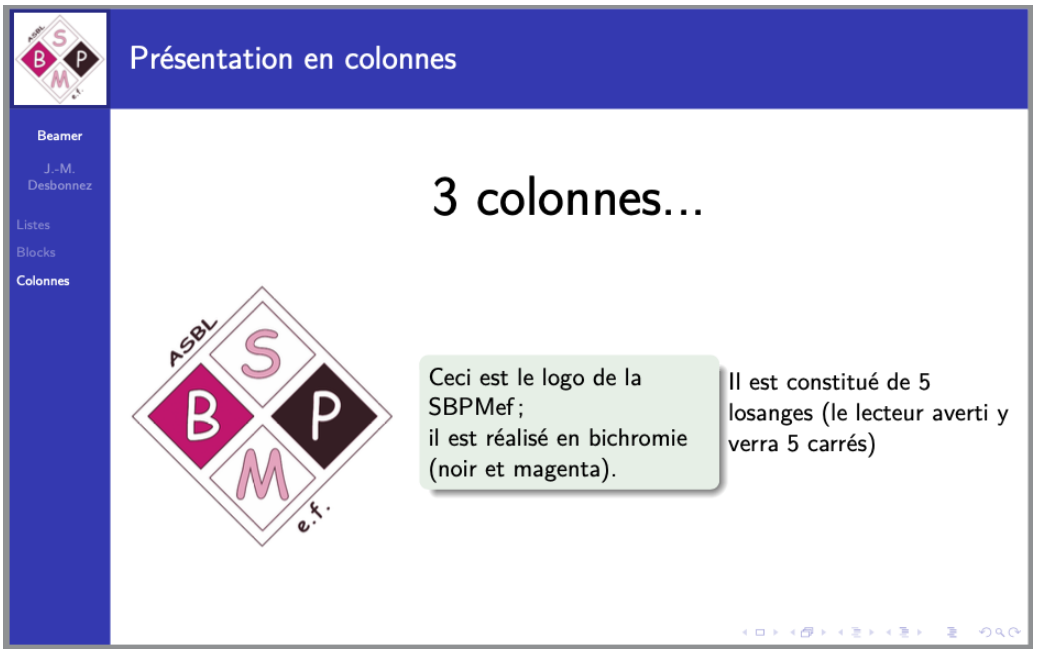
```latex
\section{Blocks}
% slide blocks
\begin{frame}
  \frametitle{Les blocks}
  \framesubtitle{pour mettre un objet en valeur}

  \begin{block}{titre : block normal}
    body : le contenu (texte, image, tableau,     équation, etc.)
  \end{block}

  \begin{minipage}{5cm}
    \begin{block}{titre : block normal, largeur 5cm}
    body : le contenu (texte, image, tableau, équation, etc.)
    \end{block}
  \end{minipage}

  \begin{center}
    \begin{minipage}{5cm}
      \begin{block}{\centering titre : block normal, largeur 5cm, centré}
        body : le contenu (texte, image, tableau, équation, etc.)\\
        \includegraphics[scale=0.15]{sbpm.png}
      \end{block}
    \end{minipage}
  \end{center}
\end{frame}
```



```
\begin{frame}
\frametitle{Les blocks, suite}

\begin{alertblock}{block alert}
Il est impossible de diviser par 0 !
\end{alertblock}

\begin{exampleblock}{block example}
Une matrice carrée a autant de lignes que de colonnes
\end{exampleblock}

\begin{definition}
$\sin \alpha = \cos \left(\frac{\pi}{2}-\alpha\right)$
\end{definition}

\begin{theorem}
$\sin^2 \alpha + \cos^2 \alpha =1$
\end{theorem}

\begin{proof}
à titre d'exercice...
\end{proof}

\end{frame}
```

5 Les colonnes

Beamer propose un environnement `columns` qui permet de placer des objets (texte, tableau, graphique, équation, etc.) les uns à côté des autres, tout comme l'environnement `minipage` associé à la commande `\hfill`. On peut choisir la largeur des colonnes.



```
\section{Colonnes}
\begin{frame}
  \frametitle{Présentation en colonnes}

  {\centerline{\Huge 3 colonnes...}}
  \vspace{1cm}

  \begin{columns}

    \begin{column}{0.33\textwidth}
      \includegraphics[width=0.95\linewidth]{sbpm.png}
    \end{column}

    \begin{column}{0.33\textwidth}
      \begin{exampleblock}{}
      Ceci est le logo de la SBPMef~; \\
      il est réalisé en bichromie (noir et magenta).
      \end{exampleblock}
    \end{column}

    \begin{column}{0.33\textwidth}
      Il est constitué de 5 losanges (le lecteur averti y verra 5 carrés)
    \end{column}

  \end{columns}
\end{frame}

```

15 block sans titre

Attention à ne pas confondre l'environnement global `columns` (au pluriel) qui contient les colonnes individuelles `column` (au singulier).

6 Les couleurs

Le choix des couleurs est déjà intimement lié au choix du thème. Si cela suffit, c'est tant mieux ; et dans ces conditions, il y a finalement peu de code à maîtriser pour obtenir rapidement un bon résultat.

S'il est nécessaire de modifier certaines couleurs, il est conseillé, afin de préserver une uniformité dans les diapositives, d'opérer ces changements dans le préambule. En effet, la multiplicité des couleurs, des polices, des tailles, etc. accentue l'effet « kitch » et détourne l'attention du public au détriment du contenu. Mais c'est un avis personnel.

Les instructions qui suivent seront expliquées sans copie d'écran associée. Au lecteur de se faire une idée... en essayant.

Rappelons que *Beamer* charge automatiquement le package *xcolor*, ce qui permet

- d'utiliser une série de couleurs prédéfinies (blue, red, magenta, pink, purple, etc.) ;
- de personnaliser des couleurs aux formats *rgb*, *cmyk* ou *gray*,
  par exemple `\definecolor{carotte}{cmyk}{0 0.56 0.85 0.05}` ;

- d'ajuster une couleur : de clair (!0) à foncé (!100),
  par exemple `\colorlet{rougeclair}{red!40}`.

Les aficionados de la couleur n'ont donc pas d'inquiétude à avoir.

Avec *beamer*, le choix d'une couleur se fait à deux niveaux :

- une couleur d'avant plan, notée *fg* (foreground), qui fixe la couleur de l'élément sur lequel elle porte (texte, puce d'item, etc.) ;
- une couleur d'arrière plan, notée *bg* (background).

6.1 Thèmes de couleurs

La première chose à faire avant de modifier toute une série de couleurs est de jeter un oeil dans les « thèmes de couleurs ». L'instruction `\usecolortheme{nom_du_thème}` permet de choisir un ensemble de couleurs prédéfinies pour tous les éléments de la présentation. Il y a par exemple *albatros*, *beetle*, *wolverine*, etc. Voir [2] pour d'autres thèmes et les rendus... en couleur. Le simple changement de thème de couleurs donne à la présentation un aspect tout à fait différent.

6.2 Couleur des blocks

Pour modifier les couleurs du titre et du corps :

- `\setbeamercolor{block title}{fg=couleur,bg=couleur}`
- `\setbeamercolor{block body}{fg=couleur,bg=couleur}`

Cas particulier : on peut modifier la couleur d'un bloc d'alerte (la couleur par défaut dépend du thème, rouge pour *PaloAlto*, bleu pour *AnnArbor*, ...) avec l'instruction `\setbeamercolor{alerted text}{couleur}`.

La couleur choisie influence également l'instruction `\alert{texte}` qui écrit `texte` dans cette couleur, ce qui est la façon la plus rapide pour mettre un texte en évidence.

6.3 Couleur et taille des items

- `\setbeamercolor{itemize item}{fg=couleur}`
- `\setbeamercolor{itemize subitem}{fg=couleur}`
- `\setbeamercolor{itemize subsubitem}{fg=couleur}`

Inopérant sur des items de type « ball »

7 La navigation

Pour que la présentation soit opérationnelle, il faut l'ouvrir avec *Adobe Acrobat Reader* ; le but est évidemment de faire défiler les diapositives. Il y plusieurs manières de faire.

Si la présentation est simple et linéaire (séquentielle), il y a

- le clavier
  - → pour avancer d'une dia : flèche droite, barre d'espace, clic
  - → pour reculer d'une dia : flèche gauche, clic droit
- un dispositif pointeur laser

Si la présentation est longue, si sa structure est complexe, si l'auditoire est amené à faire des commentaires nécessitant une navigation très souple, il y a la *barre de navigation* présente en bas de chaque diapositive. Elle est composée de 6 icônes :

1 navigation par page *pdf* (dans la partie consacrée aux *overlays*, on verra comment une même diapositive peut être composées de plusieurs pages)

2 navigation par diapositive

3 navigation par début de sous-section

4 navigation par début de section

5 fin de la présentation, juste avant l'annexe, si elle existe

6 navigation dans le fichier *pdf* complet : retour, recherche, suivant (la touche esc permet de quitter ce mode de navigation)

La barre de navigation est personnalisable.
Pour ôter la barre : `\setbeamertemplate{navigation symbols}{}`
Pour n'y faire apparaître que les icônes souhaitées :
`\setbeamertemplate{navigation symbols}{liste des icônes}` avec

```
(1) \insertslidenavigationsymbol
(2) \insertframenavigationsymbol
(3) \insertsubsectionnavigationsymbol
(4) \insertsectionnavigationsymbol
(5) \insertdocnavigationsymbol
(6) \insertbackfindforwardnavigationsymbol
```

Une présentation *beamer* est écrite en LaTeX. Le passage d'une présentation à un article (ou vice-versa) est donc une chose très aisée ! Ceci est à ajouter aussi aux nombreux avantages de *beamer*.

8 Les overlays

Chaque *frame* (diapositive) d'une présentation est en réalité une page (en mode paysage) d'un fichier au format .pdf ; ce fichier est lu par l'application *Acrobat Reader* et les pages (diapositives) successives sont projetées à l'écran.

Dans le code source LaTeX, tout le contenu situé entre le `\begin{frame}` et le `\end{frame}` est présenté d'un seul tenant.

Comme c'est souvent le cas, l'orateur souhaite faire apparaître *successivement* les différents éléments constituant la diapositive, en fonction de l'évolution de son discours.

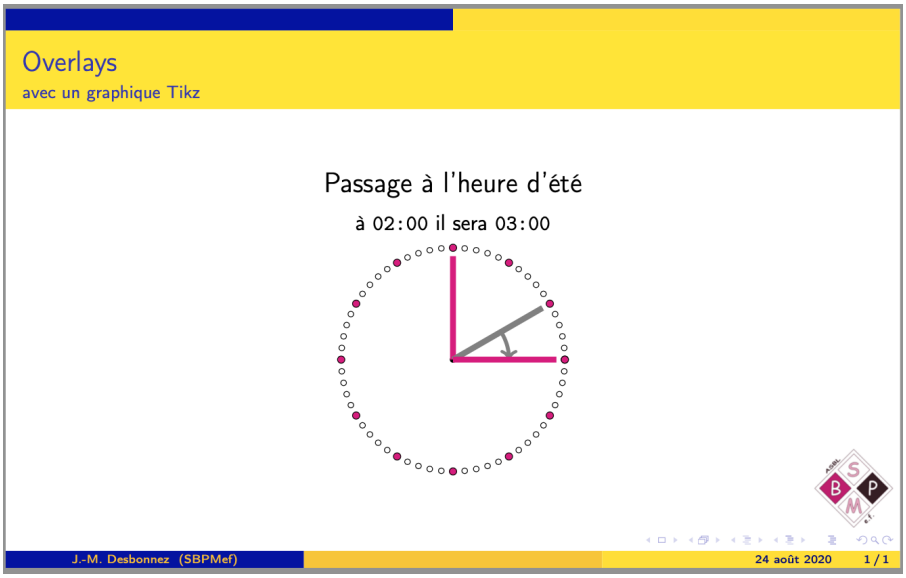
C'est comme si la diapositive est découpée en morceaux, et que les morceaux sont apparaissent les uns à la suite des autres, au bon vouloir de l'orateur.
On entend par *morceau* n'importe quelle zone de texte, les items d'une liste, les composants d'un graphique, les lignes d'une démonstration, etc.

Voilà un bon résumé de ce que sont les *overlays*. Le sujet est très vaste. On se contentera ici d'expliquer des instructions relativement simples dont la prise en main est rapide, et qui suffisent dans la majorité des applications. Le lecteur qui souhaite aller plus loin peut consulter [3] par exemple, et bien d'autres articles sur le net.

Pour être efficace : avant de créer des overlays, on commence par créer chaque frame (diapositive) avec tous les objets (textes, listes, images, etc.) qui seront présents. Lorsque les diapositives sont complètes, alors on peut décider de la manière dont les éléments vont apparaître, et placer les instructions d'overlays.

8.1 Avec l'instruction \pause

C'est la manière la plus simple de créer des *overlays*. Elle est illustrée par un dessin créé à l'aide de *Tikz* expliquant de manière visuelle le passage à l'heure d'été.



Ci-dessus, le « produit fini ».

