Visualisation interactive de données dans R avec animint2

1

Ce site est le travail en cours du projet de traduction en français du site en anglais The `animint2` Manual, financé par FabriqueREL, livraison prévu pour mars 2026.

Pour créer ce site web à partir des fichiers source, nous avons utilisé les versions suivantes :

```
  R.version.string
```

```
[1] "R version 4.5.2 (2025-10-31)"
```

```
  packageVersion("animint2")
```

```
[1] '2025.12.3'
```

Part I

Fonctions de base

2

La grammaire des graphiques

Dans ce chapitre nous expliquons la grammaire des graphiques, un modèle puissant pour décrire une vaste gamme de visualisations de données. Après avoir lu ce chapitre, vous serez en mesure

- De citer les avantages de la grammaire des graphiques par rapport aux systèmes précédents de visualisation de données
- D'installer le package R `animint2`
- De traduire les esquisses de graphique en code ggplot dans R
- D'afficher des ggplots sur des pages web en utilisant `animint2`
- De créer des ggplots multicouches (avec plusieurs geoms)
- De créer des ggplots avec facettes

2.1 Historique et objectifs de la grammaire des graphiques

La plupart des systèmes informatiques d'analyse de données offrent des fonctions de création de graphiques pour visualiser les tendances des données. Les systèmes les plus anciens proposent des fonctions très générales permettant de tracer les éléments de base d'un graphique, tels que des lignes et des points (par exemple, les packages `graphics` et `grid` dans R). Si vous utilisez l'un de ces systèmes généraux, vous devez assembler vous-même les composants pour obtenir un graphique significatif et interprétable. L'avantage des systèmes généraux réside dans le peu de limites imposées aux types de graphiques qui peuvent être créés. Leur principal inconvénient est qu'ils ne fournissent généralement pas de fonctions permettant d'automatiser les tâches courantes (axes, panneaux, légendes).

Pour remédier aux inconvénients de ces systèmes de tracé généraux, des packages de visualisation tels que `lattice` ont été développés (Sarkar, 2008). Ils disposent de plusieurs types de graphiques prédéfinis et fournissent une fonction dédiée à la création de chaque type de graphique. Par exemple, `lattice` fournit la fonction `bwplot` pour créer des diagrammes en boîte et des boîtes à moustaches. L'avantage de ces systèmes est qu'ils facilitent grandement la création de graphiques complets avec légendes et panneaux. Par contre, ils ont l'inconvénient d'offrir un ensemble de types de graphiques prédéfinis, ce qui fait qu'il n'est pas facile de créer des graphiques plus complexes.

Les nouveaux systèmes basés sur la grammaire des graphiques se situent entre ces deux extrêmes. Ensuite, Leland Wilkinson propose la grammaire des graphiques pour décrire et créer un grand éventail de graphiques (Wilkinson, 2005). Hadley Wickham implémente ultérieurement plusieurs idées de la grammaire des graphiques dans le package R `ggplot2` (Wickham, 2009). Le package `ggplot2` présente plusieurs avantages par rapport aux systèmes précédents.

- Comme les systèmes généraux, et contrairement à `lattice`, `ggplot2` impose peu de limites aux types de graphiques qui peuvent être créés (il n'y a pas de types de graphiques prédéfinis).
- Contrairement aux systèmes généraux, et à l'instar de `lattice`, `ggplot2` permet d'inclure facilement des éléments de graphique courants tels que des axes, des panneaux et des légendes.
- Puisque `ggplot2` est basé sur la grammaire des graphiques, il est nécessaire de mapper explicitement les variables de données aux propriétés visuelles. Plus loin dans ce chapitre, nous expliquerons comment ce mappage permet la création d'esquisses qui peuvent être directement traduites en code R.

Enfin, tous les systèmes évoqués précédemment sont destinés à la création de graphiques statiques, qui peuvent être visualisés aussi bien sur un écran d'ordinateur que sur du papier. Cependant, le sujet principal de ce manuel est `animint2`, un package R pour les graphiques interactifs. Contrairement aux graphiques statiques, les graphiques interactifs sont conçus pour être visualisés sur un ordinateur équipé d'une souris et d'un clavier pour interagir avec le graphique. Étant donné que de nombreux concepts des graphiques statiques sont également utiles dans les graphiques interactifs, le package `animint2` est implémenté en tant que fork du package `ggplot2`. Dans ce chapitre, nous présenterons les principales caractéristiques de `ggplot2` qui seront également utiles pour la conception de graphiques interactifs dans les chapitres suivants.

En 2013, Toby Dylan Hocking et ses étudiants ont créé le package `animint`, qui dépend du package `ggplot2`. Cependant, de 2014 à 2017, `ggplot2` a introduit de nombreux changements incompatibles avec la grammaire interactive du package `animint`, ce qui a conduit en 2018, à la création du package `animint2` qui copie les parties pertinentes du package `ggplot2`. Désormais, `animint2` peut être utilisé sans que `ggplot2` soit installé. Il est en fait conseillé d'utiliser `library(animint2)` sans utiliser `library(ggplot2)`. On peut cependant utiliser `animint2` avec des packages qui importent `ggplot2` (sans l'attacher). À titre d'exemple, voir le Chapitre 16 qui utilise le package `penaltyLearning` (lequel importe `ggplot2`).

2.2 Installer et attacher animint2

Pour installer la dernière version d' `animint2` à partir du dépôt CRAN,

```
if(!requireNamespace("animint2"))install.packages("animint2")
```

```
Loading required namespace: animint2
```

Pour installer une version de développement encore plus récente d' `animint2` depuis GitHub,

```
if(!requireNamespace("animint2")){
  if(!requireNamespace("remotes"))install.packages("remotes")
  remotes::install_github("animint/animint2")
}
```

Une fois que vous avez installé `animint2`, vous pouvez charger et attacher toutes ses fonctions exportées via :

```
library(animint2)
```

2.3 Traduire des esquisses en ggplots

Dans cette section, nous expliquons comment traduire une esquisse de graphique en code R. Dans ce livre, nous utilisons le mot « esquisse » dans le sens « Première forme, traitée à grands traits et généralement en dimensions réduites, de l'œuvre projetée » (Larousse). Nous allons souvent présenter une esquisse, dessinée à la main, pour donner l'idée du graphique qu'on veut créer, avant d'écrire le code nécessaire. Chaque esquisse va comporter certains éléments qui nous permettra de facilement écrire le code correspondant. Avant d'écrire le code R pour un graphique que vous voulez créer, nous vous encourageons de faire vos propres esquisses à la main, soit dans un cahier, soit sur un tableau. Nous utilisons comme exemple un ensemble de données de la Banque mondiale. Commençons par charger et examiner les noms des colonnes.

```
if(!requireNamespace("animint2fr")){
  if(!requireNamespace("remotes"))install.packages("remotes")
  remotes::install_github("animint/animint2fr")
}
```

```
Loading required namespace: animint2fr
```

```
data(BanqueMondiale, package="animint2fr")
names(BanqueMondiale)
```

```
 [1] "iso2c"                      "country"
 [3] "year"                       "fertility.rate"
 [5] "life.expectancy"            "population"
 [7] "GDP.per.capita.Current.USD" "15.to.25.yr.female.literacy"
 [9] "iso3c"                      "region"
[11] "capital"                    "longitude"
[13] "latitude"                   "income"
[15] "lending"                    "Region"
[17] "région"                     "espérance.de.vie"
[19] "taux.de.fertilité"          "année"
[21] "pays"                       "PIB.par.habitant.USD"
[23] "alphabétisation"            "revenu"
```

On voit 24 noms ci-dessus. Le jeu de données `BanqueMondiale` comprend des mesures telles que le taux de fertilité et l'espérance de vie pour chaque pays sur la période 1960-2010. Pour simplifier les sorties ci-dessous, nous considérons seulement le sous-ensemble pertinant pour les visualisations :

```
banque_mondiale <- subset(
  BanqueMondiale,
  is.finite(taux.de.fertilité) & is.finite(espérance.de.vie),
```

```
  select=c(
    région, pays, année, taux.de.fertilité, espérance.de.vie,
    population, revenu))
tail(banque_mondiale)
```

```
                    région     pays année taux.de.fertilité espérance.de.vie
13032 Afrique subsaharienne Zimbabwe  2006             3.551         44.70178
13033 Afrique subsaharienne Zimbabwe  2007             3.491         45.79707
13034 Afrique subsaharienne Zimbabwe  2008             3.428         47.07061
13035 Afrique subsaharienne Zimbabwe  2009             3.360         48.45049
13036 Afrique subsaharienne Zimbabwe  2010             3.290         49.86088
13037 Afrique subsaharienne Zimbabwe  2011             3.219         51.23644
      population     revenu
13032   12724308 Revenu bas
13033   12740160 Revenu bas
13034   12784041 Revenu bas
13035   12888918 Revenu bas
13036   13076978 Revenu bas
13037   13358738 Revenu bas
```

```
dim(banque_mondiale)
```

```
[1] 9852    7
```

Le code ci-dessus imprime les dernières lignes, ainsi que les dimensions (lignes 9852 et colonnes 7) du tableau de données. Supposons que nous voulions voir s'il existe une relation entre l'espérance de vie et le taux de fertilité. Nous pourrions fixer une année, puis utiliser ces deux variables de données dans un nuage de points. L'esquisse ci-dessous contient les principaux éléments de cette visualisation de données.

Figure 2.1: World Bank scatterplot

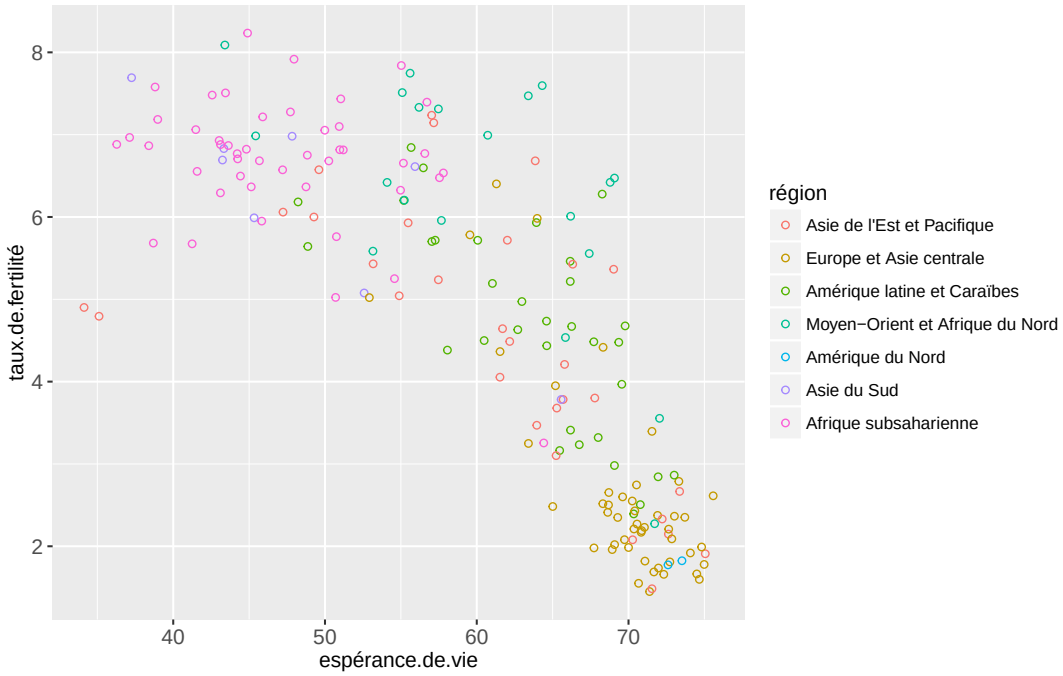
L'esquisse ci-dessus montre l'espérance de vie sur l'axe horizontal (`x`), le taux de fertilité sur l'axe vertical (`y`) et une légende pour la région. Ces éléments de l'esquisse peuvent être directement traduits en code R à l'aide de la méthode suivante. Tout d'abord, nous devons construire un tableau de données comportant une ligne par pays pour l'année 1975, et des colonnes nommées `espérance.de.vie`, `taux.de.fertilité`, et `région`. Le jeu de données `banque_mondiale` possède déjà ces colonnes, il suffit donc de considérer le sous-ensemble pour l'année 1975 :

```
banque_mondiale_1975 <- subset(banque_mondiale, année==1975)
tail(banque_mondiale_1975)
```

```
                              région           pays année taux.de.fertilité
11623      Asie de l'Est et Pacifique        Vanuatu  1975             5.929
11676      Asie de l'Est et Pacifique          Samoa  1975             5.237
12524 Moyen-Orient et Afrique du Nord Yémen, Rép. du  1975             8.089
12683           Afrique subsaharienne Afrique du Sud  1975             5.251
12895           Afrique subsaharienne         Zambie  1975             7.435
13001           Afrique subsaharienne       Zimbabwe  1975             7.395
      espérance.de.vie population             revenu
11623         55.47998      99879   Revenu moyen bas
11676         57.46951     151383   Revenu moyen bas
12524         43.40459    6676714   Revenu moyen bas
12683         54.57920   24728000 Revenu moyen élevé
12895         51.04137    4963655   Revenu moyen bas
13001         56.71702    6170266         Revenu bas
```

Le code ci-dessus imprime les dernières lignes en 1975, une ligne par pays. L'étape suivante consiste à utiliser les notes de l'esquisse pour coder un ggplot avec `aes()` qui fait correspondence entre variables et propriétés visuelles :

```
(nuage_de_points <- ggplot()+
  geom_point(
    mapping=aes(x=espérance.de.vie, y=taux.de.fertilité, color=région),
    data=banque_mondiale_1975))
```



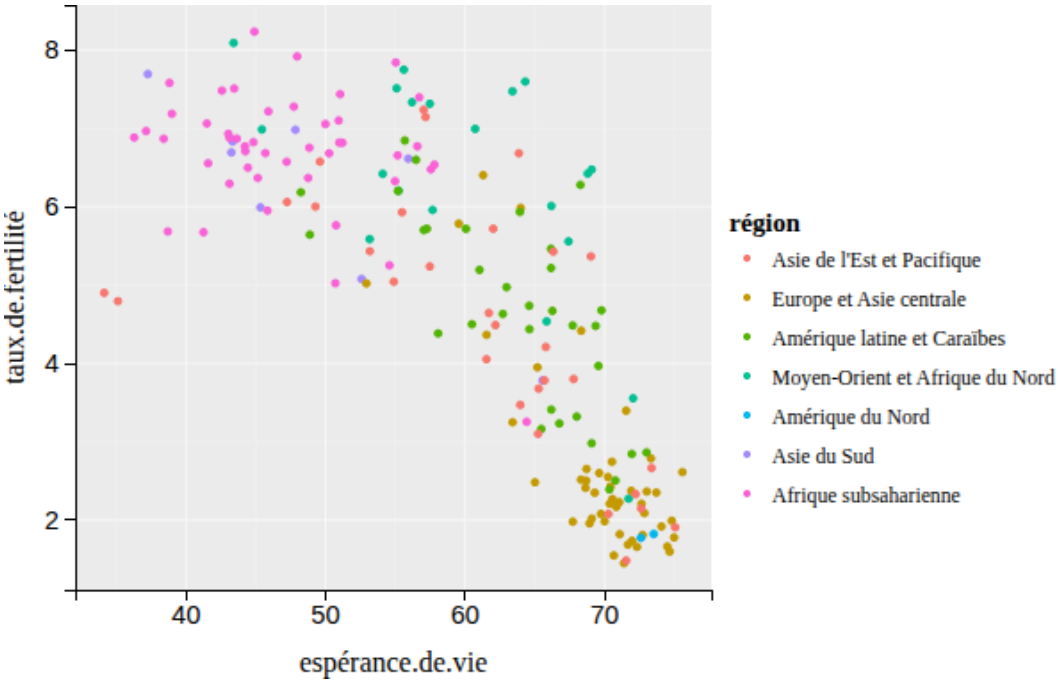
La fonction `aes()` est appelée avec les noms des propriétés visuelles (`x`, `y`, `color`) et les valeurs des variables de données correspondantes (`espérance.de.vie`, `taux.de.fertilité`,

`région`) Cette correspondance est appliquée aux variables du tableau de données `banque_mondiale_1975`, afin de créer les propriétés du `geom_point`. Le ggplot a été sauvegardé dans une variable nommée `nuage_de_points` qui, lorsqu'imprimé sur la ligne de commande R, affiche le graphique sur un périphérique graphique. Notez que nous avons automatiquement une légende `région` en couleur.

2.4 Afficher des ggplots sur des pages web à l'aide d'animint

Cette section explique comment `animint2` peut être utilisé pour afficher des ggplots sur des pages web. Le ggplot de la section précédente peut être affiché avec `animint2`, en utilisant la fonction `animint`

```
animint(nuage_de_points)
```



Lorsqu'il est visualisé dans un navigateur web, le graphique devrait paraître semblable aux versions statiques produites par les périphériques graphiques R standard, à la différence que la légende est interactive : en cliquant sur la légende, les points de cette couleur sont cachés ou affichés. A l'interne, la fonction `animint()` crée une liste de classe `"animint"`, puis R exécute la fonction `print.animint()` grâce au système d'objets S3. La fonction `animint2dir()` est appellé pour compiler la liste en un dossier de fichiers de données (CSV) et de codes (HTML, JavaScript) qui peuvent être affichés dans un navigateur web. Dans RStudio, vous devrait voir le graphique afficher sous l'onglet « Visualiseur ». Vous pouvez configurer le navigateur d'affichage avec l'option `browser`, par exemple : `options(browser="firefox")`. Lors de l'exécution du code ci-dessus, si l'animint ne s'affiche pas dans votre navigateur web pour une raison quelconque (par exemple si vous voyez une page web vierge), veuillez consulter notre FAQ qui vous aidera à trouver une solution.

Exercices: essayez de modifier le contenu du `aes()` du ggplot, puis de créer une nouvelle visualisation.

- Essayez d'utiliser la variable qualitative `revenu` dans `aes(color)` ou `aes(fill)`, et vous allez voir une légende interactive.
- Essayez `population` et `log10(population)` sur `x` ou `y`, et vous allez voir que la transformation logarithmique marche bien pour cette variable (`population` varie entre plusieurs ordres de grandeur).
- Essayer `aes(size=log10(population))` et vous allez voir une légende quantitative, qui n'est pas interactive.
- Pour avoir une légende interactive pour `population`, il faudra discretiser. Créez une nouvelle variable discrète `log10pop` avec définition `paste(round(log10(population)))`. Si vous utilisez `aes(size=log10pop)` vous allez voir une légende qualitative, qui est interactive.

2.5 Graphiques multicouches (plusieurs geoms)

Dans les ggplots, nous utilisons le mot « couche » comme synonym de « geom » comme `geom_point` que nous avons utilisé ci-dessus. Il existe d'autres geoms comme `geom_line` et `geom_path` qu'on peut utiliser pour dessiner les courbes. On appelle un graphique « multicouche » quand on utilise plusieurs geoms dans le même graphique, dessinés les uns sur les autres. Un graphique multicouche est utile lorsque vous souhaitez afficher plusieurs geoms ou ensembles de données différents dans le même graphique. Par exemple, considérons l'esquisse suivante qui ajoute une couche de `geom_path` à la visualisation de données précédente.

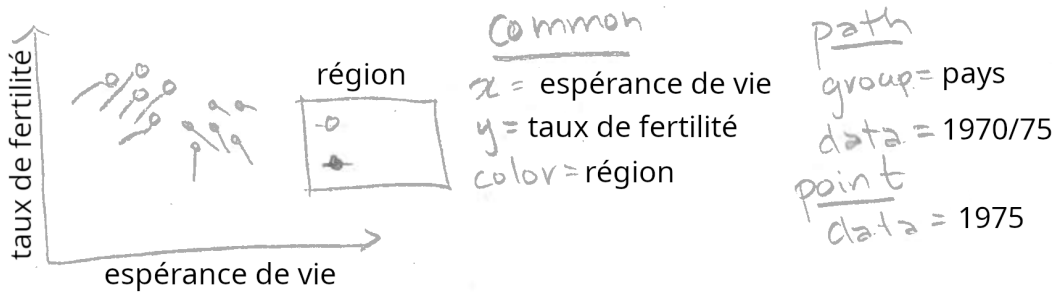


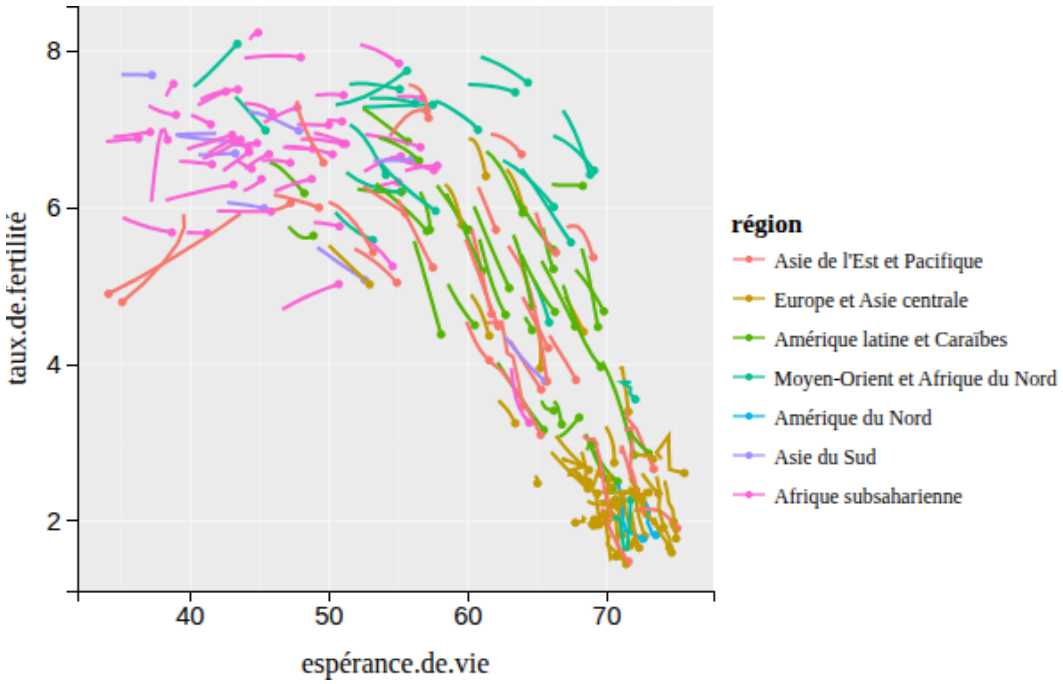
Figure 2.2: multi-layer WorldBank data viz

Notez que l'esquisse ci-dessus comprend deux geoms différents (point et path). Les deux geoms partagent une définition commune des éléments `x`, `y`, et `color`, mais avec des ensembles de données différents. En plus, le `geom_path()` contient `group=pays`, ce qui signifie qu'il faut dessiner une courbe pour chaque différente valeur de `pays`.

Nous traduisons ci-dessous cette esquisse en code R.

```
banque_mondiale_avant_1975 <- subset(
  banque_mondiale, 1970 <= année & année <= 1975)
gg_deux_couches <- nuage_de_points+
```

```
  geom_path(aes(
    x=espérance.de.vie,
    y=taux.de.fertilité,
    color=région,
    group=pays),
    data=banque_mondiale_avant_1975)
(vis.deux.couches <- animint(gg_deux_couches))
```



Dans le code ci-dessus, nous faisons un nouveau jeu de données `banque_mondiale_avant_1975`, et ensuite nous rajoutons un `geom_path` au `nuage_de_points` ce qui donnne `gg_deux_couches`, un ggplot avec deux couches. Ensuite nous sauvegardons la valeur rendue de l'appel à `animint()` dans l'objet `vis.deux.couches`, qu'on imprime pour l'afficher. Dans ce manuel, nous utiliserons souvent des noms de variables commençant par `vis` pour désigner les objets de classe `"animint"`, qui sont en fait des listes de ggplots et d'options.

Le graphique ci-dessus est une visualisation de données avec deux couches :

- le `geom_point` montre l'espérance de vie, le taux de fertilité et la région de tous les pays en 1975.
- le `geom_path` montre les mêmes variables pour les cinq années précédentes.

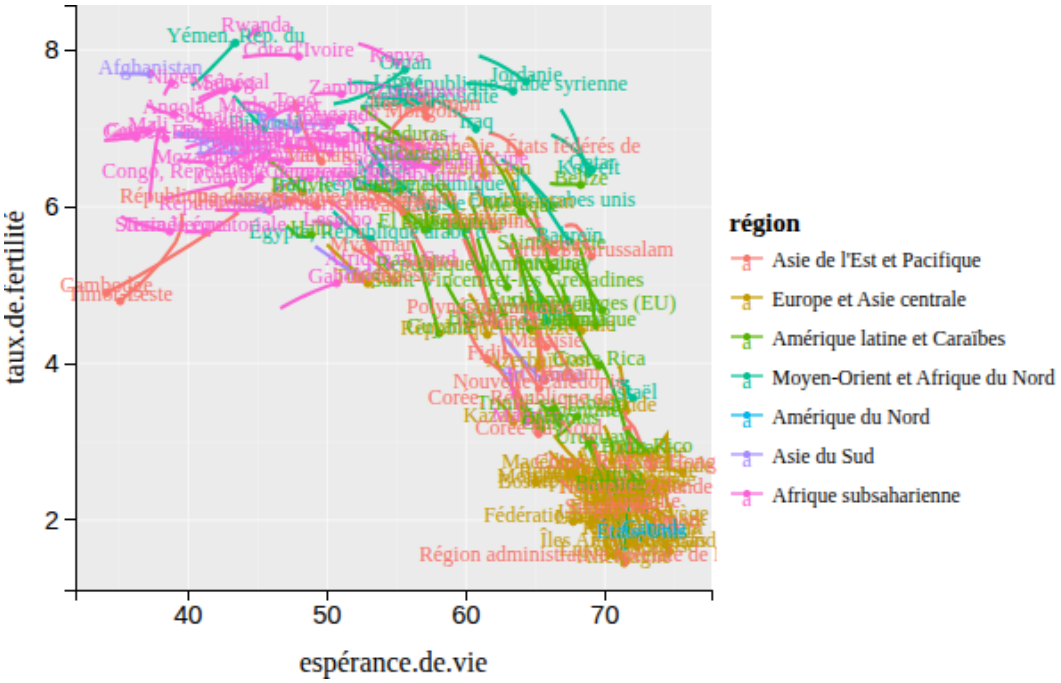
L'ajout de `geom_path` montre comment les pays ont changé au fil du temps. Les données de la plupart des pays se sont déplacées vers la droite et vers le bas, ce qui signifie une espérance de vie plus élevée et un taux de fertilité plus faible. Il y a cependant quelques exceptions. Par exemple, les deux pays d'Asie de l'Est situés en bas à gauche ont vu leur espérance de vie diminuer au cours de cette période. Par ailleurs, certains pays ont vu leur taux de fertilité augmenter.

Exercice: essayez de remplacer la légende `région` par une légende `revenu`. Indice

: vous devez utiliser la même spécification `aes(color=revenu)` pour tous les geoms. Vous pouvez utiliser `scale_color_manual` avec une palette de couleurs séquentielle, voir `RColorBrewer::display.brewer.all(type="seq")` et lire l'annexe pour plus de détails.

Pouvons-nous ajouter le nom des pays à la visualisation de données ? Ci-dessous, nous ajoutons une autre couche avec une étiquette de texte pour le nom de chaque pays.

```
gg_trois_couches <- gg_deux_couches+
  geom_text(aes(
    x=espérance.de.vie,
    y=taux.de.fertilité,
    color=région,
    label=pays),
    data=banque_mondiale_1975)
animint(gg_trois_couches)
```



Cette visualisation n'est pas très facile à lire, car il y a beaucoup d'étiquettes de texte qui se chevauchent. La légende interactive des régions aide un peu, puisqu'elle permet à l'utilisateur de masquer les données des régions sélectionnées. Cependant, ce serait encore mieux si l'utilisateur pouvait afficher et masquer le texte pour les pays individuellement. Ce type d'interaction peut être réalisé en utilisant les mots-clés `showSelected` et `clickSelects` que nous expliquons dans les chapitres 3 et 4.

Pour l'instant, passons à une des grandes forces d'`animint` : la visualisation de données à l'aide de plusieurs graphiques interconnectés.

2.6 Visualisations avec plusieurs ggplots interconnectés

L'utilisation de plusieurs ggplots est utile lorsque vous souhaitez présenter des plusieurs jeux de données, ou bien plusieurs façons de regarder un seul jeu de données. Dans ce genre de visualisation, il est courant d'utiliser au moins deux ggplots :

- un pour afficher un résumé ou survol de données, qu'on peut cliquer pour sélectionner un sous-ensemble d'intêret,
- un pour afficher plus de détails pour les données sélectionnées.

Prenons l'exemple d'une visualisation comportant deux graphiques : une série temporelle avec les données de la Banque mondiale de 1960 à 2010 (sommaire) et un nuage de points avec les données de 1975 (détails). Nous esquissons ci-dessous le graphique des séries temporelles.

