Statistiques avec CoffeeScript

CoffeeScript est un langage de programmation comme les autres, il n'est ni plus doué, ni moins doué qu'un autre[1] pour faire de la statistique. Cependant, CoffeeScript est un langage du web, ce qui le prédestine au traitement de données en ligne, et ces données peuvent très bien être statistiques. Par ailleurs, les possibilités graphiques et autres d'*alcoffeethmique* prédestine celui-ci à des activités de nature statistique.

Le principal problème en statistique est celui de la saisie des données : Comment entrer tous ces nombres dans la calculatrice ? Avec l'ordinateur, le problème est le même[2], sauf si on peut ouvrir un fichier contenant toutes les données, ce qui, pour html[3], est possible avec la technologie AJAX, ou en passant par le tableur de *Google Drive*.

Ceci dit, il est possible avec tout logiciel de mathématiques, d'obtenir des données statistiques, tout simplement en les choisissant de nature mathématique ! C'est ce qui sera fait ici, en utilisant les possibilités algorithmiques d'*alcoffeethmique* pour extraire des données statistiques, et bien entendu pour les exploiter.

I/ Nombres normaux

1) Obtention des décimales

Une question souvent ouverte est de savoir si un nombre donné est, ou non, ***normal*** en base 10. On dit qu'un nombre est normal si les fréquences de ses décimales sont égales. Pour voir si un nombre a l'air normal, il suffit donc de faire des statistiques sur ses 1000 premières décimales[4]. On va ici comparer deux nombres pour la normalité : $\sqrt{3}$, qui semble normal, et $\frac{123}{1111}$ qui ne l'est évidemment pas. Dans les deux cas, on a besoin d'un algorithme pour avoir le tableau des décimales :

- on positionne à 1000 la variable `Big.DP` qui fixe le nombre de décimales de Big[5] ;
- on fait calculer par Big la racine de 3 à 1000 décimales, avec `rac3 = Big(3).sqrt()` ;
- on convertit le résultat en chaîne de caractères avec `rac3.toString()` ;
- on ne garde que les décimales, en enlevant le 1 et la virgule, avec `lettres = rac3.toString()[2..]` ;
- enfin on crée un tableau dans lequel on place les lettres, reconverties en nombres, avec `données = (parseInt(x) for x in lettres)`.

C'est sur ce tableau qu'on va faire des statistiques.

Pour le second nombre, on fait pareil mais au début, le calcul est `Big(123).div(Big(1111))`.

---

1 À part R, qui lui est vraiment fait pour ça...
2 À moins de disposer d'un bon outil de reconnaissance de la parole, mais au risque de se ridiculiser auprès des voisins...
3 Html est le biotope idéal pour CoffeeScript, même si ce n'est pas le seul (CaRMetal par exemple).
4 Reste ensuite à démontrer la normalité conjecturée, ce qui est nettement plus difficile. Par exemple, qu'est-ce qui prouve qu'au-delà de la millionième décimale, il n'y a pas surabondance de 7 ?
5 Big est un logiciel libre, récupéré ici, et qui a été incorporé à *alcoffeethmique*. Il permet de faire du calcul en grande précision.

Pour le tableau des 1000 premières décimales de $\sqrt{3}$, le script est le suivant :

```
Big.DP = 1000
rac3 = Big(3).sqrt()
lettres = rac3.toString()[2..]
données = (parseInt(x) for x in lettres)
affiche données
```

On peut vérifier que tout s'est bien passé en remplaçant la dernière ligne par `affiche données.length` qui donne l'effectif total (théoriquement 1000).

2) Tableau d'effectifs

L'effectif du chiffre 7 est donc le nombre de fois que le 7 apparaît dans le tableau des `données`. Comme il est fastidieux de compter les 7, autant demander à CoffeeScript de le faire, avec

```
affiche données.compteLes 7
```

Le seul fait que le 7 apparaisse 98 fois sur 1000 est un indice très fort en faveur de la normalité de $\sqrt{3}$.

Alors il devient simple de constituer un tableau d'effectifs, sous forme de tableau associatif, dans lequel on place, en face de chaque `chiffre`, le nombre d'occurrences de celui-ci dans le tableau des `données` :

```
tabEff = new Array()
tabEff[chiffre] = données.compteLes(chiffre) for chiffre in [0..9]
mettreDansTableau tabEff
```

Ceci remplit le tableau dans la partie « sorties graphiques ».

Voici le tableau d'effectifs obtenu :

