Interroger Lexique avec R

Christophe Pallier

Contents

Ce document montre comment interroger et manipuler la base Lexique hors-ligne avec le logiciel R

Note: Il fait parti du cours Programmation pour les Sciences Cognitives. Merci de signaler d'éventuelles erreurs à christophe@pallier.org

Installation

1. S'ils ne sont pas déjà installés sur votre ordinateur, vous devrez installer les deux logiciels (en acceptant les options par défaut):
   - R
   - RStudio Desktop.
2. Démarrer le programme **Rstudio** puis, dans l'onglet **Console** situé dans la fenêtre en bas à gauche, copier la ligne suivante puis appuyez sur 'Entrée'

   ```
   install.packages(c('rjson', 'tidyverse'))
   ```

   Laissez RStudio se débrouiller (mais vous devrez peut-être sélectionner un serveur).

   Puis créez un nouveau projet (Menu `File/New Project/New Directory/New project`) que vous nommerez *lexique*.

   Vous êtes prêt ! Vous n'aurez plus jamais à refaire ces étapes.

Chargement de la table *lexique*

Tout se déroule dans le logiciel **RStudio**.

Dans le menu `File`, ouvrez le projet *lexique* créé dans la section précédente (Après un redémarrage de RStudio, vous pouvez utiliser le menu `File/Recent Projects` pour retrouver ce projet).

- Cliquez le menu `File/New File/R Notebook`: Un document "Untitled" apparait dans la fenêtre en haut à gauche. C'est dans ce document que nous allons entrer du code R.
- Commencez par supprimer tout ce que se trouve après la ligne 4 (ne conservez que les 4 premières lignes).
- Placez le curseur sur la ligne 5 (juste après la ligne contenant '—'), et cliquez sur le bouton *Insert* puis choisisez *R* (ou appuyer sur *Ctrl+Alt+I*).

  Copiez les lignes suivantes à l'intérieur du bloc de code qui vient d'être créé.