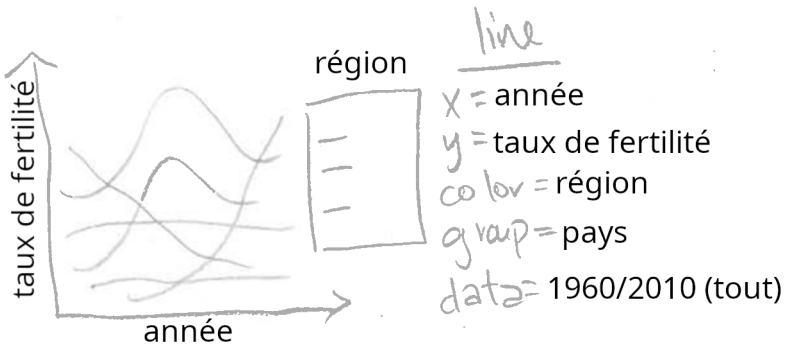


Figure 2.3: WorldBank data viz with two plots

Notez que l'esquisse ci-dessus peut être directement traduite dans le code R ci-dessous. Nous copions ensuite la visualisation existante (`vis.deux.couches`), puis nous rajoutons un ggplot à un nouvel élément nommé `timeSeries`.

```
vis.deux.graphes <- vis.deux.couches
vis.deux.graphes$timeSeries <- ggplot()+
  geom_line(aes(
    x=année,
    y=taux.de.fertilité,
    color=région,
    group=pays),
    data=banque_mondiale)
```

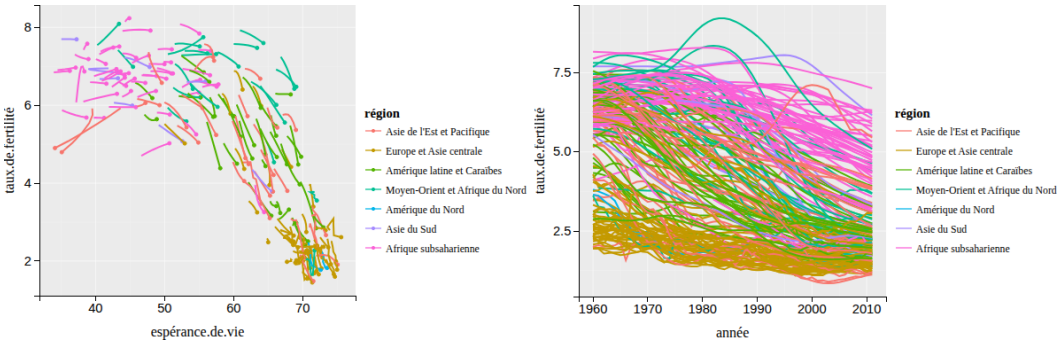
Il en résulte une liste nommée de deux éléments (les deux éléments sont des ggplots de classe `gganimint`).

```
summary(vis.deux.graphes)
```

```
           Length Class     Mode
plot1      9      gganimint list
timeSeries 9      gganimint list
```

On peut imprimer ou afficher cette liste de visualisation de données en tapant son nom. Puisque la liste contient deux ggplots, `animint2` affiche la visualisation de données sous la forme de deux graphiques liés.

```
vis.deux.graphes
```



La visualisation de données ci-dessus contient deux ggplots, qui mappent chacun différentes variables de données sur le plan horizontal `x`. La série temporelle utilise `aes(x=année)` et montre un résumé des valeurs du taux de fertilité pour toutes les années. Le nuage de points utilise `aes(x=espérance.de.vie)` et montre les détails de la relation entre le taux de fertilité et l'espérance de vie en 1975.

Essayez de cliquer sur une entrée de légende dans le nuage de points ou dans la série temporelle ci-dessus. Vous devriez voir les données et les légendes des deux graphiques se mettre à jour simultanément. Étant donné que `aes(color=région)` a été spécifié pour les deux graphiques, `animint` crée une seule variable de sélection partagée appelée `région`. Cliquer sur l'une des légendes a pour effet de mettre à jour l'ensemble des régions sélectionnées, `animint` met donc à jour les légendes et les données dans les deux graphiques en conséquence. Il s'agit du mécanisme principal utilisé par `animint` pour créer des visualisations de données interactives avec des graphiques liés, nous en discuterons plus en détail dans les deux prochains chapitres.

Exercice: utilisez `animint` pour créer une visualisation de données avec trois graphiques, en créant une liste avec trois ggplots. Par exemple, vous pouvez ajouter une série temporelle d'une autre variable de données telle que `espérance.de.vie` ou `population`.

Notez que les deux ggplots mappent la variable taux.de.fertilité sur l'axe y. Cependant, comme il s'agit de deux graphiques distincts, les plages de leurs axes y sont calculées séparément. Cela signifie que même lorsque les deux graphiques sont affichés côte à côte, les deux axes y ne sont pas exactement alignés. C'est problématique, la visualisation des données serait plus facile à décoder si chaque unité d'espace vertical représentait la même quantité de taux de fertilité. Pour obtenir cet effet, nous utilisons les facettes dans la section suivante.

2.7 Visualisation de données avec plusieurs panneaux (facettes)

Un ggplot peut avoir des facettes (panneaux), qui sont des sous-graphiques, chacun d'eux présentant un sous-ensemble de données différent. On utilise le mot « facette » pour les

fonctions `facet_*` correspondantes. L'un des principaux atouts de ggplots est qu'il est relativement facile de créer différents types de graphiques avec plusieurs facettes. Les facettes peuvent servir deux objectifs différentes :

- L'alignement des axes de plusieurs graphiques connexes contenant différents geoms. Cette technique facilite la comparaison entre plusieurs geoms différents et est également utile pour la visualisation interactive des données.
- La division des données d'un geom en plusieurs panneaux. Cette technique facilite la comparaison entre des sous-ensembles de données mais elle est moins utile pour la visualisation interactive de données (l'interactivité peut souvent la remplacer et produire le même effet comparatif pour les sous-ensembles de données).

2.7.1 Des geoms distincts dans chaque panneau (axes alignés)

Nous commençons par expliquer comment les facettes sont utiles pour aligner les axes quand on veut afficher des différents geoms dans chaque facette. Considérons l'esquisse ci-dessous qui contient un graphique avec deux panneaux.

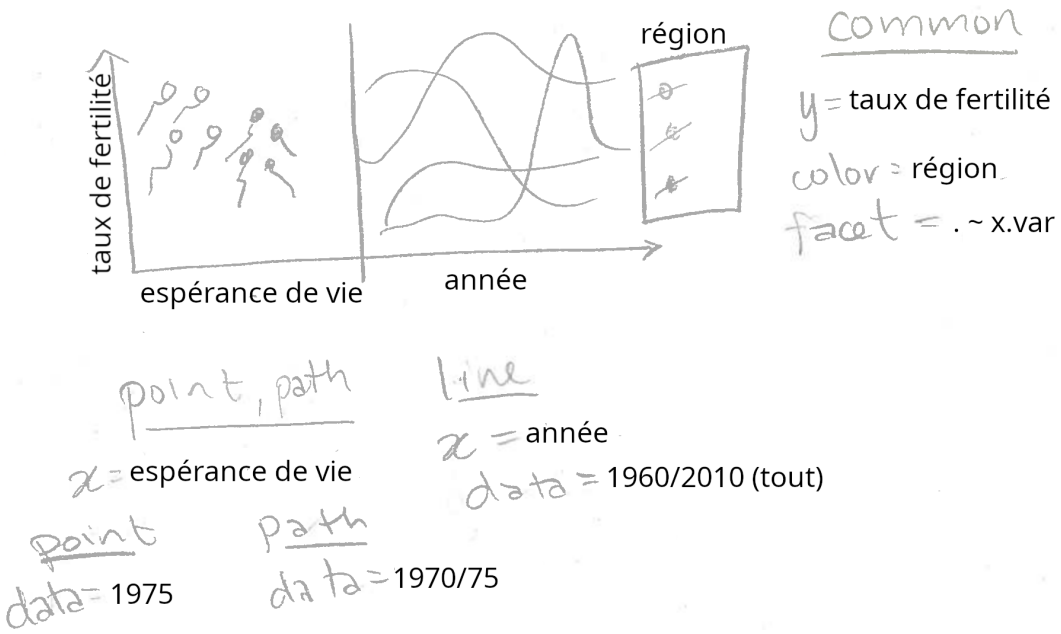
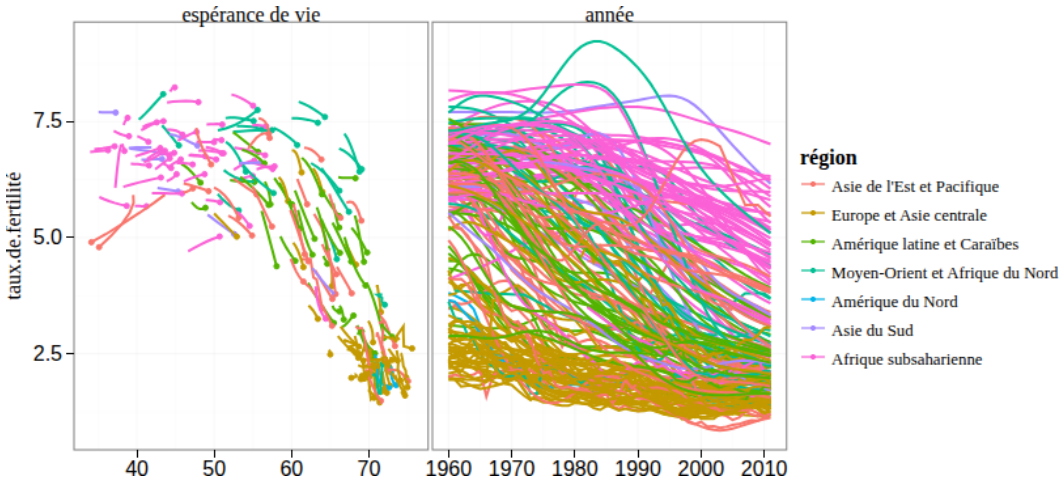


Figure 2.4: World Bank aligned plot

Notez que les deux panneaux tracent des geoms différents, chacun avec son propre `aes()`. Le `geom_point` et le `geom_path` dans le panneau de gauche ont `x=espérance.de.vie`, et le `geom_line` du panneau de droite a `x=année` . Notez également que nous avons écrit `facet=x.var`, nous devons donc ajouter une variable `x.var` à chacun des trois ensembles de données. Nous traduisons cette esquisse en code R ci-dessous.

```
add.x.var <- function(df, x.var){
  data.frame(df, x.var=factor(x.var, c("espérance de vie", "année")))
}
```

```
(vis.aligné <- animint(
  scatter=ggplot()+
    theme_bw()+
    theme_animint(width=600)+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région),
      data=add.x.var(banque_mondiale_1975, "espérance de vie"))+
    geom_path(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région,
      group=pays),
      data=add.x.var(banque_mondiale_avant_1975, "espérance de vie"))+
    geom_line(aes(
      x=année, y=taux.de.fertilité, color=région, group=pays),
      data=add.x.var(banque_mondiale, "année"))+
    xlab("")+
    facet_grid(. ~ x.var, scales="free")))
```



La visualisation de données ci-dessus contient un seul ggplot avec deux panneaux et trois couches. Le panneau de gauche montre les `geom_point` et `geom_path` et le panneau de droite montre les `geom_line`. Les panneaux ont un axe commun pour le taux de fertilité, ce qui permet de comparer directement le `geom_line` du panneau des séries temporelles avec le `geom_point` et le `geom_path` du panneau du nuage de points.

Il est à noter que nous avons utilisé la méthode `add.x.var` pour ajouter une variable `x.var` à chaque ensemble de données, puis nous avons utilisé cette variable dans la fonction `facet_grid(scales="free")`. Nous appelons cette méthode Ajouter une variable ensuite des facettes. Elle est généralement utile pour créer une visualisation de données à plusieurs panneaux avec des axes alignés. Ainsi, pour changer l'ordre des panneaux dans la visualisation, il nous suffirait de modifier l'ordre des niveaux de facteurs dans la définition de `add.x.var` .

Notez également que `theme_bw()` fait les bordures de panneaux noires et des fonds de panneaux blancs, et que `theme(panel.margin=0)` pourrait être utilisé pour supprimer l'espacement entre les panneaux. Cette élimination libère de l'espace pour les panneaux, ce qui permet de mettre l'accent sur les données. Nous appelons cette méthode Les facettes économes en espace, elle est généralement utile dans tout ggplot avec des facettes.

Dans la visualisation ci-dessus, les étiquettes de texte se chevauchent un peu, ce qui peut être corrigé par l'une des méthodes suivantes (exercice pour le lecteur).

- en utilisant l'argument `breaks` de `scale_x_continuous()`
- en réduisant la taille du texte avec `theme()` et `element_text()`
- en augmentant la largeur du graphique à l'aide de `theme_animint()` voir le Chapitre 6 pour plus d'informations.

2.7.2 Facettes pour comparaison des sous-ensembles de données

La deuxième raison d'utiliser des graphes avec plusieurs panneaux dans une visualisation de données est de comparer des sous-ensembles d'observations. Cette approche facilite la comparaison entre les sous-ensembles de données et peut être utilisée dans au moins deux situations différentes :

- L'ensemble de données d'un geom comporte trop d'observations pour être visualisé efficacement dans un seul panneau.
- Vous souhaitez comparer différents sous-ensembles de données tracées pour un geom

Prenons l'exemple de l'esquisse ci-dessous.

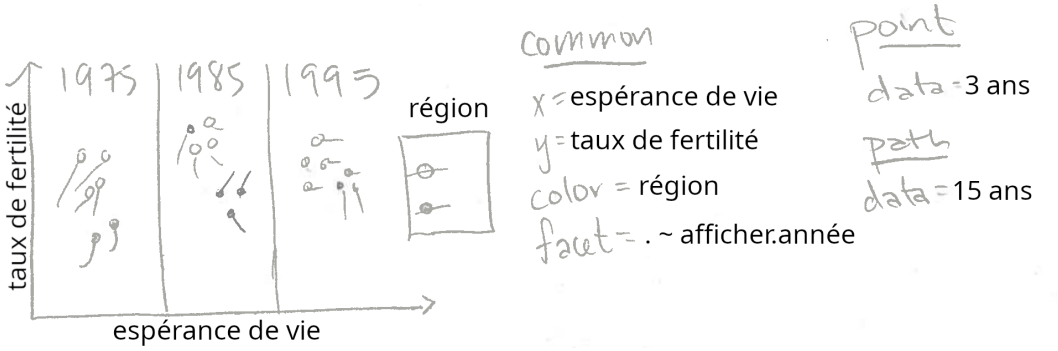


Figure 2.5: World Bank panels plot

Notez que les trois panneaux tracent les deux mêmes geoms, `geom_point` et `geom_path`. Puisque `facet=afficher.année` et qu'il y a trois panneaux, nous devrons créer des tableaux de données qui auront trois valeurs pour la variable `afficher.année`. Le `geom_point` ne contient des données que pour 3 ans, et le tableau `geom_path` a des données pour 15 ans (mais 3 valeurs de `afficher.année`). Le code ci-dessous crée ces deux ensembles de données pour trois années de l'ensemble de données de la Banque mondiale.

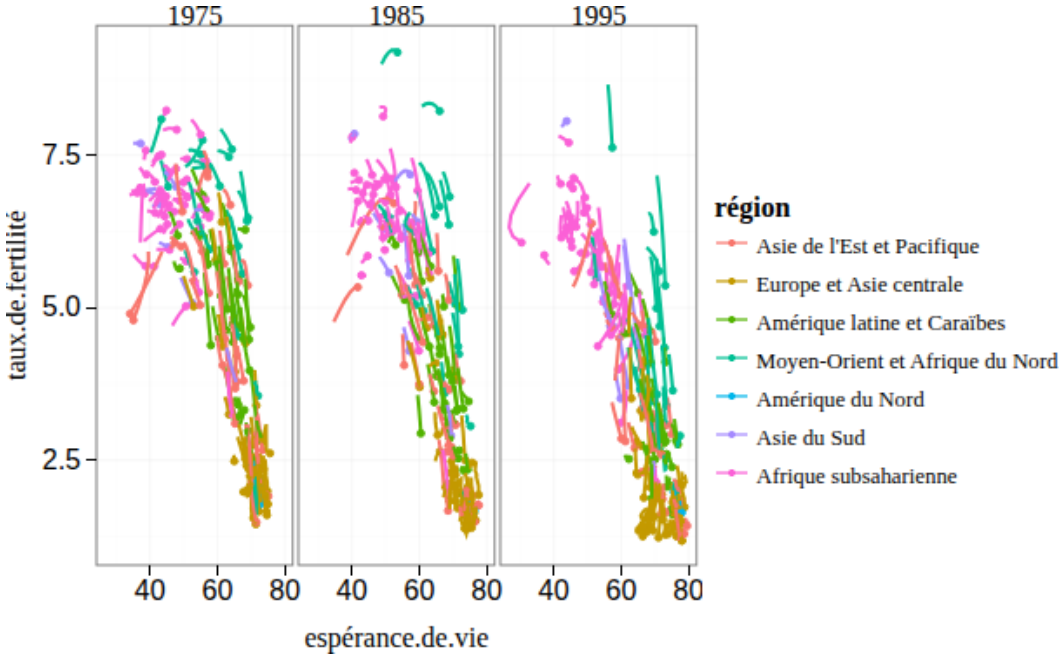
```
afficher.point.list <- list()
afficher.path.list <- list()
for(afficher.année in c(1975, 1985, 1995)){
  afficher.point.list[[paste(afficher.année)]] <- data.frame(
    afficher.année, subset(
      banque_mondiale, année==afficher.année))
  afficher.path.list[[paste(afficher.année)]] <- data.frame(
    afficher.année, subset(
      banque_mondiale,
```

```
      afficher.année - 5 <= année & année <= afficher.année))
}
afficher.point <- do.call(rbind, afficher.point.list)
afficher.path <- do.call(rbind, afficher.path.list)
```

Nous avons utilisé une boucle for sur trois valeurs de `afficher.année`, la variable que nous utiliserons plus tard dans `facet_grid`. Pour chaque valeur de `afficher.année` nous stockons un sous-ensemble de données sous la forme d'un élément nommé d'une liste. Après la boucle for, nous utilisons `do.call` avec `rbind` pour combiner les sous-ensembles de données. Il s'agit d'un exemple de la méthode Liste de tableau de données qui est généralement utile pour la visualisation interactive des données.

Ci-dessous, nous appliquons les facettes sur la variable `afficher.année` pour créer une visualisation de données avec trois panneaux.

```
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région),
      data=afficher.point)+
    geom_path(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région,
      group=pays),
      data=afficher.path)+
    facet_grid(. ~ afficher.année)+
    theme_bw())
```



La visualisation de données ci-dessus contient un seul ggplot avec trois facettes. Elle montre une plus grande partie de l'ensemble des données de la Banque mondiale que les visualisations

précédentes qui ne montraient que les données de 1975. Cependant, elle ne montre qu'un sous-ensemble de données relativement restreint. Vous pourriez être tenté d'utiliser un panneau pour afficher chaque année (pas seulement 1975, 1985 et 1995). Sachez toutefois que ce type de visualisation de données à plusieurs panneaux fonctionne bien seulement s'il y a un nombre limité de sous-ensembles de données. Au-delà d'une dizaine de panneaux, il devient difficile de voir toutes les données simultanément, et donc de faire des comparaisons significatives.

Au lieu de montrer toutes les données à la fois, nous pouvons créer une visualisation de données animée qui présente à l'observateur différents sous-ensembles de données au fil du temps. Dans le chapitre suivant nous montrerons comment le nouveau mot-clé `showSelected` peut être utilisé pour réaliser une animation, et nous révélerons plus de détails sur cet ensemble de données.

2.8 Résumé du chapitre et exercices

Ce chapitre a présenté les bases de la visualisation de données statiques avec `ggplot2`. Nous avons montré comment animint peut être utilisé pour afficher une liste de ggplots dans un navigateur web. Nous avons expliqué comment `ggplot2` facilite la création de graphiques avec plusieurs couches, et plusieurs facettes.

Exercices :

- Quels sont les trois principaux avantages de `ggplot2` par rapport aux systèmes antérieurs tels que `grid` et `lattice` ?
- Pourquoi utiliserait-on plusieurs couches dans un même graphique ?
- Quelles sont les deux raisons différentes de créer des graphiques à plusieurs facettes ? Laquelle est plus utile dans les visualisations interactives ?
- Définissons "A < B" comme signifiant qu'"un B peut contenir plusieurs A". Laquelle des affirmations suivantes est vraie ?
  - ggplot < panel
  - panel < ggplot
  - ggplot < animint
  - animint < ggplot
  - layer < panel
  - panel < layer
  - layer < ggplot
  - ggplot < layer
- Dans les facettes `vis.alignée`, pourquoi est-il important d'utiliser l'argument `scales="free"` ?
- Dans `vis.alignée` nous avons montré un ggplot avec un panneau de nuage de points à gauche et un panneau de séries temporelles à droite. Réalisez une autre version de la visualisation de données avec le panneau des séries temporelles à gauche et le panneau du nuage de points à droite.
- Dans `vis.alignée` le nuage de points affiche le taux de fertilité et l'espérance de vie, mais la série chronologique n'affiche que le taux de fertilité. Réalisez une autre version de la visualisation de données qui montre les deux séries temporelles. Indice : utilisez les panneaux horizontaux et verticaux dans `facet_grid`.
- Utilisez `aes(size=population)` dans le nuage de points pour indiquer la population de

chaque pays. Indice : `scale_size_animint(pixel.range=c(5, 10))` signifie que des cercles d'un rayon de 5/10 pixels doivent être utilisés pour représenter la population minimale/maximale.

- Créez une visualisation de données à plusieurs panneaux qui montre chaque année des données `banque_mondiale` dans un différent panneau. Quelles sont les limites de l'utilisation de graphiques statiques pour visualiser ces données ?
- Créez un `vis.alignée` en utilisant un système de tracé qui n'est pas basé sur la grammaire des graphiques. Par exemple, vous pouvez utiliser les fonctions du package `graphics` dans R (`plot`,`points`,`lines` etc.), ou `matplotlib` en Python. Quels sont les avantages de `ggplot2` et `animint` ?

Ensuite, dans le Chapitre 3 nous expliquons le mot-clé `showSelected`, qui nous permettra de préciser l'intéractivité ainsi que l'animation.

3

Le mot-clé `showSelected`

Dans ce chapitre, nous vous présentons `showSelected`, l'un des deux principaux mots-clés introduits par `animint2` pour la visualisation interactive de données. Après avoir lu ce chapitre, vous saurez :

- Utiliser le mot-clé `showSelected` dans vos esquisses pour spécifier des geoms pour lesquels seul un sous-ensemble de données doit être tracé à la fois.
- Utiliser les menus de sélection pour modifier le sous-ensemble de données tracées.
- Préciser des transitions fluides entre les sous-ensembles de données avec `aes(key)` et l'option `duration`.
- Créer des visualisations de données animées à l'aide de l'option `time`.

3.1 Esquisses avec `showSelected`

Dans le chapitre 2, nous avons expliqué comment planifier les codes nécessaires pour créer une visualisation avec une esquisse qui comprend les éléments principaux (geoms, axes, légendes, données). Dans cette section, nous allons expliquer comment le mot-clé `showSelected` peut être utilisé dans les esquisses. Le mot-clé `showSelected` est utilisé pour préciser les variables à utiliser pour sélectionner un sous-ensemble de données. Chaque geom possède son propre ensemble de données et sa propre définition de variables `showSelected`. Cela signifie que chaque geom peut afficher son propre sous-ensemble de données (qui est différent que les autres geoms).

En fait, nous avons déjà utilisé le mot-clé `showSelected` : il avait été généré automatiquement par les légendes interactives que nous avions créées dans les deux chapitres précédents. Considérons, par exemple, la courbe de Keeling du chapitre 1 présentée ci-dessous.

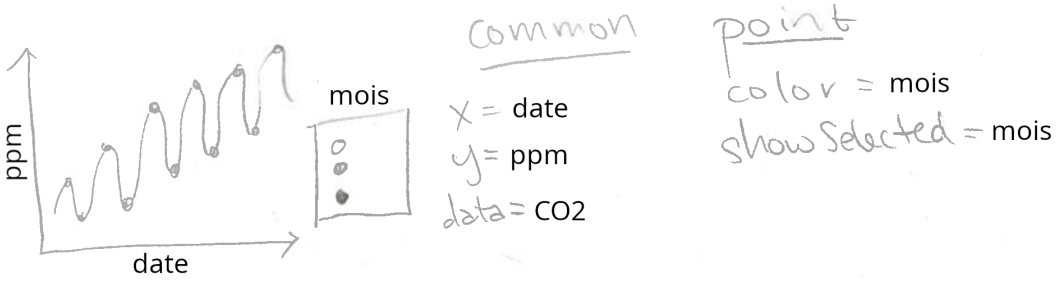


Figure 3.1: Esquisse de la courbe de Keeling

L'esquisse ci-dessus se traduit en le code R suivant.

```
ggplot(co2, aes(date, ppm))+
  geom_point(aes(color=mois), showSelected="mois")+
  geom_line()
```

Le code ci-dessus comprend `showSelected="mois"` pour le `geom_point()` ce qui signifie que le graphique affiche le sous-ensemble de données pour les mois sélectionnés. En revanche, puisque `geom_line()` n'inclut pas `showSelected`, il affiche toujours l'ensemble complet des données (peu importe les mois sélectionnés).

Prenons un autre exemple : l'esquisse ci-dessous de la première visualisation de données de la Banque mondiale, du chapitre 2.

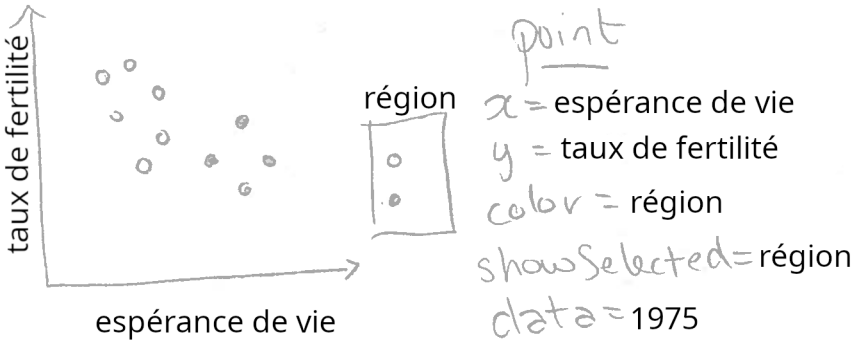


Figure 3.2: Esquisse Banque mondiale avec showSelected région

L'esquisse ci-dessus se traduit en le code R suivant.

```
ggplot()+
  geom_point(aes(
    espérance.de.vie, taux.de.fertilité, color=région),
    data=banque_mondiale_1975,
    showSelected="région")
```

Le code R contient `showSelected="région"` pour le `geom_point()`, ce qui signifie que le graphique affiche le sous-ensemble de données pour les régions sélectionnées.

Notez que l'esquisse dans chapitre 2 ne précisait pas explicitement `showSelected=région`. Nous avions simplement indiqué `aes(color=région)`, et `animint2` a automatiquement ajouté une variable `showSelected` correspondante. En général, `animint2` rajoute un mot-clé `showSelected` pour chaque variable utilisée dans une légende qualitative.

Cependant, le mot-clé `showSelected` n'est pas limité aux légendes qualitatives. Vous pouvez utiliser `showSelected` pour toutes les variables de votre choix, en précisant explicitement les variables dans l'argument `showSelected` du geom.

Chaque variable utilisée avec `showSelected` est traitée par `animint2` comme une variable de sélection. Par exemple, la visualisation de la courbe de Keeling a une variable de sélection (`mois`), tout comme la visualisation de la Banque mondiale (`région`). Pour chaque variable de sélection, `animint2` stocke les valeurs sélectionnées. Lorsque la sélection change, `animint2` met à jour le sous-ensemble de données affiché.

Les deux visualisations présentées ci-dessus ne comportent qu'une seule variable de sélection. Par contre, une visualisation de données peut avoir autant de variables de sélection que vous le souhaitez. Dans la section suivante, nous étudierons une visualisation des données de la Banque mondiale, avec les variables de sélection `région` et `année`.

3.2 Sélection de sous-ensembles avec menus

Considérons l'esquisse suivante avec une variable `showSelected` de plus et un ensemble de données différent.