chiffres	0	1	2	3	4	5	6	7	8	9
effectifs	95	97	100	97	84	93	103	98	125	108

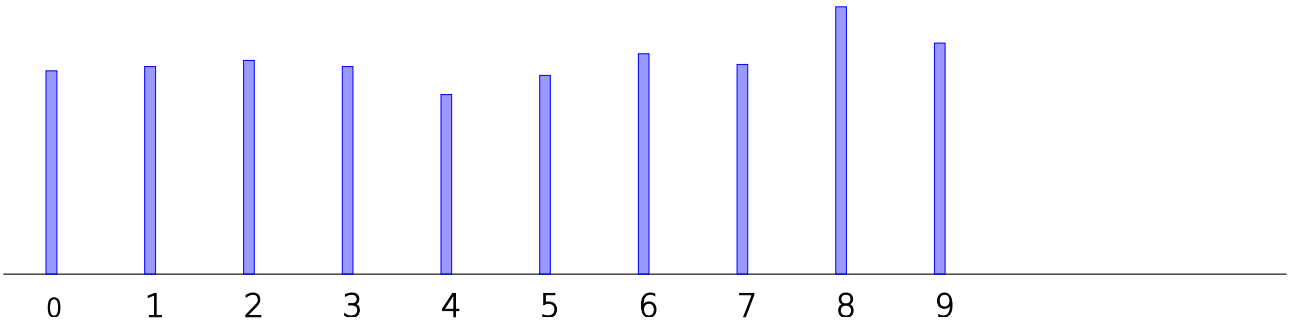
Pour vérifier l'équirépartition des chiffres, il faut faire un test de χ^2 mais cela est hors programme.

3) diagramme en bâtons

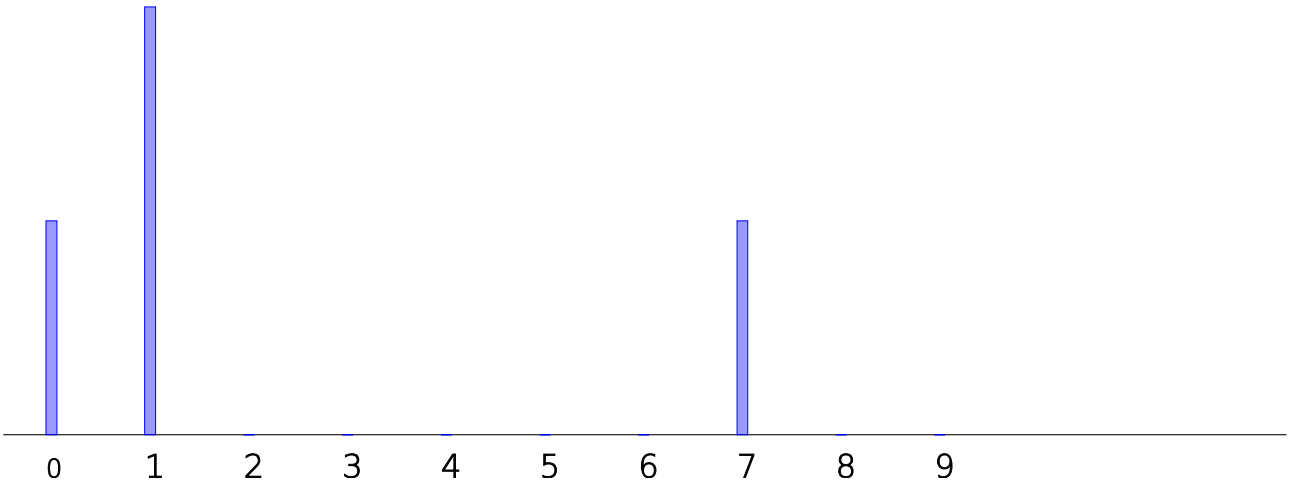
Une fois qu'on a un tableau, on peut avoir un graphique par simple conversion :

```
Big.DP = 1000
rac3 = Big(3).sqrt()
lettres = rac3.toString()[2..]
données = (parseInt(x) for x in lettres)
tabEff = new Array()
tabEff[chiffre] = données.compteLes(chiffre) for chiffre in [0..9]
diagrammeBatonsTrie tabEff, 400
```

Voici le graphique obtenu :



On y voit bien l'aspect équiréparti des décimales, donc l'apparente normalité de $\sqrt{3}$. À titre de comparaison, voici le diagramme en bâtons des 1000 premières décimales de $\frac{123}{1111}$:



4) caractéristiques de position

- médiane

Pour calculer la médiane, on délègue une fois de plus à alcoffeethmique :

```
affiche laMédianeDe données
```

On trouve 5, ce qui n'est pas surprenant au vu de ce qui précède. Le chiffre médian de $\frac{123}{1111}$ est de façon analogue, égal à 1.

- moyenne

Idem pour la moyenne de la série :

```
Big.DP = 1000
rac3 = Big(3).sqrt()
lettres = rac3.toString()[2..]
données = (parseInt(x) for x in lettres)
affiche laMoyenneDe données
```

On a alors l'affichage suivant, qui montre que le chiffre moyen est 4,665 :

```
Algorithme lancé
4.665

Algorithme exécuté en 1164 millisecondes
```

Alors que pour $\frac{123}{1111}$, le chiffre moyen est 2,25. On peut le calculer en constatant que le développement décimal de $\frac{123}{1111}$ est périodique de période « 1107 » ; or la moyenne de 1, 1, 0 et 7 est (1+1+0+7)/4=9/4=2,25.

5) caractéristiques de dispersion

- Quartiles

Pour avoir les quartiles, de façon similaire :

```
affiche lePremierQuartileDe données
affiche leDernierQuartileDe données
```

Pour $\sqrt{3}$, les quartiles sont 2 et 7 alors que pour $\frac{123}{1111}$, ce sont 1 et 7.

- Écart-type

Pareil :

```
affiche lEcartTypeDe données
```

On trouve un écart-type de 2,9214337233625485 à comparer avec l'écart-type pour $\frac{123}{1111}$ qui est 2,7726341266023544.

