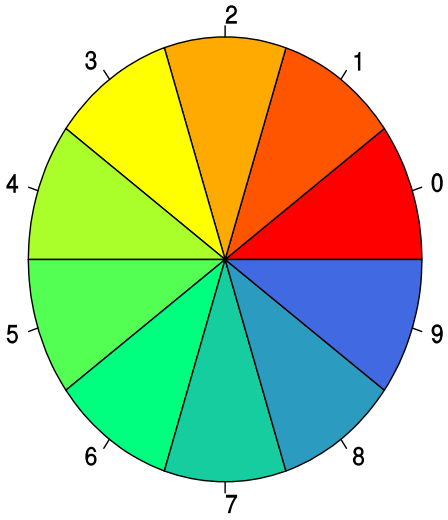
Effectif des chiffres sur le plus grand nombre premier connu

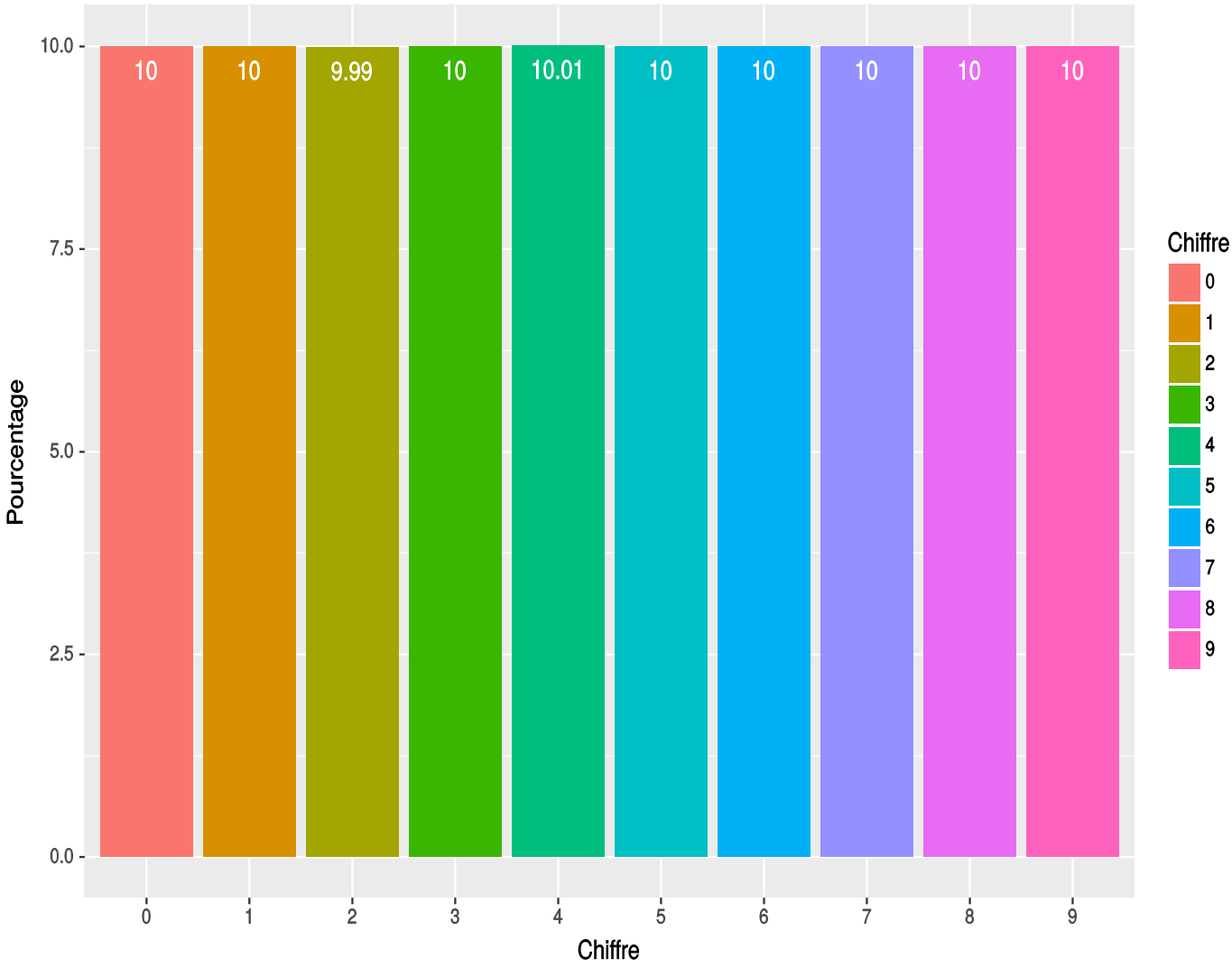
1 Introduction :