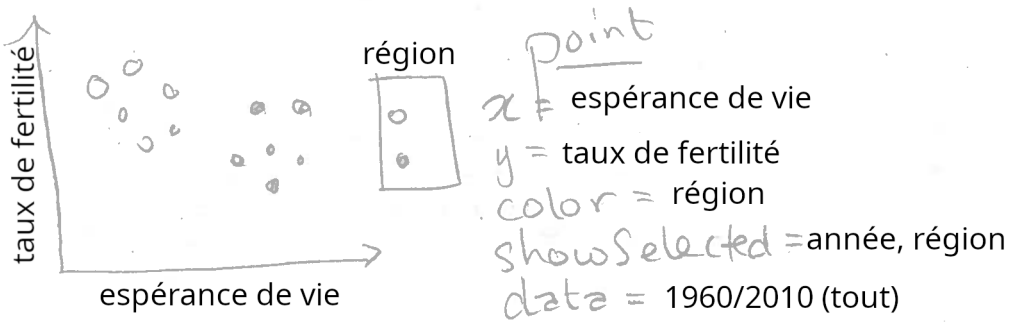
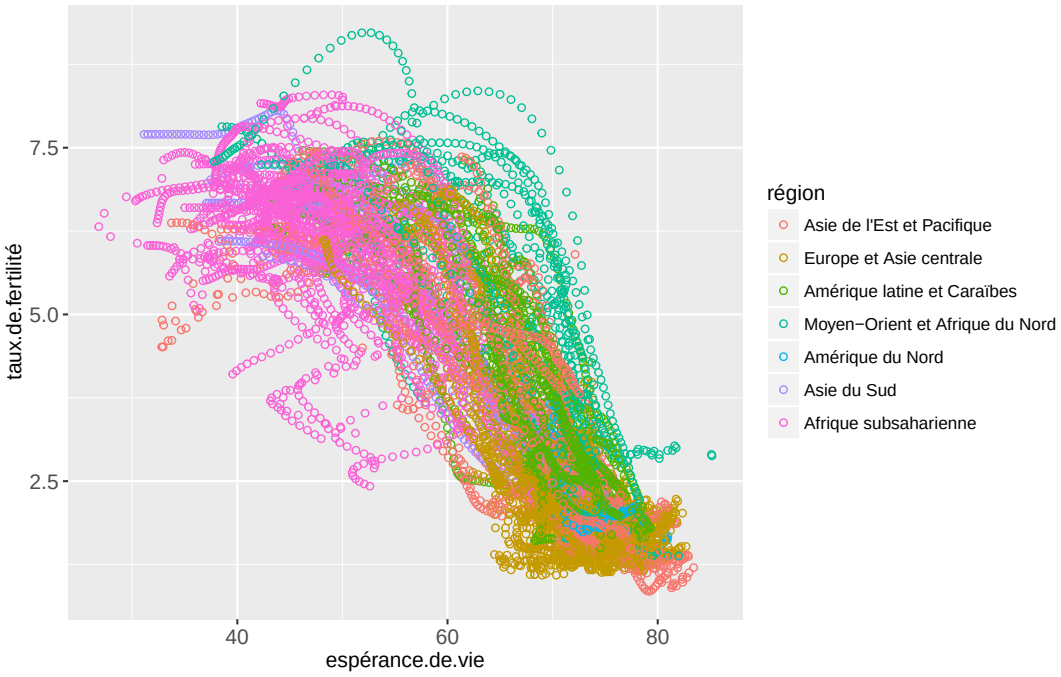


Figure 3.3: Esquisse Banque mondiale avec showSelected région et année

Notez qu'il y a deux variables `showSelected` : `région` et `année`. Notez également que les données sont spécifiées pour toutes les années (mais une seule année sera affichée à la fois, grâce au `showSelected=année`). Ci-dessous, nous traduisons l'esquisse en code R.

```
library(animint2)
data(BanqueMondiale, package="animint2fr")
nuage <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région),
    showSelected="année",
    data=BanqueMondiale)
nuage
```

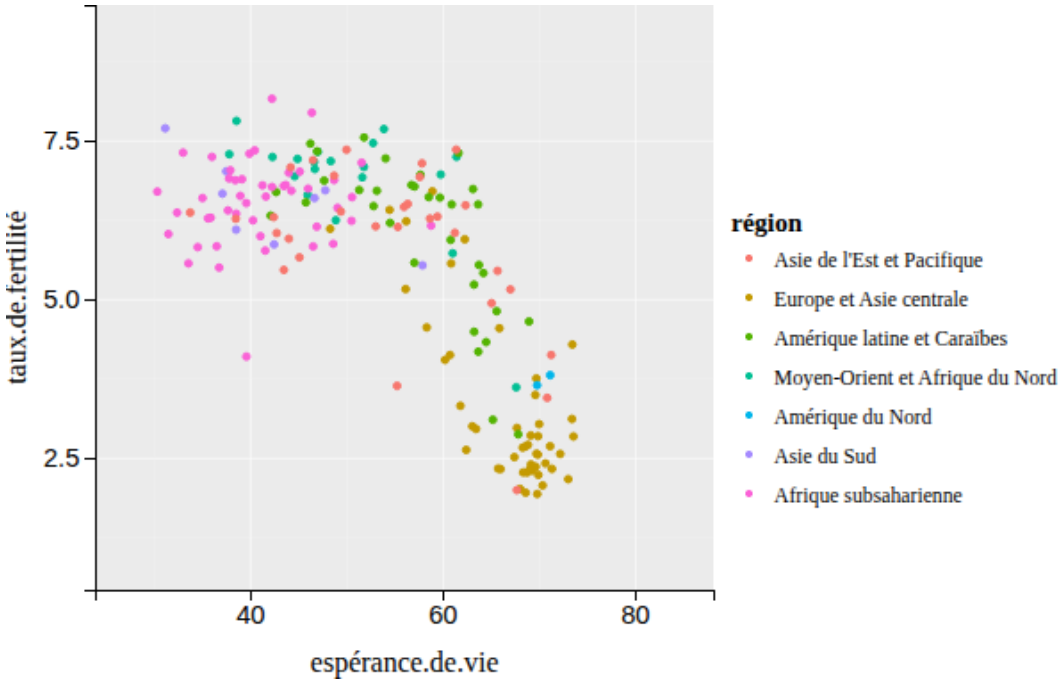
```
Warning: Removed 1490 rows containing missing values (geom_point).
```



Notez que le code ci-dessus contient le mot-clé `showSelected`, une nouveauté dans `animint2` par rapport à `ggplot2`. Le mot-clé `showSelected` est ignoré lorsque le graphique est affiché comme ci-dessus, avec les fonctions d'affichage habituelles de R, qui dessinent un nuage de points incluant toutes les années.

En revanche, donner le code ci-dessus comme argument pour `animint()` donne la visualisation interactive ci-dessous, qui dessine une année à la fois.

```
animint(nuage)
```



Notez que la visualisation ci-dessus comporte deux variables de sélection : `région` et `année` (`color=région` fait automatiquement de `région` une variable `showSelected`). Chaque variable dispose d'un menu sous la visualisation de données pour la modification de la sélection en cours. Dans cette visualisation, ces menus sont affichés par défaut. Pour les masquer, cliquez sur le bouton "Hide selection menus", et pour les réafficher sur "Show selection menus".

Les variables discrètes, telles que `région`, ont une sélection multiple par défaut, de sorte que plusieurs valeurs sont sélectionnées et affichées à la fois. Essayez de modifier la région sélectionnée dans la légende interactive et dans le menu de sélection. Lorsque vous modifiez la sélection à l'aide d'une des méthodes, la légende interactive et le menu de sélection sont tous les deux mis à jour, pour refléter la nouvelle sélection.

Nous utilisons les termes "manipulation directe" et "manipulation indirecte" pour décrire ces différentes façons de modifier la sélection. La manipulation directe est généralement plus facile à comprendre, parce qu'il s'agit de cliquer sur les objets que l'on souhaite modifier dans le graphique. De leur coté, les techniques de manipulation indirecte, telles que les menus, sont utiles dans d'autres cas (par exemple, la sélection du nom d'un pays). Dans la visualisation ci-dessus, vous pouvez modifier la valeur de `région` en utilisant soit la légende, soit le menu. L'utilisation de la légende est une technique de manipulation plus directe, puisque la légende est dessinée plus près du nuage de points qui sera mis à jour.

D'autres variables de sélection, comme `année`, ont une sélection simple par défaut, de sorte qu'une seule valeur est sélectionnée et affichée à la fois. Essayez de modifier la valeur sélectionnée pour la variable `année` à l'aide du menu de sélection. Vous devriez voir le nuage de points se mettre à jour immédiatement pour afficher le taux de fertilité et l'espérance de vie de tous les pays au cours de l'année que vous avez sélectionnée.

Exercice couches multiples : ajoutez un autre geom à ce nuage de points interactif. Comme dans le chapitre 2, vous pouvez utiliser un `geom_text` pour afficher le nom de chaque pays (facile), ou un `geom_text` pour afficher l'année sélectionnée (moyen), ou un `geom_path` pour

afficher les données des 5 années précédentes (difficile). Astuce : utilisez `showSelected=année` dans tous les geoms.

Exercice multi-plot : ajoutez une série temporelle à la visualisation de données ci-dessus. Comme dans le chapitre 2, vous pouvez utiliser un `geom_line` pour afficher le taux de fertilité de chaque pays, pour toutes les années. Ajoutez un `geom_vline` avec `showSelected=année` pour mettre en évidence l'année sélectionnée.

3.3 Transitions : l'option `duration` et `aes(key)`

Vous avez peut-être remarqué la présence de boutons sous chaque visualisation de données créée par `animint2`. Cliquez sur le bouton « Show animation controls » de la visualisation présentée ci-dessus. Vous verrez alors que ce tableau contient une ligne pour chaque variable de sélection. Les zones de texte indiquent le nombre de millisecondes utilisées pour les transitions fluides après la mise à jour de chaque variable de sélection. Pour chaque variable de sélection, la durée de transition par défaut est 0, ce qui signifie que les données seront immédiatement placées à leur nouvelle position (pas de transition fluide).

Pour illustrer l'importance des transitions fluides, essayez de changer la durée de transition de la variable `année` pour 2000. Ensuite, utilisez le menu pour modifier la valeur sélectionnée de la variable `année`. Vous devriez voir les points de données se déplacer lentement vers leurs nouvelles positions en 2 secondes.

Certaines transitions n'entraînent qu'un léger déplacement des points vers des positions proches (par exemple, 1979-1980). D'autres transitions entraînent un déplacement beaucoup plus important des points, vers des localisations plus éloignées (par exemple 1980-1981). Pourquoi?

Les transitions fluides n'ont de sens que pour les points de données qui existent à la fois avant et après la modification de la sélection. Dans le code R ci-dessous, nous calculons un tableau de contingence des points de données qui peuvent être tracés pour chacune de ces trois années.

```
trois.ans <- subset(BanqueMondiale, 1979 <= année & année <= 1981)
can.plot <- with(trois.ans, {
  (!is.na(espérance.de.vie)) & (!is.na(taux.de.fertilité))
})
table(trois.ans$année, can.plot)
```

```
     can.plot
      FALSE TRUE
1979    27  187
1980    27  187
1981    26  188
```

Le tableau de contingence ci-dessus montre clairement que 187 points peuvent être tracés en 1979 et 1980. Cependant, en 1981, il y a un point de données supplémentaire correspondant à un pays pour lequel nous n'avions pas de données en 1980. Nous présentons ci-dessous les données de ce pays, le Kosovo.

```
  subset(trois.ans, pays=="Kosovo")
```

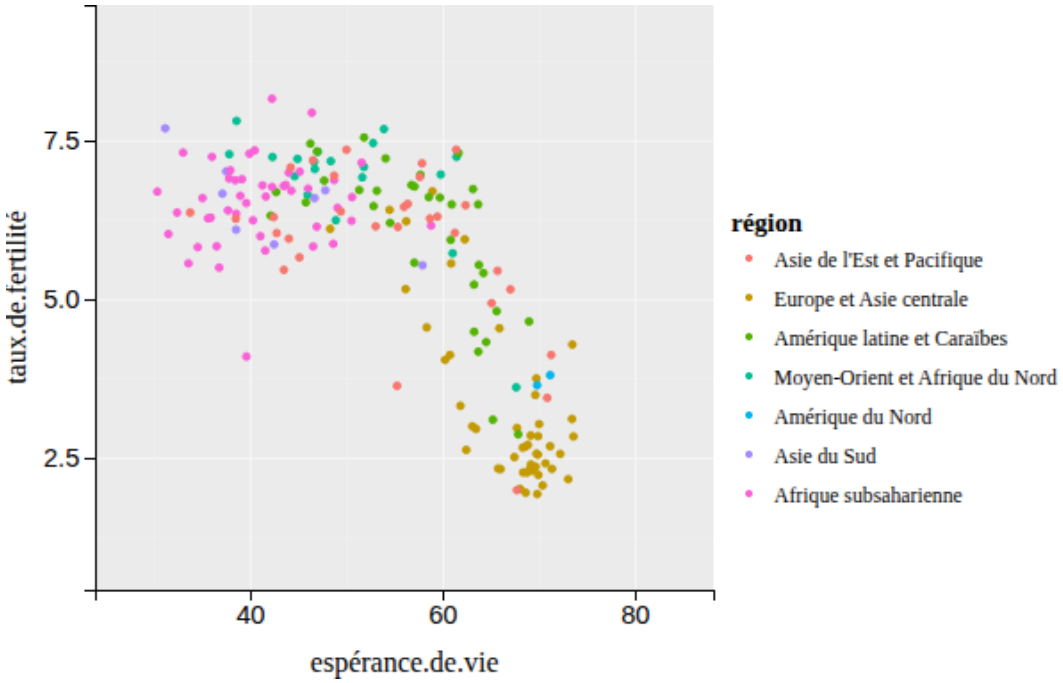
```
     iso2c country year fertility.rate life.expectancy population
5850    KV  Kosovo 1979             NA              NA    1491000
5851    KV  Kosovo 1980             NA              NA    1521000
5852    KV  Kosovo 1981         4.5758        65.93268    1552000
     GDP.per.capita.Current.USD 15.to.25.yr.female.literacy iso3c
5850                         NA                          NA   KSV
5851                         NA                          NA   KSV
5852                         NA                          NA   KSV
                                     region  capital longitude latitude
5850 Europe & Central Asia (all income levels) Pristina    20.926   42.565
5851 Europe & Central Asia (all income levels) Pristina    20.926   42.565
5852 Europe & Central Asia (all income levels) Pristina    20.926   42.565
                 income lending               Region                  région
5850 Lower middle income     IDA Europe & Central Asia Europe et Asie centrale
5851 Lower middle income     IDA Europe & Central Asia Europe et Asie centrale
5852 Lower middle income     IDA Europe & Central Asia Europe et Asie centrale
     espérance.de.vie taux.de.fertilité année   pays PIB.par.habitant.USD
5850               NA               NA  1979 Kosovo                  NA
5851               NA               NA  1980 Kosovo                  NA
5852         65.93268           4.5758  1981 Kosovo                  NA
     alphabétisation          revenu
5850              NA Revenu moyen bas
5851              NA Revenu moyen bas
5852              NA Revenu moyen bas
```

On voit bien que les données sur le taux de fertilité et sur l'espérance de vie sont absentes du tableau pour le Kosovo en 1979 et en 1980. Il ne serait donc pas possible d'effectuer une transition fluide pour ce pays entre 1980 et 1981, puisque les données ne sont présentes qu'à partir du 1981. Comment préciser qu'on veut utiliser `pays` comme clé d'identification de chaque point dessiné avec ce geom ? Dans le code ci-dessous, nous utilisons `aes(key=pays)` pour spécifier que la variable `pays` doit être utilisée pour faire correspondre les points de données avant et après la modification de la sélection.

```
nuage.key <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région,
    key=pays),
    showSelected="année",
    data=BanqueMondiale)
```

Le `aes(key)` dans le ggplot ci-dessus n'a de sens que pour la visualisation interactive des données, il est donc ignoré lorsqu'il est affiché avec les périphériques graphiques R habituels. Cependant, si nous affichons ce ggplot en utilisant `animint2`, la variable `pays` permettra des durées de transitions pertinentes. Pour spécifier une durée de transition par défaut pour la variable `année` nous utilisons l'option `duration` dans l'image de données ci-dessous.

```
(viz.duration <- animint(nuage.key, duration=list(année=2000)))
```



L'option `duration` doit être une liste nommée. Chaque nom doit être une variable de sélection et chaque valeur doit spécifier le nombre de millisecondes à utiliser pour la durée de la transition lorsque la valeur sélectionnée de cette variable est modifiée.

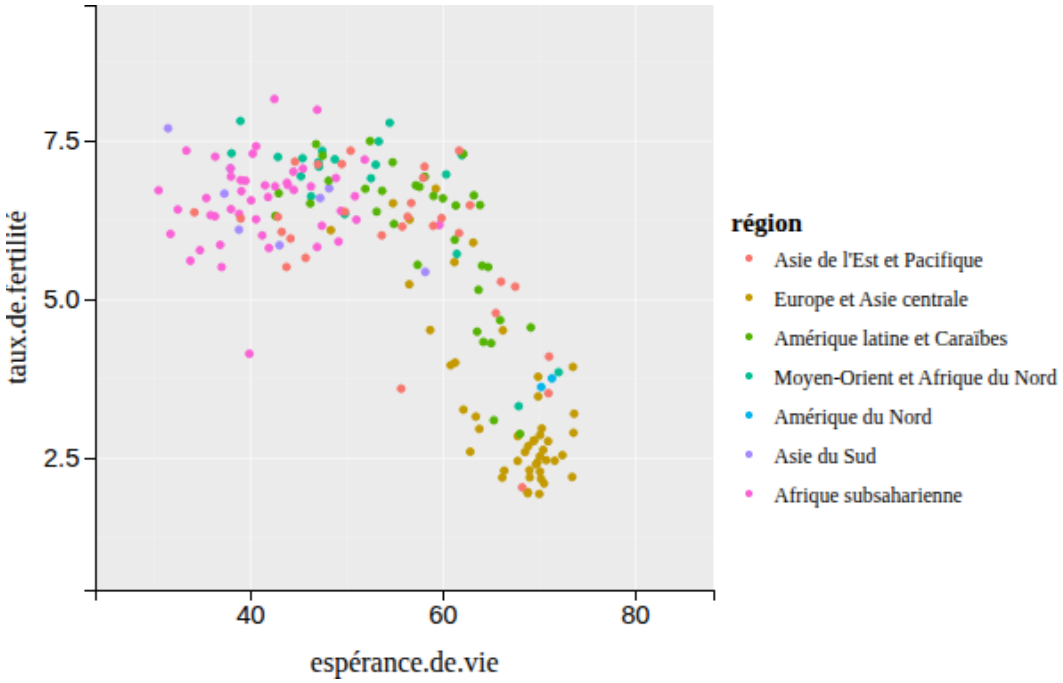
Si vous cliquez sur "Show animation controls" (Afficher les contrôles d'animation) dans le graphique ci-dessus, vous verrez que la zone de texte pour la variable année est 2000, comme spécifié dans le code R. Si vous changez la sélection de 1980 à 1981, vous devriez voir une transition correcte.

De façon générale, `aes(key)` doit être spécifié pour tous les geoms qui utilisent le mot-clé `showSelected` avec une variable qui apparaît dans l'option `duration`. Dans cet exemple, nous avons utilisé l'option `duration` pour spécifier une transition fluide pour la variable `année`. Puisque nous utilisons `showSelected=année` dans `geom_point` nous avons également spécifié `aes(key)` pour ce geom.

3.4 Animation : l'option `time`

L'option `time` permet de spécifier une variable à utiliser pour l'animation. Dans le code ci-dessous, on utilise l'option `time` pour préciser `année` comme variable d'animation, et mettre l'animation à jour toutes les 2000 millisecondes.

```
viz.duration.time <- viz.duration
viz.duration.time$time <- list(variable="année", ms=2000)
viz.duration.time
```



On dit que la visualisation ci-dessus est animée, parce que la sélection pour `année` va changer toutes les deux secondes. Pour contrôler l'animation, vous pouvez cliquer sur « Show animation controls » :

- Cliquez sur « Pause » pour arrêter l'animation.
- Cliquez sur « Play » pour la rédemarrer.
- Pour contrôler le temps entre les mises à jour de la variable d'animation, saisissez un nombre (en millisecondes) dans le champ « updates » et appuyez sur la touche d'entrée.
- Quand vous changez les délais pour l'animation et les transitions fluides, il faut que le délai d'animation « updates » soit plus grand que les autres délais, pour que les transitions fluides se terminent avant la prochaine mise à jour d'animation.

Exercice : réalisez une visualisation de données animée qui n'utilise PAS de transitions fluides. Indice : créez une liste de ggplots qui intègrent l'option `time` mais pas l'option `duration`.

3.5 Résumé du chapitre et exercices

Dans ce chapitre, nous avons expliqué l'utilisation du mot-clé `showSelected`, des menus de sélection, des transitions et de l'animation.

Exercices :

- Réalisez une version améliorée de `vis.alignée` du chapitre précédent. Au lieu de fixer l'année à 1975, utilisez `showSelected=année` pour que l'utilisateur puisse sélectionner une année. Ajoutez des geoms qui affichent l'année sélectionnée : un `geom_text` sur le nuage de points et un `geom_vline` sur la série chronologique.
- Traduisez l'un des exemples de library(animation) en animint2. Indice : dans le code de `library(animation)`, il y a toujours une boucle `for` dans l'option `time`. Au lieu

d'appeler une fonction d'affichage à l'intérieur de la boucle `for`, utilisez l'idiome liste de tableaux de données pour stocker les données qui doivent être tracées. Utilisez ensuite ces données avec `showSelected` pour créer des ggplots, et affichez-les avec `animint2`.

Dans le chapitre 4, nous vous expliquerons le mot-clé `clickSelects` qui indique un geom sur lequel on peut cliquer pour mettre à jour une variable de sélection.

4

Le mot-clé `clickSelects`

Dans ce chapitre, nous expliquons `clickSelects`, l'un des deux principaux mots-clés qu'`animint2` propose pour la visualisation interactive de données. Le mot-clé `clickSelects` précise quelle variable de séléction sera mise à jour quand on clique sur le geom correspondant. Chaque geom d'une visualisation de données possède son propre ensemble de données et sa propre définition du mot-clé `clickSelects`. Ainsi, en cliquant sur différents geoms, on peut modifier différentes variables de sélection.

Après avoir lu ce chapitre, vous serez en mesure de

- Comprendre comment les légendes interactives utilisent implicitement les `clickSelects`.
- Utiliser le mot-clé `clickSelects` dans vos esquisses de graphiques.
- Traduire vos esquisses de graphiques avec `clickSelects` en code R.
- Utiliser l'option `selector.types` pour spécifier plusieurs variables de sélection.

4.1 Les légendes interactives utilisent implicitement `clickSelects`

Dans cette section, nous expliquerons comment le mot-clé `clickSelects` est implicitement utilisé dans les légendes interactives. Si vous avez lu les chapitres précédents, vous avez déjà utilisé implicitement `clickSelects`, qui a été créé automatiquement pour les légendes interactives dans les chapitres précédents. Considérons l'esquisse de la visualisation de données de la Banque mondiale présentée au dernier chapitre.

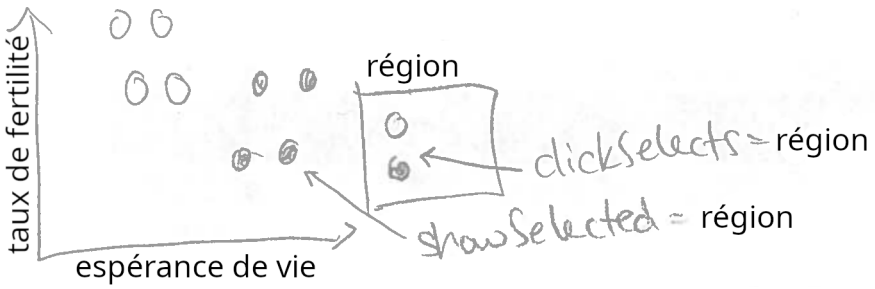


Figure 4.1: Esquisse avec clickSelects sur légende

Puisque la légende a `clickSelects=région` , cliquer sur une entrée de cette légende met à jour la variable de sélection `région`. Notez que `animint2` rend automatiquement chaque légende discrète interactive, vous n'avez donc pas besoin de spécifier explicitement `clickSelects=région` pour la légende. En fait, lorsque nous avons spécifié `color=région` pour le `geom_point animint2` fait deux choses automatiquement :

- `showSelected=région` est aussi affecté au même `geom_point` .
- `clickSelects=région` est attribué à la légende des couleurs.

Notez que les mots-clés `clickSelects` ne sont pas limités aux légendes interactives. Chaque geom a sa propre définition de `clickSelects` qui détermine la variable de sélection qui doit être mise à jour lorsqu'on clique sur ce geom. Dans la section suivante, nous donnerons plusieurs exemples sur la manière dont `clickSelects` peut être utilisé en combinaison avec `showSelected` pour créer des visualisations de données interactives.

4.2 Utiliser `clickSelects` pour identifier des points sur un nuage de points

L'objectif de cette section est de créer la visualisation suivante des données de la Banque mondiale.

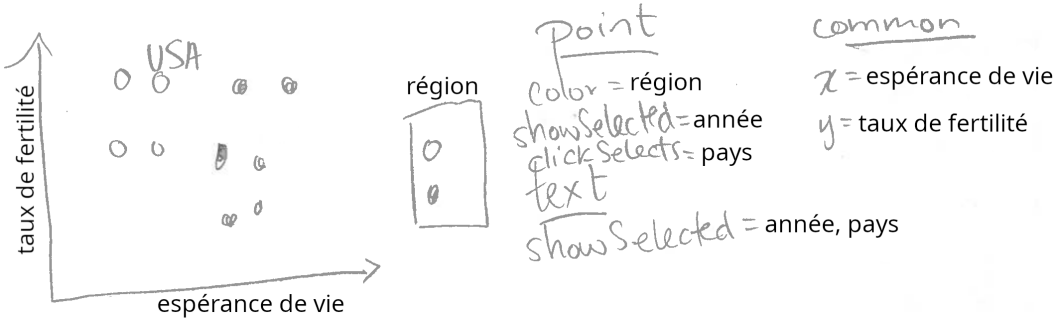
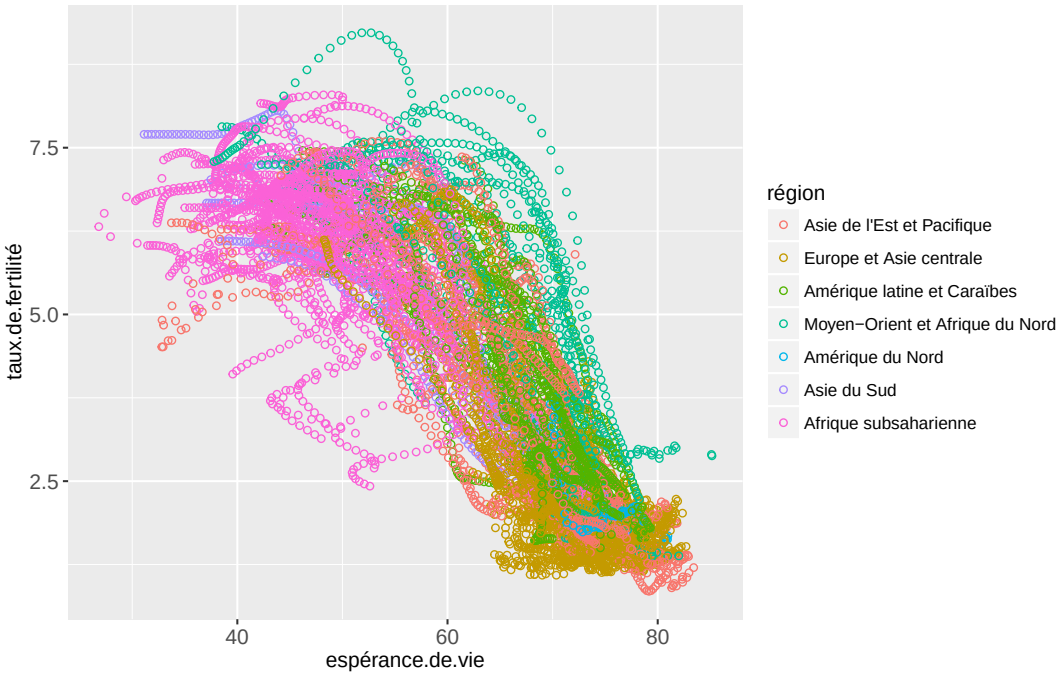


Figure 4.2: Esquisse avec texte pour un pays

Pour commencer, considérons le code R suivant qui génère un nuage de points des données de la Banque mondiale.

```
library(animint2)
data(BanqueMondiale, package="animint2fr")
(gg_nuage <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région,
    key=pays),
    showSelected="année",
    clickSelects="pays",
    data=BanqueMondiale))
```