Calculs théoriques :

Pour des chiffres équirépartis, la moyenne est (0+1+2+3+4+5+6+7+8+9)/10=4,5 (à comparer avec les 4,665 vus ci-dessus). Quant à l'écart-type, on trouve $\sqrt{\frac{33}{4}} \approx 2{,}8722813232690143$ à comparer avec les 2,9214337233625485 vus ci-dessus.

Pour les chiffres 1,1,0,7, on a vu que la moyenne est 2,25 mais l'écart-type est $\sqrt{\frac{123}{16}} \approx 2{,}7726341266023544$ qui ne sont pas loin de la valeur expérimentale trouvée ci-dessus.

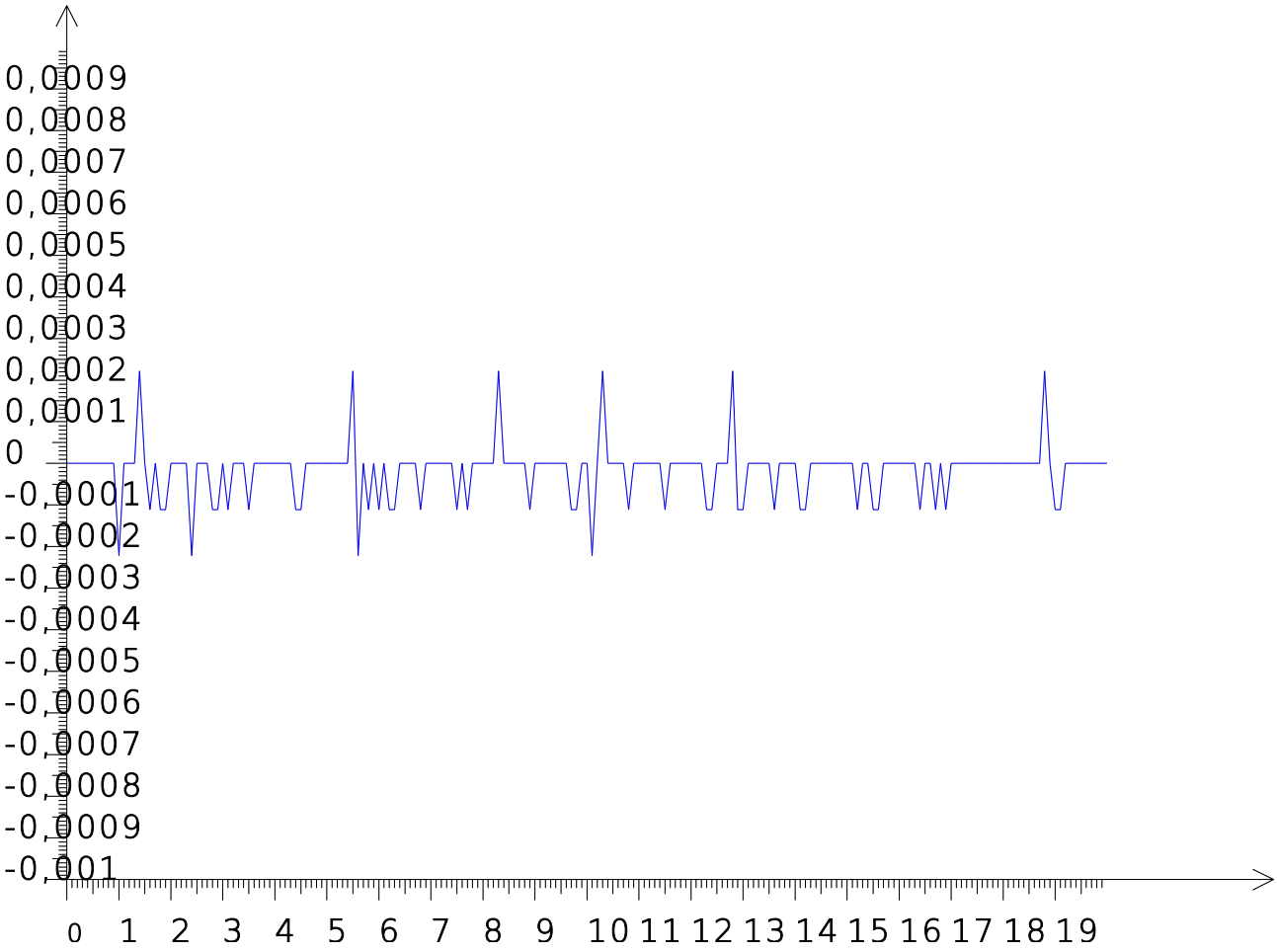
II/Erreurs d'approximation

Ici on va s'intéresser à la fonction nulle, qui, à tout réel x, associe 0. On va se concentrer sur deux expressions de la fonction nulle :

- L'expression trigonométrique $\cos^2(x)+\sin^2(x)-1$, qui est bien égale à 0 d'après le théorème de Pythagore.
- L'expression logarithmique $\ln(e^x)-x$ qui est aussi égale à 0 pour tout $x\in\mathbb{R}$.

En réalité, aucune de ces fonctions n'est vraiment nulle si on la calcule sur ordinateur, ainsi que le montrent leurs représentations graphiques sur [0;20].