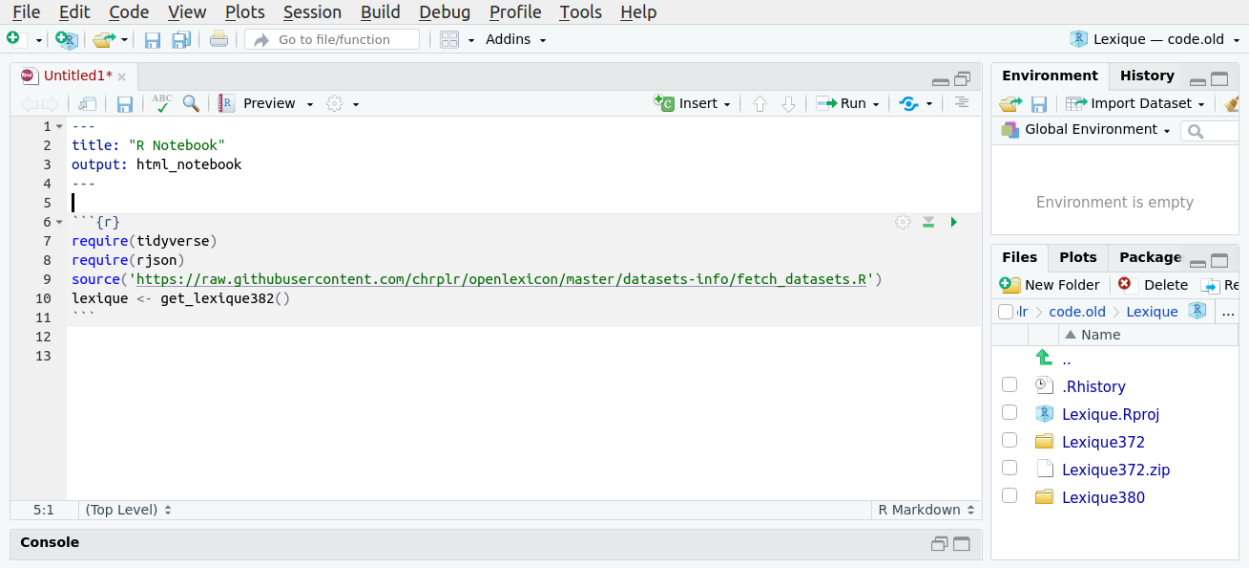


Figure 1: Téléchargment de Lexique

```
require(tidyverse)
require(rjson)
source('https://raw.githubusercontent.com/chrplr/openlexicon/master/datasets-info/fetch_datasets.R'
lexique <- get_lexique382()
```

La fenêtre `Rstudio` doit ressembler à la fiugure suivante :

En faisant bien attention que le curseur soit *à l'intérieur du bloc de code*, cliquez sur `Run/Run current chunk` (ou bien appuyez sur la petite *flèche verte*, ou bien encore sur les touches *Ctrl-Shift-Enter*).

La variable `lexique` doit maintenant apparaitre dans l'onglet **Environment** en haut à droite ; Vous pouvez cliquez dessus pour consulter la table.

Exploration de la distribution des fréquences

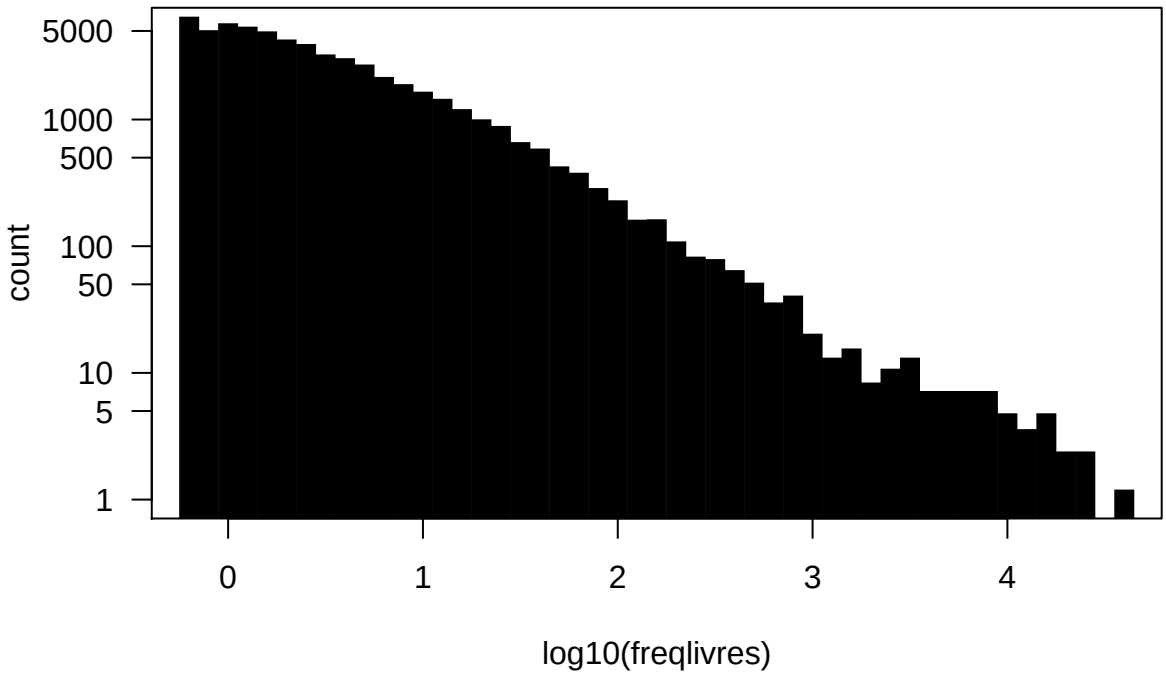
Supprimons les mots très rares, de fréquence d'occurrence inférieure à 0.5 par million dans le corpus des livres :

```
lexique1 <- subset(lexique, freqlivres > 0.5)
lexique1$logfreq <- log10(lexique1$freqlivres)
```