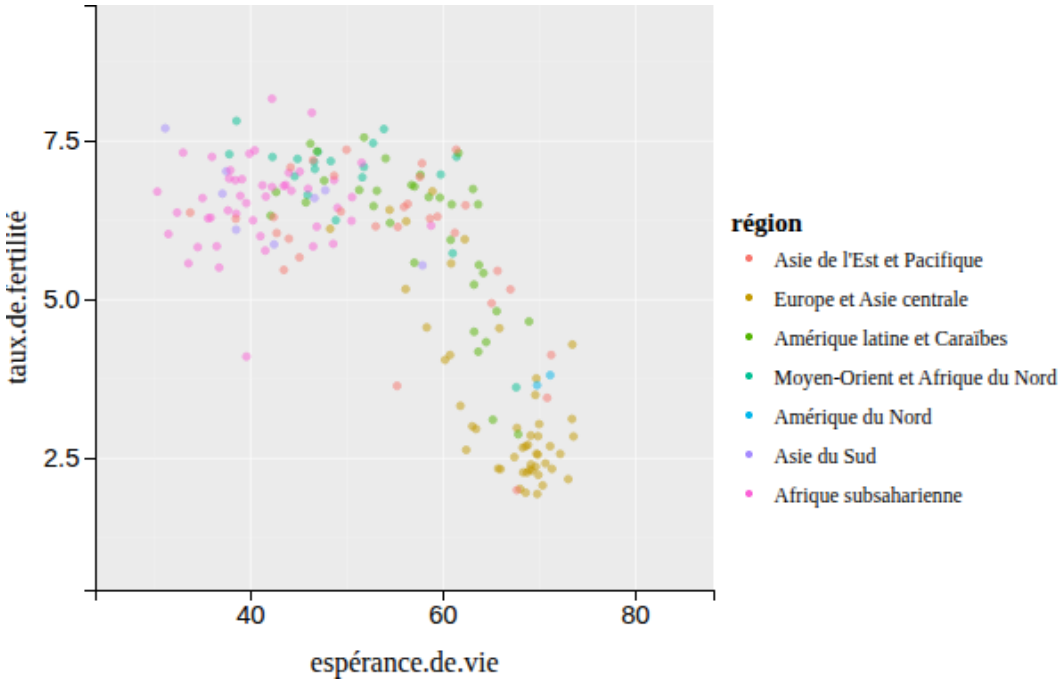
```
Warning: Removed 1490 rows containing missing values (geom_point).
```



Notez que le graphique ci-dessus n'est pas interactif, car il est affiché à l'aide du périphérique graphique R traditionnel. En revanche, afficher le même ggplot en utilisant `animint2` donne un graphique interactif :

- `showSelected="année"` indique qu'on affiche seulement les données pour l'année sélectionnée.
- `duration=list(année=2000)` indique un délai de 2 sécondes pour les transitions fluides entre les années.
- Nous avons vu dans le chapitre 3 qu'il est important d'utiliser `aes(key)` pour chaque geom avec transition fluide.
- Nous avons donc utilisé `aes(key=pays)` pour indiquer que la variable `pays` sera utilisée pour les transitions fluides entre les années.

```
(vis.nuage <- animint(
  nuage=gg_nuage,
  duration=list(année=2000)))
```



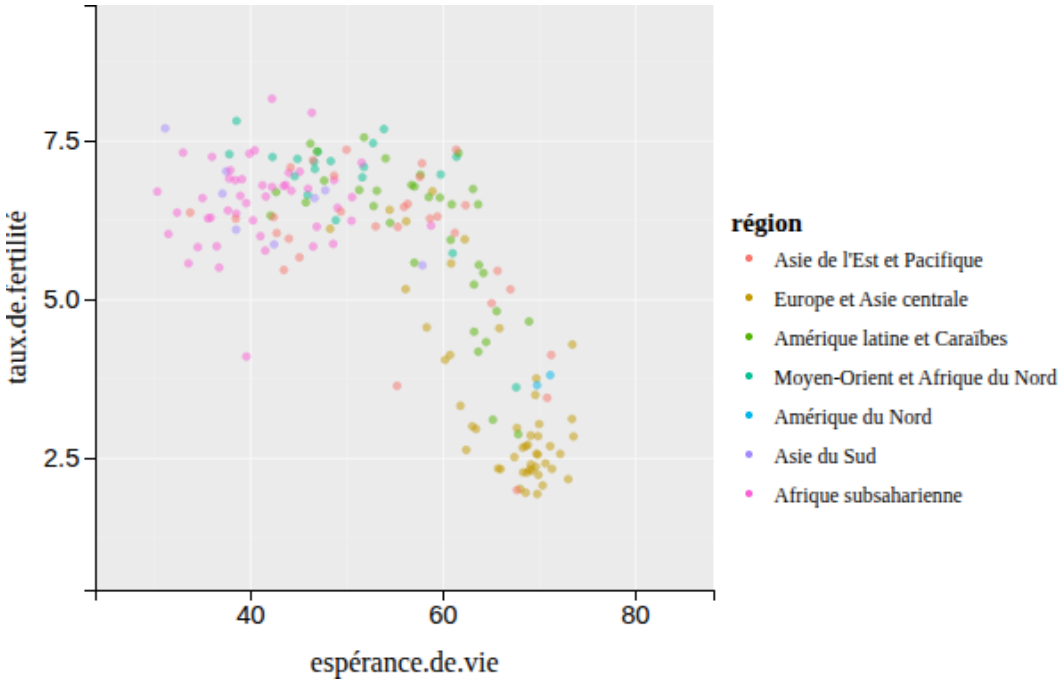
Essayez de cliquer sur des points de données dans le nuage de points ci-dessus. Vous devriez voir la valeur de la variable `pays` changer après avoir cliqué sur un point de données. Vous devriez également voir que le point de données pour le pays sélectionné est plus foncé que les autres. Cela permet de mettre en évidence la sélection en cours, cette surbrillance est générée automatiquement pour chaque geom avec `clickSelects` . Par défaut, le point sélectionné a `alpha=1` (totalement opaque, pas de transparence), et les autres points ont `alpha=0.5` (moitié semi-transparent). Ces valeurs par défaut peuvent être personnalisées ; par exemple, dans le code ci-dessous, un contour noir est utilisé pour mettre en évidence la sélection actuelle.

- `aes(fill=région)` indique que la variable `région` détermine la couleur du remplissage des cercles.
- `clickSelects=pays` indique qu'on peut cliquer pour changer la sélection de la variable `pays`,
- `color="black"` indique que le pays sélectionné aura un contour noir,
- `color_off=NA` indique que les pays non-sélectionnés auront un contour transparent.

```
animint(
  ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, fill=région,
      key=pays),
      clickSelects="pays",
      color="black",
      color_off=NA,
      showSelected="année",
      data=BanqueMondiale))
```

La visualisation de données ci-dessus affiche le nom du pays sélectionné dans le menu de sélection, mais il serait préférable de l'afficher sous forme d'une étiquette de texte sur le nuage de points. Nous pouvons le faire en ajoutant une couche `geom_text` avec deux variables `showSelected` :

```
vis.text <- vis.nuage
vis.text$nuage <- gg_nuage+
  geom_text(aes(
    x=espérance.de.vie, y=taux.de.fertilité, label=pays,
    key=pays),
    showSelected=c("année", "pays"),
    data=BanqueMondiale)
vis.text
```



Après avoir cliqué sur un point de données dans le nuage de points ci-dessus, vous devriez voir apparaître une étiquette de texte avec le nom du pays. Essayez également de changer l'année à l'aide du menu de sélection. Vous devriez voir l'étiquette de texte se déplacer dans une transition fluide en même temps que le point de données correspondant.

La visualisation de données ci-dessus contient plus d'un geom, chacun avec des caractéristiques interactives différentes. Essayez de cliquer sur le bouton « Start Tour » au bas de la visualisation, qui signifie « démarrer la visite guidée ». Il affichera une fenêtre qui explique les fonctions interactives disponibles pour le premier geom. En cliquant sur « Next », vous obtiendrez des informations sur le prochain geom, et en cliquant sur « Done » ou sur l'arrière-plan gris, vous mettrez fin à la visite guidée. La visite guidée peut être utile aux nouveaux utilisateurs, pour découvrir les fonctions interactives présentes dans chaque geom. Les informations affichées pendant la visite peuvent être personnalisées, en définissant les paramètres `help` et `title` de chaque geom (voir chapitre 6 pour un exemple).

Comme nous l'avons expliqué dans le dernier chapitre, toute variable spécifiée à l'aide du mot-clé `showSelected` d'un geom est traitée comme une variable interactive. Dans l'exemple ci-dessus, nous avons spécifié deux variables `showSelected` pour le `geom_text`. Cela signifie que l'on ne dessine qu'une étiquette de texte pour les lignes de la base de données `BanqueMondiale` qui correspondent aux valeurs actuelles des deux variables de sélection. Étant donné que chaque combinaison de `pays` et `année` a une ligne dans ces données, une seule étiquette de texte sera affichée à la fois.

Essayez de cliquer sur l'entrée de la légende qui correspond à la région du pays actuellement sélectionné (par exemple, si le Canada est sélectionné, essayez de cliquer sur l'entrée de la légende Amérique du Nord). Vous devriez voir le point disparaître, mais le texte rester affiché.

Exercice : comment faire disparaître le texte en même temps que le point ? Indice : il faut ajouter un mot-clé au `geom_text`.

Dans le dernier chapitre, nous avons introduit les termes "manipulation directe" et "manipulation indirecte" pour décrire les interactions avec les légendes et les menus. Dans la visualisation de données ci-dessus, nous pouvons modifier la valeur de la variable de sélection `pays` en cliquant sur un point de données (manipulation directe) ou en utilisant le menu de sélection (manipulation indirecte). Les deux techniques sont utiles, mais à des fins différentes :

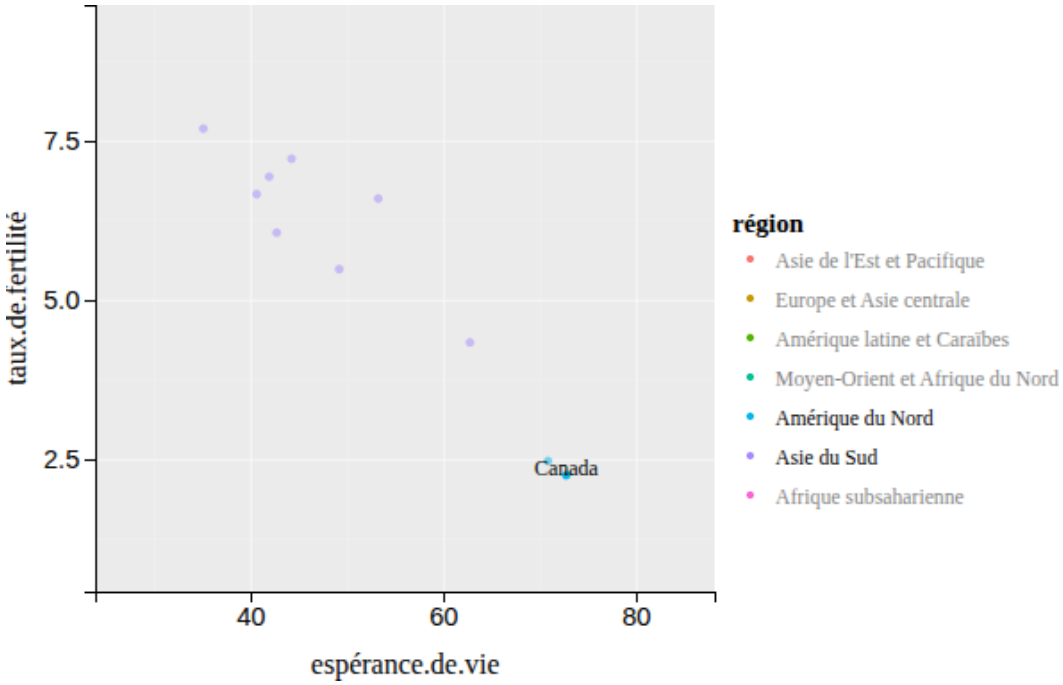
- La manipulation directe, soit de cliquer sur les points de données, est utile pour trouver le nom des pays ayant des valeurs extrêmes de taux de fertilité et d'espérance de vie. Par exemple, pour l'année 1960, en cliquant sur le point en bas à gauche du graphique, on découvre le nom du pays, le Gabon.
- La manipulation indirecte à l'aide des menus est utile pour voir la position tracée d'un pays d'intérêt. Par exemple, il serait difficile de trouver la France en cliquant sur tous les différents points, mais il est simple de trouver la France en tapant son nom dans le menu de sélection.

Notez que lorsque la visualisation ci-dessus est affichée pour la première fois, le pays sélectionné est Andorre et l'année sélectionnée est 1960. Comme les données pour Andorre sont manquantes en 1960, aucune étiquette de texte n'est affichée au départ. Pour modifier la première sélection, vous pouvez spécifier l'option `first` comme expliqué dans la section suivante.

4.3 L'option `first`

Pour spécifier la sélection qui doit être affichée lorsque la visualisation de données est exécutée pour la première fois, utilisez l'option `first`. Il doit s'agir d'une liste nommée avec des entrées pour chaque variable de sélection. Par exemple, le code ci-dessous spécifie 1970 comme première année, le Canada comme premier pays, et l'Amérique du Nord et l'Asie du Sud comme premières régions.

```
vis.first <- vis.text
vis.first$first <- list(
  année=1970,
  pays="Canada",
  région=c("Amérique du Nord", "Asie du Sud"))
vis.first
```



Notez que dans la visualisation des données ci-dessus, il n'y a qu'un seul pays sélectionné à la fois. Dans la section suivante, nous expliquerons comment l'option `selector.types` peut être utilisée pour modifier la variable de sélection `pays` en une variable de sélection multiple.

4.4 L'option `selector.types`

Dans cette section, notre objectif est de produire une version légèrement plus complexe du nuage de points de la section précédente. L'esquisse ci-dessous ne présente qu'une seule différence par rapport à l'esquisse de la dernière section : les étiquettes de texte sont affichées pour plus d'un pays.

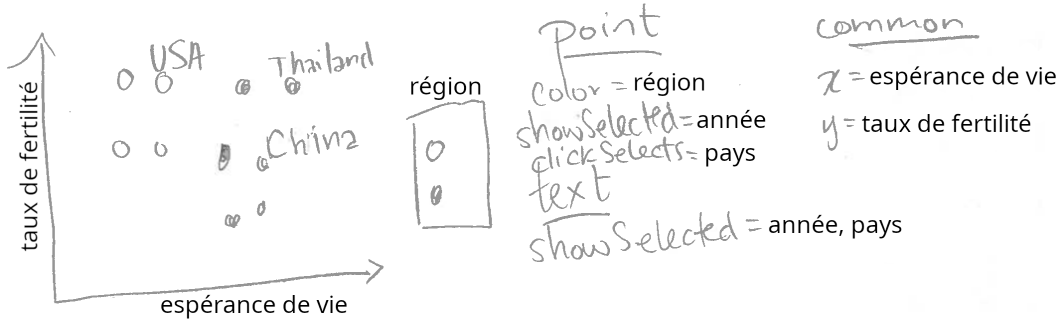


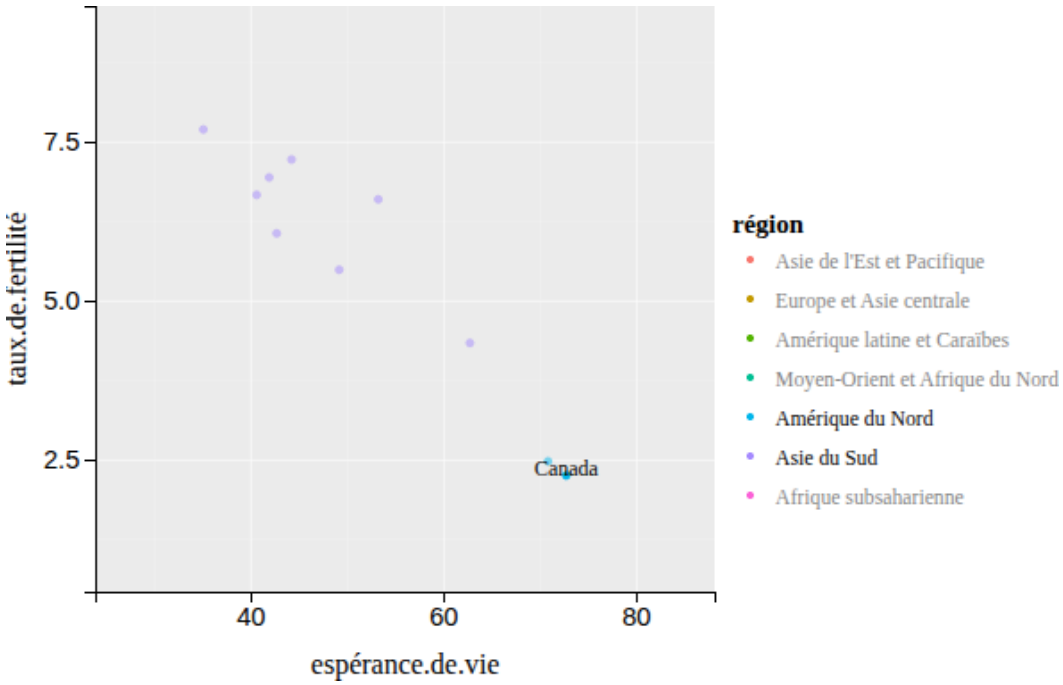
Figure 4.3: Esquisse avec texte pour plusieurs pays

Dans `animint2`, chaque variable de sélection a un type, soit simple, soit multiple. La sélection unique signifie qu'une seule valeur peut être sélectionnée à la fois. La sélection multiple signifie

qu'un nombre quelconque de valeurs peut être sélectionné à la fois. Dans les graphiques de la section précédente, la sélection multiple a été utilisée pour la variable `région` mais pas pour les variables `année` et `pays`. Comment cela se fait-il ?

Par défaut, `animint2` attribue une sélection multiple à toutes les variables qui apparaissent dans les légendes discrètes interactives, et une sélection unique aux autres variables. Cependant, une sélection unique (`"single"`) ou multiple (`"multiple"`) peut être précisée en utilisant l'option `selector.types`. Dans le code R ci-dessous, nous utilisons l'option `selector.types` pour spécifier que la variable `pays` doit être traitée comme une variable de sélection multiple.

```
vis.multiple <- vis.first
vis.multiple$selector.types <- list(pays="multiple")
vis.multiple
```



Lorsque la visualisation de données ci-dessus est affichée pour la première fois, elle montre les points de données de l'année 1970 pour chaque pays d'Amérique du Nord et d'Asie du Sud. Elle affiche également une étiquette de texte pour le Canada.

Vous avez peut-être remarqué qu'il est facile d'ajouter des pays à la sélection actuelle en cliquant sur des points de données. Normalement, le fait de cliquer sur un point de données sélectionné supprime ce pays de la sélection actuelle. Toutefois, dans cette visualisation de données en particulier, il n'est pas si facile de les supprimer, car les étiquettes de texte sont affichées au-dessus des points de données.

Exercice : Recréez la visualisation des données ci-dessus de manière à ce qu'un clic sur une étiquette de texte supprime le pays en question de l'ensemble de sélection. Indice : vous devez ajouter un mot-clé `clickSelects` au `geom_text`.

Notez que dans la visualisation de données ci-dessus, la variable année ne peut être modifiée que par le menu de sélection.

Dans la section suivante, nous ajouterons une facette avec un geom directement cliquable pour changer la variable `année`.

4.5 Sélection d'une année sur un graphique de série temporelle

L'objectif de cette section est d'ajouter un graphique de série temporelle sur lequel on peut cliquer pour changer l'année sélectionnée.

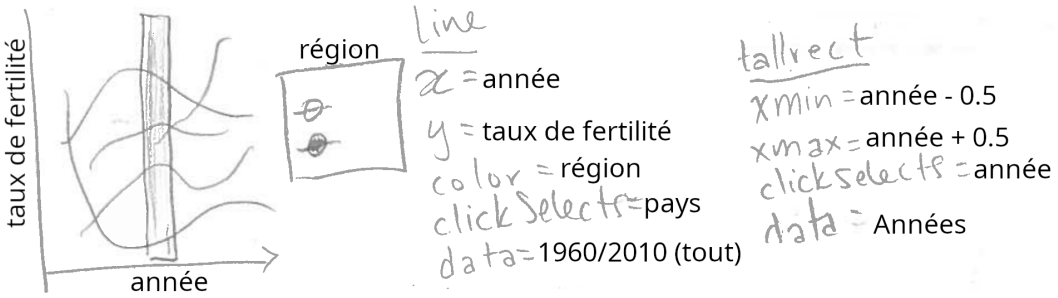


Figure 4.4: Esquisse avec série temporelle

Notez que l'esquisse ci-dessus comprend `geom_tallrect`, un nouveau geom introduit dans `animint2`. Il est qualifié de "grand" car il occupe tout l'espace vertical du graphique, il ne nécessite donc que la définition de ses limites horizontales dans `aes()` avec `xmin` et `xmax`. En spécifiant `clickSelects=année` nous indiquons que nous voulons tracer un `tallrect` pour chaque année, et que cliquer sur un `tallrect` changera l'année sélectionnée. Nous devons donc créer un nouvel ensemble de données appelé `années` avec une ligne pour chaque année des données `BanqueMondiale`.

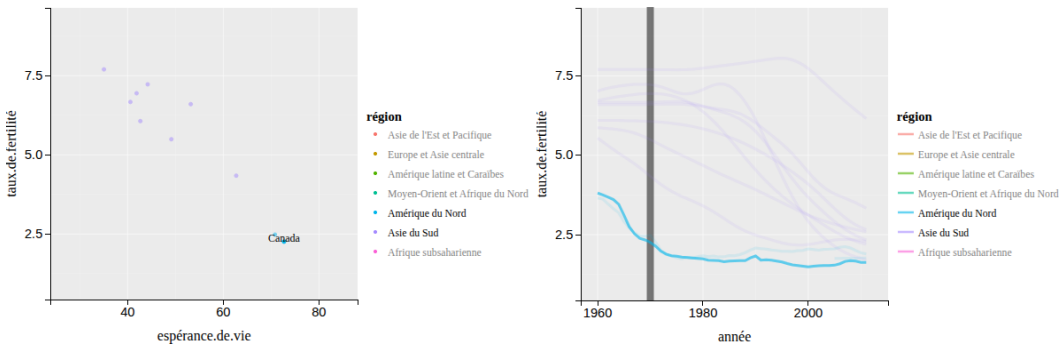
```
années <- data.frame(année=unique(BanqueMondiale$année))
head(années)
```

```
  année
1  1960
2  1961
3  1962
4  1963
5  1964
6  1965
```

Ensuite, nous ajoutons la série temporelle à la visualisation existante.

```
vis.timeSeries <- vis.multiple
vis.timeSeries$timeSeries <- ggplot()+
  geom_tallrect(aes(
    xmin=année-0.5, xmax=année+0.5),
    clickSelects="année",
    alpha=0.5,
```

```
    data=années)+
  geom_line(aes(
    x=année, y=taux.de.fertilité, group=pays, color=région),
    clickSelects="pays",
    size=3,
    alpha=0.6,
    data=BanqueMondiale)
vis.timeSeries
```



Essayez de cliquer sur l'arrière-plan de la série temporelle dans la visualisation de données ci-dessus. Vous devriez voir les points de données et les étiquettes de texte se déplacer dans une transition graduelle vers une position correspondant à l'année nouvellement sélectionnée.

Exercice : animez la visualisation de données en spécifiant l'option `time`, comme expliqué au chapitre 3.

Exercices avec multicouches : ajoutez un `geom_text` qui indique l'année en cours sur le nuage de points. Ajoutez un `geom_path` qui présente les données des 5 dernières années. Ajoutez un `geom_label_aligned` pour les pays sélectionnés dans la série temporelle.

4.6 Sélection d'une année sur une facette de série temporelle

L'objectif de cette section est d'ajouter une facette avec un graphique de série temporelle sur lequel on peut cliquer pour changer l'année sélectionnée.

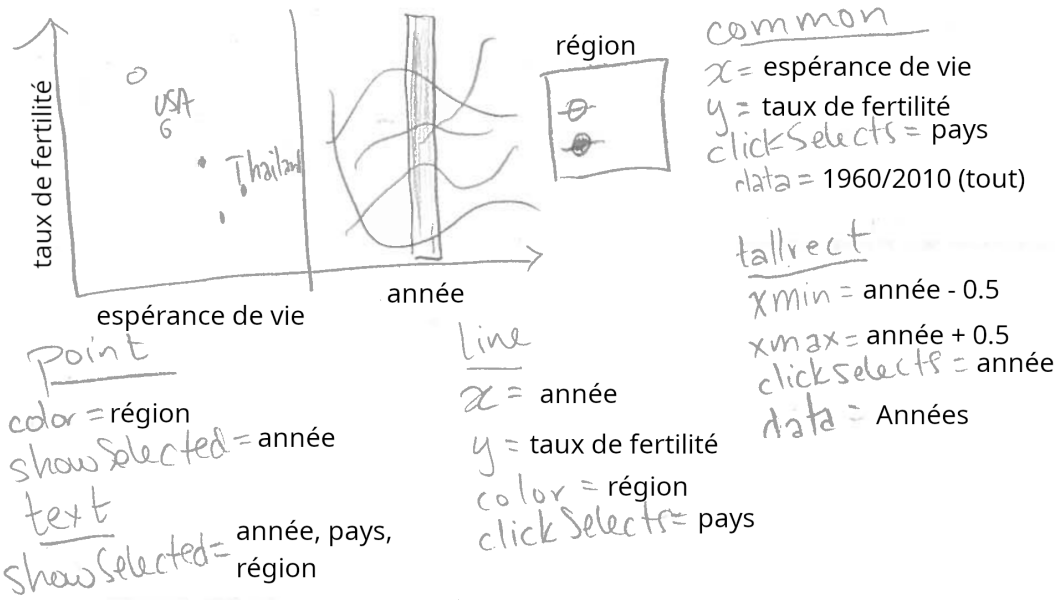


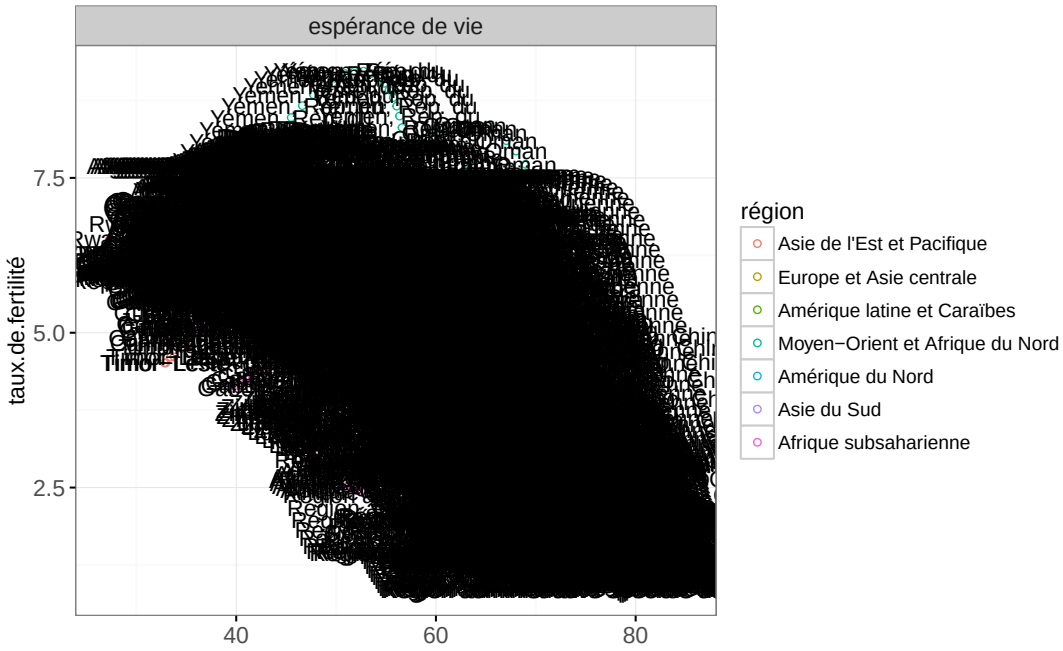
Figure 4.5: Esquisse avec facettes pour les données de la Banque mondiale

Tout d'abord, nous recréons le nuage de points de la section précédente à l'aide de la méthode Ajouter une variable ensuite des facettes qui est utile pour créer des ggplots avec des axes alignés.

```
add.x.var <- function(df, x.var){
  data.frame(df, x.var=factor(x.var, c("espérance de vie", "année")))
}
nuageFacet <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région,
    key=pays),
    showSelected="année",
    clickSelects="pays",
    data=add.x.var(BanqueMondiale, "espérance de vie"))+
  geom_text(aes(
    x=espérance.de.vie, y=taux.de.fertilité, label=pays,
    key=pays),
    clickSelects="pays",
    showSelected=c("année", "pays", "région"),
    data=add.x.var(BanqueMondiale, "espérance de vie"))+
  facet_grid(. ~ x.var, scales="free")+
  xlab("")+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))
nuageFacet
```

```
Warning: Removed 1490 rows containing missing values (geom_point).
```

```
Warning: Removed 1490 rows containing missing values (geom_text).
```