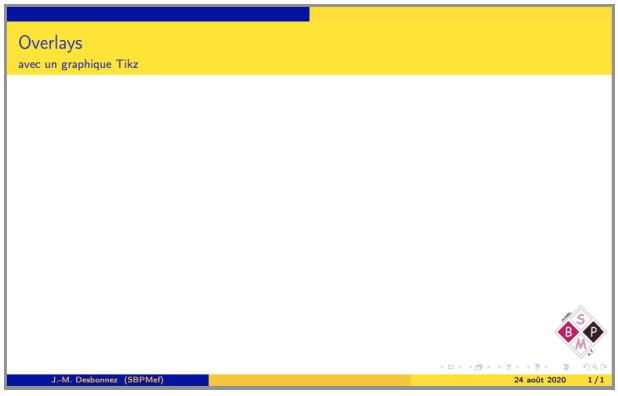
Comme dans tout document TeX, il y a un préambule, ici le minimum vital...

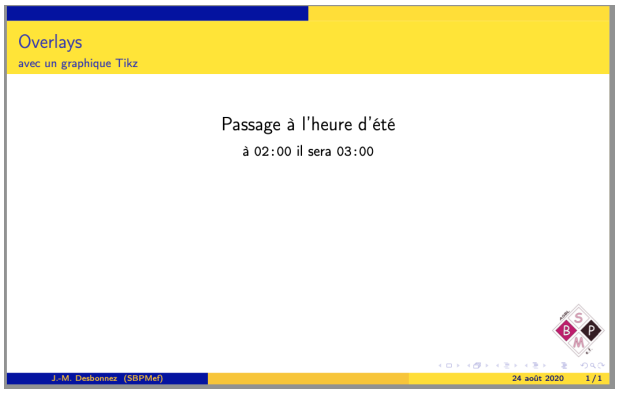
```
1   \documentclass[aspectratio=1610]{beamer}
2   \usepackage[utf8]{inputenc}
3   \usepackage[T1]{fontenc}
4   \usepackage[french]{babel}
5   \usepackage{tikz}
6
7   \usetheme{AnnArbor}
8   \hypersetup{pdfpagemode=FullScreen}
9
10  \author[J.-M. Desbonnez]{Jean-Marc \\  DESBONNEZ}
11  \institute[SBPMef]{\textit{SBPMef \\Société Belge des Professeurs
12   de Mathématique
13  \\d'expression française}}
14  \logo{\includegraphics[scale=0.15]{sbpm.png}}
```

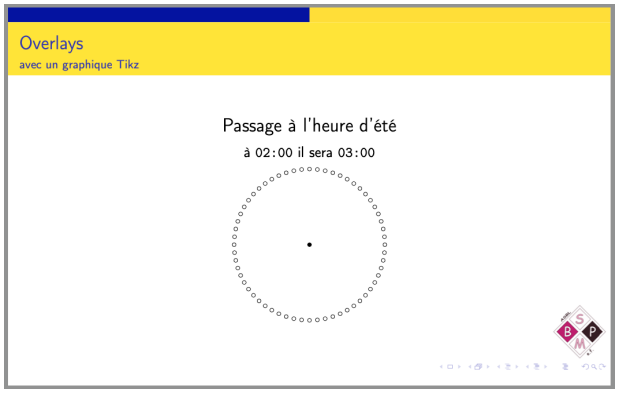
On souhaite faire apparaître successivement

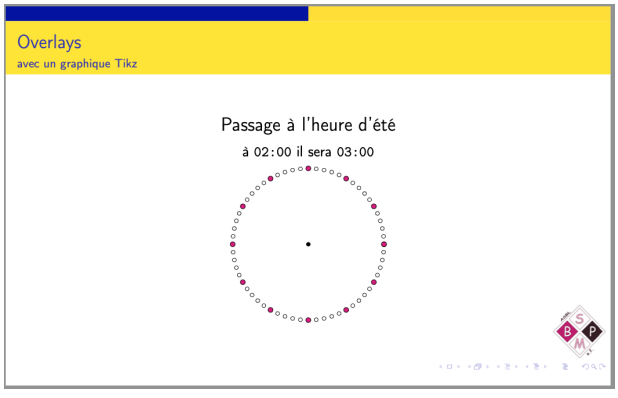
1. la diapositive vierge ;
2. le titre ;
3. 60 points représentant les minutes ;
4. 12 points représentant les heures ;
5. les aiguilles à 02:00 ;
6. les aiguilles à 03:00.

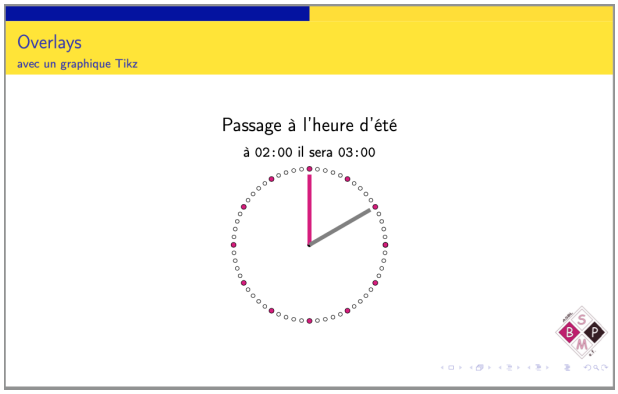
La diapositive complète sera scindée en 6 diapositives qui vont successivement se superposer à l'écran, donnant ainsi l'illusion que les éléments suivants s'ajoutent aux précédents.

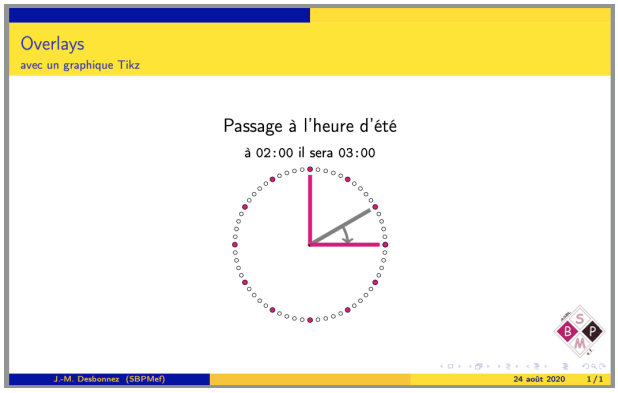












Dans le code qui suit, si on supprime les instructions \pause, on obtient le résultat de la page précédente, équivalent à la sixième diapositive créée à la fin des *overlays*.

```
15  \begin{document}
16  \begin{frame}
17  \frametitle{Overlays}
18  \framesubtitle{avec un graphique Tikz}
19          \pause
20  \begin{center}
21  {\Large Passage à l'heure d'été}\\[0.2cm]
22  à \texttt{02:00} il sera \texttt{03:00}
23          \pause
24  \medskip
25  \begin{tikzpicture}[scale=0.5]
26  % les 60 minutes
27  \foreach \a in{0,6,...,354}
28          {\draw(\a:4) circle(1mm);}
29  \draw(0,0) [fill=black]circle(1mm);
30          \pause
31  % les 12 heures
32  \foreach \a in{0,30,...,360}
33          {\draw(\a:4) [fill=magenta]circle(1.3mm);}
34          \pause
35  % il est 02:00
36  \draw [magenta,line width=3pt](0,0)--(90:3.7);
37  \draw [gray,line width=3pt](0,0)--(30:3.7);
38          \pause
39  % il est 03:00
40  \draw[<-,gray,line width=2pt](2,0) arc(0:30:1.9);
41  \draw [magenta,line width=3pt](0,0)--(0:3.7);
42  \end{tikzpicture}
43
44  \end{center}
45  \end{frame}
46  \end{document}
```

Remarques

1 Le fait de placer les instructions d'overlays augmente considérablement le nombre de pages du document .pdf ; si le fichier .pdf doit être imprimé, il est donc conseillé de l'imprimer

avant de compiler avec les instructions d'overlays, ou alors d'installer les instructions d'overlays sur une copie du fichier.

2 Pour imprimer...

- si on imprime le fichier .pdf tel quel, il y a une diapositive par feuille, par défaut en mode paysage ;
- on peut mettre deux diapositives par feuille ; prévoir à cet effet les deux instructions suivantes dans le préambule :

```
\usepackage{pgfpages} \pgfpagesuselayout{2 on 1}
[a4paper,border shrink=5mm]
```

  l'option `a4paper` force le remplissage total de la feuille, l'option `border shrink=5mm` installe une marge de 5 mm autour de chaque diapositive.

- on peut mettre quatre diapositives par feuille ; il faut forcer en plus le mode paysage :

```
\usepackage{pgfpages} \pgfpagesuselayout{4 on 1}
[a4paper,border shrink=5mm,landscape]
```

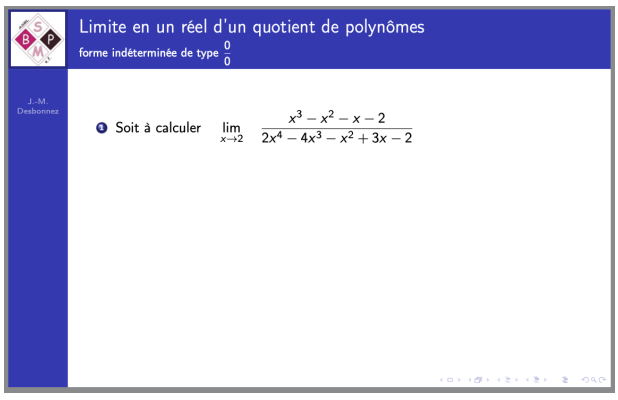
3 Fichiers engendrés par la compilation

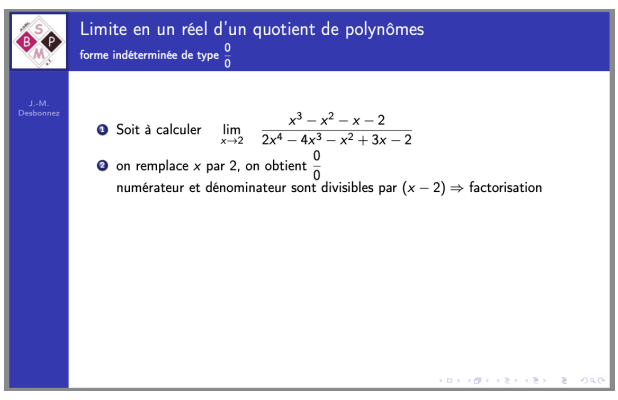
La compilation d'un document de la classe *beamer* engendre plusieurs types de fichiers : *.pdf* traditionnel, mais aussi *.nav*, *.out*, *.snm*, et *.toc*.
ce ne sont pas des fichiers de type texte, et je n'ai pas trouvé d'explications les concernant. On peut supposer aisément que *.nav* a un rapport avec la navigation dans les diapositives, et *.toc* a un rapport avec la table des matières...

8.2 Overlays automatiques pour itemize et enumerate

Les environnements *itemize* et *enumerate* sont certainement les plus utilisés dans les présentations, de même que les différents items de ces listes doivent apparaître successivement à l'écran.

Une option (très courte) permet de rendre cela automatique : `[<+->]`





Limite en un réel d'un quotient de polynômes

forme indéterminée de type $\frac{0}{0}$

J.-M. Desbonnez

1. Soit à calculer $\lim\limits_{x\to 2} \dfrac{x^3-x^2-x-2}{2x^4-4x^3-x^2+3x-2}$
2. on remplace x par 2, on obtient $\dfrac{0}{0}$
   numérateur et dénominateur sont divisibles par $(x-2) \Rightarrow$ factorisation
3. $= \lim\limits_{x\to 2} \dfrac{(x-2)(x^2+x+1)}{(x-2)(2x^3-x+1)}$

Limite en un réel d'un quotient de polynômes

forme indéterminée de type $\frac{0}{0}$

J.-M. Desbonnez

1. Soit à calculer $\lim\limits_{x\to 2} \dfrac{x^3-x^2-x-2}{2x^4-4x^3-x^2+3x-2}$
2. on remplace x par 2, on obtient $\dfrac{0}{0}$
   numérateur et dénominateur sont divisibles par $(x-2) \Rightarrow$ factorisation
3. $= \lim\limits_{x\to 2} \dfrac{(x-2)(x^2+x+1)}{(x-2)(2x^3-x+1)}$

4. $= \lim\limits_{x\to 2} \dfrac{(x^2+x+1)}{(2x^3-x+1)}$

Limite en un réel d'un quotient de polynômes

forme indéterminée de type $\frac{0}{0}$

J.-M. Desbonnez

1. Soit à calculer $\lim\limits_{x\to 2} \dfrac{x^3-x^2-x-2}{2x^4-4x^3-x^2+3x-2}$
2. on remplace x par 2, on obtient $\dfrac{0}{0}$
   numérateur et dénominateur sont divisibles par $(x-2) \Rightarrow$ factorisation
3. $= \lim\limits_{x\to 2} \dfrac{(x-2)(x^2+x+1)}{(x-2)(2x^3-x+1)}$
4. $= \lim\limits_{x\to 2} \dfrac{(x^2+x+1)}{(2x^3-x+1)}$

5. on remplace x par 2

Limite en un réel d'un quotient de polynômes

forme indéterminée de type $\frac{0}{0}$

J.-M. Desbonnez

1. Soit à calculer $\lim\limits_{x\to 2} \dfrac{x^3-x^2-x-2}{2x^4-4x^3-x^2+3x-2}$
2. on remplace x par 2, on obtient $\dfrac{0}{0}$
   numérateur et dénominateur sont divisibles par $(x-2) \Rightarrow$ factorisation
3. $= \lim\limits_{x\to 2} \dfrac{(x-2)(x^2+x+1)}{(x-2)(2x^3-x+1)}$
4. $= \lim\limits_{x\to 2} \dfrac{(x^2+x+1)}{(2x^3-x+1)}$
5. on remplace x par 2

6. $= \dfrac{7}{15}$

```
\documentclass[aspectratio=1610]{beamer}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\usepackage{tikz}
\usepackage{cancel}
\everymath{\displaystyle}
\usetheme{PaloAlto}
\hypersetup{pdfpagemode=FullScreen}
\author[J.-M. Desbonnez]{Jean-Marc  DESBONNEZ}
\institute[SBPMef]{\textit{SBPMef }}
\logo{\includegraphics[scale=0.15]{sbpm.png}}
\begin{document}
\begin{frame}
\frametitle{Limite en un réel d'un quotient de polynômes}
\framesubtitle{forme indéterminée de type $\frac{0}{0}$}
\begin{enumerate}[<+->]
\item
Soit à calculer \quad $\lim_{x \to 2} \quad \frac{x^3-x^2-x-2}{2x^4-4x^3-x^2+3x-2}$
\item
on remplace $x$ par 2, on obtient $\frac{0}{0}$ \\
numérateur et dénominateur sont divisibles par $(x-2)\Rightarrow$ factorisation
\item
$= \lim_{x \to 2} \quad \frac{\cancel{(x-2)}(x^2+x+1)}{\cancel{(x-2)}(2x^3-x+1)}$
\item
$= \lim_{x \to 2} \quad \frac{(x^2+x+1)}{(2x^3-x+1)}$
\item
on remplace $x$ par 2
\item
$=\frac{7}{15}$
\end{enumerate}
\end{frame}
\end{document}
```

8.3 Avec l'instruction \visible

Dans les deux exemples qui précèdent, un élément s'ajoute aux précédents à chaque diapositive, et tous restent en permanence visibles.

