broom & purrr & furrr

Bertrand Servin & Guilhem Huau

31 Mars 2022 - Rencontre R Toulouse

broom

Introduction

- `broom` est un package qui permet de faciliter l'extraction de résultats issus de l'ajustement de modèles statistiques
- Le package propose 3 fonctions principales
  1. **`tidy`** : "nettoie" les sorties correspondant aux estimations d'effets pour les rendre disponibles sous la forme de tableaux
  2. **`glance`** : "nettoie" les sorties correspondant au résumé de l'ajustement du modèle (R^2, variance résiduelle …)
  3. **`augment`** : ajoute des colonnes aux données modélisées (prédictions, résidus, clustering …)

Aujourd'hui : illuster l'utilisation du package sur un cas pratique

Données d'exemple

Comptage de crossovers dans des intervalles génomiques

```
crossovers = read_table('data/cocounts.txt',col_types=c('ffffiiid'))
print(crossovers %>% arrange(wstart))
```

```
## # A tibble: 108,240 × 8
##    population parent         sex   chrom wstart   wstop   nco coverage
##    <fct>      <fct>          <fct> <fct>  <int>   <int> <int>    <dbl>
##  1 Lacaune    16163764382    M     26         0 1000000     0 17120367
##  2 Lacaune    55113607937    M     26         0 1000000     0 20170025
##  3 Lacaune    55143306662    M     26         0 1000000     1 61066052
##  4 Lacaune    16167600340    M     26         0 1000000     1  6083324
##  5 Lacaune    12000382030512 M     26         0 1000000     0 10003406
##  6 Lacaune    12000327050504 M     26         0 1000000     0  8292307
##  7 Lacaune    16162660168    M     26         0 1000000     0 13161204
##  8 Lacaune    16158660525    M     26         0 1000000     0        0
##  9 Lacaune    12000367040505 M     26         0 1000000     0  5697776
## 10 Lacaune    55017706911    M     26         0 1000000     0  2249906
## # … with 108,230 more rows
```

Modéliser le nombre de crossovers pour chaque intervalle (chrom, wstart, wstop) par un effet population + sex ?

Modèle de régression de Poisson

$$Y_i \sim Poisson(\exp(\mathbf{X_i}\beta_\mathbf{i})\, o_i)$$

$$E(log(Y_i)) = \mathbf{X_i}\beta_\mathbf{i} + \log(o_i)$$

```
glm( nco ~ sex + chrom + sex:chrom + offset(log(coverage)), family=poisson)
```

Objectifs:

1. Ajuster le modèle pour chaque intervalle du jeux de données (chrom,wstart,wstop)
2. Récupérer les effets estimés
3. Prédire une valeur standardisée de E(log(Y)) pour un coverage de 1 (= taux de recombinaison) et son erreur standard
4. sous une forme propre :)

summary (old school)

```
myint = crossovers %>% filter(wstart==5000000, coverage>0)
myint.model = glm(nco ~sex+population+sex:population+offset(log(coverage)), data=myint,family='poisson')
summary(myint.model)
```

```
##
## Call:
## glm(formula = nco ~ sex + population + sex:population + offset(log(coverage)),
##     family = "poisson", data = myint)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3615  -0.3596  -0.2397  -0.1684   3.4857
##
## Coefficients:
##                      Estimate Std. Error  z value Pr(>|z|)
## (Intercept)         -17.69535    0.07433 -238.067   <2e-16 ***
## sexF                 -0.90138    0.50546   -1.783   0.0745 .
## populationSoay       -0.02706    0.13312   -0.203   0.8389
## sexF:populationSoay  -0.12757    0.54767   -0.233   0.8158
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 833.30  on 2401  degrees of freedom
## Residual deviance: 790.82  on 2398  degrees of freedom
## AIC: 1264.1
##
## Number of Fisher Scoring iterations: 6
```

tidy & glance

```
library(broom)
tidy(myint.model)
```

```
## # A tibble: 4 × 5
##   term                estimate std.error statistic p.value
##   <chr>                  <dbl>     <dbl>     <dbl>   <dbl>
## 1 (Intercept)         -17.7       0.0743  -238.     0
## 2 sexF                 -0.901     0.505     -1.78   0.0745
## 3 populationSoay       -0.0271    0.133     -0.203  0.839
## 4 sexF:populationSoay  -0.128     0.548     -0.233  0.816
```