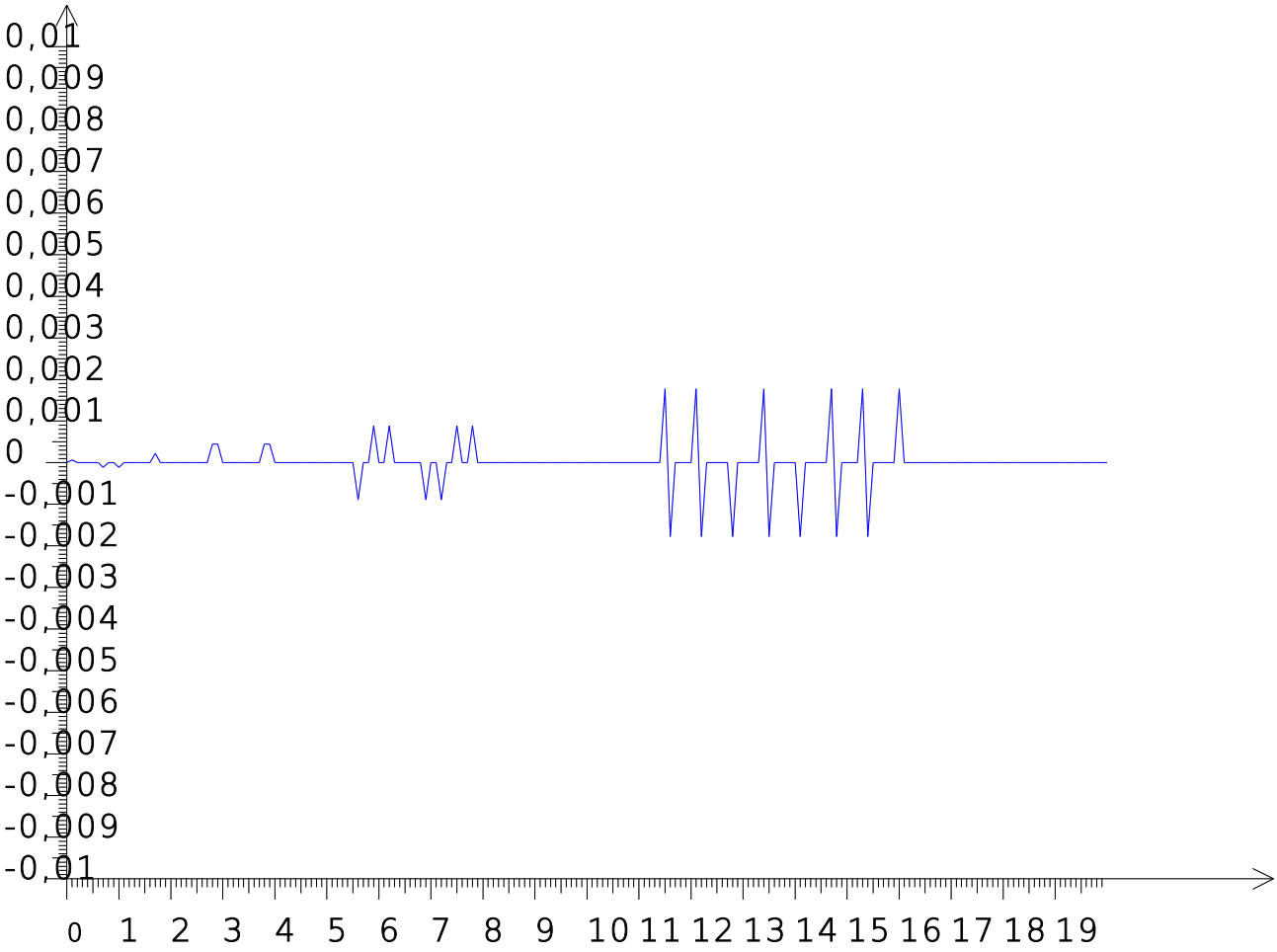
La représentation graphique de la première (ou plutôt $10^{12}\times(\cos^2(x)+\sin^2(x)-1)$ qui est aussi la fonction nulle puisque $10^{12}\times 0=0$) :



Cette représentation graphique a été obtenue avec le script suivant :

```
f = (x) ->
    1e12*(carré(sin(x))+carré(cos(x))-1)

dessineFonction f, 0.1, 20, -0.001, 0.001, 'blue'
```

Et la seconde fonction ($10^{12}\times\left(\ln\left(e^{x}\right)-x\right)$) :



La seconde fonction nulle a des erreurs qui ne sont pas systématiques (on trouve vraiment 0 assez souvent quand même) mais qui tendent à s'amplifier avec x ; alors que les erreurs d'approximation semblent rester bornées avec la première fonction. On s'attend donc à une répartition statistique différente dans les deux cas.