Avec l'instruction `\visible` on peut définir sur quelle(s) diapositive(s) un élément doit être visible.

Elle a pour syntaxe `\visible[options]{blablabla}` où `[options]` peut prendre les valeurs suivantes :

- <*n*> : blablabla visible uniquement sur la diapositive numéro n ;
- <*n,p*> : blablabla visible sur les diapositives numéros n à p ;
- <*n*–> : blablabla visible à partir de la diapositive numéro n et jusqu'à la fin ;
- <–*n*> : blablabla visible jusqu'à la diapositive numéro n.

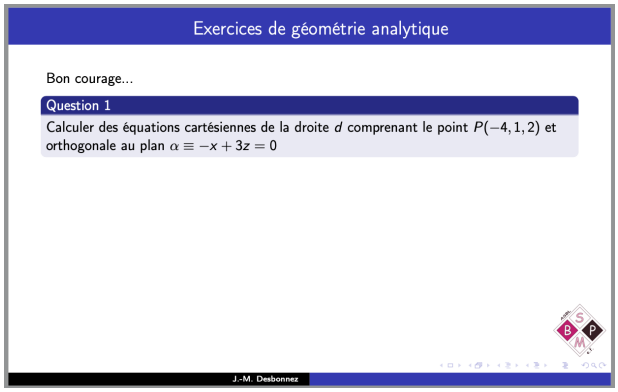
Application

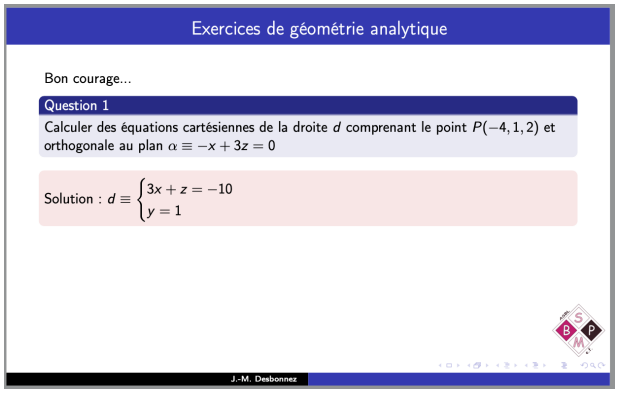
Des exercices (énoncé et solution) sont projetés sous la forme d'une présentation. D'abord l'énoncé, puis ajout de la solution. Mais pour l'énoncé suivant, le précédent et sa solution doivent disparaître.

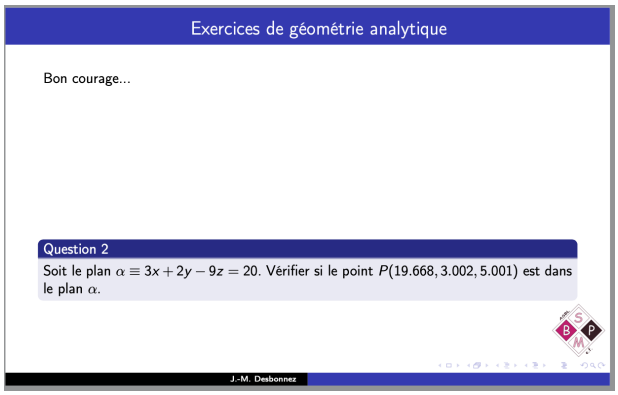
Dans l'exemple qui suit, on ne traitera que deux énoncés courts, l'essentiel étant la méthode, et non le contenu. S'il y a plus d'énoncés, il suffit d'ajouter des diapositives.

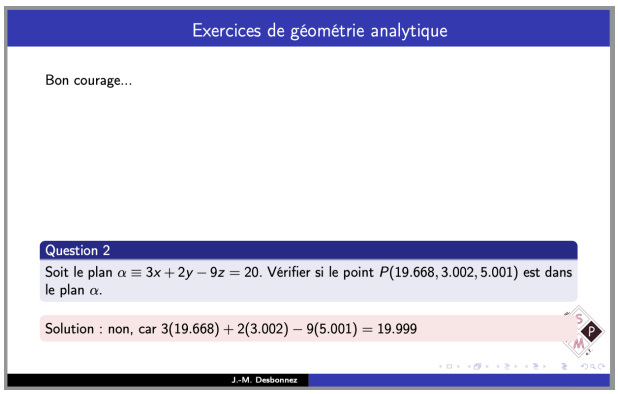
On en profitera pour faire une petite piqûre de rappel à propos des *blocks*.

Le thème choisi ici est *Copenhagen*, thème dans lequel les blocks ont leur titre sous fond de couleur. Il faudrait adapter la largeur des blocks contenant la solution, l'exercice est laissé au lecteur...









```
\documentclass[aspectratio=1610]{beamer}
\usepackage[utf8]{inputenc}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\hypersetup{pdfpagemode=FullScreen}

\usetheme{Copenhagen}

\author[J.-M. Desbonnez]{Jean-Marc \\  DESBONNEZ}
\institute[SBPMef]{\textit{SBPMef}}
\logo{\includegraphics[scale=0.15]{sbpm.png}}

\begin{document}

\begin{frame}
\frametitle{\centerline {Exercices de géométrie analytique}}

\visible<1-> {Bon courage...}

\visible<1,2>{
\begin{block}{Question 1}
Calculer des équations cartésiennes de la droite $d$ comprenant le point
$P(-4,1,2)$ et orthogonale au plan $\alpha\equiv -x+3z=0$
\end{block}}

\visible<2>{
\begin{alertblock}{}
Solution : $d \equiv \begin{cases} 3x+z=-10\\y=1 \end{cases}$
\end{alertblock}}

\visible<3,4>{
\begin{block}{Question 2}
Soit le plan $\alpha \equiv 3x+2y-9z=20$. Vérifier si le point
 $P(19.668,3.002,5.001)$ est dans le plan $\alpha$.
\end{block}}

\visible<4>{
\begin{alertblock}{}
Solution : non, car $3(19.668)+2(3.002)-9(5.001)=19.999$
\end{alertblock}}

\end{frame}

\end{document}
```

9 Effets de transition, vidéo, son

Ce n'est pas le point fort de *Beamer*... mais est-ce vraiment indispensable dans une présentation de type scientifique ? Le lecteur trouvera beaucoup d'explications détaillées dans [3].

Il y a une vingtaine d'effets de transitions, il serait malvenu de simplement les copier ici. Les quelques essais que j'ai réalisé n'ont pas tous été convaincants...

L'insertion d'une vidéo est complexe : il y a plusieurs formats (j'ai testé, en vain, .mpg, .mov et .avi) puis j'ai abandonné.

L'insertion d'un son, honnêtement, je n'ai pas testé.

10 Quelques trucs utiles

\tableofcontents

C'est une instruction qui, ordinairement, crée la table des matières d'un document. Dans la classe *Beamer*, elle crée une nouvelle diapositive contenant une liste numérotée des sections (pour autant évidemment que la présentation soit organisée en sections).

Chaque point de la liste est un lien hypertexte vers la première diapositive de la section concernée.

Diapositive trop longue

C'est au concepteur à gérer le contenu des diapositives et à en créer une suivante lorsque la précédente est « complète ».

Deux options permettent de gérer des contenus « trop longs » :

- `\begin{frame}[allowframebreaks]`

  Cette option permet au concepteur de ne pas trop s'inquiéter de la longueur du contenu : si celui-ci est trop long, une nouvelle diapositive (nouvelle page dans le fichier .pdf) est automatiquement créée.

- `\begin{frame}[shrink]`

  Lorsque le contenu d'une diapositive est « légèrement » trop long, cette option a pour effet de réduire la taille de la police afin que le tout tienne sur une page.

Diapositive vide

L'option *plain* permet de créer une diapositive « vierge » sans titre ni bandeau de navigation, ne contenant que la barre de navigation (si celle-ci n'a pas été masquée). Cela permet par exemple un gain de place pour un contenu volumineux. `\begin{frame}[plain]`

Couleur d'arrière plan

Il est possible de définir une couleur d'arrière plan pour l'ensemble des diapositives. L'instruction est à placer dans le préambule.

```
\setbeamercolor{normal text}{bg=blue!10}
```

L'instruction qui précède définit un arrière plan (*bg pour background*) léger bleu (10%).

Nous avons abordé *Beamer* sous la forme d'une initiation ; cela suffit pour beaucoup de présentations.

Si vos besoins sont plus sophistiqués, il y a de quoi les satisfaire, n'hésitez pas à consulter [3].

Vous avez peut-être déjà entendu dire « J'ai fait un *PowerPoint* »...
vous pouvez dire maintenant « J'ai fait un *Beamer* » !

Références

[1] www.cpt.univ-mrs.fr/~masson/latex/cours-LaTeX-A4.pdf.

[2] mcclinews.free.fr/latex/beamergalerie/completsgalerie.html.

[3] tug.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf.

[4] https://fr.wikipedia.org.

Je pose et je retiens, une histoire de multiplication... sans multiplication

Résumé. *À l'occasion d'une rencontre très amicale, Philippe* Tilleuil *me parle d'une étrange table de multiplication...*

1 Souvenirs d'école primaire

Nous sommes tous passés, du moins à partir d'une certaine génération, par l'apprentissage des multiplications avec un schéma comme ci-dessous, où l'on calcule, par exemple, 5698×7.

$$\begin{array}{cccccl} & 4 & 6 & 5 & & \leftarrow \text{chiffres retenus} \\ & 5 & 6 & 9 & 8 & \\ & & & \times & \mathbf{7} & \\ \hline 3 & 9 & 8 & 8 & 6 & \leftarrow \text{chiffres posés} \end{array}$$

Il se décode de la manière suivante, refrain qui résonne peut-être encore dans les mémoires :

1. $7 \times 8 = 56$; je pose 6 et je retiens 5 ;
2. $7 \times 9 = 63$; j'ajoute 5, ce qui donne 68 ; je pose 8 et je retiens 6 ;
3. $7 \times 6 = 42$; j'ajoute 6, ce qui donne 48 ; je pose 8 et je retiens 4 ;
4. $7 \times 5 = 35$; j'ajoute 4, ce qui donne 39 ; j'écris 39 car tous les chiffres du nombre à multiplier ont été épuisés.

Le processus nécessite 4 étapes, car le nombre à multiplier (ici par 7) est composé de 4 chiffres.

La première et la dernière étape sont particulières, et n'ont pas la même forme que les étapes intermédiaires.

2 L'école primaire est terminée...

Dans le milieu du XIX^{e} siècle, Chaim Zelig Slonimsky (1810–1904), éditeur hébreu, mathématicien, astronome, écrivain scientifique et rabin, invente un procédé de multiplication, qui sera appelé *Théorème de* Slonimsky.

En 1990, le néerlandais Peter Roubos construit un « calculateur » à partir de ce procédé [1].

En 1846, August Leopold Crelle (1780–1855), mathématicien et architecte allemand démontre le théorème en utilisant des fractions issues des suites de Farey, et le publie dans un journal allemand [2], pp. 215–229, et en français !

Après la page 277, on trouve un tableau (28 lignes et 110 colonnes) dont est extrait le tableau ci-après, limité ici pour l'exemple à la multiplication par 7, mais on peut en extraire les multiplications par 0 ☺, 1 ☺, 2, ..., 9.

Slonimsky a en fait conjecturé qu'il y avait exactement 28 types de retenues possibles.