`term` est désormais une colonne plutôt qu'un nom de ligne

```
glance(myint.model)
```

```
## # A tibble: 1 × 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          833.    2401  -628. 1264. 1287.     791.        2398  2402
```

augment

La fonction `augment` permet de fabriquer un `tibble` des valeurs prédites par le modèle.

```
augment(myint.model, type.predict = 'response')
```

```
## # A tibble: 2,402 × 10
##      nco sex   population `offset(log(coverag… .fitted .resid .std.resid    .hat
##    <int> <fct> <fct>                     <dbl>   <dbl>  <dbl>      <dbl>   <dbl>
##  1     1 M     Lacaune                    16.9   0.454  0.697      0.698 2.51e-3
##  2     1 M     Lacaune                    17.2   0.640  0.415      0.416 3.54e-3
##  3     3 M     Lacaune                    18.2   1.65   0.940      0.944 9.13e-3
##  4     0 M     Lacaune                    15.9   0.165 -0.575     -0.575 9.13e-4
##  5     0 M     Lacaune                    16.5   0.310 -0.787     -0.788 1.71e-3
##  6     0 M     Lacaune                    16.3   0.248 -0.704     -0.705 1.37e-3
##  7     2 M     Lacaune                    17.1   0.549  1.51       1.51  3.03e-3
##  8     1 M     Lacaune                    17.5   0.831  0.180      0.180 4.59e-3
##  9     0 M     Lacaune                    17.0   0.475 -0.975     -0.976 2.62e-3
## 10     0 M     Lacaune                    15.4   0.103 -0.454     -0.455 5.71e-4
## # … with 2,392 more rows, and 2 more variables: .sigma <dbl>, .cooksd <dbl>
```

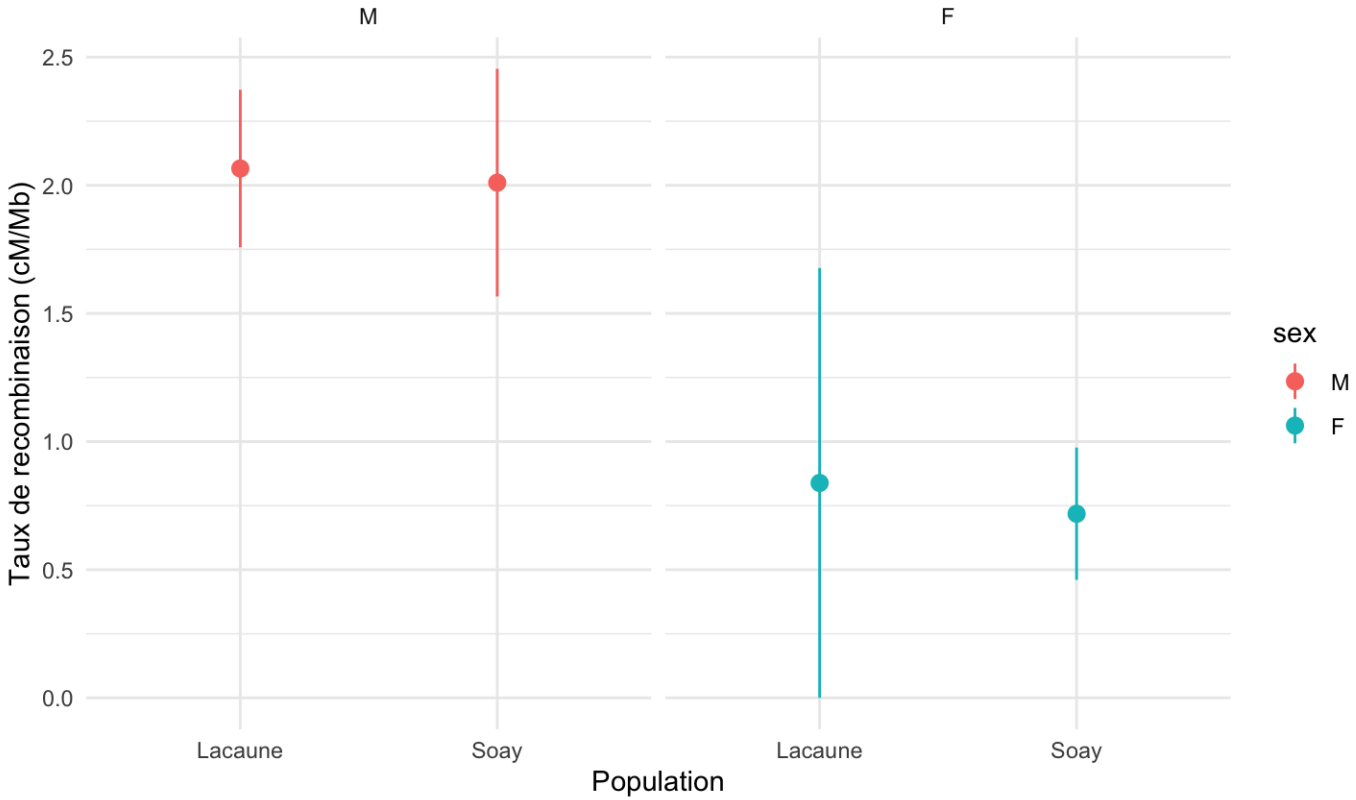
Dans ce cas, la valeur prédite prend en compte l'informativité (coverage). Pour calculer des valeurs standardisées (à informativité égale), on peut passer un nouveau jeu de données