La découverte du dernier plus grand nombre premier connu a été confirmée le 3 janvier 2018.
Il s'agit du nombre $2^{77\,232\,917}-1$ qui s'écrit avec 23 249 425 chiffres en base 10, le précédent nombre était $2^{74\,207\,281}-1$ qui s'écrit avec 22 338 618 chiffres en base 10.
Le dernier nombre premier connu est disponible ici :
http://www.mersenne.org/primes/digits/M77232917.zip

En remarquant que $2^0 + 2^1 + 2^2 + 2^3 + ... + 2^n = 2^{n+1} - 1$, l'écriture binaire de ce nombre est une suite de 1, de longueur 77 232 918. On peut se demander si les chiffres de l'écriture de ce nombre en base 10 sont mieux équirépartis qu'en base 2.

Chiffre	Effectif
0	2 325 846
1	2 324 106
2	2 323 306
3	2 325 845
4	2 326 305
5	2 325 065
6	2 324 655
7	2 324 051
8	2 326 039
9	2 324 207





On présente différentes manières de compter les chiffres du plus grand nombre premier : avec Python, R, et Sqlite3, en comparant le temps pris par chaque langage.

Le fichier décompressé contenant le nombre premier est : M77232917.txt et fait environ 22Mo.

2 Avec Python :

On peut directement ouvrir le fichier, le parcourir caractère par caractère et compter en mettant le résultat dans une liste où l'élément d'indice i est l'effectif du chiffre i :

Code source : effectif-premier.py

```
fichier=open('M77232917.txt','r')
listeEff=[0]*10
N=fichier.read()

for nbr in N:
    if nbr !='\n' :
        nbr=int(nbr)
        listeEff[nbr]+=1
fichier.close()
print(listeEff)
```

Ligne de commande bash :

```
python3 effectif-premier.py
```

Affichage :
[2325846, 2324106, 2323306, 2325845, 2326305, 2325065, 2324655, 2324051, 2326039, 2324207]

Ainsi, le 0 apparaît 2 325 846 fois.

Reformatage des données pour obtenir un chiffre par ligne.

Le fichier d'entrée initial (M77232917 .txt) est une longue suite de chiffres, afin de faciliter le traitement de cette donnée avec R, on va reformater l'entrée en mettant un chiffre par ligne.

On peut faire cela par exemple avec la commande sed. En effet, on peut considérer le nombre comme un mot sur l'alphabet {0,1,2,3,4,5,6,7,8,9} et utiliser des expressions régulières. On remplace ici chaque occurrence d'un chiffre, par ce même chiffre suivi d'un retour à la ligne. Ensuite on élimine les lignes vides, le fichier initial contenant des sauts de lignes tous les 100 caractères. On créé ainsi un nouveau fichier nb_1er.csv

Ligne commande bash :

```
sed -r 's/[0-9]/&\n/g' M77232917.txt | sed '/[0-9]/!d' > nb_1er.csv
```

La taille du fichier obtenu est le double du fichier initial.

On peut aussi reformater le fichier avec Python sans avoir recours aux expressions régulières :

```
source=open('M77232917.txt')
cible=open('nb_1er-python.csv','w')
N=source.read()

for nbr in N:
     if nbr !='\n' :
          cible.write(nbr+'\n')
source.close()
cible.close()
```

3 Avec R

On utilise la bibliothèque data.table qui permet de lire plus rapidement les csv que la fonction standard.
Code source :effectif-premier.R

```
library(data.table)
nb_premier<-fread("nb_1er.csv", header=FALSE, colClasses = 'integer', col.names = 'Chiffre')
EffTable<-table(nb_premier$Chiffre)
print(EffTable)
```

Affichage :

```
  0       1       2       3       4       5       6       7       8       9
2325846 2324106 2323306 2325845 2326305 2325065 2324655 2324051 2326039 2324207
```

4 Avec SQLite3

On peut également mettre les chiffres dans une table de base de données, afin de bénéficier des optimisations liées à l'indexation.