Multiplication par 7

	0	re	1	re	2	re	3	re	4	re	5	re	6	re	7	re	8	re	9	re
1	0	1	7	1	4	6	1	9	8	12	5	15	2	18	9	20	6	24	3	28
2	0	1	7	2	4	6	1	9	8	12	5	15	2	18	9	20	6	24	3	28
3	0	1	7	2	4	6	1	9	8	12	5	15	2	18	9	20	6	24	3	28
4	1	1	8	2	5	6	2	9	9	12	6	15	3	18	0	21	7	24	4	28
5	1	1	8	2	5	6	2	9	9	12	6	15	3	18	0	21	7	24	4	28
6	1	1	8	2	5	6	2	9	9	12	6	15	3	18	0	21	7	24	4	28
7	1	1	8	2	5	7	2	9	9	12	6	15	3	18	0	21	7	24	4	28
8	1	1	8	3	5	7	2	9	9	12	6	15	3	19	0	21	7	24	4	28
9	2	1	9	3	6	7	3	9	0	13	7	15	4	19	1	21	8	24	5	28
10	2	1	9	3	6	7	3	10	0	13	7	15	4	19	1	21	8	25	5	28
11	2	1	9	3	6	7	3	10	0	13	7	15	4	19	1	21	8	25	5	28
12	2	1	9	3	6	7	3	10	0	13	7	15	4	19	1	21	8	25	5	28
13	3	1	0	4	7	7	4	10	1	13	8	15	5	19	2	21	9	25	6	28
14	3	1	0	4	7	7	4	10	1	14	8	15	5	19	2	21	9	25	6	28
15	3	1	0	4	7	8	4	10	1	14	8	15	5	19	2	22	9	25	6	28
16	3	1	0	4	7	8	4	10	1	14	8	16	5	19	2	22	9	25	6	28
17	4	1	1	4	8	8	5	10	2	14	9	16	6	19	3	22	0	26	7	28
18	4	1	1	4	8	8	5	10	2	14	9	16	6	19	3	22	0	26	7	28
19	4	1	1	4	8	8	5	10	2	14	9	16	6	19	3	22	0	26	7	28
20	4	1	1	5	8	8	5	10	2	14	9	16	6	20	3	22	0	26	7	28
21	5	1	2	5	9	8	6	10	3	14	0	17	7	20	4	22	1	26	8	28
22	5	1	2	5	9	8	6	11	3	14	0	17	7	20	4	22	1	27	8	28
23	5	1	2	5	9	8	6	11	3	14	0	17	7	20	4	23	1	27	8	28
24	5	1	2	5	9	8	6	11	3	14	0	17	7	20	4	23	1	27	8	28
25	5	1	2	5	9	8	6	11	3	14	0	17	7	20	4	23	1	27	8	28
26	6	1	3	5	0	9	7	11	4	14	1	17	8	20	5	23	2	27	9	28
27	6	1	3	5	0	9	7	11	4	14	1	17	8	20	5	23	2	27	9	28
28	6	1	3	5	0	9	7	11	4	14	1	17	8	20	5	23	2	28	9	28

Le tableau est composé de 28 lignes, numérotées de 1 à 28 et de 10 doubles colonnes.

Dans chaque double colonne, le chiffre noir est le chiffre à poser, et le chiffre rouge est le numéro de ligne où l'on trouvera la suite du calcul.

Dans la ligne d'en-tête, on trouve en noir les chiffres constituant le nombre que l'on multiplie par 7 (ici 5698).

Voici la manière de l'utiliser, pour le calcul décrit en tête d'article.

1 La première étape est particulière ; il faut commencer par multiplier 8 (par 7) ; on recherche 8 dans la ligne d'en-tête, et à la ligne 1, colonne « 8 » on trouve 6. C'est le premier nombre à poser. La suite est à la ligne 24.

2 Il faut maintenant multiplier 9 (par 7) ; on recherche 9 dans la ligne d'en-tête, et à la ligne 24, colonne « 9 » on trouve 8. C'est le deuxième nombre à poser. La suite est à la ligne 28.

3 Il faut maintenant multiplier 6 (par 7) ; on recherche 6 dans la ligne d'en-tête, et à la ligne 28, colonne « 6 » on trouve 8. C'est le troisième nombre à poser. La suite est à la ligne 20.

4 Il reste à multiplier 5 (par 7) ; on recherche 5 dans la ligne d'en-tête, et à la ligne 20, colonne « 5 » on trouve 9. C'est le quatrième nombre à poser. La suite est à la ligne 16.

5 Il n'y a plus de chiffre à multiplier (par 7), on imagine 0 ; on recherche 0 dans la ligne d'en-tête, et à la ligne 16, colonne « 0 » on trouve 3. On est revenu à la ligne 1, c'était le dernier chiffre à poser.

3 Allo, Excel ?

Cette table de multiplication est une grande matrice, dans laquelle on se promène au gré des lignes et des colonnes, et donc un terrain de jeu idéal pour les fonctions de recherche INDEX() et EQUIV() du tableur *Excel*. Le lecteur se fera un plaisir de consulter [3] en guise de rappel.

On n'y échappe pas, il faut encoder toute la table... le pavé numérique va chauffer...
Les couleurs n'ont aucune influence sur la suite.

Comme de bonne habitude, on opte pour une numérotation des colonnes en 1, 2, 3, ... ce qui permet de mieux s'y retrouver dans la feuille, et surtout de voir que les formules copiées sont exactement identiques.

	1	2	3	4	5	⋯	20	21
1		0	re	1	re	⋯	9	re
2	1	0	1	7	1	⋯	3	28
3	2	0	1	7	2	⋯	3	28
4	3	0	1	7	2	⋯	3	28
⋯	⋯	⋯	⋯	⋯	⋯	⋯	⋯	28
29	28	6	1	3	5	⋯	9	28

Pour une meilleure compréhension des formules,

- la plage `L2C1:L29C21` est nommée `table7` ;
- la plage `L1C1:L1C21` est nommée `colonne7` (en-tête de colonnes de la table7).

Rappelons que les noms de plages sont connus dans tout le classeur.

Dans une autre feuille, plaçons les données ; dans un premier temps, on prend les mêmes que dans l'exemple initial, ce qui permettra de tester les résultats.

	1	2	3	4	5	6
1	**0**	5	6	9	8	
2	×	**7**				
3						
4						**1**

Le zéro non significatif en `L1C1` est obligatoire pour permettre la copie de la formule `L3C5` jusqu'en `L3C1`.
Le résultat de la multiplication sera dans la plage `L3C1:L3C5` ; on verra qu'il n'y aura qu'une seule formule en `L3C5`, que l'on copiera jusqu'en `L3C1` ; le 1 en `L4C6` est indispensable à cet effet.
La plage `L4C2:L4C6` contiendra les numéros de lignes des colonnes « re » où il faudra lire les résultats suivants. Une seule formule, également copiable.
Pas d'inquiétude, on explique...

en L3C5

Il faut rechercher la position (n° colonne) du 8 (`L1C5` ou `L(-2)C` par rapport à `L3C5`) dans la plage `colonne7` ; on obtient 18.

$$\underbrace{EQUIV(L(-2)C;colonne7;0)}_{18}$$

Dans `table7`, on recherche l'élément situé à la ligne 1 (`L4C6` ou `L(1)C(1)` par rapport à `L3C5`) colonne 18 ; on obtient 6.

$$\underbrace{=INDEX(table7;L(1)C(1);\underbrace{EQUIV(L(-2)C;colonne7;0)}_{18})}_{6}$$

en L4C5

On va y mémoriser le numéro de ligne (colonne « re ») pour la suite des opérations ; il se situe dans la ligne 1 (`L4C6` ou `LC(1)` par rapport à `L4C5`), une colonne à droite de celle où l'on a trouvé le 6 ; on obtient 24.

$$\underbrace{=INDEX(table7;LC(1);\underbrace{1+EQUIV(L(-2)C;colonne7;0)}_{19})}_{24}$$

d'où

	1	2	3	4	5	6
1	**0**	5	6	9	8	
2	×	**7**				
3					6	
4					24	**1**

On va appliquer maintenant le même raisonnement pour le chiffre suivant, et voir ainsi que les formules sont identiques.

en L3C4

Il faut rechercher la position (n° colonne) du 9 (`L1C4` ou `L(-2)C` par rapport à `L3C4`) dans la plage `colonne7` ; on obtient 20.

$$\underbrace{EQUIV(L(-2)C;colonne7;0)}_{20}$$

Dans `table7`, on recherche l'élément situé à la ligne 24 (`L4C5` ou `L(1)C(1)` par rapport à `L3C4`) colonne 20 ; on obtient 8.

$$\underbrace{=INDEX(table7;L(1)C(1);\underbrace{EQUIV(L(-2)C;colonne7;0)}_{20})}_{8}$$

en L4C4

On va y mémoriser le numéro de ligne (colonne « re ») pour la suite des opérations ; il se situe dans la ligne 24 (`L4C5` ou `LC(1)` par rapport à `L4C4`), une colonne à droite de celle où l'on a trouvé le 8 ; on obtient 28.

$$\underbrace{=INDEX(table7;LC(1);\underbrace{1+EQUIV(L(-2)C;colonne7;0)}_{21})}_{28}$$

d'où

	1	2	3	4	5	6
1	**0**	5	6	9	8	
2	×	**7**				
3				8	6	
4				28	24	**1**

Après copie des formules, on obtient

	1	2	3	4	5	6
1	**0**	5	6	9	8	
2	×	**7**				
3	3	9	8	8	6	
4		16	20	28	24	**1**

Il suffit de modifier les données dans `L1C2:L1C5` pour tester d'autres multiplications par 7.

Multiplier par 7 un nombre plus long

Il suffit d'ajouter des chiffres à la ligne 1, et de ne pas oublier de placer le chiffre 1 à l'extrémité de la ligne 4, une colonne après le dernier chiffre du nombre à multiplier. Et copier (vers la droite) les formules des lignes 3 et 4.

Multiplier par 1, 2, ..., 9

Il faut commencer par encoder toutes les tables extraites du journal de Crelle... bon amusement.

Pour chaque table, nommer les plages `table1`, `co1`, `table2`, `co2`, `table3`, `co3`, ..., `table9`, `co9`.

L'auteur de cet article a eu l'idée de génie de placer le 7 en position fixe `L2C2` et pas en `L2C5` sous le 8 (comme dans la version « école primaire ») ce qui permet de toujours trouver le chiffre multiplicateur dans la même cellule, même si le nombre à multiplier s'allonge.

Il faut modifier comme suit les deux formules :

- remplacer `colonne7` par
  `CHOISIR(L2C2;co1;co2;co3;co4;co5;co6;colonne7;co8;co9)`

- remplacer `table7` par
  `CHOISIR(L2C2;table1;table2;table3;table4;table5;table6;table7;table8;table9)`

Les formules deviennent nettement plus longues, mais c'est bien connu, quand on aime, on ne compte pas.

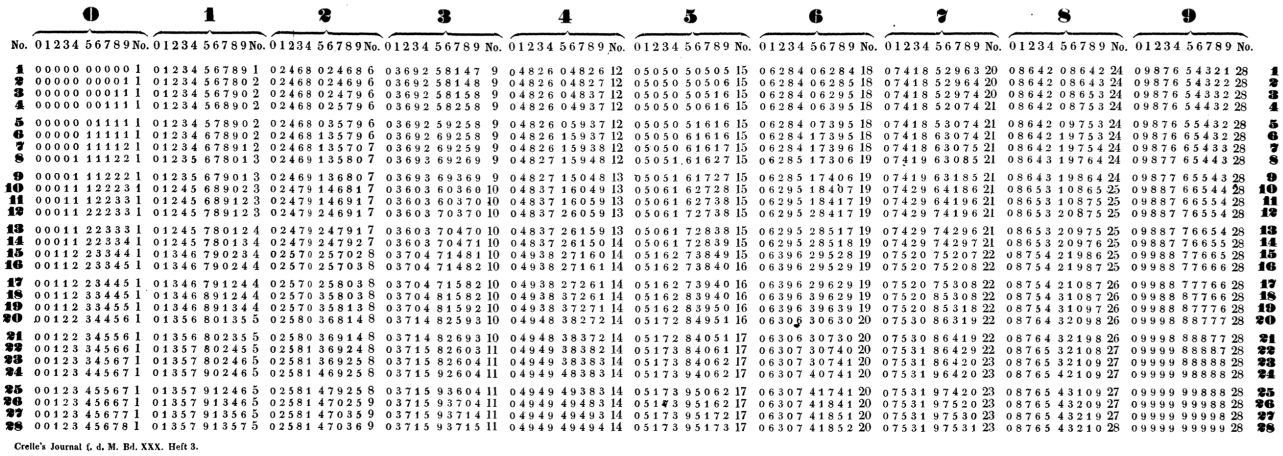


Fig. 26 *Les tables de* Crelle, *dans [2]*

Philippe Tilleuil fait remarquer que si on prend n'importe quel nombre, aussi grand soit-il, et ne contenant que des 1, 2, 4, 5, 7, 8 et 9, son produit par 3 ne contient jamais de chiffre 0.

Dans la table ci-dessus, regarder les larges colonnes numérotées en gras 1, 2, 4, 5, 7, 8 et 9, et pour chacune d'elle, parcourir les colonnes dont l'intitulé est 3 ; le chasseur de zéros rentre bredouille !