1) Tableau de valeurs

Les données statistiques seront ici obtenues directement à partir d'un tableau de valeurs : Celles de la fonction pour des valeurs de x équidistantes entre 1 et 2. En convertissant ce tableau de valeurs en ensemble, on voit qu'il n'y en a que 4, comme l'examen de la représentation graphique permettait de le conjecturer :

```
Algorithme lancé
{-0.0002220446049250313,0.0002220446049250313,-
0.00011102230246251565,0}

Algorithme exécuté en 21 millisecondes
```

Le script ayant donné ce résultat est celui-ci :

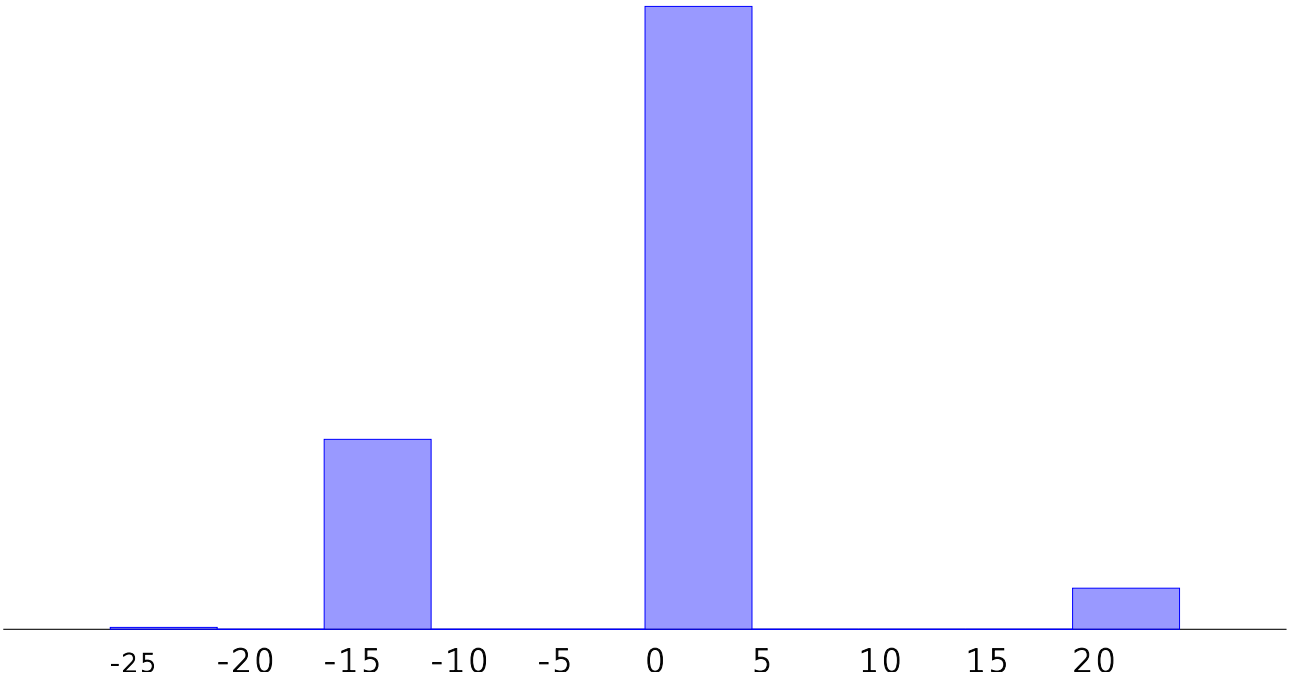
```
h = 0.001
f = (x) ->
    1e12*(carré(sin(x))+carré(cos(x))-1)

tabVal = (f(1+n*h) for n in [1..1000])
affiche new Ensemble tabVal
```

2) Histogramme

Pour avoir l'histogramme de ces valeurs, on multiplie la fonction nulle par 10^{17} au lieu de 10^{12} pour avoir des bornes entières :

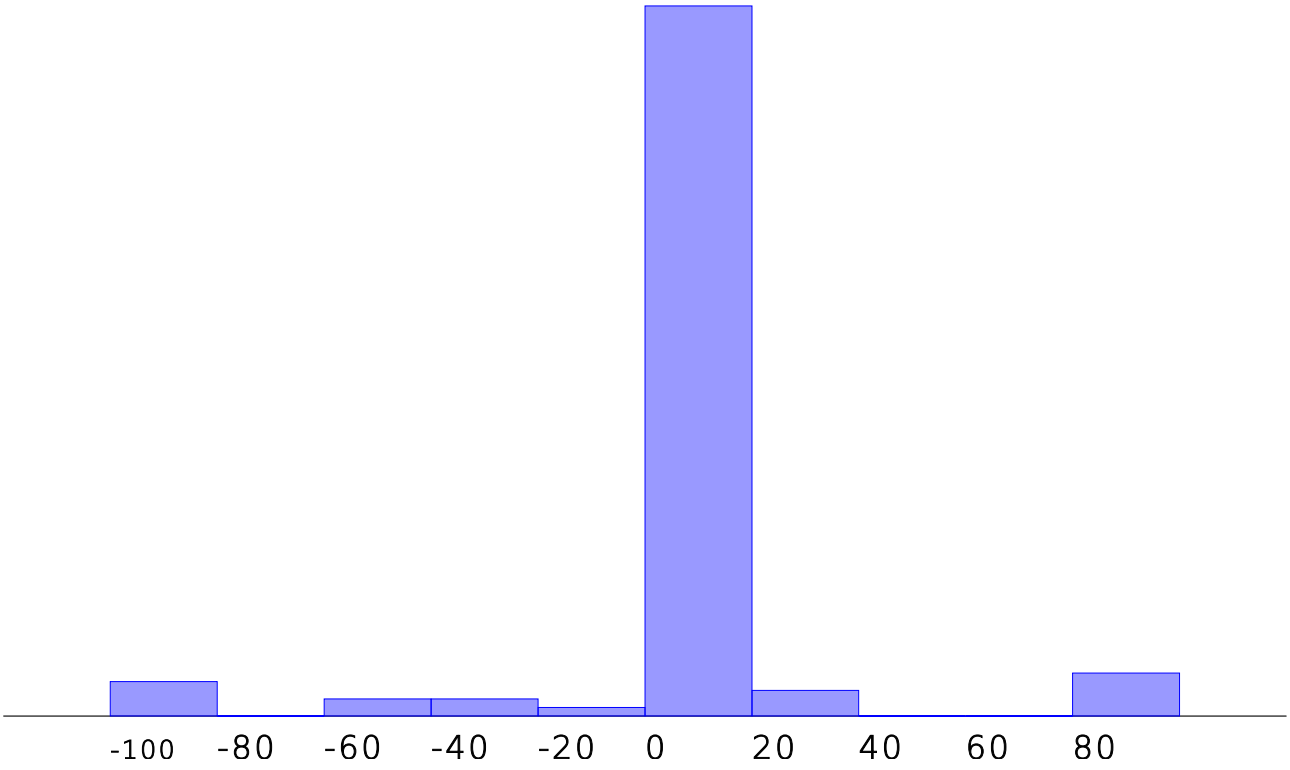
```
h = 0.001
f = (x) ->
    1e17*(carré(sin(x))+carré(cos(x))-1)

tabVal = (f(1+n*h) for n in [1..1000])
histogramme tabVal, -25, 25, 10, 1000
```