augment avec nouvelles données

```
myint.new = myint %>% select(population,sex) %>% distinct %>% mutate(coverage=1e8)
myint.pred = augment(myint.model, newdata=myint.new,
                     type.predict='response', se_fit=TRUE)
myint.pred
```

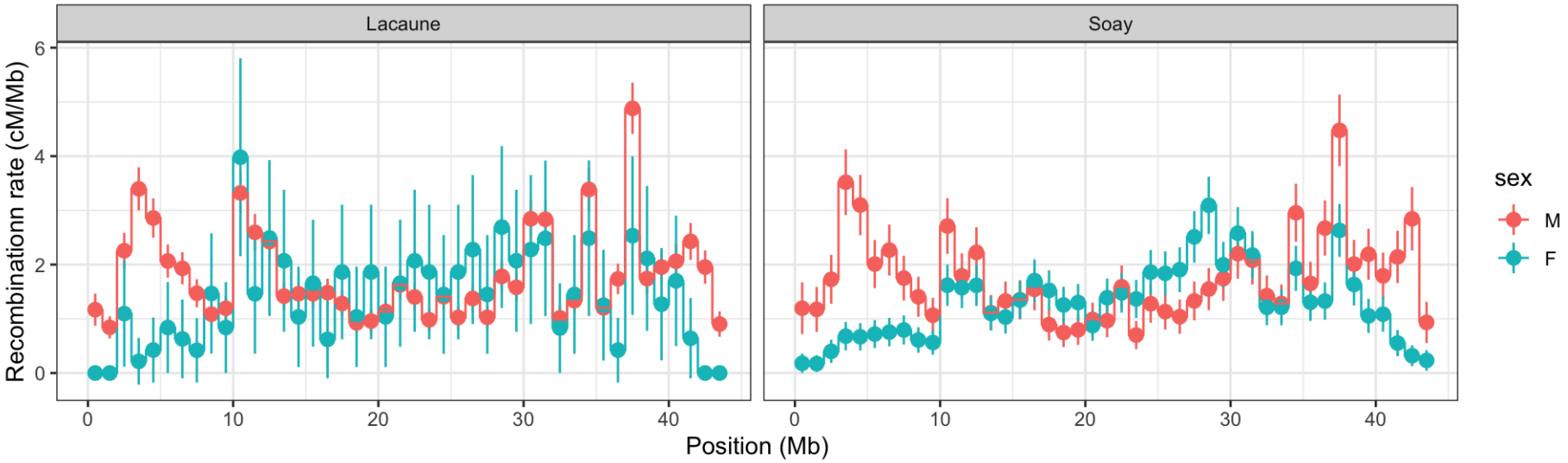
```
## # A tibble: 4 × 5
##   population sex    coverage .fitted .se.fit
##   <fct>      <fct>     <dbl>   <dbl>   <dbl>
## 1 Lacaune    M     100000000   2.07    0.154
## 2 Lacaune    F     100000000   0.839   0.419
## 3 Soay       F     100000000   0.718   0.129
## 4 Soay       M     100000000   2.01    0.222
```