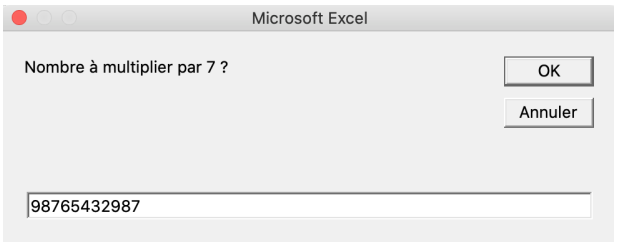
4 Macro sur le gâteau

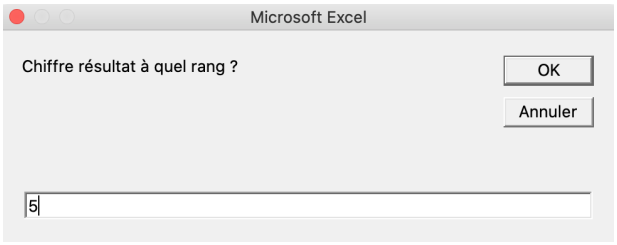
Après mûre réflexion, on constate que nous avons fait une multiplication sans faire aucune multiplication...

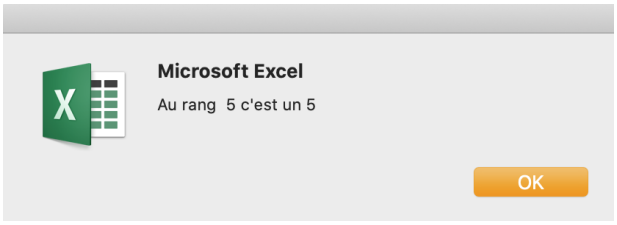
La table de Crelle permet donc de calculer le chiffre d'un rang donné du résultat d'une multiplication, sans faire la multiplication ; il suffit de suivre un chemin autant de fois que nécessaire.

C'est le principe de la macro qui suit. Nous nous sommes limités à la multiplication par 7 puisque la table a déjà été encodée.

Ce que l'on verra à l'écran







$$98\,765\,432\,987 \times 7 = 691\,358\,030\,909$$

La macro

```
1   Option Explicit
2   Dim k, cn, col, chr, nombre, rang, ligne_suiv, fin As Integer
3   Dim nombre_string As String
4   Dim table7, colonne7 As Variant
5
6   Sub mult7()
7       table7 = Sheets("7").Range(Cells(2, 1), Cells(29, 21))
8       colonne7 = Sheets("7").Range(Cells(1, 1), Cells(1, 21))
9
10      nombre = InputBox("Nombre à multiplier par 7 ?")
11      nombre_string = "0" & CStr(nombre)
12      rang = InputBox("Chiffre résultat à quel rang ?")
13
14      If Int(Log(nombre) / Log(10)) = Int((Log(nombre) + Log(7)) / Log(10)) Then
15          fin = rang + 1
16      Else
17          fin = rang
18      End If
19
20      ligne_suiv = 1 'ligne suivante dans la table de Crelle
21      For k = Len(nombre_string) To fin Step -1
22          cn = CInt(Right(Left(nombre_string, k), 1))
23          col = CInt(Application.Match(cn, colonne7, 0))
24          chr = CInt(Application.Index(table7, ligne_suiv, col))
25          ligne_suiv = CInt(Application.Index(table7, ligne_suiv, 1 + col))
26      Next
27      MsgBox "Au rang  " & rang & " c'est un " & chr
28  End Sub
```

Les explications

- Lignes **1** à **4** : déclaration des variables

  → *k* : compteur de chemins dans la table ;

  → *nombre* : le nombre qui doit être multiplié par 7 ;

  → *cn* : un chiffre du nombre (extrait de droite à gauche) ;

  → *chr* : le chiffre résultat qui est « posé ;

  → *col* : le numéro de colonne dans la plage *colonne7* ;

  → *rang* : le rang auquel on s'intéresse ;

  → *ligne_suiv* : la ligne suivante à parcourir dans la table *table7* ;

  → *nombre_string* : le nombre auquel on a ajouté le chiffre 0 en préfixe.

- Lignes **7** et **8** : « 7 » est le nom de la feuille dans laquelle a été encodée la table ; VBA ne reconnait pas les noms de plages qui ont été définis dans le classeur, il faut donc les re-définir. Pfffff.

- Ligne **11** : la fonction CStr() convertit un nombre en chaîne de caractère.

- Lignes **14** à **18** : calcul du nombre de chemins à suivre dans la table *table7* avant d'arriver au rang cherché. Ce nombre dépend de la longueur du résultat $nombre \times 7$ que l'on doit comparer à la longueur de *nombre* ; ces deux longueurs peuvent différer de 1 unité .
  Pour les calculer, on passe par les logarithmes ; en VBA, il n'y a pas de fonction logarithme en base 10, et le logarithme népérien s'écrit *log*...
  Si $7 * nombre$ est plus long que *nombre*, il y a une étape supplémentaire à effectuer.

- Ligne **20** : on initialise la ligne de recherche suivante, comme on l'a fait dans la cellule `L4C6`.

- Ligne **21** à **26** : on parcourt de droite à gauche les chiffres du nombre à multiplier par 7 avec une combinaison des fonctions RIGHT() et LEFT() ; ces fonctions opèrent sur des *chaînes de caractères* ; la fonction CInt() convertit une chaîne de caractères en nombre entier.
  En VBA, la fonction INDEX() se traduit par Application.Index() et la fonction EQUIV() se traduit par Application.Match().
  Les formules des lignes **22** à **25** sont les mêmes que celles où l'on a calculé les résultats de la plage `L3C1:L3C5` dans l'exemple initial.

Fig. 27 *Le calculateur de* Roubos*, dans [1]*

Références

[1] https://locomat.loria.fr/reconstructions/weiss/SloniMultE.pdf

[2] Crelle A. L., Démonstration d'un théorème de Mr. Slonimky sur les nombres, avec une application au calcul de chiffres. *Journal für die reine und angewandte Mathematik*, XXX (1845), pp. 215–229 ; table : p.277.

[3] Desbonnez J.-M., Le tableur Excel et les fonctions de recherche dans les tableaux. *Losanges*, 48, pp. 50–61, 2020.

Persistance multiplicative et additive des nombres

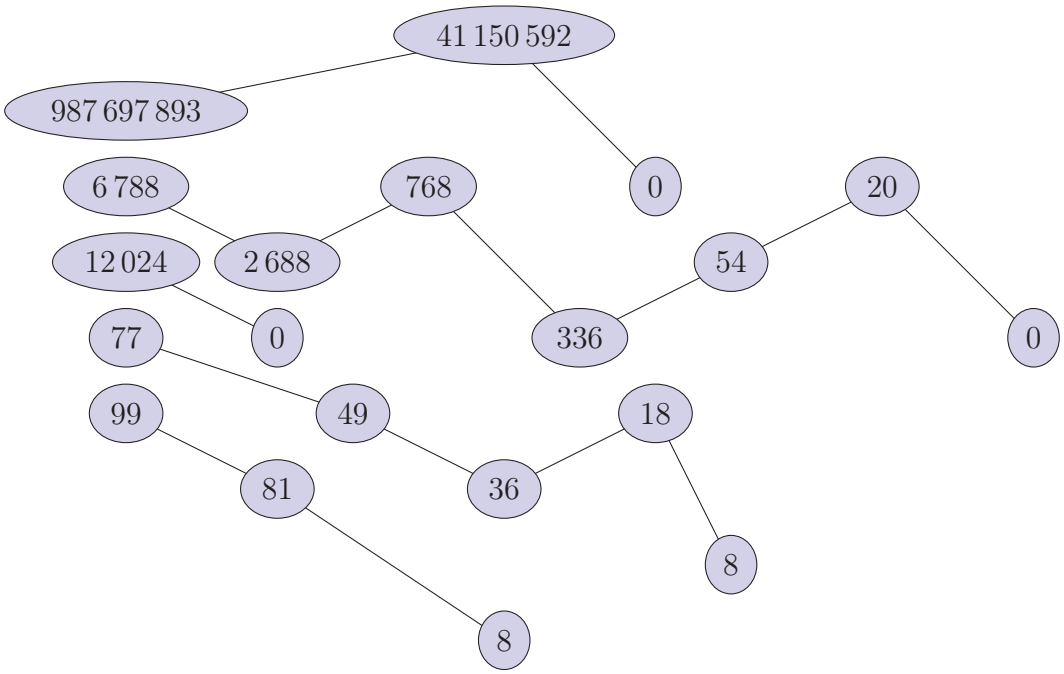
Résumé. *On choisit un nombre naturel (en base 10) et on fait le produit — la somme — des chiffres qui le composent. Quoi de plus simple ? On fait de même avec le résultat obtenu, puis encore, et encore... jusqu'à ce que le résultat soit un nombre composé d'un seul chiffre. Le processus est-il long ? Va-t-il s'arrêter ?*

L'occasion de faire un peu de programmation...

1 Persistance multiplicative

L'article le plus ancien traitant de ce sujet, datant de 1973, a été écrit par un mathématicien britanico-américain Neil Sloane (1939 —).

Le procédé est décrit dans le résumé. Voici quelques cheminements.



On appelle *persistance multiplicative* d'un entier n, que l'on pourrait noter $PM(n)$, le nombre d'étapes nécessaires avant d'obtenir un unique chiffre. Les cinq exemples précédents montrent que $PM(99) = 2$, $PM(77) = 4$, $PM(12\,024) = 1$, $PM(6\,788) = 6$ et $PM(987\,697\,893) = 2$.

Le résultat final (dernier chiffre restant) est appelé *racine numérique multiplicative* ou *résidu*.

Quelques réflexions arithmétiques immédiates

- dès qu'un nombre contient le chiffre 0, c'est l'arrêt immédiat du processus de calcul ; en conséquence, si n contient un chiffre 0, alors $PM(n) = 1$ et si n contient un chiffre pair et un chiffre 5, alors $PM(n) = 2$;
- la multiplication étant commutative, on peut changer l'ordre des chiffres constituant le nombre ; en particulier, si on classe en ordre croissant tous les chiffres composant un nombre n_1, on obtient un nombre n_2 tel que $PM(n_1) = PM(n_2)$;
- 1 étant l'élément neutre pour la multiplication, on peut supprimer tous les chiffres 1 constituant un nombre n ; on obtient un autre nombre qui a la même persistance multiplicative.

Pour aller plus loin

Il a été démontré que la persistance multiplicative moyenne des nombres compris entre 1 et n tend vers 1 lorsque n tend vers l'infini.

Par contre, la persistance multiplicative maximum vaut 11 ; c'est encore à l'heure actuelle une conjecture. Le plus petit nombre n, les chiffres étant classés en ordre croissant, pour lequel $PM(n) = 11$ est 277 777 788 888 899.

Ce résultat a été vérifié jusqu'à $n = 10^{500}$ par le mathématicien Mark DIAMOND puis jusqu'à $n = 10^{20\,000}$ par Benjamin CHAFFIN.

Le lecteur intéressé trouvera dans [1], [2] et [3] beaucoup d'autres informations. Il serait toutefois indécent de ne pas illustrer ici ce calcul de persistance multiplicative par quelques algorithmes. Nous avons choisi le tableur *Excel* et le langage de programmation *VBA*, pour cette première partie.

Calcul de $PM(n)$ pour un nombre n donné, avec affichage des résultats intermédiaires

La cellule `A1` contient le nombre n dont on souhaite calculer la persistance multiplicative. Les cellules qui suivent (`A2`, `A3`, `A4`,...) contiendront les résultats des étapes successives jusqu'à l'obtention d'un chiffre unique.

Un bouton est associé à la macro principale appelée `persistance_multiplicative()`, c'est le moyen le plus souple pour exécuter une macro.

Pour simplifier le code de la macro principale, on utilise deux *fonctions* :

- la fonction `effacer()` (lignes 23–26) qui efface le contenu de la plage `A2:A11`, les résultats d'éventuels calculs précédents ; le nombre maximum d'étapes étant estimé à 11, l'effacement jusqu'à `A12` suffit ;
- la fonction `calcul_produit(nombre)` (lignes 15–21) qui calcule le produit des chiffres qui compose le nombre précisé en argument. Ce calcul est très similaire au calcul d'une factorielle. La fonction `Len(...)` fournit la longueur (*lenght*) de l'argument en termes de nombre de caractères (nombre de chiffres en ce qui nous concerne) tandis que la combinaison `Right(Left(...,k),1)` fournit le k^{e}caractère de l'argument ;

la variable qui renvoie le résultat de calcul de la fonction **doit** porter le même nom que la fonction elle-même (ligne 20).

Quant à la macro principale (lignes 4–13) elle utilise une répétitive dont la condition d'arrêt est que la longueur du dernier résultat intermédiaire soit égale à 1 (il ne reste plus que 1 chiffre). C'est la variable *etape* qui gère le passage à l'étape suivante, et aussi à l'adresse de la cellule suivante qui recevra le résultat du calcul. Comme cette variable a été initialisée à 1 [(1)] **avant** de commencer les calculs (ligne 6), il faut lui retrancher 1 pour obtenir la valeur de la persistance multiplicative (ligne 12).