1) On créé une base de données (nb_1er.db) pour y loger la table, avec bash cela donne :

```
sqlite3 nb_1er.db
```

2)Puis dans la ligne de commande sqlite3, on créé une table Nombre :

```
CREATE TABLE Nombre("chiffre" int);
```

3)On importe les données dans la table Nombre :
.*mode* csv
.*import* nb_1er.csv Nombre

4)On ajoute un indexe

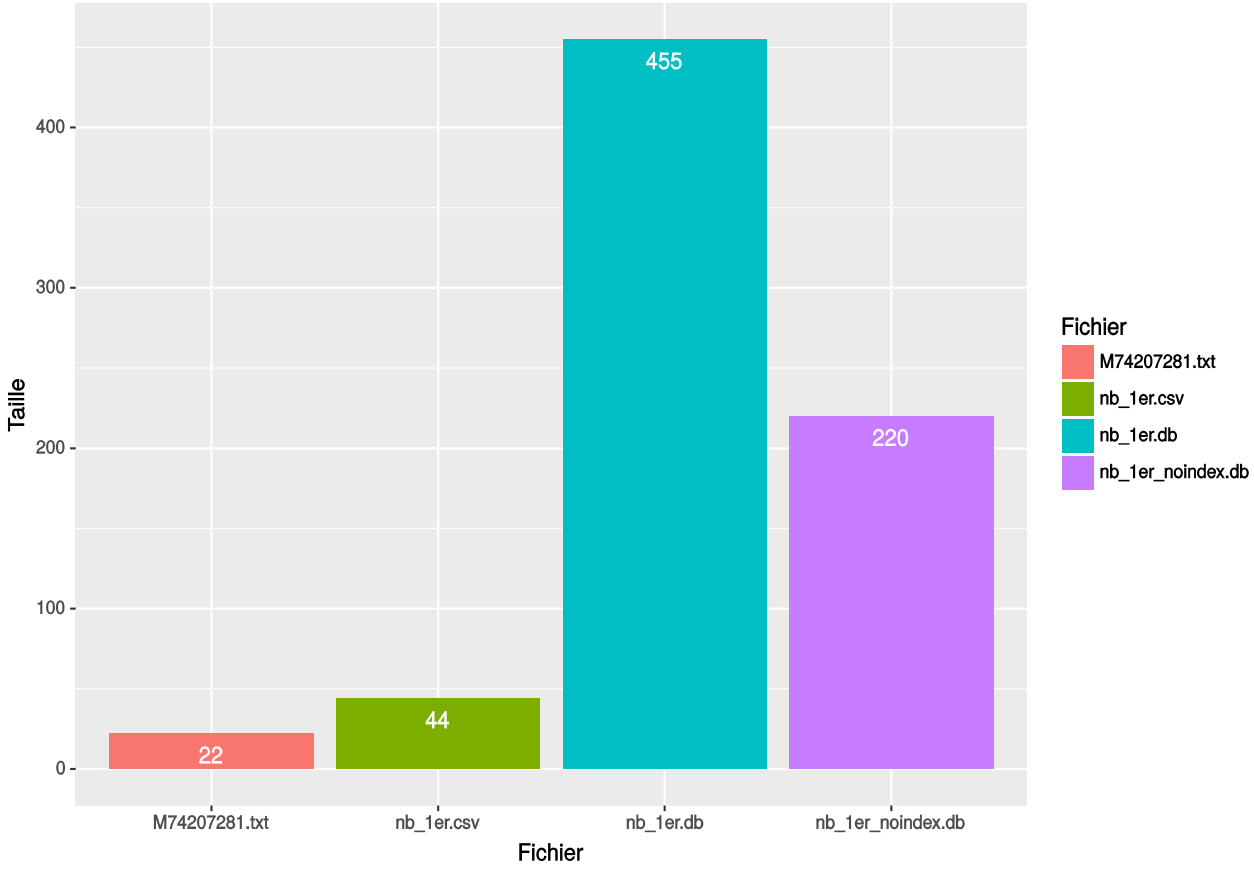
```
CREATE INDEX ix_Nombre_chiffre on Nombre(chiffre);
```