Puis calculons l'histogramme des fréquences, et affichons le sur un graphique avec des axes logarithmiques:

```
with(lexique1, {
  histdata <- hist(logfreq, plot=FALSE, nclass=50)
  plot(histdata$breaks[-1], histdata$count, log="y", type='h', lwd=10, lend=2, las=1, xlab='log10(freql
  }
)
```

```
## Warning in xy.coords(x, y, xlabel, ylabel, log): 1 y value <= 0 omitted
## from logarithmic plot
```



Note: On retrouve la loi de Zipf, c'est à dire une relation à peu près linéaire sur ce graphique log-log, qui reflète une distribution en loi de puissance.

Recherche de mots

Supposons que vous vouliez extraire les lignes de la table lexique correspondant, par exemple, aux mots *bateau, avion, maison, arbre*.

Créez un bloc de code (Ctrl-Alt-I) et copiez-y le code suivant :

```
items <- c('bateau', 'avion', 'maison', 'arbre')

selection <- subset(lexique, ortho %in% items)

head(selection)
```

```
## # A tibble: 4 x 35
##   ortho phon  lemme cgram genre nombre freqlemfilms2 freqlemlivres
##   <chr> <chr> <chr> <chr> <chr> <chr>          <dbl>         <dbl>
## 1 arbre aRbR  arbre NOM   m     s               81.7         209.
## 2 avion avj§  avion NOM   m     s              128.           78.0
## 3 bate~ bato  bate~ NOM   m     s              125.           82.4
## 4 mais~ mEz§  mais~ NOM   <NA>  s              606.          575.
## # ... with 27 more variables: freqfilms2 <dbl>, freqlivres <dbl>,
## #   infover <chr>, nbhomogr <int>, nbhomoph <int>, islem <int>,
## #   nblettres <int>, nbphons <int>, cvcv <chr>, p_cvcv <chr>,
## #   voisorth <int>, voisphon <int>, puorth <dbl>, puphon <dbl>,
## #   syll <chr>, nbsyll <int>, `cv-cv` <chr>, orthrenv <chr>,
## #   phonrenv <chr>, orthosyll <chr>, cgramortho <chr>, deflem <dbl>,
## #   defobs <dbl>, old20 <dbl>, pld20 <dbl>, morphoder <chr>, nbmorph <dbl>
```

Exécutez ce code (Ctrl-Shift-Enter). La variable `selection` doit contenir les lignes pertinentes. Vous pouvez inspecter son contenu en cliquant sur son nom dans l'onglet **Environment** situé dans la fenêtre en haut à

droite de RStudio.

Pour sauvegarder les résultats obtenus dans un fichier, exéctuez la commande suivante:

```
write_tsv(selection, 'selection.tsv')
```

Cela doit créer un fichier `selection.tsv` dans le répertoire du projet (*lexique*).

Notez que les fichiers ayant l'extension `tsv` (tab-separated-values) peuvent être ouverts avec Excel, ou OpenOffice Calc, ou même avec n'importe quel éditeur de texte. Note: le package `readr` de R fournit aussi des fonctions `write_excel_csv` et `write_excel_csv2` qui peuvent intéresser certains.