Voici l'histogramme obtenu (en réalité un diagramme en bâtons vu la rareté des éventualités) :



On voit que 0 n'est pas toujours égal à 0...

Pour la fonction nulle calculée par logarithme, l'histogramme est plus complexe :



Le script ayant dessiné cet histogramme est ici :

```
h = 0.1
f = (x) ->
    1e17*(ln(exp(x))-x)

tabVal = (f(n*h) for n in [1..100])
histogramme tabVal, -100, 100, 10, 100
```

3) Moyenne et écart-type

Pour les séries statistiques ci-dessus, voici les moyennes et écart-types :

- Version Pythagore $10^{17}\times\left(\cos^2(x)+\sin^2(x)-1\right)$, moyenne -2,220446049250313 et écart-type 6,2803698347351[6] ;
- Version logarithme $10^{17}\times\left(\ln(e^x)-x\right)$, moyenne 0,2914335439641036 et écart-type 27,87349103633203.

6 L'écart-type des erreurs est proche de $2\pi\times10^{-17}$; est-ce un hasard ?

III/ Statistique et bio-informatique

Comme exemple de caractère qualitatif, on va voir ici la répartition des bases nucléiques dans un gène de caféier : Celui qui code la fabrication d'une enzyme appelée caféine-synthase 7, une protéine permettant de transformer la 7-methylxanthine en caféine. Bref le gène qui, dans le *coffea arabica*, code la synthèse de la caféine. Ce gène est dans l'ARN du caféier, et il est donc représenté par une chaîne de caractères parmi les suivants :

- A pour adénine
- C pour cytosine
- G pour guanine
- T pour thymine

La première étape, c'est de se procurer l'ADN correspondant, en le téléchargeant sur Internet. Il est répertorié sur la base de données du NIH. Il est fourni, sous la forme d'un document texte, en annexe du présent document.

Ensuite, on lance *alcoffeethmique*, on écrit

```
gene = ""
```

Puis on fait un copier-coller entre le contenu du fichier caffeine.txt et l'emplacement entre les guillemets, de façon à avoir quelque chose qui commence comme ceci :

```
gene = "ATTATGTACG(...)
```