Voici le code *VBA* ; les numéros de lignes ont été ajoutés artificiellement pour faciliter leur référencement dans les explications, et ne font donc pas partie du code.

```
1   Option Explicit
2   Dim etape, nombre, n, pm, prod, k As Integer
3
4   Sub persistance_multiplicative()
5     effacer
6     etape = 1
7     Do While Len(Cells(etape, 1)) <> 1
8       pm = calcul_produit(Cells(etape, 1))
9       etape = etape + 1
10      Cells(etape, 1) = pm
11    Loop
12    MsgBox("Persistance multiplicative : " & etape-1)
13  End Sub
14
15  Function calcul_produit(nombre)
16    prod = 1
17    For k = 1 To Len(nombre)
18        prod = prod * Right(Left(nombre,k),1)
19    Next
20    calcul_produit = prod
21  End Function
22
23  Function effacer()
24    Range("A2:A12").Select
25    Selection.ClearContents
26    Range("a1").Select
27  End Function
```

	A
1	6788
2	2688
3	768
4	336
5	54
6	20
7	0

Voici un exemple d'exécution.
La persistance multiplicative est affichée à l'écran dans une boîte de dialogue (ligne 12).

Calcul de $PM(n)$ pour une série de nombres n donnés, sans affichage des résultats intermédiaires

La plage `A1:A...` contient une série de nombres dont on souhaite calculer la persistance multiplicative. La plage `B1:B...` contiendra les résultats.

Une boucle va parcourir chacune des cellules de la plage `A1:A...` et s'arrêtera dès qu'une cellule vide est rencontrée.

1. On ne peut pas l'initialiser à 0 car elle sert à l'adressage d'une cellule, et il n'y a pas de « ligne 0 ».

```
1   Option Explicit
2   Dim  nombre, pm, prod, k, ligne As Integer
3   Dim temp As String
4
5   Sub liste_persistances_multiplicatives()
6           ligne = 1
7           Do While Not (IsEmpty(Cells(ligne, 1)))
8                   temp = Cells(ligne, 1)
9                   pm = 1
10                  Do While Len(temp) <> 1
11                          temp = calcul_produit(temp)
12                          pm = pm + 1
13                  Loop
14                  Cells(ligne, 2) = pm - 1
15                  ligne = ligne + 1
16          Loop
17  End Sub
18
19  Function calcul_produit(nombre)
20      prod = 1
21      For k = 1 To Len(nombre)
22          prod = prod * Right(Left(nombre, k), 1)
23      Next
24      calcul_produit = prod
25  End Function
```

La ligne 3 assure que le nombre dans la cellule A... sera traité comme une chaîne de caractères.

Voici les persistances multiplicatives obtenues pour 11 nombres « bien choisis », résultats que l'on retrouve, entre autres, dans [1].

	A	B
1	10	1
2	25	2
3	39	3
4	77	4
5	679	5
6	6 788	6
7	68 889	7
8	2 677 889	8
9	26 888 999	9
10	377 8888 999	10
11	2 777 777 888 888 899	11

2 Persistance additive

Comme annoncé dans le résumé, on additionne les chiffres qui composent un nombre, puis on recommence avec le résultat obtenu jusqu'à obtenir un nombre à un seul chiffre, appelé *racine numérique additive*.

La persistance additive est le nombre d'étapes nécessaires.

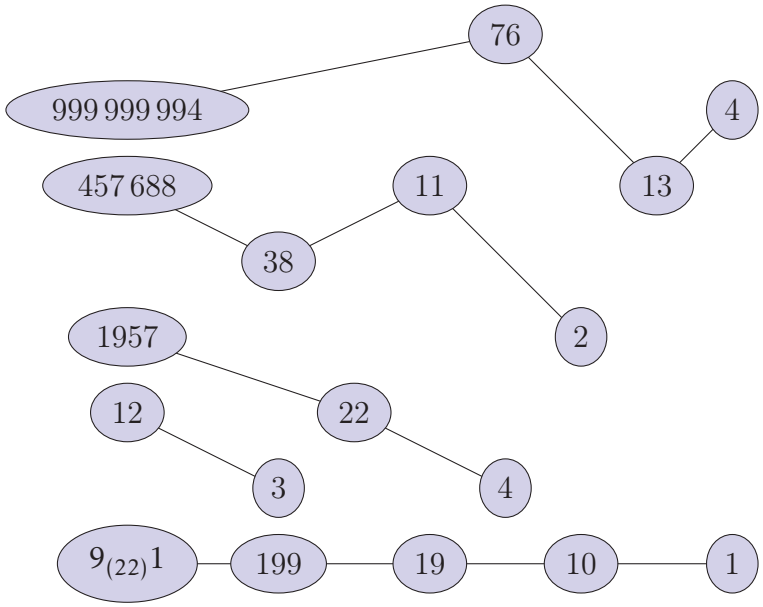
La recherche de la racine numérique additive est (était ?) un procédé « bien connu » des élèves sortant de l'école primaire, puisqu'elle est utilisée pour effectuer une « preuve par 9 », cette racine numérique étant le reste de la division par 9 du nombre de départ.

On peut obtenir des persistances additives aussi grandes que l'on souhaite.

Voici un schéma qui permet de construire un procédé de fabrication :

$$2 \longleftarrow 11 \longleftarrow \underbrace{11\,111\,111\,111}_{\text{11 chiffres 1}} \longleftarrow \underbrace{\cdots}_{\text{11 111 111 111 chiffres 1}} \longleftarrow \cdots$$

Quelques cheminements...



$9_{(22)}1$ signifie un nombre composé de 22 chiffres 9 consécutifs, suivis de 1.

Pour varier les plaisirs, nous avons ici utilisé le langage *Python* pour les deux algorithmes qui suivent.

Le premier est une fonction, baptisée somchiffres(), qui calcule la somme des chiffres constituant un nombre.

```
1   def somchiffres(nombre):
2           som=0
3           for chiffre in str(nombre):
4                   som=som+int(chiffre)
5           return som
```

La fonction str() (ligne 3) convertit un nombre en chaîne de caractères.

La fonction int() (ligne 4) convertit une chaîne en nombre entier.

L'instruction de la ligne 3 parcourt automatiquement chaque caractère du « mot » *nombre*, à savoir chaque chiffre, qui est ensuite (ligne 4) converti en nombre pour être cumulé au résultat précédent.

On peut utiliser la fonction par exemple sous la forme print(somchiffres(123456789))

Le second permet de calculer la persistance additive d'un nombre donné. Il fait appel à la fonction qui précède.

```
6   pers_add=0
7   nombre=99999999999999999999991
8   temp=nombre
9   while len(str(temp)) != 1:
10          temp=somchiffres(temp)
11          pers_add=pers_add+1
12  print("la persistance additive de "+ str(nombre) +" vaut " + str(pers_add))
```

Références

[1] Delahaye J.-P., La persistance des nombres. *Pour la Science*, 430, Août 2013.

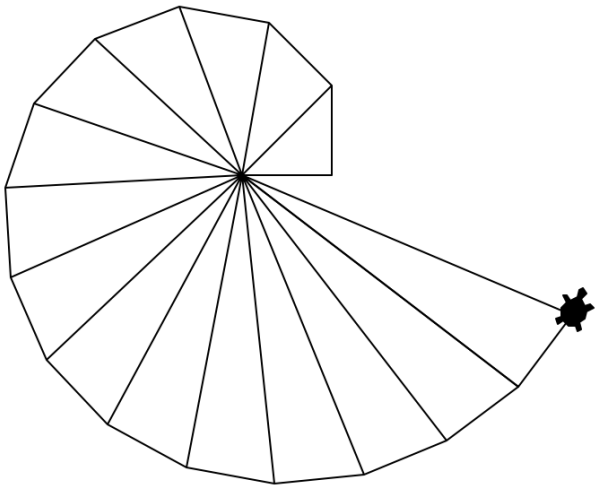
[2] http://villemin.gerard.free.fr.

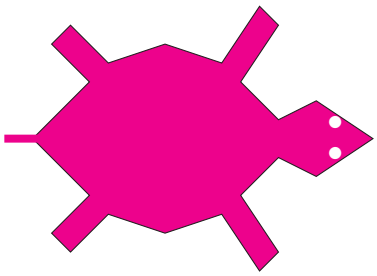
[3] https://automaths.blog/2017/08/26/des-entiers-persistants/.

Le serpent, l'arbre et l'escargot, sans beurre ni ail

Résumé. *Dans un article « Un rectangle nommé d'argent (2) » ([4]), Laure* Ninove *fait référence à l'*escargot de Pythagore*, qui permet de construire, à la règle et au compas, les segments de longueur* $\sqrt{n}$ *avec* n *entier.*

Je propose de troquer ici la règle et le compas contre le langage Python *et son module* Turtle *qui implémente le langage* Logo*. Voici l'escargot, et l'actrice principale,* turtle*. Dans la foulée, intéressons-nous aussi à l'*arbre de Pythagore*, ce qui nous permettra de retrouver une excellente amie, la* récursivité*.*





1 Python, turtle, Logo, tortue

Dans [2] et [1], respectivement à propos de la suite de Padovan et des L-systèmes, on utilise le langage *Logo* et sa célèbre tortue, inventés par Seymour Papert. Pour résumer simplement, ce langage permet de dessiner en déplaçant une petite tortue à l'aide d'instructions très simples comme *avance*, *tourne à gauche*, *tourne à droite*, etc.

Modernité oblige, le célèbre et très puissant langage *Python* permet d'utiliser un module nommé *turtle* qui permet de dessiner avec quasiment les mêmes instructions que *Logo*. C'est ce module qui a été utilisé dans [5], chapitre 4, pour dessiner certains types de fractals.

2 L'escargot, lentement, mais sûrement

Peu d'instructions seront nécessaires, les explications qui suivent permettront de les comprendre facilement.

from turtle import *

Permet de charger toutes les instructions du module *turtle*.

shape("turtle")

Permet de donner au pointeur la forme d'une tortue (par défaut, c'est une pointe de flèche). Ce n'est pas du tout indispensable, juste pour le plaisir.

mode("standard")

La point de départ de la tortue est le point de coordonnée (0,0). L'instruction `mode()` permet de déterminer l'orientation initiale de la tortue et le sens de calcul des angles positifs.

Il y a deux modes simples : « standard » et « logo ». Dans le mode *standard*, l'orientation initiale de la tortue est l'Est (la droite) et les angles positifs sont calculés dans le sens contraire des aiguilles d'une montre ; dans le mode *logo*, l'orientation initiale est le Nord (le haut) et les angles positifs sont calculés dans le sens des aiguilles d'une montre.

Rappelons que les longueurs s'expriment en *pixels* et les angles en *degrés*.

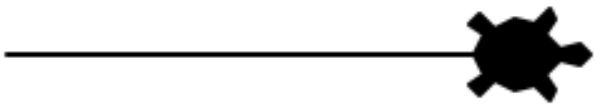
forward(distance), left(angle)

Respectivement avancer d'une certaine distance et tourner à gauche d'un certain angle, ce n'est pas bien compliqué.

```
from turtle import *

shape("turtle")
mode("standard")
```

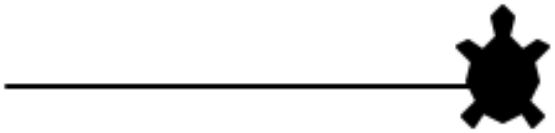
Position initiale de la tortue.

```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
```



La tortue avance de 50 pixels, et ne change pas d'orientation.

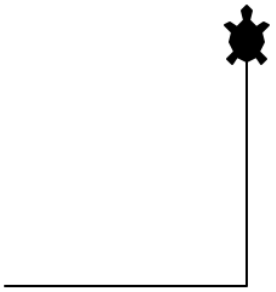
```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
left(90)
```



La tortue avance de 50 pixels, puis change
son orientation en tournant à gauche de 90 degrés.

L'escargot est composé d'une suite de triangles rectangles, le premier ayant 1 pour côtés de l'angle droit. Disons donc que 1 équivaut à 50 pixels, et nous venons de dessiner le premier côté du premier triangle. Voici pour le second côté, facile, mais gare à l'hypoténuse...

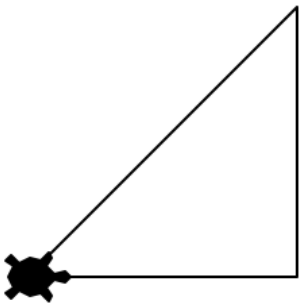
```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
left(90)
forward(50)
```



position(), home(), setposition(coord)

Pour dessiner l'hypoténuse, il suffirait de l'instruction `home()` qui ramène la tortue en position (0,0) tout en dessinant, et qui la remet dans son orientation initiale.

```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
left(90)
forward(50)
home()
```

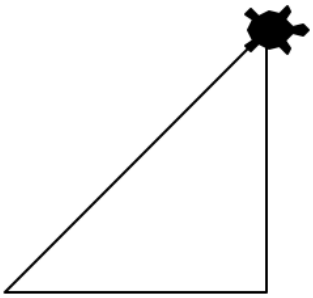


L'hypoténuse du premier triangle est le premier côté de l'angle droit du deuxième triangle ; le problème est qu'il faut retourner d'où l'on vient pour dessiner le second côté... il faut pour cela avoir mémorisé cette position. C'est le rôle de l'instruction `position()` qui retourne la dernière position sous la forme d'un couple (x,y) (mais cela ne se voit pas) qu'il suffit de placer dans une variable, que nous avons nommée p.

La fonction `setposition(p)` dessine le segment entre la position actuelle, (0,0), et p.