augment avec nouvelles données



Application à un grand nombre d'intervalles

- `broom` devient particulièrement intéressant quand on ajuste des modèles à des sous-tableaux
- Dans ce cas là on le combine avec les packages `tidyr` (fonctions `nest/unnest`) et `purrr` (fonction `map`).



purrr

Introduction

- `purrr` est un package du tidyverse fait pour travailler avec des listes et des vecteurs dans une logique de **programation fonctionnelle**. Il propose différentes variantes de la fonction `map` qui permet d'appliquer une fonction à chaque élément d'une liste, ainsi que des fonctions supplémentaires permettant de travailler avec des listes.
- Un même schéma dans la syntaxe des fonctions: `function(.x, .f, ...)`
  - .x est une liste
  - .f est la fonction à appliquer
  - … sont les autres arguments à passer à la fonction

Comparaison avec `apply`

`apply` family

- Rapidité
- Base R

`map` family

- Simplicité
- Typage de la sortie
- Consistence
- *Tidyverse*

Comparaison avec **apply**

Rapidité d'exécution

```
my_list <- c(11:20)
mbm <- microbenchmark(
    lapply = lapply(my_list, function(x) x^2),
    map = map(my_list, function(x) x^2),
    times        = 10000
)
```

```
## # A tibble: 2 × 2
##   expr   `mean(time)`
##   <fct>         <dbl>
## 1 lapply        3618.
## 2 map           5857.
```

```
## Warning in microbenchmark(lapply = lapply(my_list, function(x) x^2), map
## = map(my_list, : less accurate nanosecond times to avoid potential integer
## overflows
```