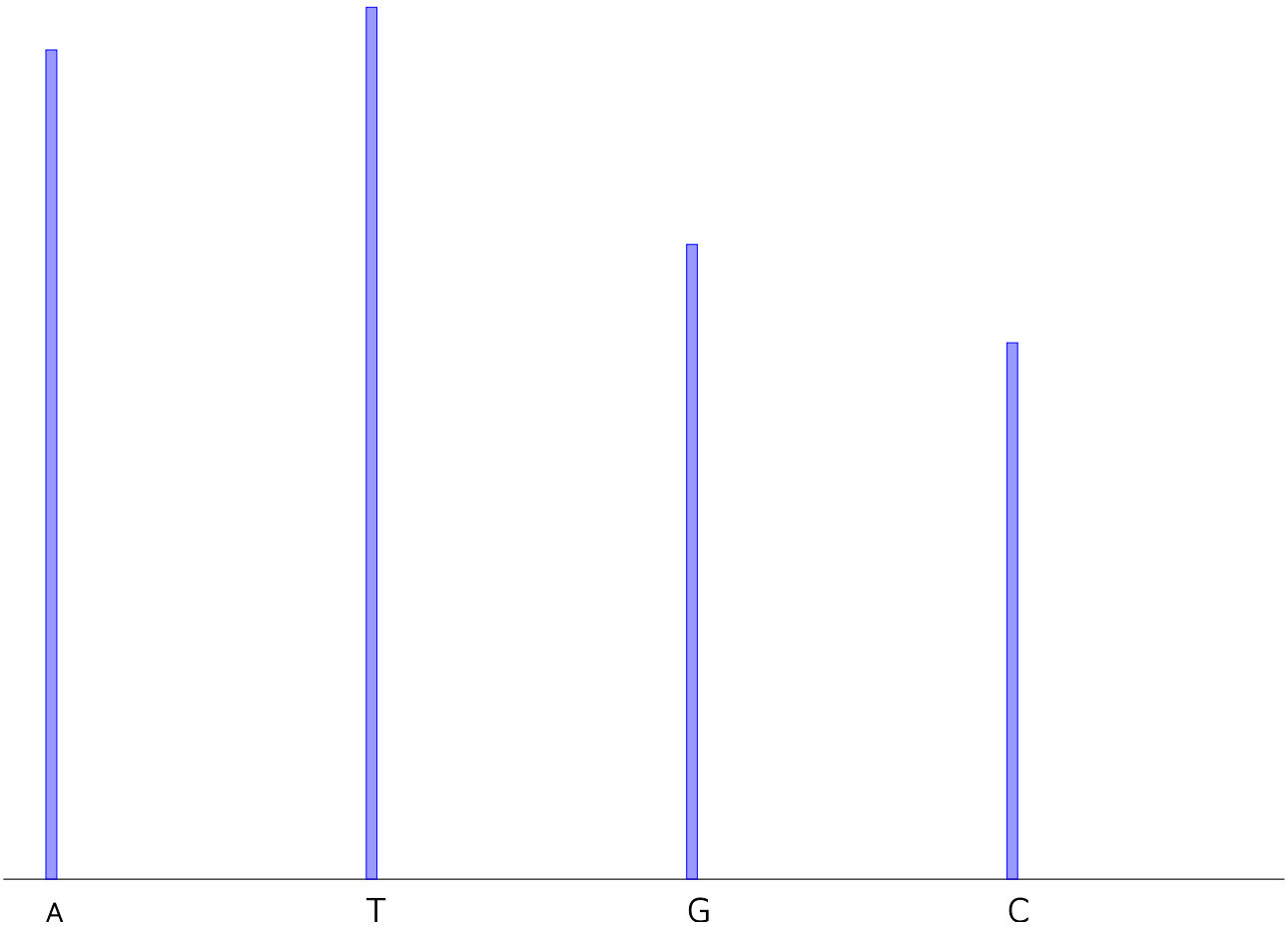
Le tableau d'effectifs des bases nucléiques s'obtient alors ainsi :

```
bases = (x for x in gene)
tabEff =  new Sac()
for base in bases
    tabEff.ajoute base
affiche tabEff
```

Ce qui produit l'affichage suivant :

```
Algorithme lancé
{"A":388,"T":408,"G":297,"C":251}

Algorithme exécuté en 19 millisecondes
```

Ce tableau d'effectifs montre que les bases nucléiques ne sont pas équiréparties dans l'ADN du caféier. On peut le confirmer en dessinant le diagramme en bâtons :



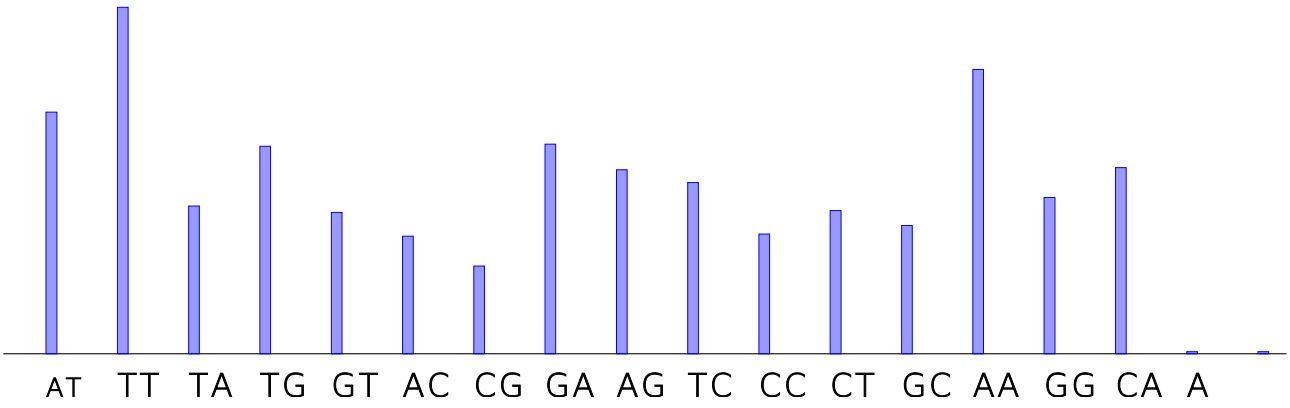
Le script ayant servi à dessiner ce diagramme en bâtons est celui-ci :

```
gene = "ATTATGTACG(...)
bases = (x for x in gene)
tabEff =  new Sac()
for base in bases
    tabEff.ajoute base
diagrammeBatons tabEff.effectifs
```