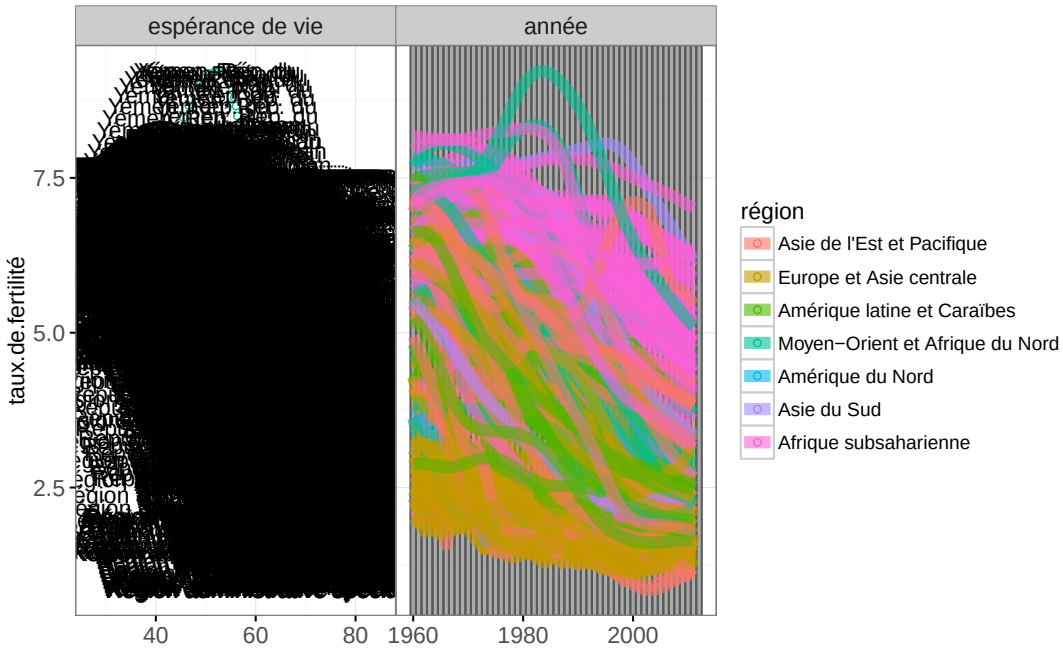
Notez que le ggplot ci-dessus utilise les mêmes définitions `aes` que le nuage de points de la section précédente. La seule différence est que nous avons utilisé un jeu de données `BanqueMondiale` enrichi avec une variable `x.var` supplémentaire que nous utilisons avec `facet_grid`. Ci-dessous, nous ajoutons des geoms pour un graphique de série temporelle qui est aligné sur l'axe du taux de fertilité.

```
nuage_série_temporelle <- nuageFacet+
  geom_tallrect(aes(
    xmin=année-0.5, xmax=année+0.5),
    clickSelects="année",
    alpha=0.5,
    data=add.x.var(années, "année"))+
  geom_line(aes(
    x=année, y=taux.de.fertilité, group=pays, color=région),
    clickSelects="pays",
    size=3,
    alpha=0.6,
    data=add.x.var(BanqueMondiale, "année"))
nuage_série_temporelle
```

```
Warning: Removed 1490 rows containing missing values (geom_point).
```

```
Warning: Removed 1490 rows containing missing values (geom_text).
```

```
Warning: Removed 759 rows containing missing values (geom_path).
```



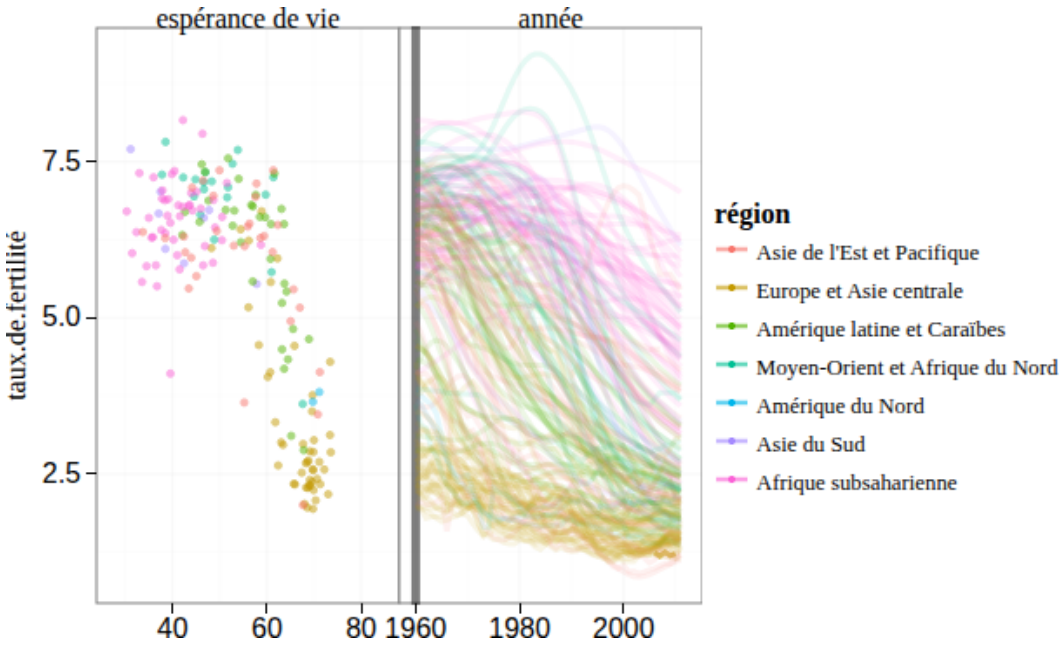
Les deux geom définis ci-dessus occupent un nouveau `facet` pour la valeur d'`année` de la variable `x.var` (définie par la fonction `add.x.var`). Puisque ces deux geoms ont des définitions différentes de `clickSelects`, un clic sur chaque geom mettra à jour le graphique d'une manière différente. Notez que pour le `geom_line` nous spécifions `size=3` ce qui signifie une largeur de trait de 3 pixels. En général, il est judicieux d'augmenter la taille des geoms avec `clickSelects`, afin de les rendre plus faciles à cliquer.

Notez également que nous avons spécifié `alpha=0.5` pour le `geom_tallrect` et `alpha=0.6` pour le `geom_line`. Puisque ces deux geoms définissent `clickSelects`, certaines lignes et rectangles seront sélectionnées, et d'autres ne le seront pas. Les valeurs `alpha` précisent l'opacité des objets sélectionnés, et les autres objets auront une valeur d'opacité de 0,5 inférieure à cette valeur. Dans l'exemple ci-dessus, les lignes non sélectionnées auront `alpha=0.1` et les `tallrects` non sélectionnés auront `alpha=0` (complètement transparent).

Depuis septembre 2023, il est également possible de spécifier l'opacité, le remplissage et la couleur des objets qui ne sont pas actuellement sélectionnés (`alpha_off`, `fill_off`, `color_off`). Les utilisateurs peuvent spécifier ces paramètres dans le geom (pas dans le `aes`) afin de créer librement une apparence différente pour les éléments sélectionnés et non sélectionnés, au lieu d'être contraints de dépendre du comportement décrit ci-dessus. Pour plus d'informations, voir la discussion et l'exemple dans chapitre 6, section Préciser le mode d'affichage de l'état de la sélection .

Enfin, nous utilisons le code R ci-dessous pour afficher le nouveau nuage de points aligné avec la série temporelle.

```
(vis.facets <- animint(nuage_série_temporelle))
```



La visualisation de données interactives ci-dessus contient un nouveau panneau avec des lignes qui montrent une série temporelle de taux de fertilité sur toutes les années. Puisque nous avons précisé `clickSelects=pays` pour le `geom_line`, un clic sur une ligne met à jour l'ensemble des pays sélectionnés. Puisque nous avons spécifié `clickSelects=année` pour le `geom_tallrect`, un clic sur un `tallrect` met à jour l'année sélectionnée.

Exercice : ajoutez les options `time`, `duration`, `first` et `selector.types` à la visualisation de données ci-dessus.

4.7 Résumé du chapitre et exercices

Dans ce chapitre, nous avons vu l'utilisation de `clickSelects` , l'un des deux principaux mots-clés introduits par `animint2` pour la conception de visualisations de données interactives. Nous avons utilisé l'ensemble de données de la Banque mondiale pour montrer comment `clickSelects` peut être utilisé pour définir différentes interactions pour chacun des geoms tracés. Nous avons expliqué comment l'option `first` peut être utilisée pour préciser les valeurs à utiliser lorsque le `animint2` est affiché pour la première fois. Nous avons également expliqué comment l'option `selector.types` peut être utilisée pour préciser le type de sélection (unique ou multiple).

Exercices :

- Jusqu'à présent, nous avons vu trois façons différentes de modifier les variables de sélection : (1) utiliser des légendes interactives, (2) utiliser des menus de sélection et (3) cliquer sur les données à l'aide de `clickSelects`. Classez ces trois techniques de manipulation, de la plus directe, à la moins directe. Quelle technique privilégier et selon quelles circonstances ?
- Quand `geom_point(clickSelects=something, alpha=0.75)` est affiché avec le périphérique graphique R habituel, quel est le degré d'opacité/transparence pour tous

les points de données ? Lorsque `animint2` affiche le même geom, certains points seront sélectionnés et d'autres non. Quelle est l'opacité/transparence des points sélectionnés ? Quelle est l'opacité/transparence des points non sélectionnés ?
- Ajouter `aes(size=population)` aux points du nuage de points de la Banque mondiale. La légende de la taille est-elle interactive ? Pourquoi ?
- Ajoutez un `geom_text` au nuage de points de la Banque mondiale qui montre l'année sélectionnée.
- Ajoutez un `geom_text` à la série chronologique de la Banque mondiale pour afficher le nom des pays sélectionnés.
- Ajoutez un `geom_path` au nuage de points de la Banque mondiale pour afficher les données des cinq dernières années.
- Utilisez l'option `time` pour créer une version animée de `vis.facets`.
- Utilisez `help` et `title` pour chaque geom dans `vis.facets` afin de créer un tour guidé qui fournit plus d'informations sur ce qui est affiché dans chaque geom.

Dans le Chapitre 5 nous expliquons les différentes méthodes pour publier et partager des animints sur le web.

5

Partage

Traduction de l'anglais Ch05-sharing

Dans ce chapitre, nous expliquons plusieurs méthodes pour partager vos visualisations de données interactives sur le web. Après avoir lu ce chapitre, vous serez en mesure de visualiser des animints :

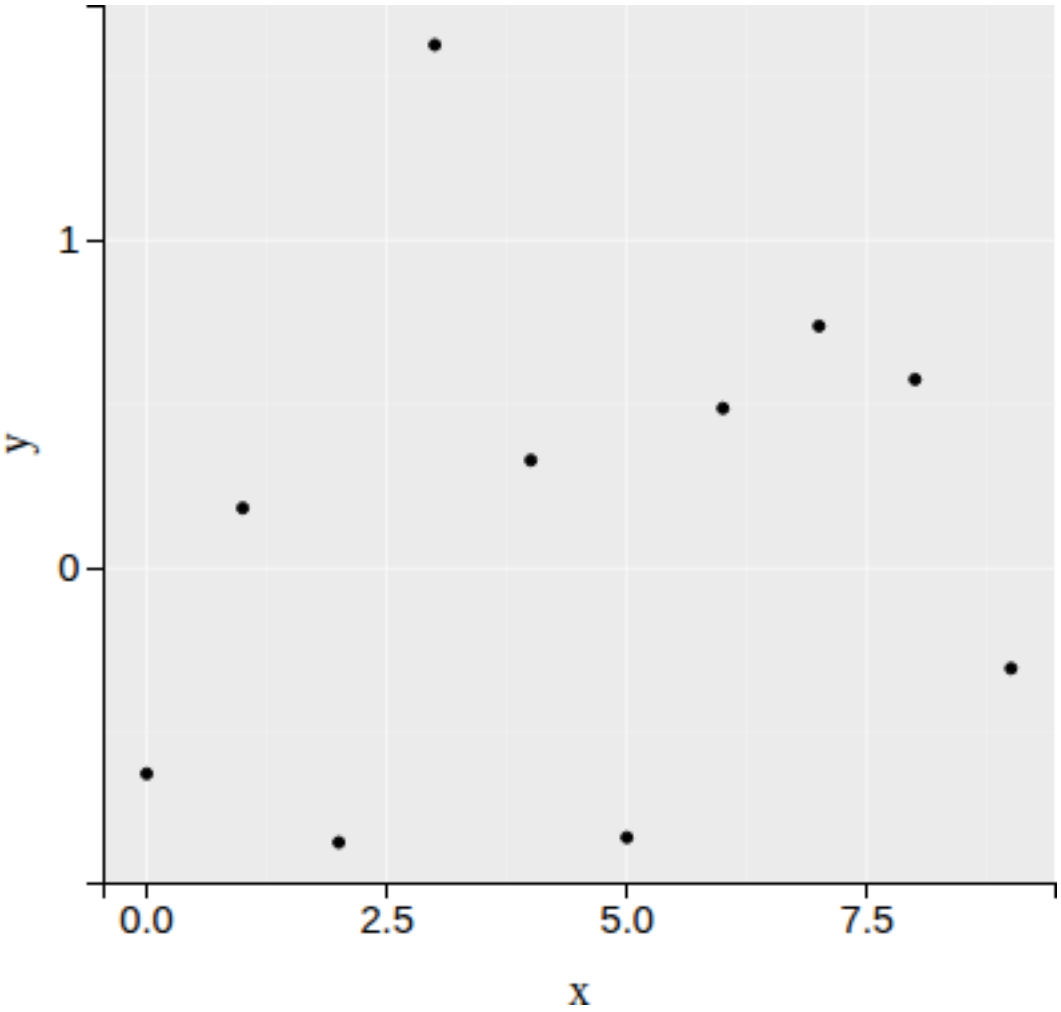
- à partir d'un dossier local sur votre ordinateur personnel;
- dans des documents R Markdown;
- en utilisant n'importe quel serveur web, y compris NetlifyDrop;
- publiés en utilisant GitHub Pages et organisés dans une galerie.

5.1 Compiler un animint dans un dossier local

Lorsque vous testez différentes visualisations de données interactives, il est utile de les prévisualiser sur votre ordinateur personnel avant de les publier sur le web. Cette section traite de deux méthodes pour compiler les animints dans un répertoire local.

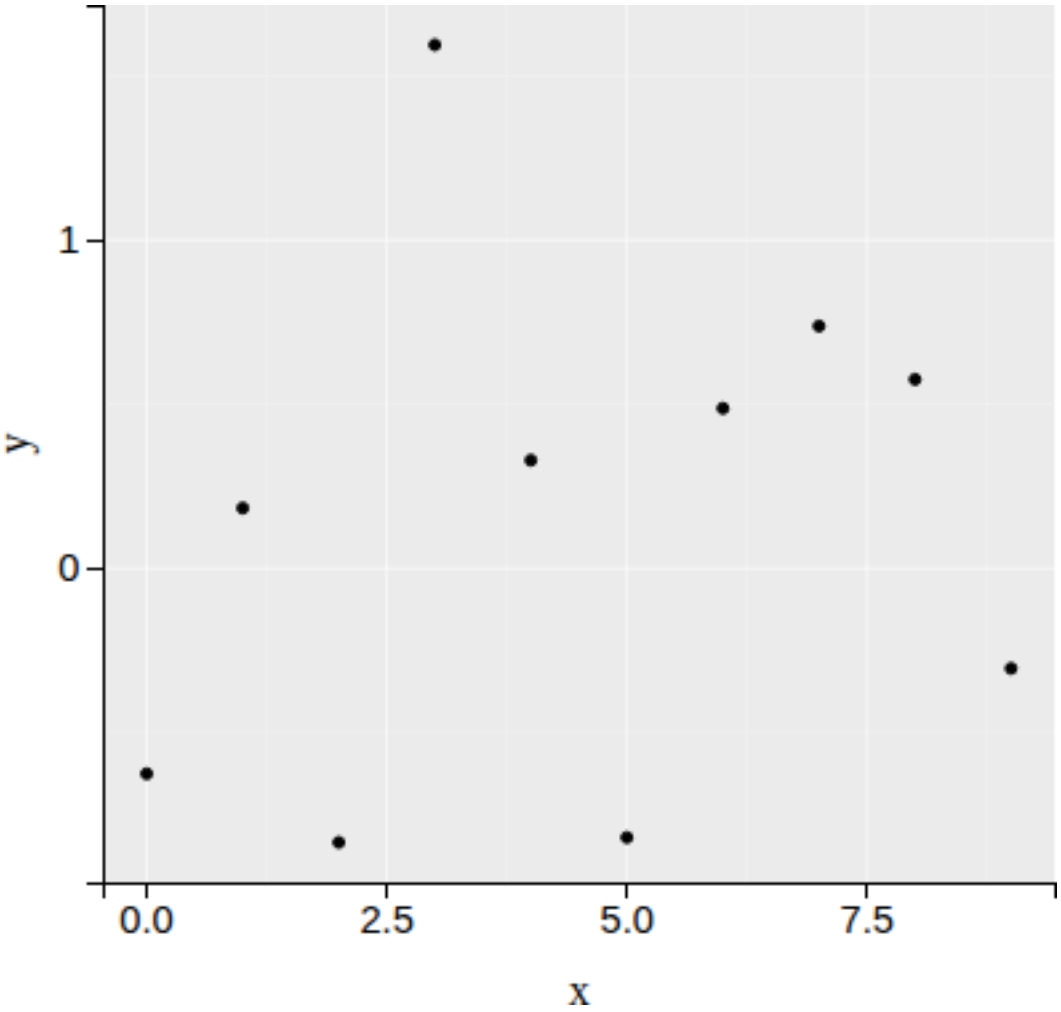
Dans les chapitres précédents, nous n'avons abordé qu'une seule méthode pour créer des visualisations de données interactives. Si `vis` est un animint (liste de ggplots et d'options avec la classe animint), alors `print(vis)`va compiler cet animint dans un dossier temporaire, en utilisant un code comme celui-ci,

```
set.seed(1)
dix.points <- data.frame(x=0:9, y=rnorm(10))
library(animint2)
animint(
  point=ggplot()+
    geom_point(aes(
      x, y),
      data=dix.points))
```



Le code ci-dessus sauvegarde l'animint dans un nouveau dossier temporaire. Plutôt que de sauvegarder chaque animint dans un dossier temporaire séparé, vous pouvez définir un dossier de sortie via l'argument `out.dir` de la commande `animint`. Si vous voulez sauvegarder l'animint dans le fichier `"Ch05-partager-dix-points"` utilisez :

```
animint(
  point=ggplot()+
    geom_point(aes(
      x, y),
      data=dix.points),
  out.dir="Ch05-partager-dix-points")
```



Si le dossier parent d'`out.dir` n'existe pas, vous générerez une erreur (vous pouvez utiliser `dir.create` pour créer le parent si nécessaire). Si `out.dir` n'existe pas, il sera créé. Si `out.dir` existe (et contient un fichier nommé `animint.js`), tous les fichiers de ce répertoire seront écrasés. Pour afficher la visualisation, il suffit d'aller à `Ch05-partager-dix-points/index.html` dans un navigateur web (ce qui devrait être fait automatiquement / par défaut). Si la page web est vide, il se peut que vous deviez configurer votre navigateur pour autoriser l'exécution du code JavaScript local, comme expliqué dans notre FAQ.

En interne, R appelle la méthode S3 `print.animint`, laquelle appelle `animint2dir` afin de compiler la visualisation dans un nouveau dossier temporaire sur votre ordinateur personnel. En général, nous déconseillons d'appeler `animint2dir` directement, mais cela peut être utile si vous voulez éviter d'ouvrir plusieurs fenêtres similaires dans le navigateur lorsque vous révisez un animint. Vous pouvez empêcher l'ouverture par défaut d'une fenêtre du navigateur avec :

```
vis <- animint(
  point=ggplot()+
    geom_point(aes(
```

```
      x, y),
      data=dix.points))
animint2dir(
  vis,
  out.dir="Ch05-partager-dix-points",
  open.browser=FALSE)
```

5.2 Publier en R Markdown

Pour inclure un animint dans un document R Markdown, utilisez `animint(...)` à l'intérieur d'un bloc de code R. R exécutera la méthode S3 `knit_print.animint`, qui compile l'animint dans un dossier local, nommé en fonction du nom du bloc de code R. Par exemple, un bloc de code nommé `vis-facettes` sera sauvegardé dans le dossier `visfacettes`. Veillez à placer chaque animint dans son propre morceau de code (ne mettez pas deux animints dans le même morceau de code).

5.3 Publier sur un serveur web

Comme les animints ne sont que des dossiers contenant des fichiers HTML, TSV et JavaScript, vous pouvez les publier sur n'importe quel serveur web en copiant simplement le dossier sur ce serveur.

Par exemple, j'ai exécuté le code dans https://github.com/animint/animint2/blob/master/inst/examples/WorldBank-facets.R pour créer le `WorldBank-facets` dossier sur mon ordinateur personnel. J'ai copié ce dossier sur le serveur web de mon laboratoire à l'aide de la commande `rsync -r WorldBank-facets/ monsoon.hpc.nau.edu:genomic-ml/WorldBank-facets/` et je peux ainsi le visualiser sur https://rcdata.nau.edu/genomic-ml/WorldBank-facets/

Si vous n'avez pas accès à un serveur web personnel ou par le biais d'un laboratoire, vous pouvez utiliser l'une des méthodes décrites ci-dessous, elles sont gratuites et accessibles à tous.

5.4 Publier sur Netlify Drop

Netlify Drop est un service d'hébergement de sites web statiques. Pour y publier votre visualisation de données, il suffit de faire glisser un dossier sur cette page web (il peut s'agir d'un dossier issu de `animint2dir` ou de `rmarkdown::render` si votre animint est à l'intérieur de Rmd, comme décrit ci-dessus). Une fois le téléversement terminé, vous obtiendrez un lien qui vous permettra de visualiser les fichiers contenus dans ce dossier. Aucun enregistrement ou connexion n'est nécessaire, mais si vous n'avez pas de compte enregistré, vos données seront supprimées après une heure. Vous pouvez souscrire à un compte gratuit si vous

souhaitez que vos vis données soient disponibles pour une plus longue période.

5.5 Publier sur GitHub Pages

GitHub Pages est un service d'hébergement de sites web statiques et peut être utilisé pour publier des animints. Pour publier un animint sur GitHub Pages, vous avez besoin d'un compte GitHub et des packages `gert` (pour exécuter git à partir de R), `gh` (pour utiliser l'API GitHub à partir de R), et `gitcreds` (pour interagir avec le stockage des identifiants de git et faciliter l'authentification lors de l'envoi à GitHub). Tout d'abord, installez ces packages. Si vous n'avez pas de compte GitHub, vous pouvez vous inscrire gratuitement. Ensuite, assurez-vous d'indiquer à R le nom et le courriel à utiliser pour la publication des commits git :

```
gert::git_config_global_set("user.name", "<votre_identifiant>")
gert::git_config_global_set("user.email", "<votre_courriel>")
```

Vous pouvez ensuite utiliser `animint2pages(vis, "nouveau_dépôt")` pour publier votre visualisation dans la branche `gh-pages` de `votre_identifiant_github/nouveau_dépôt` (à noter que `votre_identifiant_github` n'est pas écrit dans le code, puisque `gitcreds` obtiendra cette information à partir du stockage des identifiants de git). Un message devrait s'afficher vous indiquant l'URL ou le lien pour accéder à votre visualisation de données. Il faut compter quelques minutes (généralement pas plus de cinq) entre le moment où vous lancez `animint2pages` et celui où la visualisation est publiée sur GitHub Pages.

Si vous voulez mettre à jour un animint qui a déjà été publié sur GitHub Pages, vous pouvez simplement lancer `animint2pages(nouvelle_version, "dépôt_existant")` qui mettra à jour la branche `gh-pages` du dépôt spécifié.

Attention, GitHub Pages impose une limite de 100MB par fichier. Pour la plupart des visualisations, cette limite ne devrait pas poser de problème. Si votre visualisation de données contient un fichier TSV de plus de 100 Mo, vous pouvez utiliser l'option chunk_vars pour convertir ce grand fichier en plusieurs fichiers plus petits.

5.6 Organiser des animints dans une galerie

Une galerie est une collection de méta-données pour un ensemble de visualisations publiées sur GitHub Pages. Par exemple, la galerie principale des animints https://animint.github.io/gallery/ (en anglais) est une page web contenant des liens vers divers animints, organisés en tableau de contingence. Nous avons aussi une galerie en français.

- Il y a une ligne pour chaque animint.
- La première colonne `viz.link` montre une capture d'écran de l'animint, qui renvoie à la page web de l'animint.
- La deuxième colonne `title` indique le nom de la visualisation (extrait de l'option `title` de chaque animint).
- La troisième colonne `links` contient des liens vers son dépôt sur GitHub, ainsi que son

code source et éventuellement une vidéo.

Une galerie est un dépôt sur GitHub qui doit avoir deux fichiers sources dans la branche `gh-pages` :

- `repos.txt` (liste des dépôts sur GitHub qui contiennent des animints, un propriétaire/dépôt par ligne) et
- `index.Rmd` (source de la page web avec des liens vers animints).

Pour ajouter un nouvel animint à la galerie qui est publiée sur la branche `gh-pages` de `votre_identifiant_github/galerie` vous pouvez utiliser la méthode suivante :

- créez un nouveau animint dans le code R, et assurez-vous de définir les options `source` et `title`;
- utilisez `animint2pages(vis, "dépôt_de_vis")` pour publier cet animint sur la branche `gh-pages` de `votre_identifiant_github/dépôt_de_vis`;
- prenez une capture d'écran de cet `animint2`, et validez/poussez cette capture d'écran dans un fichier nommé `Capture.PNG` (sensible à la casse), dans la branche`gh-pages` de `votre_identifiant_github/dépôt_de_vis` ;
- ajoutez `votre_identifiant_github/dépôt_de_vis` au fichier `repos.txt` dans la branche `gh-pages` de `votre_identifiant_github/galerie`;
- exécutez le code R `animint2::update_gallery("path/to/galerie")` (notez qu'un clone du dossier de la galerie doit être présent sur le système où vous exécutez cette fonction, et que le GitHub distant doit être nommé `origin`). Le code R lira `galerie/repos.txt`, puis les méta-données (`title`, `source`, `Capture.PNG`) de chaque dossier qui n'est pas déjà présent dans `galerie/meta.csv`, il écrira ensuite les fichiers de méta-données mis à jour dans la galerie, affichera les fichiers de `galerieo/index.Rmd` dans `galerie/index.html`, validera les fichiers et les poussera vers `origin`.
- la galerie mise à jour devrait pouvoir être consultée sur le web, à l'adresse `https://votre_identifiant_github.github.io/galerie` après quelques minutes (généralement pas plus de cinq).

5.7 Résumé du chapitre et exercices

Ce chapitre explique comment partager des animints sur le web.

Exercices :

- Créez un animint en utilisant les options mentionnées dans ce chapitre : `out.dir` (nom du dossier où sauvegarder l'animint sur votre ordinateur), `source` (lien vers le code source R utilisé pour créer l'animint), `title` (description de l'animint).
- Utilisez Netlify Drop pour publier cette animation sur le web.
- Utilisez `animint2pages` pour publier cet animint dans un nouveau dépôt GitHub. Faites une capture d'écran et enregistrez la en tant que `Capture.PNG` dans la branche `gh-pages` de ce dépôt. Ajoutez ce dépôt à la galerie principale de l'animint repos.txt en soumettant une Pull Request sur GitHub.
- Créez votre propre galerie animint, et ajoutez y au moins deux de vos propres animints.

Dans le Chapitre 6 nous vous expliquerons les différentes options qui peuvent être utilisées pour personnaliser un animint.

6

Nouveautés

Ce chapitre donne une liste complète des nouvelles fonctionnalités qu'`animint2` introduit dans la grammaire des graphiques (qui n'existe pas dans `ggplot2`). Après avoir lu ce chapitre, vous comprendrez comment personnaliser vos graphiques `animint2` via

- les `aes()` `href`, `tooltip` et `id` pour les caractéristiques propres à l'observation ;
- les éléments nommés de `clickSelects` et `showSelected` pour spécifier plusieurs variables de sélection à la fois ;
- l'option `chunk_vars` spécifique au geom ;
- les paramètres de geom `color_off`, `fill_off`, et `alpha_off` pour spécifier comment l'état de la sélection est affiché;
- les paramètres de geom `help` et `title`, texte qui est affiché dans la visite guidée.
- les légendes et les options de hauteur/largeur propres au graphique
- les options globales.

6.1 Options spécifiques à l'observation (nouveaux `aes()`)

Cette section explique les nouveaux `aes()` reconnus par `animint2`.

6.1.1 Revue des `aes()` introduits précédemment

Nous allons commencer par discuter des nouveaux `aes()` que nous avons déjà introduits dans les chapitres précédents.

Chapitre 3 a également introduit `aes(key)` afin de désigner une variable à utiliser pour des transitions fluides et interprétables.

6.1.2 Hyperliens utilisant `aes(href)`

On peut rajouter un hyperlien avec `aes(href)`. Par exemple, si nous avons une carte des États-Unis, on peut rajouter un lien vers le page Wikipédia de chaque état. On commence par charger les données pour tracer les frontières des états.