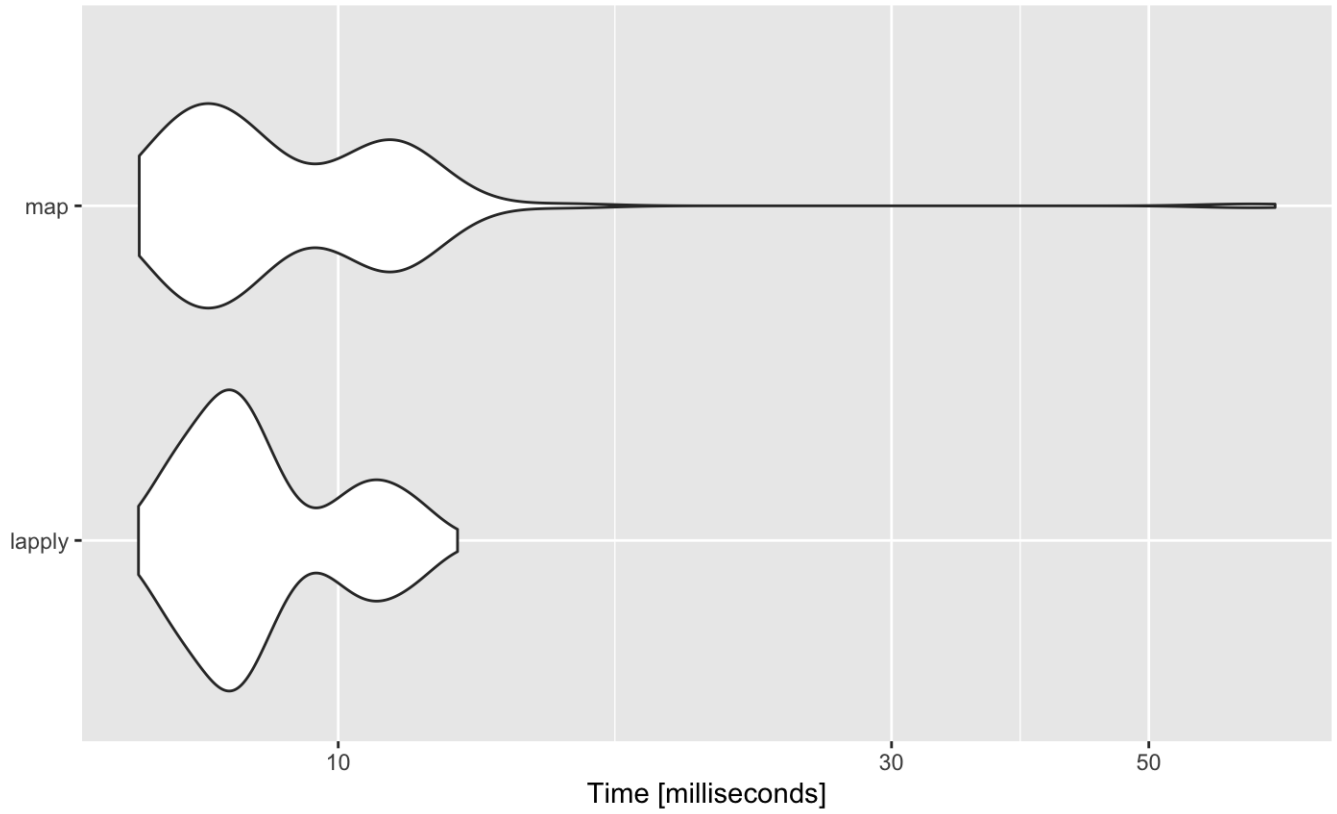
Comparaison avec **apply**

Rapidité d'exécution

```
myint = crossovers %>% filter(wstart==5000000, coverage>0)
crossovers_list <- myint %>% group_split(population)
mbm <- microbenchmark(
    lapply = lapply(crossovers_list, function(x) {
        myint.model = glm(nco ~sex+offset(log(coverage)), data=x,family='poisson')
        tidy(myint.model)}),
    map = map(crossovers_list, function(x) {
        myint.model = glm(nco ~sex+offset(log(coverage)), data=x,family='poisson')
        tidy(myint.model)}),
    times        = 200
)
```

Comparaison avec **apply**

Rapidité d'exécution




```
## Coordinate system already present. Adding new coordinate system, which will replace the exi
```

Travailler avec plusieurs listes

map2 et pmap

```
list1 <- c(11:20)
list2 <- c(1:10)

res <- map2(list1, list2, function(x,y) x*y)


list3 <- list1 + list2
list_full <- list(list1, list2, list3)

res <- pmap(list_full, function(a, b, c) a + b * c)
res <- pmap(list_full, function(...) ..1 + ..2 * ..3)
res <- pmap(list_full, ~ ..1 + ..2 * ..3)
```

Autres fonctions

Appliquer une fonction avec uniquement des effets de bord: `walk`

```
walk(month.name[1:4], print)
```

```
## [1] "January"
## [1] "February"
## [1] "March"
## [1] "April"
```

Appliquer une fonction de façon recursive: `reduce`

```
list1 <- c(11:20)
reduce(list1, sum)
```

```
## [1] 155
```

Autres fonctions

Filtres

- `keep()`: conserve les élements qui passent un test logique
- `discard()`: supprime les élements qui passent un test logique
- `some()`: Renvoie `TRUE` si certains (>1) élements passent un test

Application avec conditions

S'applique avec `map` et `modify`

- `..._if`: Applique la fonction uniquement si un test logique est passé
- `..._at`: Applique la fonction uniquement pour les élements sélectionnés
- `..._depth`: Applique la fonction à un certain niveau de liste

Autres fonctions

Gérer les erreurs avec `safely`

```
x <- list(1, "e", 3)
# Base R
lapply(x, sqrt)
```

```
# purrr package
safe_sqrt <- safely(sqrt)
safe_result_list <- map(x, safe_sqrt) %>% transpose
safe_result_list$result
```

```
## [[1]]
## [1] 1
##
## [[2]]
## NULL
##
## [[3]]
## [1] 1.732051
```

Aller plus vite avec `furrr`

- `furrr` est un package qui va faire le lien entre `purrr` et `future`. Ce dernier sert à effectuer des opérations de façon asynchrone (en parallèle).