5)Pour avoir l'effectif de chaque nombre il suffit d'exécuter les requêtes : (effectif-premier.sql)

```
SELECT count(*) from (SELECT * from Nombre where Chiffre=1);
SELECT count(*) from (SELECT * from Nombre where Chiffre=2);
SELECT count(*) from (SELECT * from Nombre where Chiffre=3);
SELECT count(*) from (SELECT * from Nombre where Chiffre=4);
SELECT count(*) from (SELECT * from Nombre where Chiffre=5);
SELECT count(*) from (SELECT * from Nombre where Chiffre=6);
SELECT count(*) from (SELECT * from Nombre where Chiffre=7);
SELECT count(*) from (SELECT * from Nombre where Chiffre=8);
SELECT count(*) from (SELECT * from Nombre where Chiffre=9);
```

5 Comparaisons de la taille des données et effectifs

On peut remarquer que la différence de taille entre :
-le fichier texte initial : M77232917.txt : 22 Mo
-le fichier csv avec un chiffre par ligne : nb_1er.csv : 44Mo
-le fichier contenant la base de donnée sans indexation : 220Mo
-le fichier contenant la base de donnée indexée : 455Mo



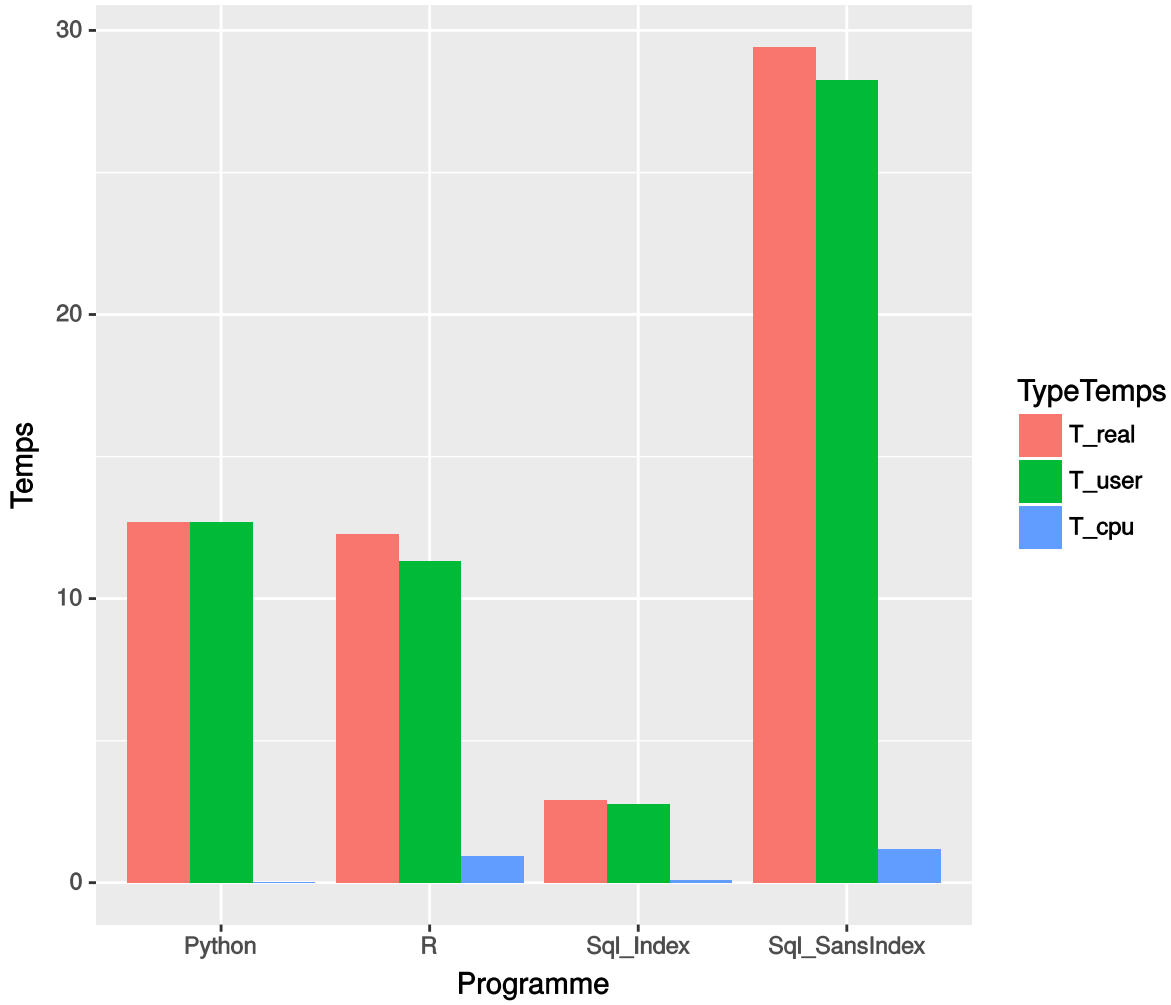
On peut s'attendre à ce qu'avec l'indexation, le temps pour compter l'effectif de chaque chiffre est inférieur au temps mis par le programme Python ou R.