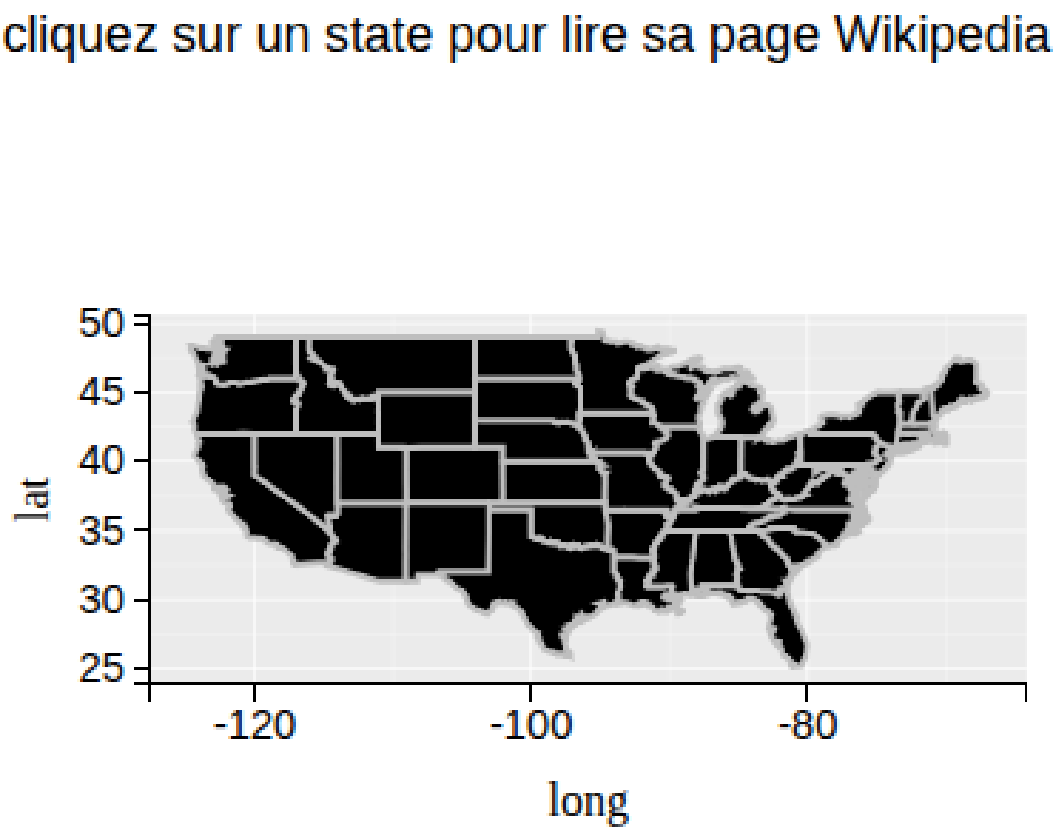
```
library(animint2)
if(!requireNamespace('maps'))install.packages('maps')
```

```
Loading required namespace: maps
```

```
USpolygons <- map_data("state")
```

Notez dans le code ci-dessus qu'il faut installer le package `maps` pour obtenir ces données. Ensuite, on utilise le code ci-dessous pour tracer la carte. On utilise `geom_polygon()` avec `aes(href)` qui donnera des hyperliens vers le page Wikipédia pour chaque état.

```
animint(
  map=ggplot()+
    ggtitle("cliquez sur un state pour lire sa page Wikipedia")+
    coord_equal()+
    geom_polygon(aes(
      x=long, y=lat, group=group,
      href=paste0("http://fr.wikipedia.org/wiki/", region)),
      data=USpolygons, fill="black", colour="grey"))
```



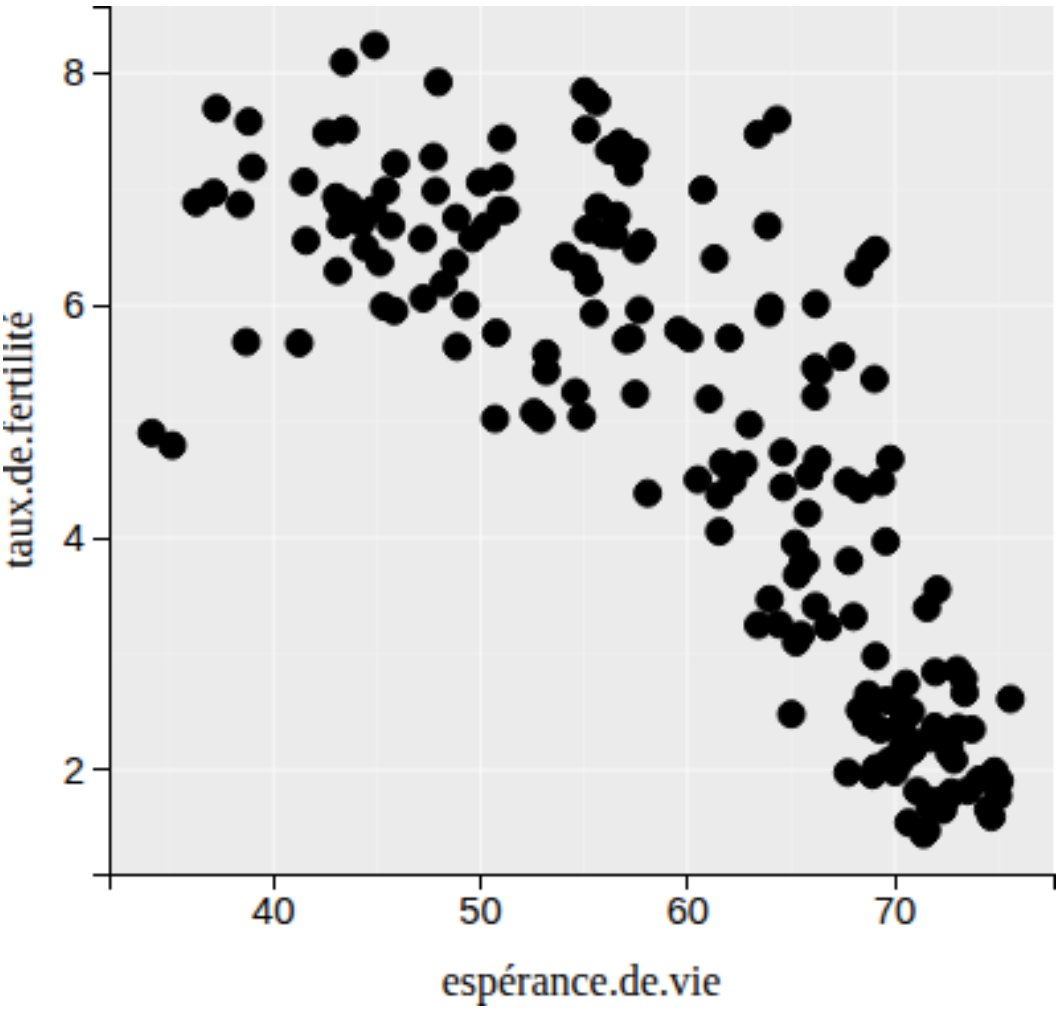
Essayez de cliquer sur un état dans la visualisation de données ci-dessus. Vous devriez voir la page Wikipédia correspondante s'ouvrir dans un nouvel onglet.

6.1.3 Les infobulles utilisant `aes(tooltip)`

Les infobulles sont de petites fenêtres d'information textuelle qui apparaissent lorsque vous survolez un élément à l'écran avec le curseur. Dans `animint()`, vous pouvez utiliser `aes(tooltip)` pour définir le message spécifique à chaque observation qui s'affichera. Par exemple, nous l'utilisons pour afficher la population et le nom du pays dans le nuage de

points des données de la Banque mondiale ci-dessous.

```
data(BanqueMondiale, package="animint2fr")
BanqueMondiale1975 <- subset(BanqueMondiale, année == 1975)
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité,
      tooltip=paste(pays, "population =", population)),
      size=5,
      data=BanqueMondiale1975))
```



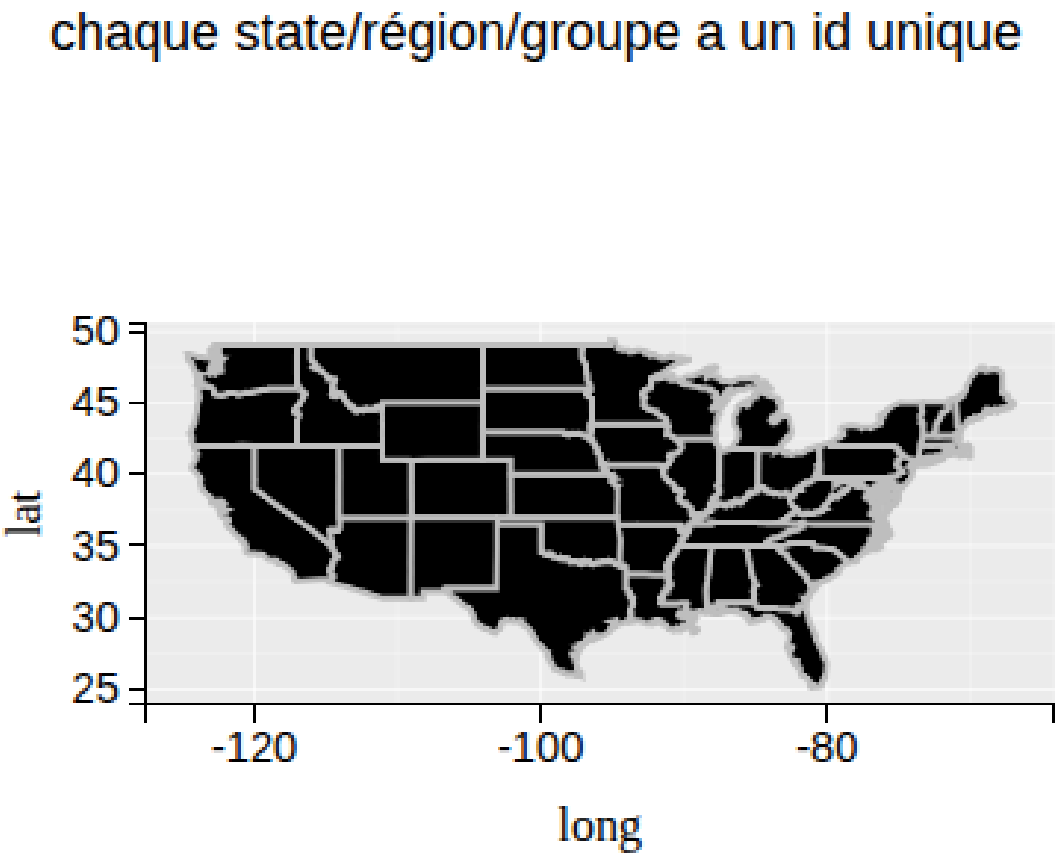
Placez le curseur sur l'un des points de données. Vous devriez voir apparaître une petite boîte contenant le nom du pays et la population pour ce point de données.

Notez qu'une infobulle de la forme "valeur variable" est définie par défaut pour chaque geom avec `aes(clickSelects)`. Par exemple, un geom avec `aes(clickSelects=année)` affiche l'infobulle par défaut "année 1984" pour une observation à l'année 1984. Vous pouvez modifier cette valeur par défaut en spécifiant explicitement `aes(tooltip)`.

6.1.4 Attribut HTML id utilisant `aes(id)`

Puisque tout ce qui est tracé par `animint()` est affiché comme un élément SVG dans une page web, vous pourriez vouloir spécifier un élément attribut HTML id à l'aide de `aes(id)` comme ci-dessous.

```
animint(
  map=ggplot()+
    ggtitle("chaque state/région/groupe a un id unique")+
    coord_equal()+
    geom_polygon(aes(
      x=long, y=lat, group=group,
      id=gsub(" ", "_", paste(region, group))),
      data=USpolygons, fill="black", colour="grey"))
```



Notez que pour qu'un identifiant soit bien défini il ne doit pas contenir d'espaces, nous utilisons donc `gsub` pour convertir les espaces en tirets bas (underscores). Notez également que `paste` est utilisé pour ajouter un numéro de groupe, car il peut y avoir plus d'un polygone par état/région, et chaque identifiant doit être unique sur une page web. Les développeurs d'`animint2` utilisent cette fonctionnalité pour tester le code de l'afficheur JavaScript de l'animint .

6.1.5 Noms de sélecteurs dynamiques basés sur les données en utilisant des `clickSelects` et `showSelected` nommés

Chapitre 3 introduit `showSelected` pour désigner un geom qui n'affiche que le sous-ensemble sélectionné de ses données.

Chapitre 4 introduit `clickSelects` pour désigner un geom sur lequel on peut cliquer pour modifier une variable de sélection.

En général, les noms de sélecteurs sont définis dans `showSelected` ou `clickSelects`. Par exemple, `showSelected=c("année", "pays")` signifie que deux variables de sélection seront créées (nommées `année` et `pays`). Cependant, cette méthode devient peu pratique si vous avez de nombreux sélecteurs dans votre visualisation des données. Considérons par exemple le cas théorique suivant (le code présenté dans cette section n'est pas directement exécutable). Supposons que vous souhaitez utiliser 20 noms de variables de sélecteurs différents, `selector1value`… `selector20value`. La façon habituelle de définir votre visualisation de données serait la suivante

```
vis <- list(
  points=ggplot()+
    geom_point(clickSelects="selector1value", data=data1)+
    ...
<!-- comment -->
geom_point(clickSelects="selector20value", data=data20)
)
```

Cependant, cette méthode est mauvaise car elle viole le principe DRY (Don't Repeat Yourself). Une autre façon de procéder serait d'utiliser une boucle `for`:

```
vis <- list(points=ggplot())
for(selector.name in paste0("selector", 1:20, "value")){
  data.for.selector <- all.data.list[[selector.name]]
<!-- comment -->
vis$points <- vis$points +
    geom_point(clickSelects=selector.name, data=data.for.selector)
}
```

Cette méthode est mauvaise car elle est lente pour construire `vis` et la visualisation compilée prend potentiellement beaucoup d'espace disque puisqu'il y a au moins un fichier TSV créé pour chaque `geom_point`. La méthode à privilégier est de rajouter des noms dans le vecteur utilisé pour `clickSelects` ou `showSelected`. Les noms doivent être utilisés pour indiquer la colonne qui contient le nom de la variable du sélecteur. Par exemple :

```
vis <- list(
  points=ggplot()+
    geom_point(
      clickSelects=c(selector.name="selector.value"),
      data=all.data)
)
```

Le compilateur d'`animint2` parcourt le data.frame `all.data` et crée des sélecteurs pour chacune des valeurs distinctes de `all.data$selector.name`. Lorsqu'on clique sur l'un des

points de données, le sélecteur correspondant est mis à jour avec la valeur indiquée par `all.data$selector.value`.

De la même manière, vous pouvez rajouter des noms au vecteur `showSelected`, au lieu de créer plusieurs geoms, ayant une valeur différente pour `showSelected`.

Cette fonctionnalité évite non seulement les répétitions dans la définition de la visualisation des données, mais elle augmente aussi l'efficacité sur le plan computationnel. Pour un exemple détaillé avec des mesures de temps d'exécution et d'espace disque, voir le chapitre 14 .

6.2 Options du geom

Dans `animint2`, il existe plusieurs options de personalisation pour les geoms. Par exemple, `chunk_vars` permet de spécifier la division des ensembles de données pour leur stockage sur disque. Quant aux paramètres `*_off`, ils sont utilisés pour définir l'affichage d'un geom `clickSelects` lorsqu'il n'est pas sélectionné. De plus, les paramètres `help` et `title` peuvent être définis, pour ajouter des informations à la visite guidée.

6.2.1 Le paramètre de geom `chunk_vars`

Le paramètre `chunk_vars` définit les variables de sélection qui sont utilisées pour diviser l'ensemble de données en morceaux distincts (fichiers TSV) à télécharger. Un fichier TSV est créé pour chaque combinaison de valeurs des variables `chunk_vars`. Plus le nombre de variables de sélection spécifiées dans `chunk_vars` est élevé, plus l'ensemble de données sera divisé en fichiers TSV distincts, chacun ayant une taille plus petite.

L'option `chunk_vars` doit être spécifiée comme argument d'une fonction `geom_*`, et sa valeur doit être un vecteur de caractères contenant le nom des variables de sélection. Lorsque `chunk_vars=character(0)`, soit un vecteur de caractères de longueur zéro, toutes les données sont stockées dans un seul fichier TSV. À l'autre extrême, lorsque `chunk_vars` contient toutes les variables de `showSelected`, un fichier TSV est créé pour chaque combinaison des valeurs de ces variables (beaucoup de fichiers TSV, chacun de petite taille).

En général, le compilateur `animint2` choisit une valeur par défaut raisonnable pour `chunk_vars`, mais vous pouvez spécifier `chunk_vars` si la visualisation des données se charge lentement ou occupe trop d'espace sur le disque. Si la visualisation des données se charge lentement, vous devriez ajouter des variables de sélection à `chunk_vars` afin de réduire la taille du premier fichier TSV à télécharger. Si la visualisation des données occupe trop d'espace sur le disque, vous pouvez retirer les variables de sélection de `chunk_vars` pour réduire le nombre de fichiers TSV. De nombreux petits fichiers TSV peuvent occuper plus d'espace disque qu'un seul fichier TSV, car certains systèmes de fichiers stockent une quantité constante de métadonnées pour chaque fichier.

Pour illustrer l'utilisation de `chunk_vars`, considérez la visualisation suivante de l'ensemble de données `breakpoints`.

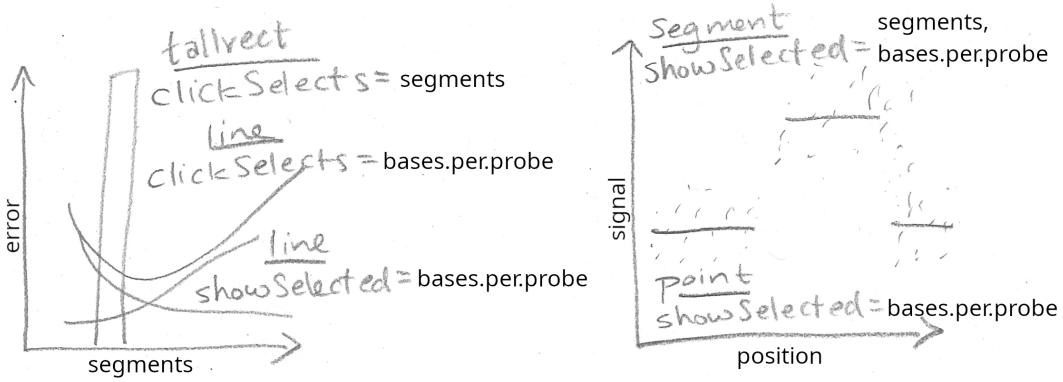
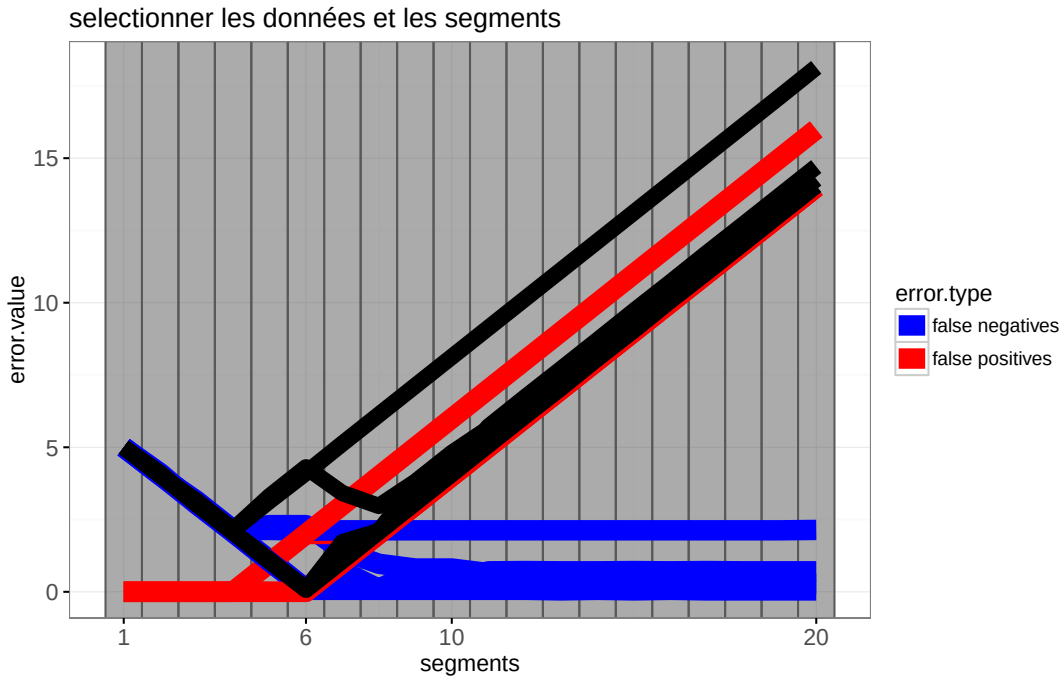


Figure 6.1: Visualisation des données 'breakpoints'

L'esquisse ci-dessus est composée de deux graphiques. Nous commençons par créer le graphique des courbes d'erreurs situé à gauche.

```
data(breakpoints)
only.error <- subset(breakpoints$error, type=="E")
only.segments <- subset(only.error,bases.per.probe==bases.per.probe[1])
library(data.table)
fp.fn.names <- rbind(
  data.table(error.type="false positives", type="FP"),
  data.table(error.type="false negatives", type=c("I", "FN")))
error.dt <- data.table(breakpoints$error)
error.type.dt <- error.dt[fp.fn.names, on=list(type)]
fp.fn.dt <- error.type.dt[, list(
  error.value=sum(error)
), by=.(error.type, segments, bases.per.probe)]
errorPlot <- ggplot()+
  ggtitle("selectionner les données et les segments")+
  theme_bw()+
  geom_tallrect(aes(
    xmin=segments-0.5, xmax=segments+0.5),
    clickSelects="segments",
    data=only.segments,
    alpha=1/2)+
  geom_line(aes(
    segments, error.value, color=error.type,
    group=paste(bases.per.probe, error.type)),
    showSelected="bases.per.probe",
    data=fp.fn.dt,
    size=5)+
  scale_color_manual(values=c(
    "false positives"="red", "false negatives"="blue"))+
  geom_line(aes(
    segments, error, group=bases.per.probe),
    clickSelects="bases.per.probe",
    data=only.error,
```

```
    size=4)+
  scale_x_continuous(breaks=c(1, 6, 10, 20))
errorPlot
```

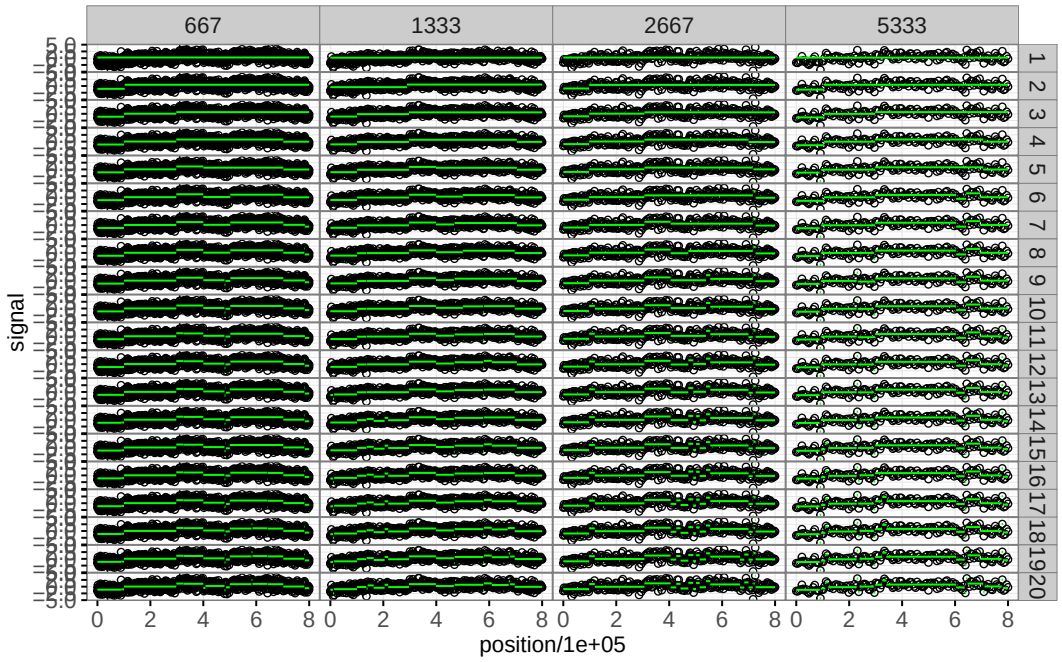


Le graphique ci-dessus comprend un `geom_tallrect` avec `clickSelects=segments` et un `geom_line` avec `clickSelects=bases.per.probe`. Il sera utilisé pour sélectionner les données et le modèle dans le graphique ci-dessous.

```
signalPlot <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  theme_animint(height=800)+
  geom_point(aes(
    position/1e5, signal),
    showSelected="bases.per.probe",
    shape=1,
    data=breakpoints$signals)+
  geom_segment(aes(
    first.base/1e5, mean, xend=last.base/1e5, yend=mean),
    showSelected=c("segments", "bases.per.probe"),
    color="green",
    data=breakpoints$segments)
```

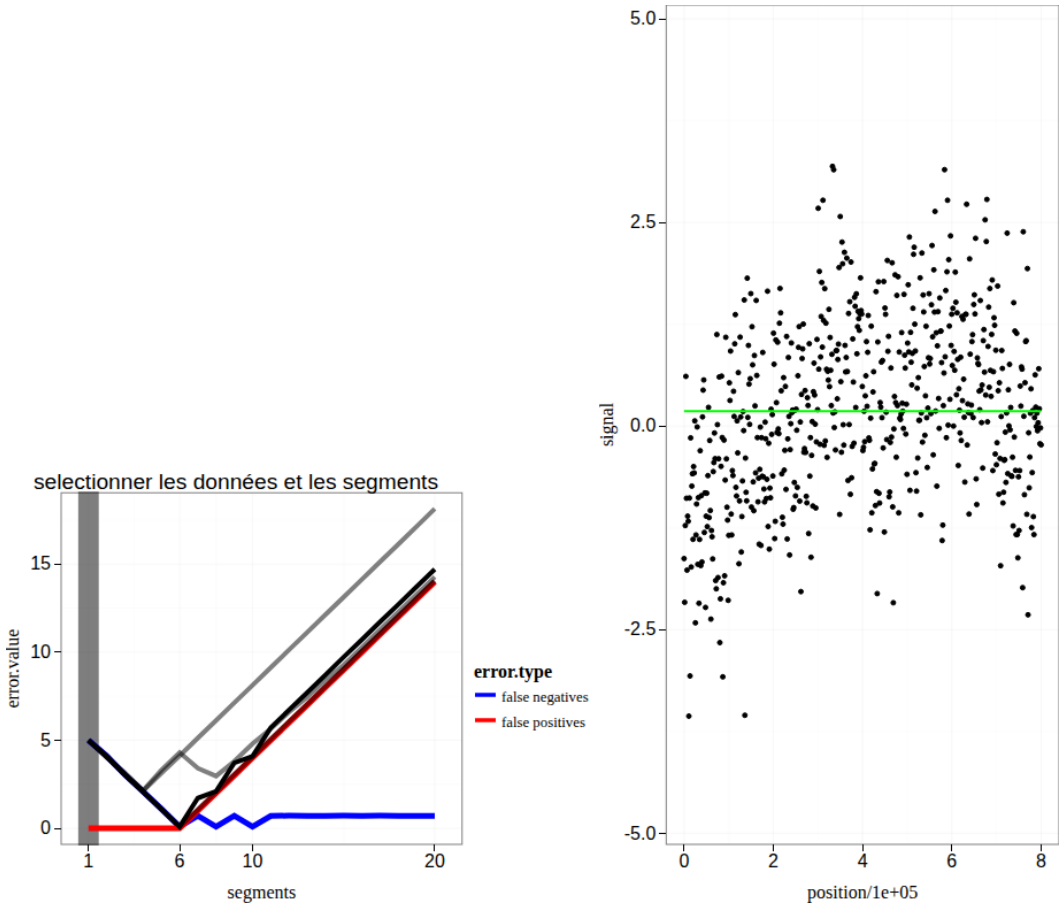
```
Warning in geom_point(aes(position/1e+05, signal), showSelected =
"bases.per.probe", : animint2 web rendering only supports shape=21
```

```
signalPlot+facet_grid(segments ~ bases.per.probe)
```



Le graphique non interactif ci-dessus comporte 80 `facets`, un pour chaque combinaison des deux variables `showSelected`, c'est-à dire `bases.per.probe` et `segments`. Nous présentons ci-dessous une version interactive dans laquelle un seul de ces `facets` sera affiché.

```
(viz.chunk.vars <- animint(
  errorPlot,
  signal=signalPlot+
    geom_vline(aes(
      xintercept=base/1e5),
      showSelected=c("segments", "bases.per.probe"),
      color="green",
      chunk_vars=character(),
      linetype="dashed",
      data=breakpoints$breaks)))
```



Cliquez sur le bouton « Show download status table » , et vous devriez voir des décomptes de chunks (fichiers TSV). Notez que `geom6_vline_signal` n'a qu'un seul morceau, puisque `chunk_vars=character()` est spécifié pour l'élément `geom_vline` dans le code R ci-dessus. Si une autre valeur de `chunk_vars` était spécifiée, cela créerait un nombre différent de fichiers TSV, mais l'apparence de la visualisation de données devrait être la même.