Si vous avez une liste de mots plus longue, il serait fastidieux d'écrire la ligne `items <- ...`. Plus simplement vous pouvez utiliser:

```
items = scan(what='characters')
```

Et coller la liste de mots. Pour finir la liste, entrez une ligne vide (en appuyant deux fois successivement sur la touche *Entrée*), puis, à nouveau:

```
selection <- subset(lexique, ortho %in% items)
```

Comme la fonction `scan` permet aussi de lire la liste dans un fichier externe, si vous avez une liste de mots dans fichier `liste.txt`, vous pouvez extraire les lignes de lexique qui contiennent spécifiquement ces mots avec le code suivant:

```
liste <- scan('liste.txt', what='characters')
selection <- subset(lexique, ortho %in% liste)
```

Sélection par critères

Supposons que vous vouliez sélectionner tous les noms de 5 lettres, singuliers, de fréquence lexicale (films) comprise entre 10 et 100. Voici la ligne magique:

```
selection = subset(lexique, cgram=='NOM' & nombre != 'p' & nblettres==5 & freqlivres > 10 & freqlivres <
head(selection)
```

```
## # A tibble: 6 x 35
##   ortho phon  lemme cgram genre nombre freqlemfilms2 freqlemlivres
##   <chr> <chr> <chr> <chr> <chr> <chr>          <dbl>         <dbl>
## 1 abîme abim  abîme NOM   m     s               6.01          20.6
## 2 achat aSa   achat NOM   m     s               9.75          17.0
## 3 acier asje  acier NOM   m     s              13.9           34.5
## 4 adieu adj2  adieu NOM   m     s              44.4           38.0
## 5 affût afy   affût NOM   m     s               1.42          11.4
## 6 agent aZ@   agent NOM   m     s             118.            39.3
## # ... with 27 more variables: freqfilms2 <dbl>, freqlivres <dbl>,
## #   infover <chr>, nbhomogr <int>, nbhomoph <int>, islem <int>,
## #   nblettres <int>, nbphons <int>, cvcv <chr>, p_cvcv <chr>,
## #   voisorth <int>, voisphon <int>, puorth <dbl>, puphon <dbl>,
## #   syll <chr>, nbsyll <int>, `cv-cv` <chr>, orthrenv <chr>,
## #   phonrenv <chr>, orthosyll <chr>, cgramortho <chr>, deflem <dbl>,
## #   defobs <dbl>, old20 <dbl>, pld20 <dbl>, morphoder <chr>, nbmorph <dbl>
```

Le symbole `&` signifie 'et'. L'expression `nombre != p` signitif que la valeur dans la colonne *nombre* ne doit pas être `p`.

Sélection par “pattern”

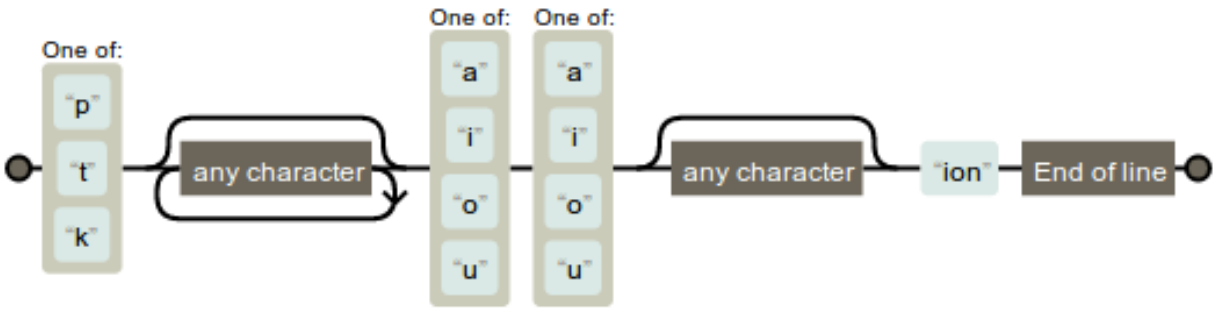
Les expressions régulières

Les expressions réulières, ou **regex**, sont des “patterns” qui permettent de rechercher des mots ayant certaines propriétés. Par exemple n’importe `a.b` désigne un mot contenant un `a` et un `b` séparés par une lettre quelconque. Voici d’autre exemples:

- `^maison$` : recherche le mot “maison” exactement
- `^anti` : recherche tous les mots commençant par “anti”
- `^(jour|nuit|matin|soir)$` : “jour” ou “nuit” ou “matin” ou “soir” (permet de rechercher une liste de mots)
- `ion` : recherche les mots qui contiennent la chaine “ion” dans n’importe quelle position
- `ion$` : mots se terminant par “ion”
- `^pr` : mots commençant par “pr”
- `^p..r$` : mots de quatre lettres commençant par “p”, finisant pas “r”
- `^p.*r$` : mots commencant par ‘”p’ et finissant par “r”
- `[aeiou][aeiou]` : mots contenant 2 voyelles successives
- `^[aeiou]` : mots commençant par une voyelle
- `^[^aeriou]` : mots ne commençant pas par une voyelle

Il existe de nombreux tutoriaux sur les regex sur le web, notamment http://regextutorials.com/intro.html

Une expression régulière décrit un automate de transitions à états finis. Le site https://regexper.com/ vous permet de visualiser l’automate associé à une regex. Par exemple `[ptk].*[aiou][aiou].?ion$` correspond à l’automate fini:



Recherches dans R avec grepl

R permet d’effectuer des recherches par pattern grâce à la fonction `grepl`. La syntaxe est `grepl(`*regex*, *variable*`)` pour rechercher les lignes où la variable “matche” la regex (Voir la doc R de grepl.

Cette fonction permet de localiser les lignes qui ‘matchent’ une expression, ou bien, en la niant avec le signe `!`, de supprimer des lignes qui matchent un pattern.

Voici quelques exemples:

- Pour obtenir tous les mots qui finissent par `tion` :

```
lexique %>% filter(grepl("tion$", ortho)) -> selection2
head(selection2)
```

```
## # A tibble: 6 x 35
##   ortho phon  lemme cgram genre nombre freqlemfilms2 freqlemlivres
##   <chr> <chr> <chr> <chr> <chr> <chr>          <dbl>         <dbl>
```

```
## 1 abdi~ abdi~ abdi~ NOM   f     s               0.05          1.96
## 2 abdu~ abdy~ abdu~ NOM   f     s               0.05          0
## 3 aber~ abER~ aber~ NOM   f     s               1.16          4.46
## 4 abje~ abZE~ abje~ NOM   f     s               0.51          2.3
## 5 abju~ abZy~ abju~ NOM   f     s               0             0.47
## 6 abla~ abla~ abla~ NOM   f     s               0.45          1.35
## # ... with 27 more variables: freqfilms2 <dbl>, freqlivres <dbl>,
## #   infover <chr>, nbhomogr <int>, nbhomoph <int>, islem <int>,
## #   nblettres <int>, nbphons <int>, cvcv <chr>, p_cvcv <chr>,
## #   voisorth <int>, voisphon <int>, puorth <dbl>, puphon <dbl>,
## #   syll <chr>, nbsyll <int>, `cv-cv` <chr>, orthrenv <chr>,
## #   phonrenv <chr>, orthosyll <chr>, cgramortho <chr>, deflem <dbl>,
## #   defobs <dbl>, old20 <dbl>, pld20 <dbl>, morphoder <chr>, nbmorph <dbl>
```

Encore une fois, vous pouvez sauvegarder ces résultats avec:

```
write_tsv(selection2, 'mots-en-tion.tsv')
```