- Possibilité d'utiliser un serveur distant ou juste du multicoeur sur sa machine
- Syntaxe très simple

Aller plus vite avec **furrr**

```
library(furrr)
```

```
## Loading required package: future
```

```
#Configurer future
plan(multisession, workers = availableCores())

#Utilise furrr
res <- future_map(crossovers_list, function(x) {
       myint.model = glm(nco ~sex+offset(log(coverage)), data=x,family='poisson')
       broom::tidy(myint.model)})
```

broom + p(f)urrr

dplyr nest

avant

```
## # A tibble: 108,240 × 9
##    population parent         sex   chrom wstart   wstop   nco coverage interval
##    <fct>      <fct>          <fct> <fct>  <int>   <int> <int>    <dbl> <chr>
##  1 Lacaune    16163764382    M     26         0 1000000     0 17120367 26:0-100…
##  2 Lacaune    55113607937    M     26         0 1000000     0 20170025 26:0-100…
##  3 Lacaune    55143306662    M     26         0 1000000     1 61066052 26:0-100…
##  4 Lacaune    16167600340    M     26         0 1000000     1  6083324 26:0-100…
##  5 Lacaune    12000382030512 M     26         0 1000000     0 10003406 26:0-100…
##  6 Lacaune    12000327050504 M     26         0 1000000     0  8292307 26:0-100…
##  7 Lacaune    16162660168    M     26         0 1000000     0 13161204 26:0-100…
##  8 Lacaune    16158660525    M     26         0 1000000     0        0 26:0-100…
##  9 Lacaune    12000367040505 M     26         0 1000000     0  5697776 26:0-100…
## 10 Lacaune    55017706911    M     26         0 1000000     0  2249906 26:0-100…
## # … with 108,230 more rows
```

dplyr nest

après

```
co.nested = crossovers %>% filter(coverage>0) %>
print(co.nested)
```

```
## # A tibble: 44 × 2
## # Groups:   interval [44]
##    interval            data
##    <chr>               <list>
##  1 26:0-1000000        <tibble [1,878 × 8]>
##  2 26:1000000-2000000  <tibble [1,945 × 8]>
##  3 26:2000000-3000000  <tibble [2,207 × 8]>
##  4 26:3000000-4000000  <tibble [2,336 × 8]>
##  5 26:4000000-5000000  <tibble [2,363 × 8]>
##  6 26:5000000-6000000  <tibble [2,402 × 8]>
##  7 26:6000000-7000000  <tibble [2,423 × 8]>
##  8 26:7000000-8000000  <tibble [2,432 × 8]>
##  9 26:8000000-9000000  <tibble [2,440 × 8]>
## 10 26:9000000-10000000 <tibble [2,443 × 8]>
## # … with 34 more rows
```

```
co.nested$data[[1]] %>% select(-chrom,-wstart,-wstop)
```

```
## # A tibble: 1,878 × 5
##    population parent         sex     nco coverage
##    <fct>      <fct>          <fct> <int>    <dbl>
##  1 Lacaune    16163764382    M         0 17120367
##  2 Lacaune    55113607937    M         0 20170025
##  3 Lacaune    55143306662    M         1 61066052
##  4 Lacaune    16167600340    M         1  6083324
##  5 Lacaune    12000382030512 M         0 10003406
##  6 Lacaune    12000327050504 M         0  8292307
##  7 Lacaune    16162660168    M         0 13161204
##  8 Lacaune    12000367040505 M         0  5697776
##  9 Lacaune    55017706911    M         0  2249906
## 10 Lacaune    12000317050511 M         2 75162719
## # … with 1,868 more rows
```