L'hypoténuse est dessinée deux fois (on pourrait lever la tortue, la déplacer, puis la baisser, mais ce sont deux instructions supplémentaires), mais ce qui ne nous arrange pas, c'est qu'elle retrouve son orientation initiale, et il nous faut un angle droit par rapport à l'hypoténuse qui vient d'être dessinée... il faut tourner à gauche, mais de quel angle ?

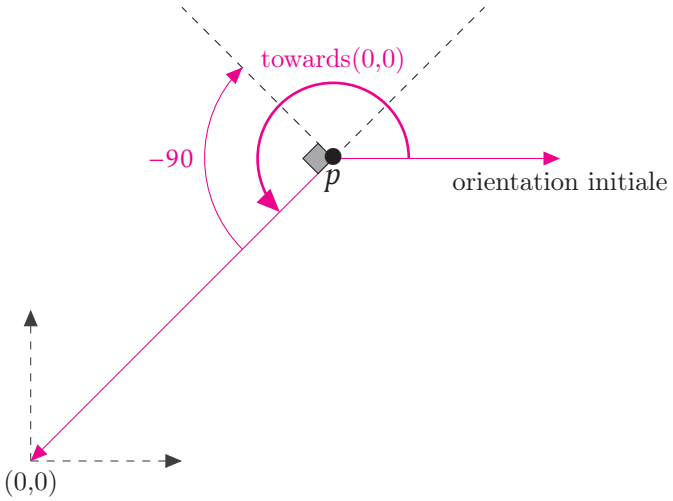
```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
left(90)
forward(50)
p=position()
home()
setposition(p)
```



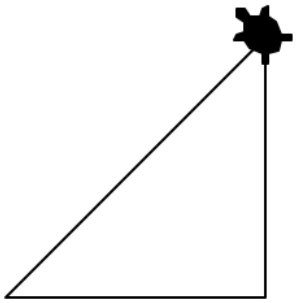
Pour le premier triangle, c'est facile : 135°. Mais pour les autres ?

towards(x,y)

Cette fonction retourne l'angle entre l'orientation initiale (ici, horizontale droite) et l'orientation de la ligne joignant la position actuelle (ici, p) et un point de coordonnée (x,y).
`towards(0,0)` retourne donc l'angle entre l'orientation initiale et l'hypoténuse du triangle. En soustrayant 90° à cet angle, on obtient la direction perpendiculaire à l'hypoténuse, voir dessin ci-contre.




```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
left(90)
forward(50)
p=position()
home()
setposition(p)
left(towards(0,0)-90)
```



La tortue est maintenant prête pour dessiner le deuxième triangle, avec la même séquence des 5 instructions `forward(50) p=position() home() setposition(p) left(towards(0,0)-90)`, et ainsi de suite pour les triangles suivants.
Il suffit donc de répéter ces 5 instructions autant de fois que le nombre de triangles souhaités.

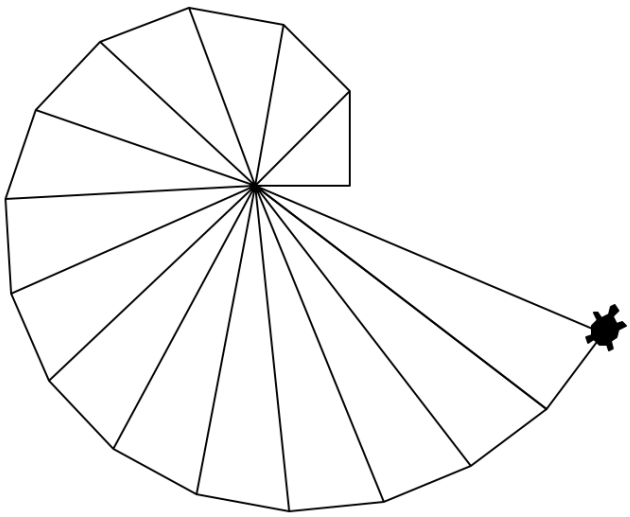
```
from turtle import *

shape("turtle")
mode("standard")
forward(50)
left(90)
for i in range(8):
    forward(50)
    p=position()
    home()
    setposition(p)
    left(towards(0,0)-90)
```

Ci-contre les instructions qui permettent de dessiner 8 triangles successifs de l'escargot, la longueur unitaire des côtés étant fixée à 50 pixels.

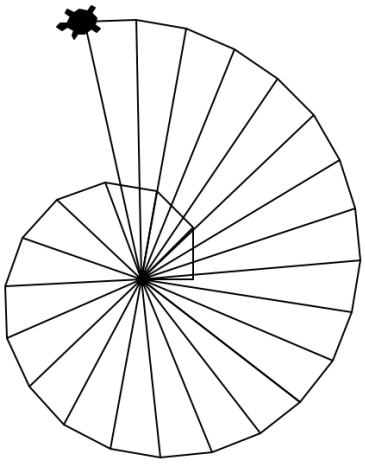
On peut facilement généraliser le processus en créant une fonction à deux variables, que nous avons nommée *escargot*, permettant de choisir la longueur unitaire (u) et le nombre de triangles (n).

Ci-après le script final pour dessiner les 15 premiers triangles avec une longueur unitaire de 50 pixels. La fonction `mainloop()` permet de fixer le dessin à l'écran, le temps nécessaire pour l'admirer...

```
# escargot.py

from turtle import *

shape("turtle")
mode("standard")

def escargot(u,n):
    forward(u)
    left(90)
    for i in range(n):
        forward(u)
        p=position()
        home()
        setposition(p)
        left(towards(0,0)-90)

escargot(50,15)
mainloop()
```



Il est vrai que dans la nature, les escargots se présentent le plus souvent sous la forme *escargot(30,25)*...



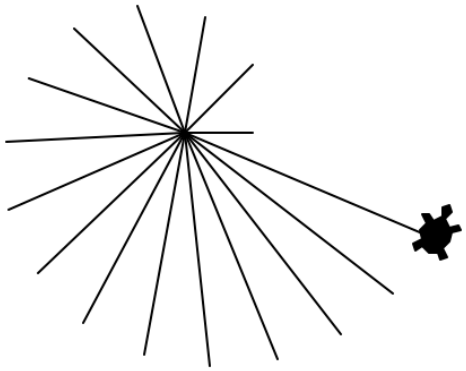
3 Prolongations

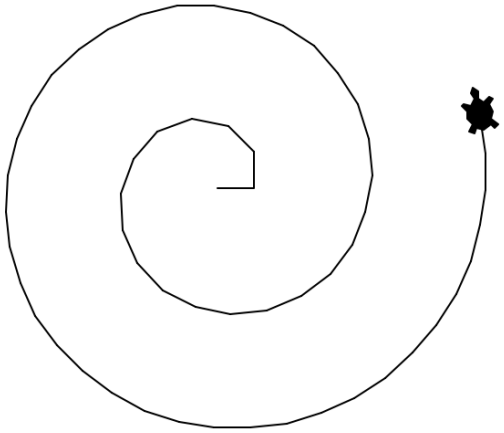
3.1 Hypoténuses et spirale

À partir du script précédent, il est facile de ne dessiner que les hypoténuses, pour visualiser la suite des nombres $\sqrt{n}$, ou que les côtés de longueur unitaire, pour visualiser la spirale, dite *spirale de Théodore* DE CYRÈNE (465–398 av J.-C.).

Deux instructions supplémentaires seront nécessaires : `up()` ou `penup()` lève la tortue et `down()` ou `pendown()` baisse la tortue.

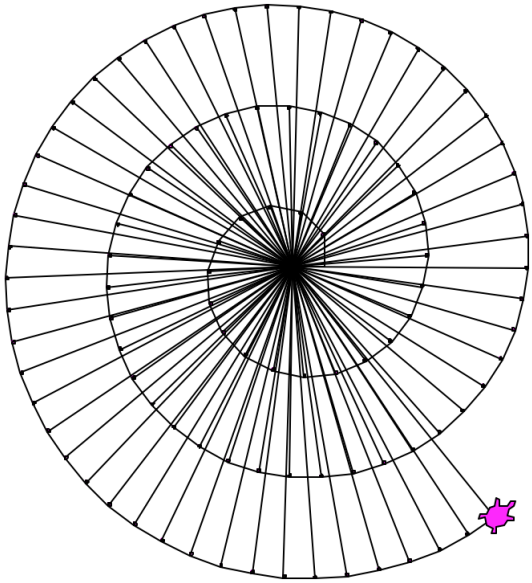
Le lecteur se fera un plaisir de tester... il suffit d'encadrer les bonnes instructions par `up()`et `down()`.





3.2 Points alignés ?

Dans la figure ci-dessous, on a marqué d'un point l'extrémité de chaque hypoténuse.



```
from turtle import *
shape("turtle")
mode("standard")
fillcolor("magenta")
def escargot(u,n):
    forward(u)
    left(90)
    for i in range(n):
        forward(u)
        begin_fill()
        circle(1)
        end_fill()
        p=position()
        home()
        setposition(p)
        left(towards(0,0)-90)
escargot(20,100)
```

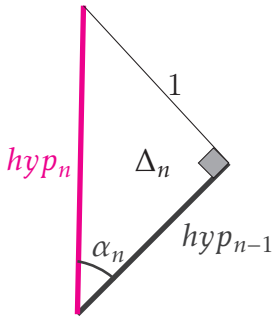
En faisant plusieurs tours, on peut se demander si, à un moment donné, deux points et le centre seront alignés... la réponde est NON !

Théorème : *Une droite passant par l'origine et un des points ne contient aucun autre de ces points* (Erich TEUFFEL, 1958). ([6]).

3.3 Nombre de triangles dans le « premier tour »

Comme le montre la figure de la page 272, après un certain nombre de triangles dessinés, le suivant « chevauche » le premier. En calculant la somme cumulée des angles au centre, on arrêtera de dessiner les triangles lorsque cette somme cumulée dépasse 2π.

Il faudra faire appel à notre amie trigonométrie et donc utiliser le module *math* de Python ; ce module *math* s'exprime en radians, mais nous allons utiliser une fonction de conversion pour afficher en degrés l'évolution du cumul des angles.



Pour calculer l'angle α_n du triangle Δ_n, on utilisera la formule

$$\alpha_n = \arctan\left(\frac{1}{hyp_{n-1}}\right) = \arctan\left(\frac{1}{\sqrt{n-1}}\right)$$

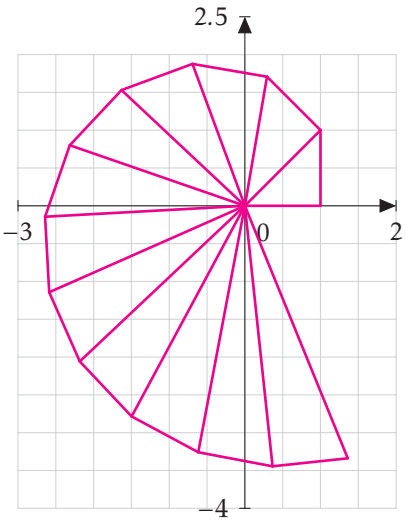
```
from turtle import *
from math import *
shape("turtle")
mode("standard")

def escargot(u):
    n=1
    angle=pi/4
    forward(u)
    p=position()
    left(90)
    while angle <= 2*pi :
        n=n+1
        angle=angle+atan(1/sqrt(n))
        angled=degrees(angle)
        print(angled)
        forward(u)
        p=position()
        home()
        setposition(p)
        left(towards(0,0)-90)
    print(n)

escargot(30)
mainloop()
```

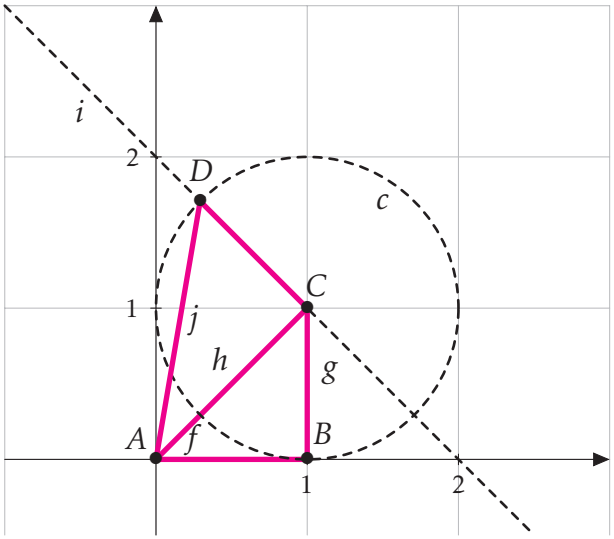
La fonction escargot() a perdu un argument, car le nombre n de triangles n'est plus fixé à l'avance.
La répétitive est initialisée avec le premier triangle ($n = 1$) et le premier angle ($angle = \frac{\pi}{4}$).
Les fonctions atan(), sqrt(), degrees() du module *math* calculent, sans grande suprise, un *arc tangente*, une *racine carrée* et une *conversion radians-degrés*.

Lors de l'exécution, en plus de l'escargot, on obtient les résultats suivants :
80.26438968275465
110.26438968275465
136.82944085983263
160.92428341194332
183.13193771053983
203.83674876517526
223.30796939966595
241.74291822258795
259.29131883638024
276.06997371734064
292.17208746932664
307.6734470362636
322.63666446957075
337.11417665550067
351.1504201234272
364.7834423487936
17

3.4 L'escargot et Géogébra

Pour qui n'est pas porté sur la programmation, on peut aussi utiliser des applications dédiées au dessin géométrique. *Géogébra* en est une. Voici une méthode simple pour dessiner les triangles de l'escargot. C'est peut-être l'occasion d'introduire *Géogébra* au cours de mathématique...