Ci-dessous, nous utilisons le programme en ligne de commande `du` pour déterminer l'utilisation du disque de la visualisation de données pour différents choix de `chunk_vars`.

```
tsvSizes <- function(segment.chunk.vars){
  vis <- list(
    error=errorPlot,
    signal=signalPlot+
      geom_vline(aes(
        xintercept=base/1e5),
        showSelected=c("segments", "bases.per.probe"),
        color="green",
        chunk_vars=segment.chunk.vars,
        linetype="dashed",
        data=breakpoints$breaks)
  )
```

```
    info <- animint2dir(vis, open.browser=FALSE)
    cmd <- paste("du -ks", info$out.dir)
    kb.dt <- fread(cmd=cmd)
    setnames(kb.dt, c("kb", "dir"))
    tsv.vec <- Sys.glob(paste0(info$out.dir, "/*.tsv"))
    is.geom6 <- grepl("geom6", tsv.vec)
    data.frame(
      kb=kb.dt$kb, geom6.tsv=sum(is.geom6), other.tsv=sum(!is.geom6))
  }
  chunk_vars_list <- list(
    neither=c(),
    bases.per.probe=c("bases.per.probe"),
    segments=c("segments"),
    both=c("segments", "bases.per.probe"))
  sizes.list <- lapply(chunk_vars_list, tsvSizes)
  (sizes <- do.call(rbind, sizes.list))
```

```
                 kb geom6.tsv other.tsv
neither         844         1        12
bases.per.probe 848         5        12
segments        904        19        12
both           1132        76        12
```

Le tableau ci-dessus indique le nombre de kilo-octets pour la visualisation des données, ainsi que le nombre de fichiers TSV pour `geom6_vline_signal` et les autres geoms. Notez comment le choix de `chunk_vars` affecte le nombre de fichiers TSV et l'utilisation de l'espace disque. Depuis `chunk_vars` n'a été spécifié que pour les `geom6_vline_signal` le nombre de fichiers TSV pour les autres geoms ne change pas. Lorsque `segments` et `bases.per.probe` sont tous les deux spécifiés pour `chunk_vars`, il y a 76 fichiers TSV pour `geom6_vline_signal`, et la visualisation des données occupe 1132 kilo-octets. En revanche, `chunk_vars=character()` ne produit qu'un seul fichier TSV pour `geom6_vline_signal`, et la visualisation des données occupe 844 kilo-octets.

En conclusion, l'option spécifique au geom `chunk_vars` détermine le nombre de fichiers TSV créés pour chaque geom. Lors du choix de la valeur de `chunk_vars`, vous devez tenir compte à la fois l'utilisation du disque et le temps de chargement. Un petit nombre de gros fichiers occupent moins d'espace disque, mais sont plus lents à télécharger que de nombreux petits fichiers.

6.2.2 Préciser le mode d'affichage de l'état de la sélection

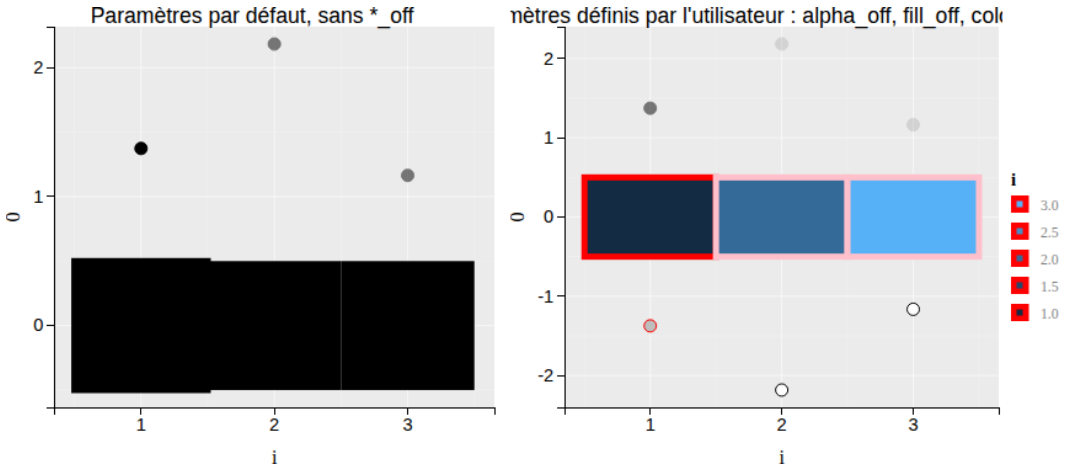
`Animint2` a des valeurs par défaut raisonnables pour l'affichage de l'état de sélection. En particulier,

- lorsqu'il y a un `rect` ou un `tile` avec `clickSelects`, nous utilisons le noir pour `color/border` pour montrer les éléments qui sont sélectionnés, et le transparent pour les éléments qui ne sont pas sélectionnés.
- pour tout autre geom avec `clickSelects`, nous utilisons une opacité complète `alpha` pour afficher les éléments sélectionnés, et une opacité réduite `alpha-0.5` pour les éléments non sélectionnés.

Les valeurs par défaut expliquées ci-dessus sont illustrées dans le premier graphique ci-dessous.

Ces valeurs par défaut peuvent être personnalisées en utilisant les paramètres du geom `alpha_off`, `fill_off` et `color_off` comme dans le code ci-dessous,

```
N <- 3
set.seed(1)
demo_df <- data.frame(i=1:N, num=rnorm(N,2))
animint(
  defaults=ggplot()+
    ggtitle("Paramètres par défaut, sans *_off")+
    geom_tile(aes(
      i, 0),
      size=5,
      clickSelects="i",
      data=demo_df)+
    geom_point(aes(
      i, num),
      size=5,
      clickSelects="i",
      data=demo_df),
  off=ggplot()+
    ggtitle("Paramètres définis par l'utilisateur  : alpha_off, fill_off, color_off")+
    geom_tile(aes(
      i, 0, fill=i),
      clickSelects="i",
      color="red",
      color_off="pink",
      size=5,
      data=demo_df)+
    geom_point(aes(
      i, num),
      size=5,
      alpha=0.5,
      alpha_off=0.1,
      clickSelects="i",
      data=demo_df)+
    geom_point(aes(
      i, -num),
      size=5,
      alpha=1,
      alpha_off=1,
      color="red",
      color_off="black",
      fill="grey",
      fill_off="white",
      clickSelects="i",
      data=demo_df))
```



Notez que lorsque vous utilisez l'une de ces propriétés visuelles dans le mapping `aes`, elle ne doit pas être spécifiée en tant que paramètre geom. Par exemple, dans le tile ci-dessus, nous avons utilisé `aes(fill)`, donc `fill` et `fill_off` ne doivent pas être spécifiés en tant que paramètres pour ce geom (afin qu'il soit clair que `fill` est utilisé pour afficher les valeurs des données, et non l'état de sélection).

6.2.3 Spécification du texte de la visite guidée

Depuis janvier 2025, `animint2` prend en charge une visite guidée, qui affiche des informations sur les interactions possibles avec chaque geom. Pour personnaliser ce qui est affiché pour chaque geom, vous pouvez spécifier les paramètres `help` et `title`, comme dans le code ci-dessous.

```
animint(
  scatter=ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région),
      size=5,
      showSelected="année",
      clickSelects="pays",
      help="Un point est tracé pour chaque pays dans l'année sélectionnée",
      alpha=0.7,
      data=BanqueMondiale)+
    geom_text(aes(
      x=espérance.de.vie, y=taux.de.fertilité, label=pays),
      data=BanqueMondiale,
      title="Pays sélectionné",
      showSelected=c("année","pays")),
  first=list(
    pays="France",
    année=1980))
```

Dans le code ci-dessus, nous spécifions `help` pour `geom_point`, qui contrôle le sous-texte pour ce geom après avoir cliqué sur le bouton "Start Tour" au bas de la visualisation de données. Après avoir cliqué sur le bouton "Next", nous pouvons voir le `title` qui a été spécifié dans le code, affiché en haut de la fenêtre de visite, pour le `geom_text`. Ce mécanisme peut être utilisé pour fournir des informations supplémentaires utiles aux utilisateurs de votre visualisation de données, afin qu'ils puissent comprendre plus facilement ce qui est affiché et quelles interactions sont possibles.

6.3 Options spécifiques au graphique

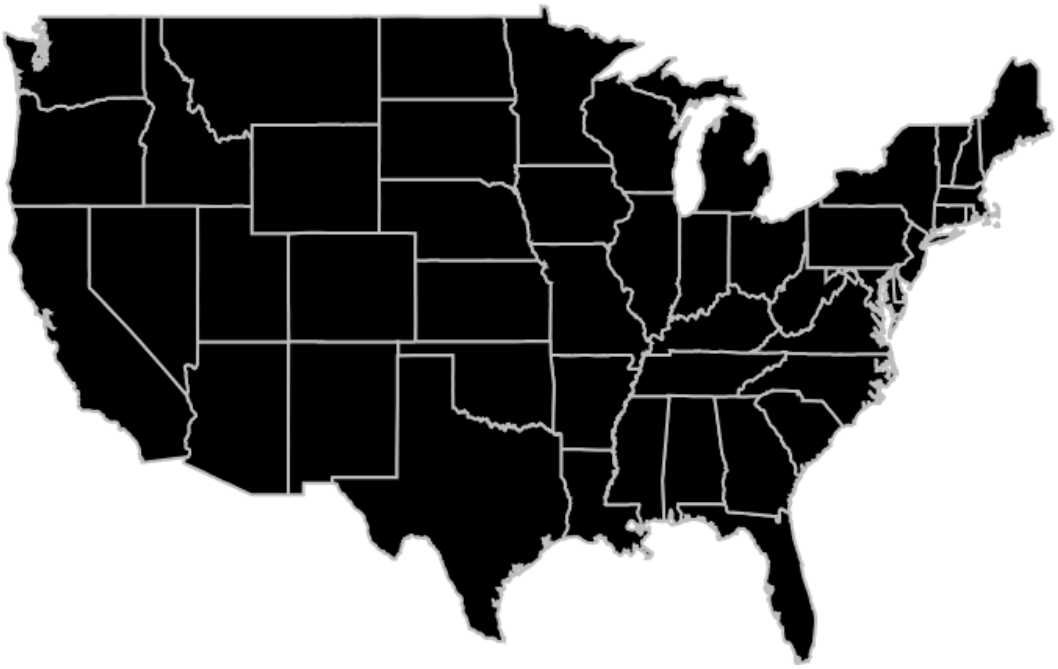
Cette section traite des options qui sont spécifiques à un ggplot d'une visualisation de données. La fonction `theme_animint` est utilisée pour attacher des options `animint2` aux objets ggplot.

6.3.1 Hauteur et largeur du graphique

Les options `width` et `height` permettent de spécifier les dimensions (en pixels) d'un ggplot affiché par `animint()`. Par exemple, considérons le remaniement suivant du graphique des États-Unis :

```
animint(
  map=ggplot()+
    theme_animint(width=750, height=500)+
    theme(
      axis.line=element_blank(),
```

```
      axis.text=element_blank(),
      axis.ticks=element_blank(),
      axis.title=element_blank(),
      panel.border=element_blank(),
      panel.background=element_blank(),
      panel.grid.major=element_blank(),
      panel.grid.minor=element_blank())+
    geom_polygon(aes(
      x=long, y=lat, group=group),
      data=USpolygons, fill="black", colour="grey"))
```



Notez que le graphique ci-dessus a été généré avec une largeur de 750 pixels et une hauteur de 500 pixels, en raison des options `theme_animint`. Si l'une de ces options n'est pas définie pour un ggplot, `animint()` utilise une valeur par défaut de 400 pixels.

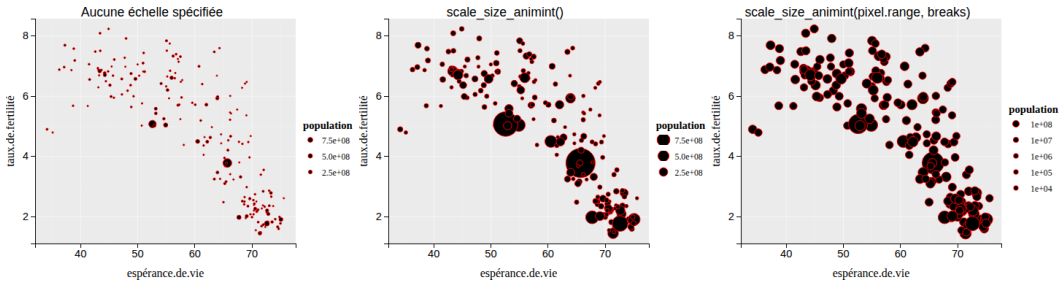
Notez également que `theme` a été utilisé pour spécifier plusieurs éléments vides. Cela a pour effet de supprimer les axes et l'arrière-plan, et est généralement utile pour générer des cartes.

6.3.2 Échelle de taille en pixels

L'échelle `scale_size_animint` doit être utilisée dans tous les ggplots où vous précisez `aes(size)`. Pour comprendre pourquoi, consultez les exemples suivants.

```
nuage1975 <- ggplot()+
  geom_point(
    aes(x=espérance.de.vie, y=taux.de.fertilité, size=population),
    BanqueMondiale1975,
    shape=21,
```

```
    color="red",
    fill="black")
(viz.scale.size <- animint(
  ggplotDefault=nuage1975+
    ggtitle("Aucune échelle spécifiée"),
  animintDefault=nuage1975+
    ggtitle("scale_size_animint()")+
    scale_size_animint(),
  animintOptions=nuage1975+
    ggtitle("scale_size_animint(pixel.range, breaks)")+
    scale_size_animint(pixel.range=c(5, 15), breaks=10^(10:1))))
```



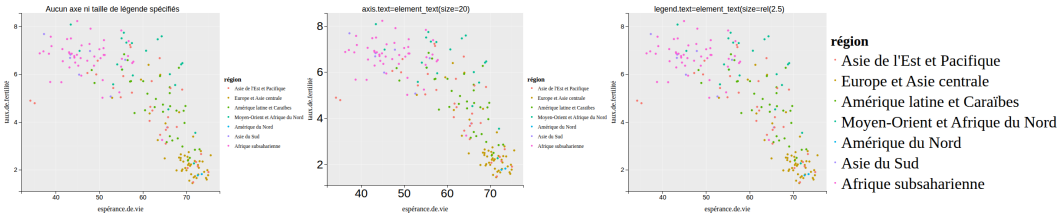
Le premier ggplot ci-dessus n'a pas d'échelle définie, il utilise donc l'échelle par défaut de ggplot2, posant ainsi deux problèmes. Le premier problème est qu'il semble que tous les pays aient à peu près la même taille, à l'exception des deux plus grands pays. Ce problème peut être résolu en ajoutant simplement `scale_size_animint()` au ggplot, ce qui donne le deuxième graphique ci-dessus. Cependant, un second problème est que les entrées de la légende ne montrent pas toute l'étendue des données. Ce problème est résolu dans le troisième graphique ci-dessus, en spécifiant manuellement l'option `breaks` à utiliser pour les entrées de légende. Notez que l'argument `pixel.range` peut également être utilisé pour définir le rayon du plus grand et du plus petit cercle.

6.3.3 Taille du texte des axes et de la légende

La syntaxe de définition des axes et de la taille du texte de la légende(en pixels) est quasiment la même que celle de `ggplot2`. À l'intérieur de `theme`, vous pouvez directement utiliser des nombres pour modifier la taille de la police, ou vous pouvez utiliser `rel()` pour définir la taille relative.

```
nuage1975 <- ggplot()+
  geom_point(aes(
    x=espérance.de.vie, y=taux.de.fertilité, color=région),
    data=BanqueMondiale1975)
(viz.text.size <- animint(
  animintDefault=nuage1975+
    theme_animint(width=500, height=500)+
    ggtitle("Aucun axe ni taille de légende spécifiés"),
  animintAxesOptions=nuage1975+
    theme_animint(width=500, height=500)+
    theme(axis.text=element_text(size=20))+
```

```
    ggtitle("axis.text=element_text(size=20)"),
  animintLegendOptions=nuage1975+
    theme_animint(width=500, height=500)+
    theme(
      legend.title=element_text(size=24),
      legend.text=element_text(size=rel(2.5)))+
    ggtitle("legend.text=element_text(size=rel(2.5)")))
```



Cela vous permet de modifier la taille de la police tout en ajustant la taille du graphique, afin d'obtenir un rendu plus cohérent.

Notez que la taille de police par défaut dans `animint2` est de 11px pour les axes et de 16px pour la légende.

6.4 Options globales de la visualisation des données

Les options globales de la visualisation des données sont tous les éléments nommés de la liste `vis` qui ne sont pas des ggplots.

6.4.1 Révision des options globales introduites précédemment

Chapitre 3 a introduit l'option `duration` qui permet de spécifier la durée des transitions en douceur.

Chapitre 3 a introduit l'option `time` qui permet de spécifier une variable de sélection qui est automatiquement mise à jour (animation).

Chapitre 4 a introduit l'option `first` qui permet de spécifier la sélection lorsque la visualisation des données est générée pour la première fois.

Chapitre 4 a introduit l'option `selector.types` qui permet de spécifier plusieurs variables de sélection.

6.4.2 Titre de la page web avec l'option `title`

L'option `title` doit être une chaîne de caractères, et sera utilisée pour définir l'élément `\<title>` de la page web. Il n'est pas utile d'utiliser l'option `title` dans un document Rmd tel que cette page. Un titre peut et doit être utilisé avec `animint2dir` comme dans le code ci-dessous.

```
vis.title <- viz.scale.size
vis.title$title <- "Plusieurs échelles de taille"
animint2dir(vis.title, "Ch06-viz-title")
```

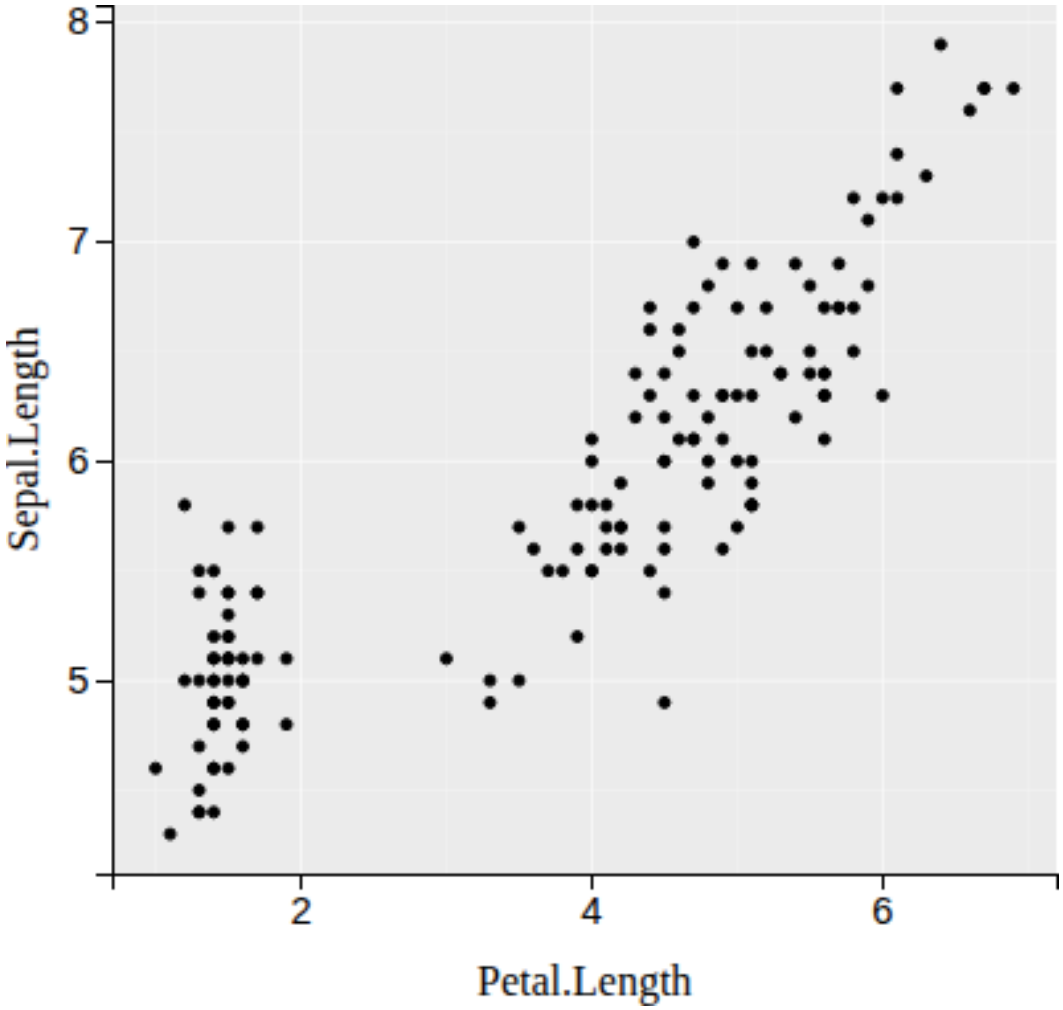
Notez que `viz.scale.size` possède déjà trois ggplots, chacun avec un `ggtitle`. Ajouter l'option globale `title` a pour effet de définir un titre pour la page web .

Le chapitre 5 a introduit la fonction animint2pages qui permet de publier un `animint2` sur GitHub Pages. Il faut que l'`animint2` définisse l'option `title` car ces méta-données sont nécessaires à l'organisation de l'`animint2` dans une galerie .

6.4.3 Lier le code R avec l'option source

L'option `source` doit être une chaîne de caractères : un lien vers le code source R qui a été utilisé pour créer l'`animint2`.

```
animint(
  demo=ggplot()+
    geom_point(aes(
      Petal.Length, Sepal.Length),
      data=iris),
  source=paste0(
    "https://github.com/animint/animint-manual-fr",
    "/blob/main/Chapitres/Ch06/Ch06_source.Rmd"))
```



Notez ci-dessus comment il y a un lien source au bas de la visualisation des données.

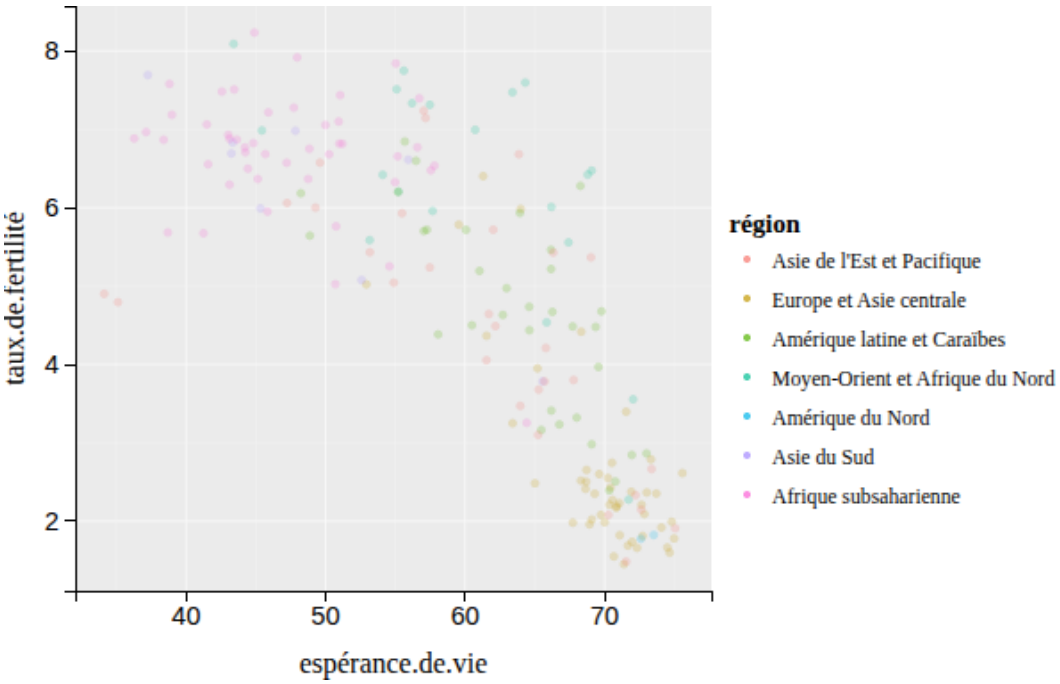
Le chapitre 5 a introduit la fonction animint2pages qui permet de publier un `animint2` sur GitHub Pages. Il faut que l'`animint2` définisse l'option `source` car ces méta-données sont nécessaires à l'organisation de l'`animint2` dans une galerie .

6.4.4 Lier une vidéo

L'option `video` doit être une chaîne de caractères : un lien vers une vidéo qui montre des interactions typiques avec l'`animint2`. Ce mécanisme peut être utilisé pour aider les utilisateurs de votre visualisation de données à comprendre ce qui est affiché, et quelles interactions ils peuvent utiliser.

```
animint(
  video="https://vimeo.com/1050117030",
  scatter=ggplot()+
    geom_point(aes(
      x=espérance.de.vie, y=taux.de.fertilité, color=région),
```

```
        clickSelects="pays",
        alpha=0.7,
        data=BanqueMondiale1975))
```



Dans la visualisation de données ci-dessus, remarquez le lien “vidéo” qui apparaît en bas à droite. En cliquant sur ce lien, on accède à une vidéo qui a été enregistrée pour expliquer une visualisation de données plus complexe basée sur les données de la Banque Mondiale. L'idée est que vous pouvez enregistrer une vidéo pour chacun de vos animints, puis inclure un lien vers cette vidéo à l'aide de ce mécanisme, afin que vos utilisateurs puissent comprendre plus facilement ce qui est affiché et quelles interactions sont possibles.

6.4.5 Afficher ou masquer les menus de sélection avec l'option `selectize`

L'option `selectize` doit être une liste nommée de valeurs booléennes. Les noms doivent être des variables de sélecteur, et les valeurs doivent indiquer si vous souhaitez ou non générer un menu de sélection via selectize.js. Par défaut, `animint2` va générer un menu de sélection pour chaque variable de sélection, à deux exceptions près :

- les variables de sélection basées sur les données qui sont définies à l'aide de namedclickSelects/showSelectedvariables .
- les variables de sélection qui ont beaucoup de valeurs (elles sont lentes à générer).

Ces valeurs par défaut devraient convenir à la grande majorité des animints. Pour ceux qui souhaitent voir un exemple du fonctionnement de l'option `selectize`, veuillez consulter PredictedPeaks dans le code source de `animint2` .

6.5 Résumé du chapitre et exercices

Ce chapitre a expliqué plusieurs options de personnalisation des animints au niveau de l'observation, du geom, du graphique et au niveau global.

Exercices :

- Créez d'autres versions de `viz.chunk.vars` avec différentes valeurs de `chunk_vars` pour les `geom_point` et `geom_segment`. Comment le choix des `chunk_vars` affecte-t-il l'apparence de la visualisation ? L'espace disque ? Le temps de chargement ?

Ensuite, Chapitre 7 explique les limites de l'implémentation actuelle de `animint2`.

7

Limitations

Ce chapitre expose les limites d'`animint2` pour certaines tâches de visualisation de données interactives et propose quelques solutions de contournement à utiliser dans ces situations. Après avoir lu ce chapitre, vous saurez comment

- Utiliser une variable normalisée lorsque différents sous-ensembles `showSelected` ont des valeurs très différentes pour une variable que vous souhaitez afficher.
- Calculer les statistiques pour chaque sous-ensemble `showSelected` au lieu d'utiliser les fonctions `stat_*` de `ggplot2`.
- Ajouter des données une par une à un ensemble de variables à sélection multiple, plutôt que d'utiliser un pinceau de sélection rectangulaire.
- Évite l'utilisation de `vjust` et d'étiquettes à lignes plusieurs lignes dans `geom_text`.
- Ordonner les graphiques sur la page.
- Éviter l'utilisation des options `theme` non supportées.
- Utiliser des facettes avec plusieurs variables par axe.
- Utiliser le package de serveur web `shiny` pour modifier de manière interactive la cartographie esthétique d'un `animint`, ou pour effectuer des calculs basés sur des valeurs sélectionnées.

Toute idée visant à améliorer `animint2` et contourner l'une de ces limites est la bienvenue. Les développeurs d'`animint2` seraient plus qu'heureux d'accepter votre Pull Request .