Utilisation avec broom

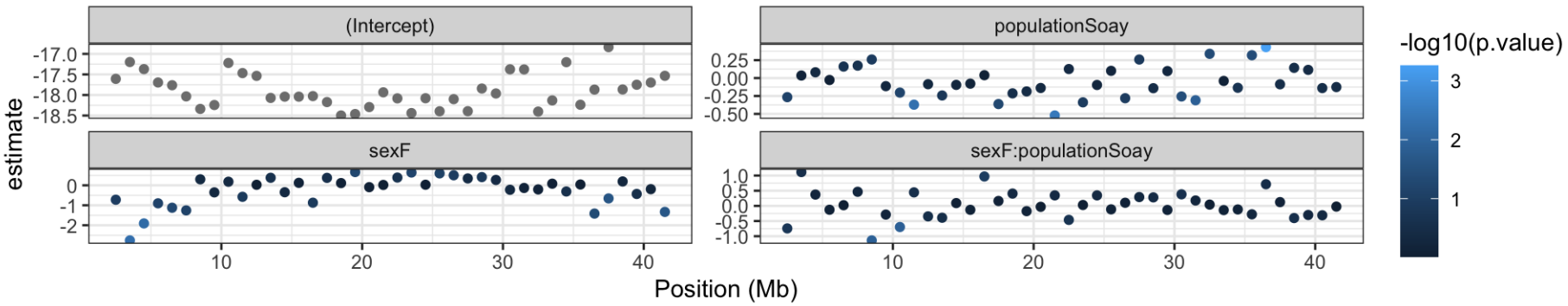
```
poisson_regression = function(df) {
  glm(nco ~ sex*population + offset(log(coverage)), data=df, family=poisson)
}

glm.analysis = crossovers %>% filter(coverage>0) %>% group_by(interval) %>% nest() %>%
  mutate( fit = future_map(data, poisson_regression),
          estimates = map(fit, tidy), ## broom
          glanced = map(fit, glance) ## broom
          )
glm.analysis
```

```
## # A tibble: 44 × 5
## # Groups:   interval [44]
##    interval            data                fit    estimates        glanced
##    <chr>               <list>              <list> <list>           <list>
##  1 26:0-1000000        <tibble [1,878 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  2 26:1000000-2000000  <tibble [1,945 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  3 26:2000000-3000000  <tibble [2,207 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  4 26:3000000-4000000  <tibble [2,336 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  5 26:4000000-5000000  <tibble [2,363 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  6 26:5000000-6000000  <tibble [2,402 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  7 26:6000000-7000000  <tibble [2,423 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  8 26:7000000-8000000  <tibble [2,432 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
##  9 26:8000000-9000000  <tibble [2,440 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
## 10 26:9000000-10000000 <tibble [2,443 × 8]> <glm>  <tibble [4 × 5]> <tibble [1 …
## # … with 34 more rows
```

Estimations

```
glm.analysis %>% select(interval, estimates) %>% unnest(estimates) %>% print(n=6)
```

```
## # A tibble: 176 × 6
## # Groups:   interval [44]
##   interval           term                estimate std.error statistic p.value
##   <chr>              <chr>                  <dbl>     <dbl>     <dbl>   <dbl>
## 1 26:0-1000000       (Intercept)          -18.3        0.127 -144.       0
## 2 26:0-1000000       sexF                 -15.9      885.      -0.0179   0.986
## 3 26:0-1000000       populationSoay         0.0229     0.237    0.0966   0.923
## 4 26:0-1000000       sexF:populationSoay   14.0      885.       0.0158   0.987
## 5 26:1000000-2000000 (Intercept)          -18.6        0.121 -153.       0
## 6 26:1000000-2000000 sexF                 -15.9      895.      -0.0178   0.986
## # … with 170 more rows
```