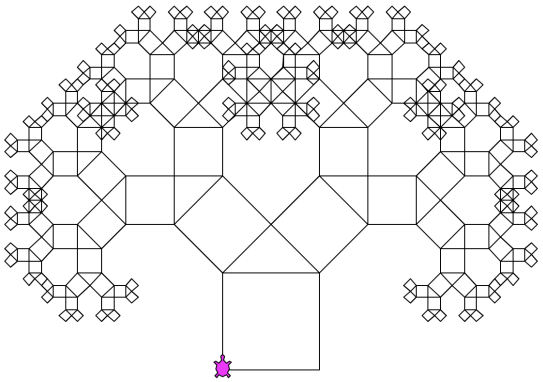
Le lecteur est supposé connaître les manipulations de base du logiciel. Les noms des objets dépendent de l'ordre de leur construction. On se réfèrera au dessin ci-dessus. Afin de l'alléger, on y a représenté en pointillés ce qui sert à la construction, et en magenta ce que l'on veut finalement représenter.

- Avec l'outil Segment, dessiner le premier triangle passant par les points de coordonnées (0,0), (1,0) et (1,1).
- Avec l'outil Perpendiculaire, tracer une perpendiculaire à l'hypoténuse h passant par le point C, soit la droite i.
- Pour obtenir sur cette perpendiculaire un segment de longueur 1, tracer, avec l'outil Cercle, un cercle de centre C et de rayon 1, soit c.
- Avec l'outil Point, marquer le point d'intersection entre le cercle c et la droite i, soit D.
- Avec l'outil Segment, dessiner l'hypoténuse du deuxième triangle, soit AD.
- Et ainsi de suite pour les triangles suivants.
- Reste ensuite à désactiver l'affichage de la grille, des axes, des cercles, des perpendiculaires, et des noms des objets.

4 L'arbre... un fractal

Ce n'est sans doute pas PYTHAGORE qui s'en est occupé, mais le nom provient du fait que l'arbre est constitué de carrés construits sur les côtés de triangles rectangles, éléments représentatifs du théorème bien connu.

Dans ce premier exemple, les triangles sont rectangles isocèles ; ci-contre 8 niveaux dans la *récursivité*.



La *récursivité* a été introduite dans [3] pour dessiner la ligne fractale et le flocon de KOCH. Rappelons simplement que c'est un outil qui permet à une procédure de s'appeler elle-même, et l'arbre de PYTHAGORE est un fractal *auto-similaire*.

```
from turtle import *
from math import *
shape("turtle")
mode("logo")
speed(0)

pile =[]
def push():
  pile.append((heading(),position()))

def pop():
  h, p = pile.pop()
  setheading(h)
  up()
  setposition(p)
  down()

def carre(c):
  for i in range(4):
      forward(c)
      right(90)
  up()
  forward(c)
  down()
```

La *récursivité* est très chronophage ; speed(0) accélère l'affichage.

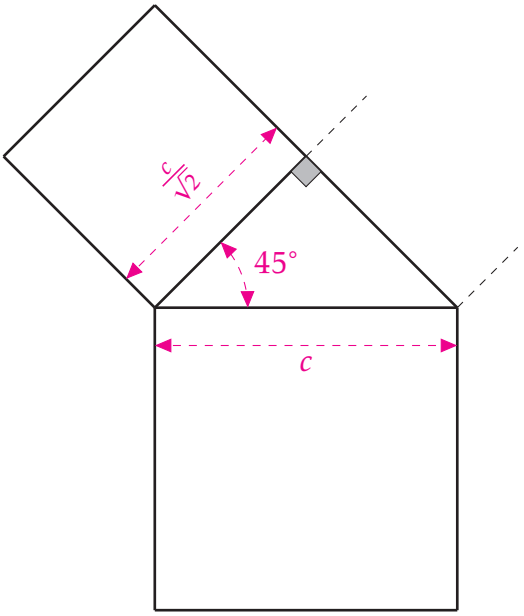
pile est un tableau dans lequel seront mémorisées les coordonnées de la tortue.

La fonction push() mémorise le cap (heading()) et la position (position()) de la tortue.

La fonction pop() restaure la dernière coordonnée mémorisée, dans la variable h pour le cap et dans la variable p pour la position. Les fonctions setheading(h) et setposition(p) redonnent à la tortue le cap et la position ainsi restaurés. setposition(p) doit être exécuté sans dessiner, d'où l'encadrement par up() et down().

La fonction carre(c) dessine un carré de côté c. C'est habituellement la première fonction que l'on découvre dans toute initiation au langage *Logo*. Les 3 dernières instructions placent la tortue dans le coin supérieur gauche après le tracé du carré.

Voilà pour le début.



Il faut aussi parler de dimensions...

Lorsque, à une étape donnée, on a dessiné un carré de côté c, les carrés de l'étape suivante auront un côté $c\frac{\sqrt{2}}{2} = \frac{c}{\sqrt{2}}$.

Et il faudra aussi un changement d'orientation de 45°.

```
def arbre(long,n):

    if n==0:
        return
    push()
    carre(long)
    left(45)

    # carré gauche
    arbre(long/sqrt(2),n-1)
    right(90)

    up()
    forward(long/sqrt(2))
    down()

    # carré droit
    arbre(long/sqrt(2),n-1)
    pop()
```

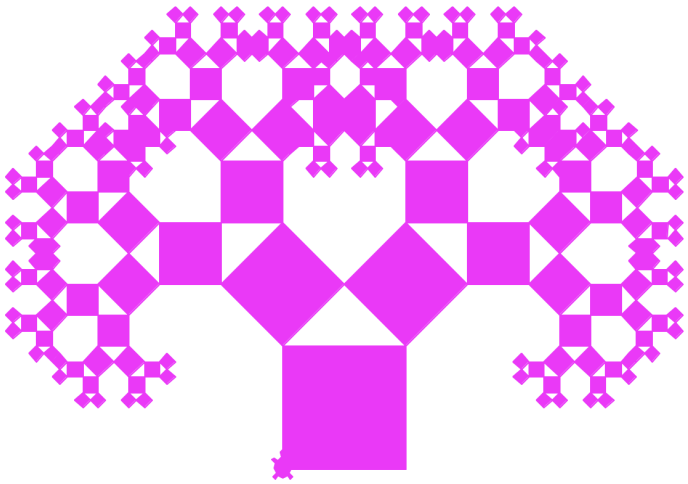
La fonction arbre(long,n) dessine l'arbre, *long* est la longueur du côté du carré et n le niveau.

Si $n = 1$, on ne dessine qu'un seul carré (le tronc), car à l'appel suivant de la fonction arbre(...,n-1) le niveau n devient nul et la procédure s'arrête.

Si $n = 2$, on dessine le premier carré ; la tortue est dans le coin supérieur gauche et pointe vers le haut. Rotation à gauche de 45°, dessin du carré « gauche » de niveau suivant ($n = 1$) avec longueur adaptée. La tortue est de retour au coin supérieur gauche du « carré-tronc », orientation nord-ouest. Rotation à droite de 90°. Il faut avancer le long de la base du carré gauche pour débuter le tracé du carré droit. Ce trajet est normalement tracé « par dessus » le côté du carré gauche. Mais quand il n'y a pas de carré gauche (dernier niveau), le trait se retrouve seul. Il suffit alors de lever et baisser le crayon avant et après. Etc.

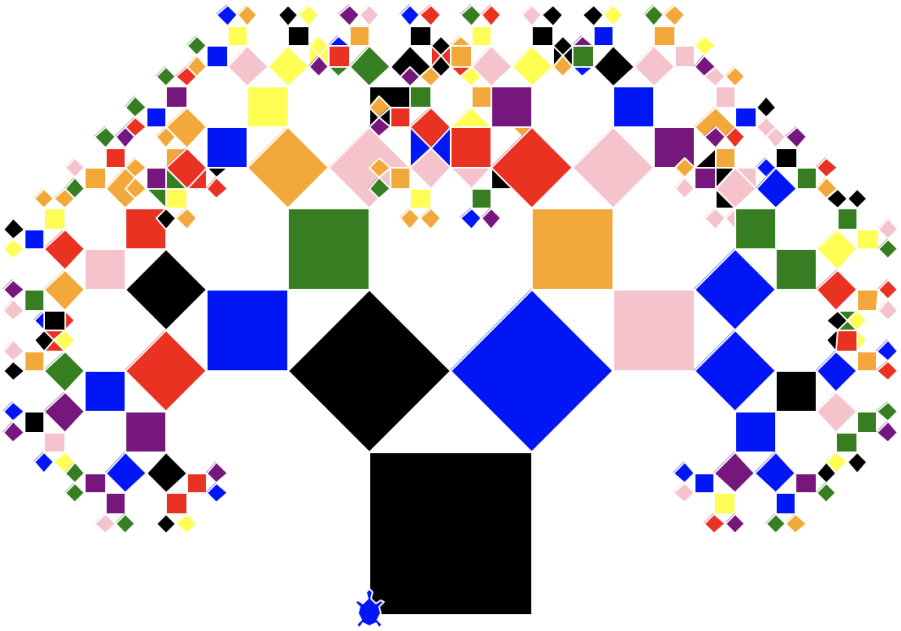
On peut remplacer un simple tracé de carrés pas des carrés « remplis »...

```
def carre(c):
    pencolor("magenta")
    fillcolor("magenta")
    begin_fill()
    for i in range(4):
        forward(c)
        right(90)
    end_fill()
    up()
    forward(c)
    down()
```



... avec des couleurs aléatoires

```
import random
couleurs  = ["red","green","blue","orange","purple","pink","yellow","black"]
def carre(c):
    pencolor("white")
    fillcolor(random.choice(couleurs))
    begin_fill()
    for i in range(4):
        forward(c)
        right(90)
    end_fill()
    up()
    forward(c)
    down()
```



arbre(100,8)

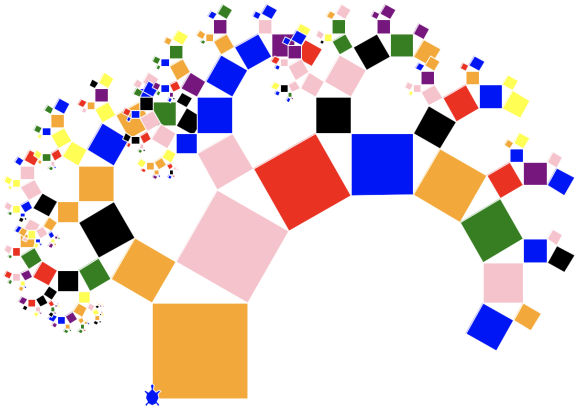
Triangles rectangles non isocèles

Il faut ajouter un paramètre supplémentaire (*angle*) à la fonction arbre().

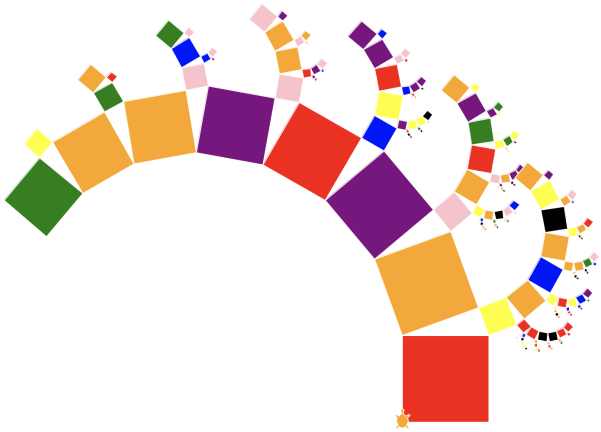
La tortue s'exprime en radians...

Le lecteur reconnaîtra les formules de calcul de la longueur des côtés de l'angle droit d'un triangle rectangle...

```
def arbre(long,n,angle):
    if n==0:
        return
    push()
    carre(long)
    left(angle)
    arbre(long*cos(pi*angle/180),n-1,angle)
    right(90)
    penup()
    forward(long*cos(pi*angle/180))
    pendown()
    arbre(long*sin(pi*angle/180),n-1,angle)
    pop()

arbre(100,8,60)

mainloop()
```



arbre(100,8,60)



arbre(100,8,20)

Références

[1] Desbonnez J.-M., Aristid Lindenmayer et Seymour Papert, pour des courbes à gogo. *Losanges*, 40, pp. 49–60. 2018.

[2] Desbonnez J.-M., La suite de Padovan, d'argent et de plastique. *Losanges*, 36, pp. 46–54. 2017.

[3] Desbonnez J.-M., vonKoch, la Tortue et Ruby. *Losanges*, 34, pp. 49–54. 2016.

[4] Ninove L., Un rectangle nommé d'argent (2). *Losanges*, xx, pp. yy–zz. 2021.

[5] Noël G. et Desbonnez J.-M., *Raconte-moi des histoires de fractals*. SBPM, 2020.

[6] Teuffel E., Eine Eigenschaft der Quadratwurzel Schneke. *Math Phys, Semester berichte*, 6, pp. 148–152. 1958.