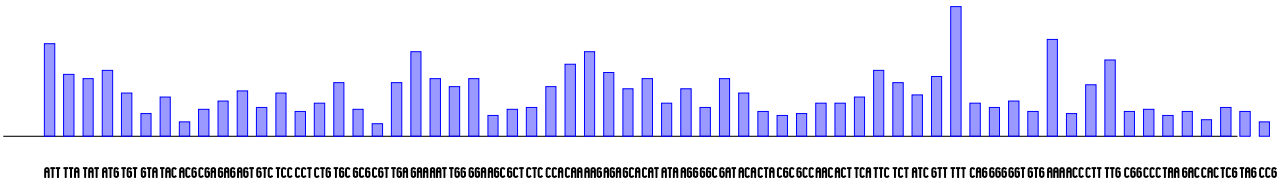
Cependant, en génétique, ce ne sont pas les bases nucléiques qui sont importantes, mais les suites de bases, en particulier les triplets de bases qui se suivent. Pour faire une statistique sur les doublets de base, on modifie légèrement le script précédent :

```
gene = "ATTATGTACG(...)
bases = (gene[n..n+1] for n in [0..gene.length])
tabEff =  new Sac()
for base in bases
    tabEff.ajoute base
diagrammeBatons tabEff.effectifs
```

Le nouveau diagramme en bâtons est plus complexe, comme on pouvait s'en douter :



Et pour les triplets de bases (en remplaçant n+1 par n+2 dans le script ci-dessus), on a un diagramme caractéristique dont les experts de Las Vegas déduiraient probablement que c'est du café :



Chacune des activités présentées ici peut donner lieu à des TP assez courts pour être faits en classe (avec pédagogie différenciée, en multipliant les exemples[7]). Mais aussi, l'export en tableau html et en svg permettent de faire des rapports de TP numériques. Enfin, l'activité sur l'alphabet {A,C,G,T} peut être modifiée pour faire un peu de cryptanalyse avec l'algorithme d'al-Kindi pour décoder un message en comptant les lettres les plus fréquentes qui y sont.

Annexe : Le gène de la caféine :

ATTATGTACGAGTCCTGCGTATGAATGGAGCTCCAAGAAGTCCTGCATATGAATGGAGGCGAAGGCGATACAAGCTACGCCAAGAACTC
ATTCTACAATCTGTTTCTCATCAGGGTGAAACCTATCCTTGAACAATGCATACAAGAATTGTTGCGGGCCAACTTGCCCAACATCAACAA
GTGCATTAAAGTTGCGGATTTGGGATGCGCTTCTGGACCAAACACACTTTTAACAGTTCGGGACATTGTACAAAGTATTGACAAAGTTG
GCCAGGAAAAGAAGAATGAATTAGAACGTCCCACCATTCAGATTTTTCTGAATGATCTTTTCCAAAATGATTTCAATTCGGTTTTCAAGT
CGCTGCCAAGCTTCTACCGCAAACTTGAGAAAGAAAATGGATGCAAAATAGGATCATGCCTGATAGGCGCAATGCCTGGCTCTTTCTAC
GGCAGACTCTTCCCCGAGGAGTCCATGCATTTTTTACACTCTTGTTACTGTTTGCATTGGTTATCTCAGGTTCCCAGCGGTTTGGTGACTG
AATTGGGGATCAGTGCGAACAAAGGGTGCATTTACTCTTCCAAAGCAAGTCGTCCGCCCATCCAGAAGGCATATTTGGATCAATTTACG
AAAGATTTTACCACATTTCTTAGGATTCATTCGGAAGAGTTGATTTCACGTGGCCGAATGCTCCTTACTTGGATTTGCAAAGAAGATGAA
TTCGAGAACCCGAATTCCATAGACTTACTTGAGATGTCAATAAACGACTTGGTTATTGAGGGACATCTGGAGGAAGAAAAATTGGACAG
TTTCAATGTTCCAATCTATGCACCTTCAACAGAAGAAGTAAAGTGCATAGTTGAGGAGGAAGGTTCTTTTGAAATTTTATACCTGGAGAC
TTTTAAGGTCCCTTATGATGCTGGCTTCTCTATTGATGATGATTACCAAGGAAGATCCCATTCCCCAGTATCCTGCGATGAACATGCTAGA
GCAGCGCATGTGGCATCTGTCGTTAGATCAATTTTCGAACCCATCGTCGCAAGTCATTTTGGAGAAGCTATTTTACCTGACTTATCCCACA
GGATTGCGAAGAATGCAGCAAAGGTTCTCCGCTCGGGCAAAGGCTTCTATGATAGTGTTATCATTTCTCTCGCCAAAAAGCCGGAGAAG
GCAGACATGTAAAAGTTTGTTTTTAGTTGGTTTTTGTGTTGTGGTTTGTTTTTAGATATGGGGAAAGGATTTAATGCGGTTCGGGGTCTAT
CGGGTATTGTACTTTTTTATATTTTTTGGTGTCTAATTATATTATATTATTAGTTTGTATTTGTAATAAGAAAAAAAAAAAAAAAA

Alain Busser
LPO Roland-Garros
Le Tampon

7 Par exemple, la fonction nulle peut aussi être vue au collège comme `carré(x)-x*x` ou `carré(x+1)-x*x-2*x-1`...