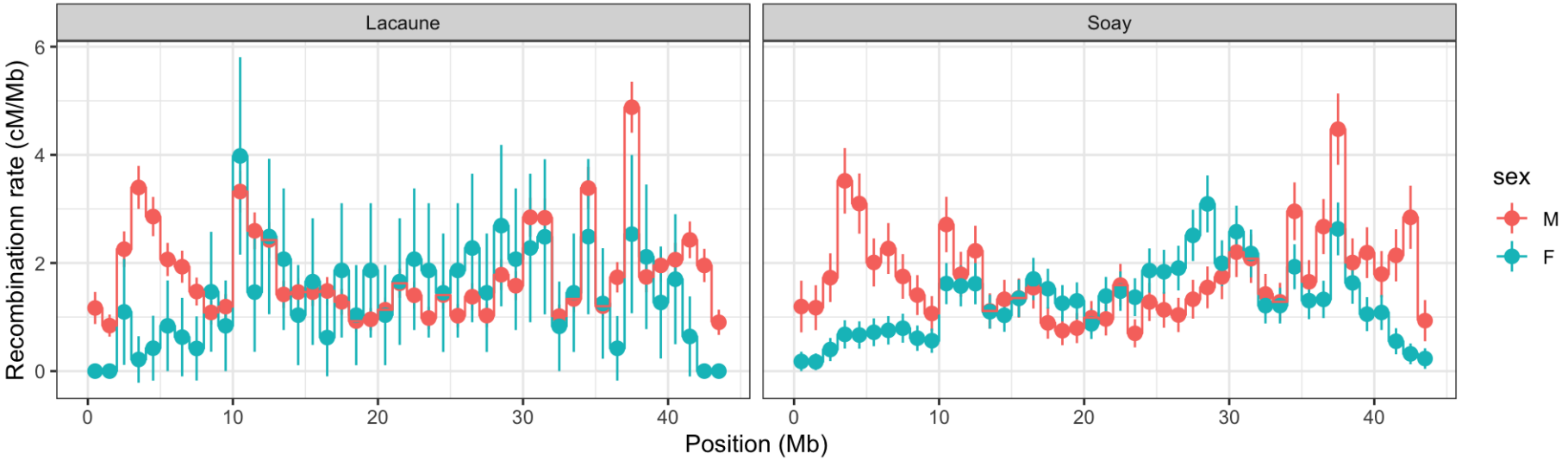


Prédictions

```
glm.predictions = glm.analysis  %>% unnest(data) %>% group_by(interval,sex,population) %>%
    summarize(n=n()) %>%  mutate(coverage=1e8) %>% ungroup() %>% group_by(interval) %>%
    nest() %>% rename(newdata=data)

pred = glm.predictions %>% left_join(glm.analysis %>% select(interval,fit),by="interval") %>%
  mutate(predictions = future_map2(fit, newdata, ~augment(.x, newdata=.y,
                                                          se_fit=TRUE,type.predict="response"))) %>%
  unnest(predictions)
print(pred, n=8)
```

```
## # A tibble: 176 × 9
## # Groups:   interval [44]
##   interval    newdata    fit    sex   population     n coverage  .fitted .se.fit
##   <chr>       <list>     <list> <fct> <fct>      <int>    <dbl>    <dbl>   <dbl>
## 1 26:0-10000… <tibble […> <glm>  M     Lacaune      520      1e8  1.17e+0 1.48e-1
## 2 26:0-10000… <tibble […> <glm>  M     Soay         424      1e8  1.20e+0 2.39e-1
## 3 26:0-10000… <tibble […> <glm>  F     Lacaune      262      1e8  1.48e-7 1.31e-4
## 4 26:0-10000… <tibble […> <glm>  F     Soay         672      1e8  1.80e-1 8.98e-2
## 5 26:1000000… <tibble […> <glm>  M     Lacaune      523      1e8  8.42e-1 1.02e-1
## 6 26:1000000… <tibble […> <glm>  M     Soay         443      1e8  1.17e+0 2.07e-1
## 7 26:1000000… <tibble […> <glm>  F     Lacaune      268      1e8  1.02e-7 9.10e-5
## 8 26:1000000… <tibble […> <glm>  F     Soay         711      1e8  1.73e-1 7.73e-2
## # … with 168 more rows
```

Prédictions



Conclusions

- `broom` permet de “tidyfier” les sorties de fonctions de modélisation (lm, glm, …).
- voir `broom.mixed` pour les modèles mixtes (`lme4`, `nlme` …)
- `purrr` facilite l’application de fonctions à des listes
- couplés avec `dplyr`, ceci permet de répliquer facilement un même modèle à des sous-tableaux de données
- `furrr` facilite la parallélisation sur plusieurs coeurs / un cluster de calcul (à optimiser cependant)
- syntaxe “tidyverse” concise et explicite (`... Explicit is better than implicit. Simple is better than complex ...`)
- packages très intéressants pour l’analyse statistique en grande dimension