- Pour lister tous les mots contenant un cluster de consonnes plosives, mais pas debut de mot:

```
lexique %>% filter(grepl('.[ptkbdg][ptkbdg]', phon)) -> selection3
head(selection3)
```

```
## # A tibble: 6 x 35
##   ortho phon  lemme cgram genre nombre freqlemfilms2 freqlemlivres
##   <chr> <chr> <chr> <chr> <chr> <chr>          <dbl>         <dbl>
## 1 abdi~ abdi~ abdi~ NOM   f     s               0.05          1.96
## 2 abdi~ abdi~ abdi~ NOM   f     p               0.05          1.96
## 3 abdi~ abdi~ abdi~ VER   <NA>  <NA>            0.47          2.77
## 4 abdi~ abdi~ abdi~ VER   <NA>  <NA>            0.47          2.77
## 5 abdi~ abdi~ abdi~ VER   <NA>  <NA>            0.47          2.77
## 6 abdi~ abdi~ abdi~ VER   <NA>  <NA>            0.47          2.77
## # ... with 27 more variables: freqfilms2 <dbl>, freqlivres <dbl>,
## #   infover <chr>, nbhomogr <int>, nbhomoph <int>, islem <int>,
## #   nblettres <int>, nbphons <int>, cvcv <chr>, p_cvcv <chr>,
## #   voisorth <int>, voisphon <int>, puorth <dbl>, puphon <dbl>,
## #   syll <chr>, nbsyll <int>, `cv-cv` <chr>, orthrenv <chr>,
## #   phonrenv <chr>, orthosyll <chr>, cgramortho <chr>, deflem <dbl>,
## #   defobs <dbl>, old20 <dbl>, pld20 <dbl>, morphoder <chr>, nbmorph <dbl>
```

- L'opérateur `filter`peut être appeler plusieurs fois pour affiner progressivement la recherche.

  Par exemple, pour obtenir tous les mots de 8 lettres qui ne finissent pas `ent`:

```
lexique %>% filter(nblettres == 8) %>% filter(!grepl("ent$", ortho)) -> selection4
head(selection4)
```

```
## # A tibble: 6 x 35
##   ortho phon  lemme cgram genre nombre freqlemfilms2 freqlemlivres
##   <chr> <chr> <chr> <chr> <chr> <chr>          <dbl>         <dbl>
## 1 a gi~ adZj~ a gi~ ADV   <NA>  <NA>            0             0.27
## 2 a pr~ apRi~ a pr~ ADV   <NA>  <NA>            1.04          3.85
## 3 a pr~ apRi~ a pr~ NOM   m     <NA>            0.41          0.47
## 4 abai~ abEsE abai~ VER   <NA>  <NA>            4.93         18.0
## 5 abai~ abese abai~ VER   <NA>  <NA>            4.93         18.0
## 6 abai~ abEs  abai~ VER   <NA>  <NA>            4.93         18.0
## # ... with 27 more variables: freqfilms2 <dbl>, freqlivres <dbl>,
## #   infover <chr>, nbhomogr <int>, nbhomoph <int>, islem <int>,
```

```
## #   nblettres <int>, nbphons <int>, cvcv <chr>, p_cvcv <chr>,
## #   voisorth <int>, voisphon <int>, puorth <dbl>, puphon <dbl>,
## #   syll <chr>, nbsyll <int>, `cv-cv` <chr>, orthrenv <chr>,
## #   phonrenv <chr>, orthosyll <chr>, cgramortho <chr>, deflem <dbl>,
## #   defobs <dbl>, old20 <dbl>, pld20 <dbl>, morphoder <chr>, nbmorph <dbl>
```

Chargement d'autres tables d'OpenLexicon

Le projet OpenLexicon comprend de nombreuses tables lexicales (voir https://chrplr.github.io/openlexicon/datasets-info/).

Pour accéder à ces tables, vous pouvez utiliser la fonction `fetch_datatset` définie dans https://raw.githubusercontent.com/chrplr/openlexicon/master/datasets-info/fetch_datasets.R.

Par exemple, pour charger la table SUBTLEX-US:

```r
source('https://raw.githubusercontent.com/chrplr/openlexicon/master/datasets-info/fetch_datasets.R')
subtlexus <- readRDS(fetch_dataset('SUBTLEX-US', format='rds')$datatables[[1]])
```

```
## Warning in fetch_dataset("SUBTLEX-US", format = "rds"): You already have
## the file /home/cp983411/openlexicon_datasets/SUBTLEXus.rds which seems up
## to date.
```