6 Comparaison des temps d'exécutions

On utilise la fonction time de bash pour comparer les temps d'exécution :

```
time python3 effectif-premier.py
time Rscript effectif-premier.R
time sqlite3 nb_1er_noindex.db < effectif-premier.sql
```

Programme	T_real	T_user	T_cpu
Python	13.203	13.059	0.100
R	13.686	12.257	2.470
Sql_SansIndex	32.215	30.339	1.844
Sql_Index	3.174	2.973	0.188



R et Python sont comparables, et l'indexation permet de rendre le calcul des effectifs 4 fois plus rapide qu'avec R et Python.

7 Test d'uniformité de la répartition

On compare les effectifs des chiffres obtenus (que l'on note c_i) avec l'effectif théorique de chaque chiffre qui est 2 324 943. ($\frac{1}{10} \times 23249\,425 \approx 2\,324\,943$).

Chiffre	Effectif	Effectif Théorique
0	2 325 846	2 324 943
1	2 324 106	2 324 943
2	2 323 306	2 324 943
3	2 325 845	2 324 943
4	2 326 305	2 324 943
5	2 325 065	2 324 943
6	2 324 655	2 324 943
7	2 324 051	2 324 943
8	2 326 039	2 324 943
9	2 324 207	2 324 943

On calcule alors

$$K = \sum_{i=0}^{9} \frac{(c_i - 2324943)^2}{2324943}$$

On trouve $K = 4,0864653456$
Il y a $(10-1) \times (2-1) = 9$ degrés de libertés, et on trouve une p-value de $1 - 0,1 = 0,9$.
On accepte ainsi l'hypothèse d'indépendance, et les chiffres sont bien uniformément répartis.

Bien-entendu, R aurait pu faire le calcul à partir des effectifs. En reprenant le script R donné au paragraphe 3 (effectif-premier.R), il suffit d'écrire :
chisq.test(EffTable)

Pour obtenir le résultat.

Remarque :
Le recours à sqlite3 peut se faire directement ou via des bibliothèques de fonctions aussi bien en python qu'en R.

Création de la base de données avec Python (fichier : nb_1er_db.py)

```
import sqlite3

#Ouverture du fichier initial et récupération du nombre
f=open('M77232917.txt','r')
N=f.read()

#Creation de la base et table
db=sqlite3.connect('premierbdd.db')
cur=db.cursor()
cur.execute("""CREATE TABLE Nombre("chiffre" int)""")

for nbr in N:
     if nbr !='\n' :
          #nbr=int(nbr)
          cur.execute("INSERT INTO Nombre VALUES(?)",nbr)

f.close()
db.commit()
cur.close()
db.close()
```

Création avec R (fichier : nombre_premier_db.R)

```
#Lecture du nombre premier
library(data.table)
nb_premier<-fread("nb_1er.csv", header=FALSE, colClasses =
'integer', col.names = 'Chiffre')

#Creation de la base de données
library(sqldf)
db<-dbConnect(SQLite(), dbname="premierbdd.db")
dbWriteTable(db,"NbrPremier",nb_premier)
#dbGetQuery(db,"SELECT count(*) from (SELECT * from NbrPremier
where Chiffre=1)")
dbDisconnect(db)
```