7.1 Utiliser des variables normalisées pour travailler avec des échelles standardisées

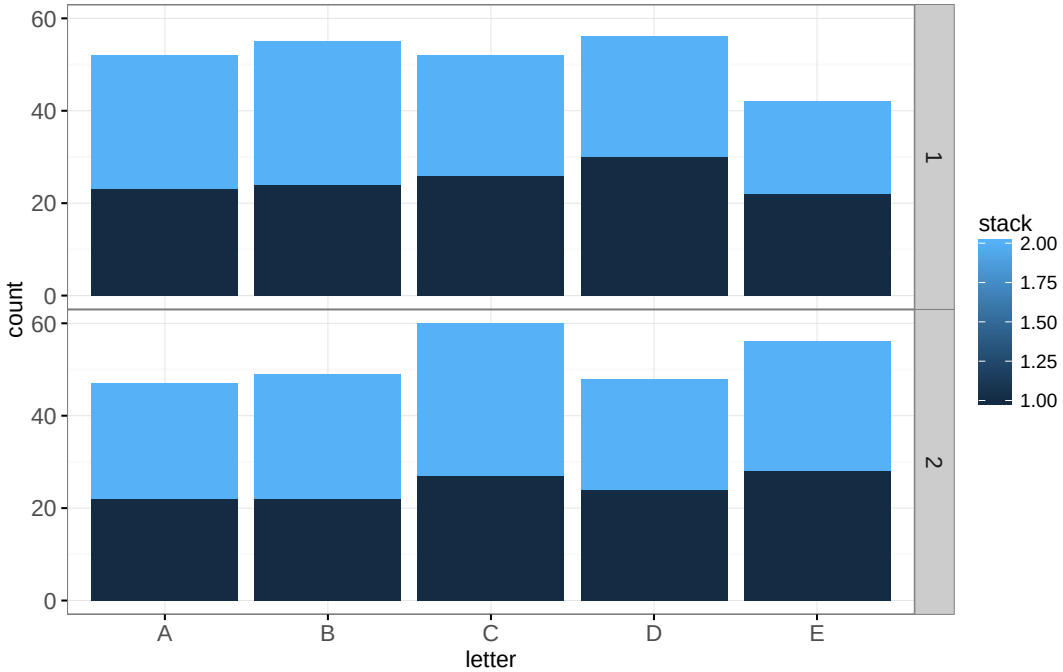
Nous implémentons les axes et les légendes de la même manière que dans `ggplot2` : en les calculant une seule fois lors de la première génération du graphique. Par conséquent, les axes et les légendes de chaque graphique animint ne sont pas interactifs. Pour la plupart des visualisations de données animées les axes fixes permettent de comprendre facilement comment les données changent en même temps que la variable `time`.

Dans certaines situations, il serait utile d'avoir des axes qui se mettent à jour de manière interactive, par exemple lorsque différents sous-ensembles `showSelected` ont des valeurs très différentes pour les variables qui sont affichées avec un axe ou une légende. Dans ce cas, il serait utile d'avoir des axes interactifs qui se mettent à jour et changent en même temps que les données. Nous disposons d'une prise en charge expérimentale pour cela, voir la mise à jour axestest pour plus de détails.

7.2 Calcul des statistiques pour chaque sous-ensemble showSeleted (ou réalisation d'un sous-ensemble).

`Animint` ne prend pas en charge les statistiques `ggplot2` avec `showSelected`. Par exemple, considérons le ggplot à facettes ci-dessous.

```
set.seed(1)
library(data.table)
random.counts <- data.table(
  letter = c(replicate(4, LETTERS[1:5])),
  count = c(replicate(4, rbinom(5, 50, 0.5))),
  stack = rep(rep(1:2, each = 5), 2),
  facet = rep(1:2, each = 10))
library(animint2)
ggstat <- ggplot() +
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_bar(
      aes(letter, count, fill = stack),
      showSelected="facet",
    data = random.counts,
    stat = "identity",
    position="stack"
  )
ggstat+facet_grid(facet ~ .)
```



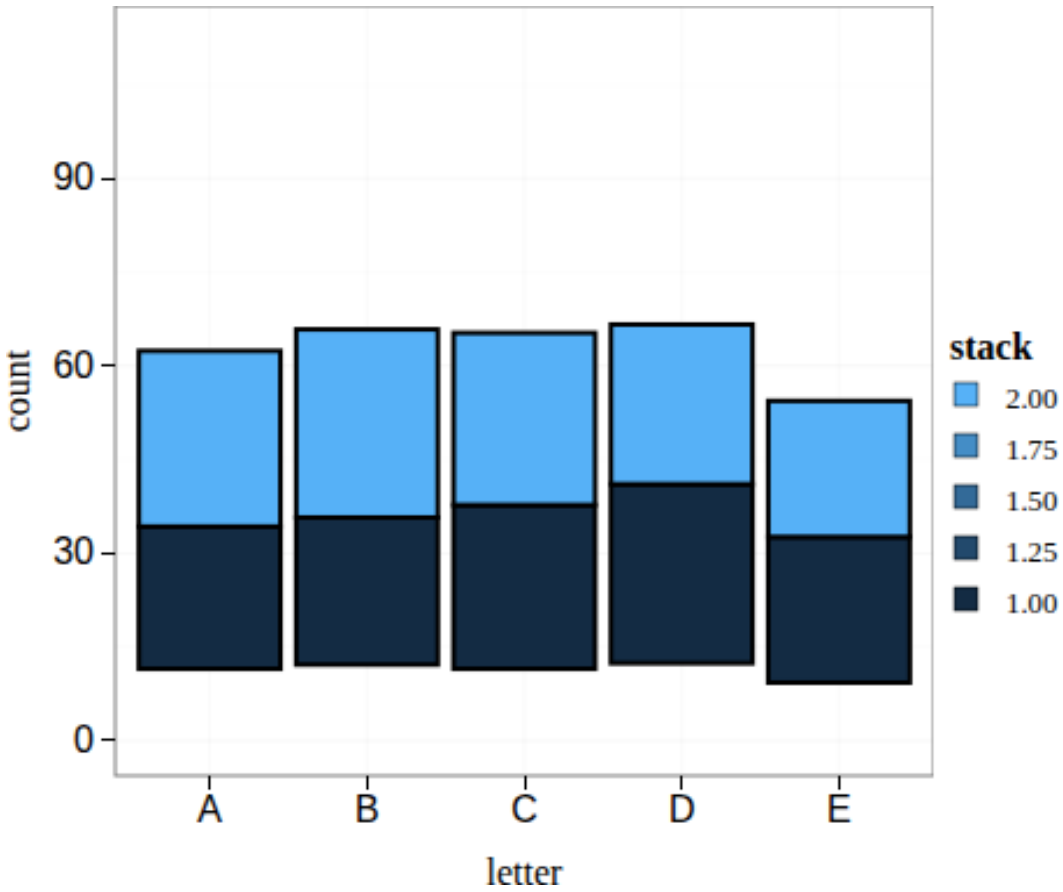
Utiliser `showSelected` au lieu des facettes ne donne pas le résultat escompté.

```
  animint(
    plot = ggstat,
    time = list(variable = "facet", ms = 1000),
    duration = list(facet = 1000))
```

```
mapping: x = letter
y = count
fill = stack
showSelected1 = facet
geom_bar: width = NULL
na.rm = FALSE
stat_identity: na.rm = FALSE
position_stack
```

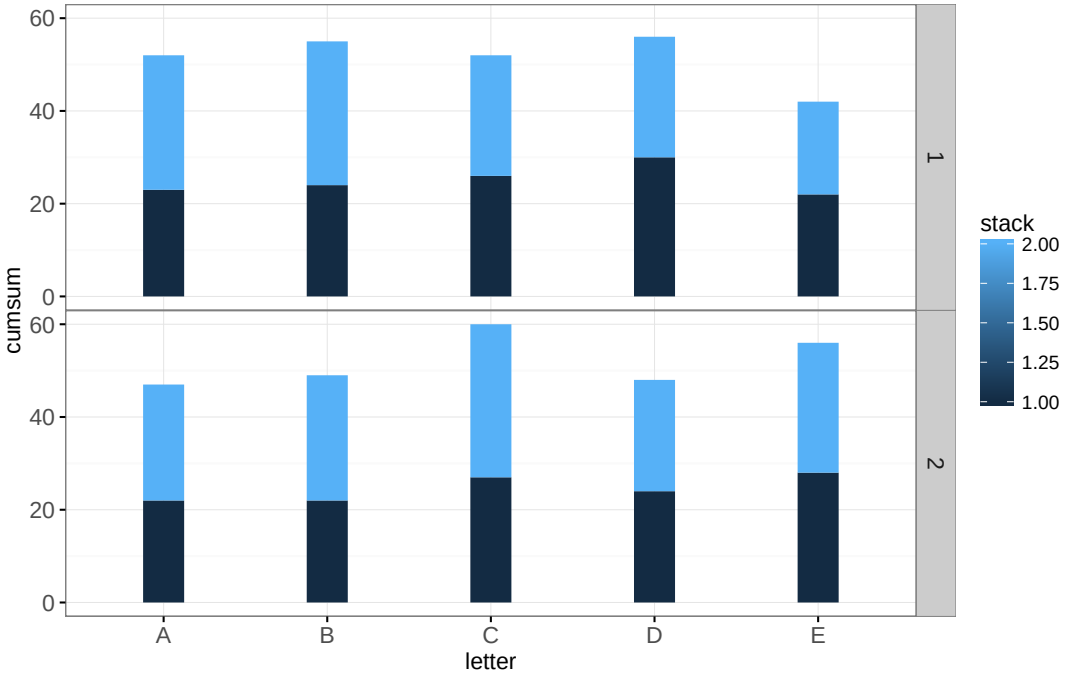
```
Warning in f(...): showSelected only works with position=identity, problem:
geom1_bar_plot
```

```
Warning in issueSelectorWarnings(meta$geoms, meta$selector.aes, meta$duration):
to ensure that smooth transitions are interpretable, aes(key) should be
specifed for geoms with showSelected=facet, problem: geom1_bar_plot
```



Une solution de contournement consiste à calculer ce que vous voulez afficher, puis à utiliser `position=identity`.

```
random.cumsums <- random.counts[, list(
  cumsum=cumsum(count),
  count=count,
  stack=stack),
  by=.(letter, facet)]
ggidentity <- ggplot() +
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_segment(aes(
    x=letter, xend=letter,
    y=cumsum, yend=cumsum-count,
    color=stack),
    showSelected="facet",
    data = random.cumsums,
    size=10,
    stat = "identity",
    position="identity")
ggidentity+facet_grid(facet ~ .)
```

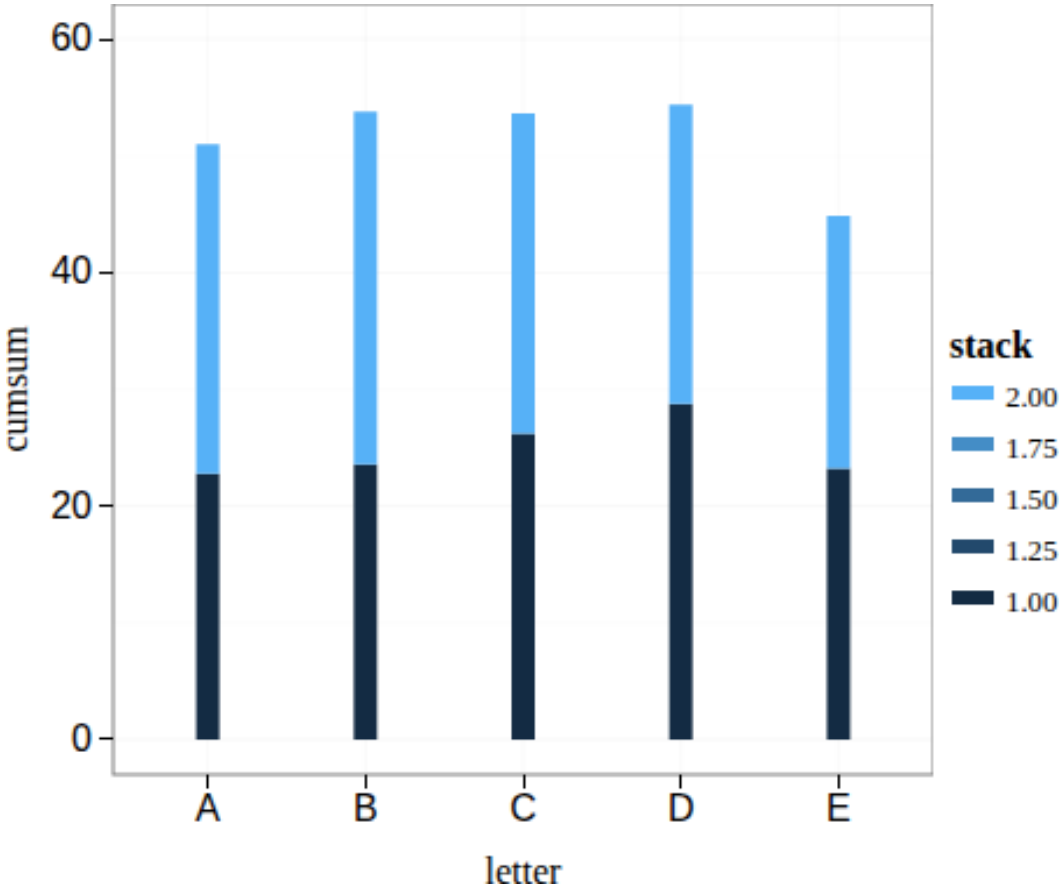


Notez que nous avons utilisé `geom_segment` au lieu de `geom_bar` mais que l'apparence des facettes reste inchangée.

```
animint(
  plot = ggidentity,
  time = list(variable = "facet", ms = 1000),
  duration = list(facet = 1000))
```

```
Warning in issueSelectorWarnings(meta$geoms, meta$selector.aes, meta$duration):
```

```
to ensure that smooth transitions are interpretable, aes(key) should be
specifed for geoms with showSelected=facet, problem: geom1_segment_plot
```



7.3 Ajouter des valeurs à un ensemble de sélection multiple, une à la fois

`animint2` ne prend pas en charge la brosse rectangulaire ou le lasso pour définir interactivement un ensemble de valeurs prises en charge. Au lieu de cela, `animint` prend en charge la sélection multiple en ajoutant des valeurs une par une à l'ensemble de sélection multiple. Il faut donc utiliser l'option selector.types pour déclarer une variable de sélection multiple.

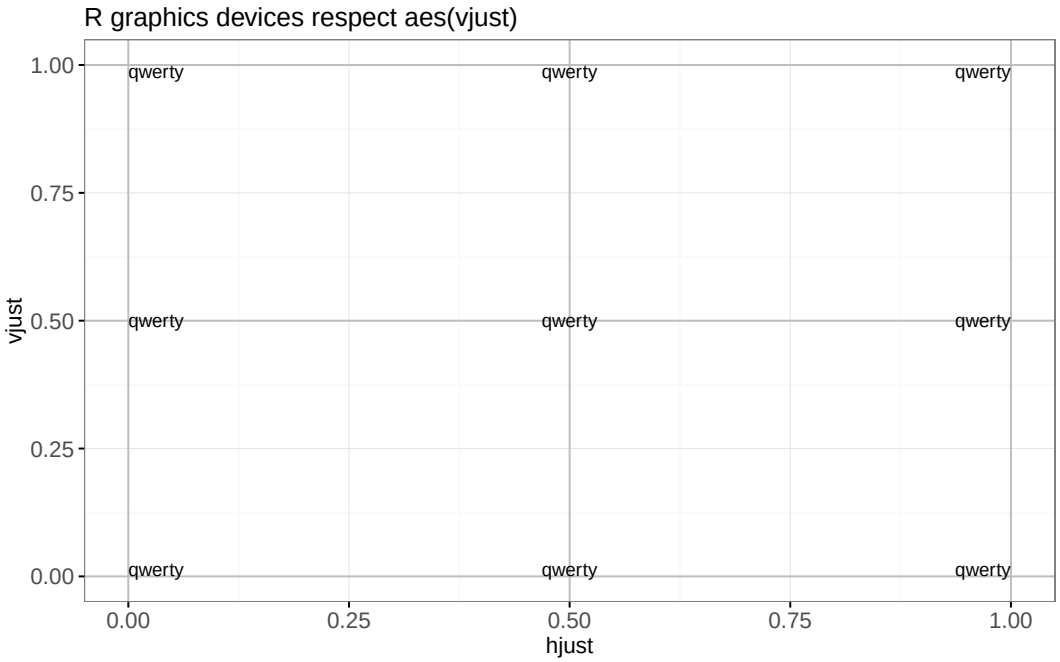
7.4 Ajuster y au lieu d'utiliser vjust

Pour l'alignement horizontal du texte, animint prend en charge l'utilisation de `hjust` dans R; les valeurs les plus courantes sont : 0 pour l'alignement à gauche, 0,5 pour l'alignement au

milieu et 1 pour l'alignement à droite. `animint2` traduit ces trois valeurs `hjust` en propriété `text-anchor` dans la visualisation de données générée.

Cependant, `animint` ne prend pas en charge l'alignement vertical du texte à l'aide de `vjust` dans R, car il n'y a pas de propriété qui puisse être utilisée pour l'alignement vertical du texte en SVG.

```
line.df <- data.frame(just=c(0, 0.5, 1))
text.df <- expand.grid(
  vjust=line.df$just,
  hjust=line.df$just)
gg.lines <- ggplot()+
  theme_bw()+
  geom_vline(aes(xintercept=just), data=line.df, color="grey")+
  geom_hline(aes(yintercept=just), data=line.df, color="grey")
gg.vjust <- gg.lines+
  geom_text(aes(
    hjust, vjust, label="qwerty", hjust=hjust, vjust=vjust),
    size=10,
    data=text.df)
gg.vjust+ggtitle("R graphics devices respect aes(vjust)")
```

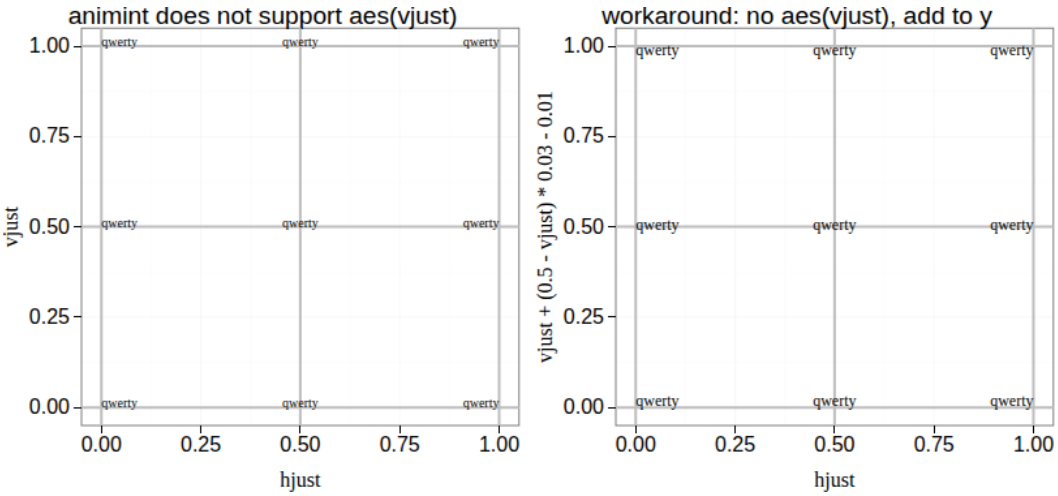


Notez comment `hjust` et `vjust` sont respectés dans le ggplot statique ci-dessus. En revanche, considérons l'`animint` ci-dessous. Le graphique de gauche devrait être le même que le ggplot ci-dessus, mais il y a des différences évidentes en termes de placement vertical des éléments de texte.

```
(viz.just <- animint(
  vjust=gg.vjust+
    ggtitle("animint does not support aes(vjust)"),
  workaround=gg.lines+
    ggtitle("workaround: no aes(vjust), add to y")+
    geom_text(aes(
      hjust, vjust + (0.5-vjust)*0.03 - 0.01,
      label="qwerty", hjust=hjust),
      data=text.df)))
```

```
[1] 0.5 1.0
```

```
Warning in vjustWarning(g.data$vjust): geom_text currently only supports
vjust=0, but you may want to try geom_label_aligned, which supports vjust
values 0, 0.5, and 1
```



La solution de contournement dans `animint2` est illustrée dans le panneau de droite ci-dessus. Vous pouvez ajuster les valeurs de la position verticale `y` des éléments de texte pour lesquels vous auriez utilisé `vjust`.

Il est possible d'implémenter une prise en charge de `vjust` dans `animint2`, mais nous n'avons pas encore eu le temps d'y travailler. Toute contribution en ce sens sera accueillie avec grand plaisir : n'hésitez pas à soumettre une Pull Request. Nous avons déjà une issue expliquant comment l'implémenter. .

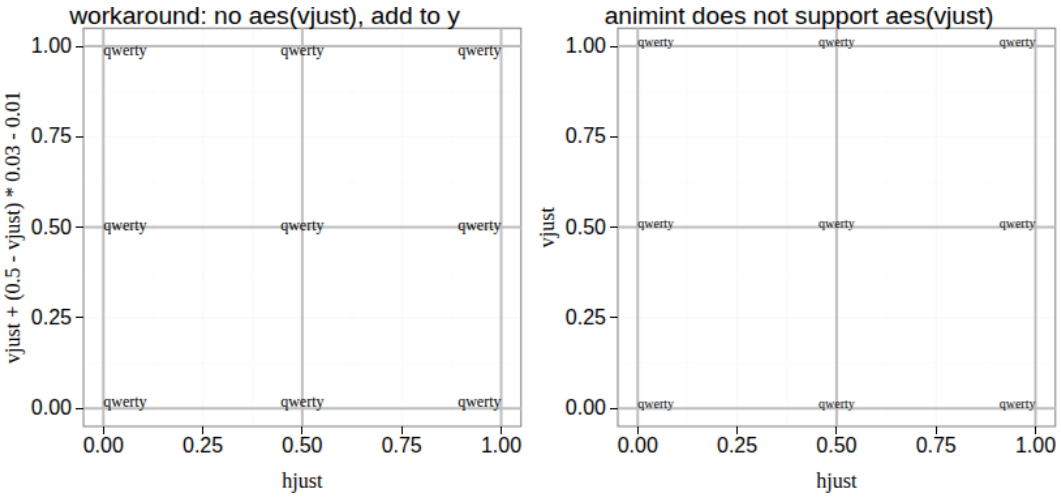
7.5 Ordonner les graphiques sur la page

Actuellement, la seule façon d'organiser une visualisation des données multiplot est d'utiliser l'ordre des ggplots dans la liste `animint viz`. Les graphiques apparaîtront sur la page web selon leur ordre dans la liste `animint viz`. Par exemple, comparez la visualisation de données ci-dessous avec la visualisation de données de la section précédente.

```
animint(
  first=viz.just$workaround,
  second=viz.just$vjust)
```

```
[1] 0.5 1.0
```

```
Warning in vjustWarning(g.data$vjust): geom_text currently only supports
vjust=0, but you may want to try geom_label_aligned, which supports vjust
values 0, 0.5, and 1
```



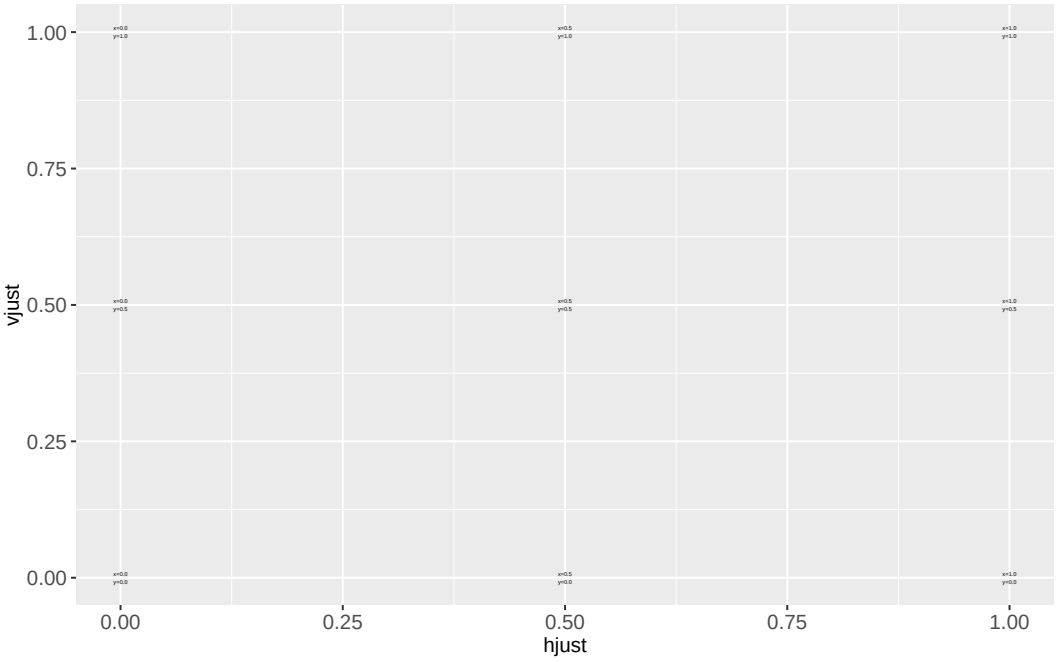
Notez que l'ordre des graphiques est inversé par rapport à la visualisation de données de la section précédente. La principale limite de cette méthode de mise en page des graphiques est que seul l'ordre peut être contrôlé. Par exemple, selon la largeur de l'élément de page web, le deuxième graphique apparaîtra soit sous le premier graphique, soit à sa droite.

Si vous avez une idée pour une meilleure façon de définir la mise en page des graphiques dans une animint, merci de nous en faire part !

7.6 Éviter les sauts de ligne dans les étiquettes de texte

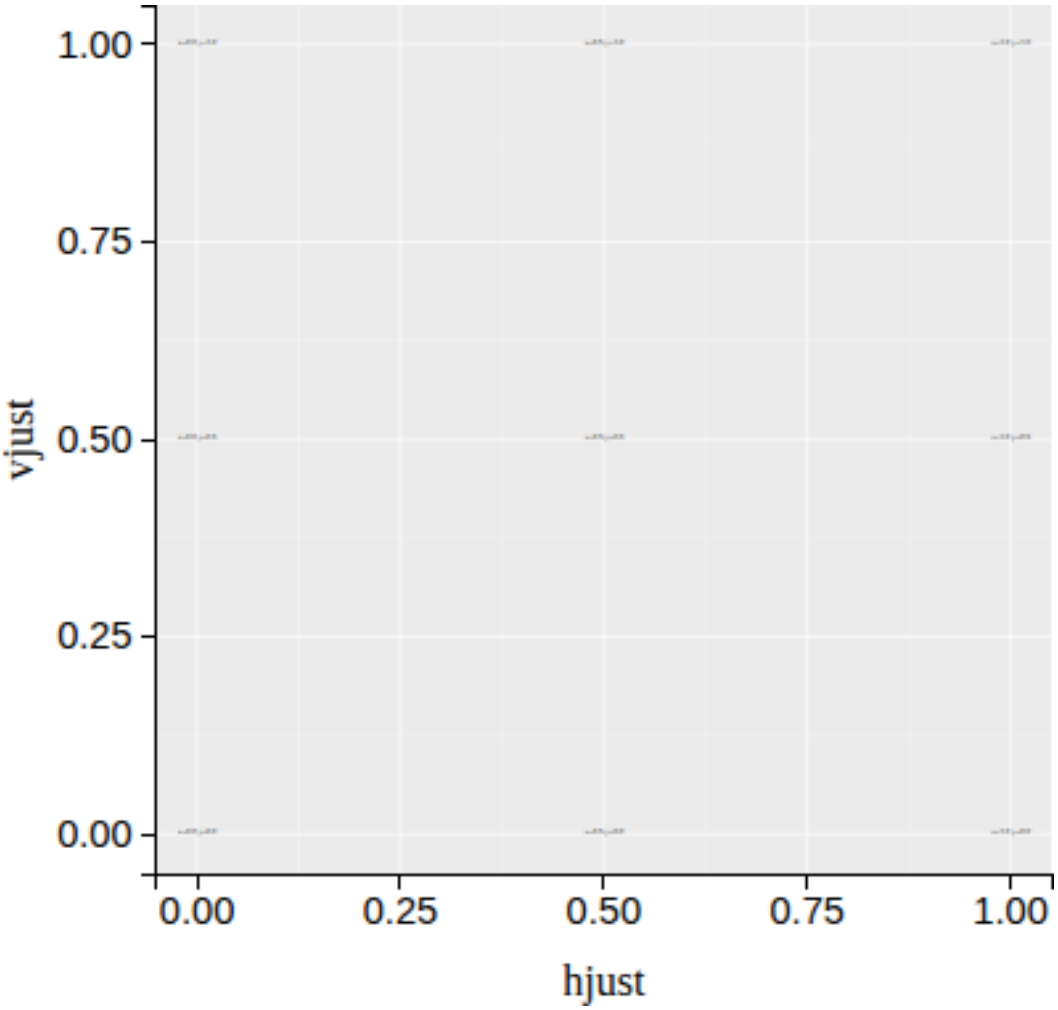
Lors de l'affichage d'un ggplot à l'aide des périphériques graphiques R habituels, l'utilisation d'un saut de ligne ou d'une nouvelle ligne `\n` dans une étiquette `geom_text` entraîne l'apparition de plusieurs lignes de texte sur le graphique.

```
gg.return <- ggplot()+
  geom_text(aes(
    hjust, vjust, label=sprintf("x=%.1f\ny=%.1f", hjust, vjust)),
    size=3,
    data=text.df)
gg.return
```



Cependant, `animint2` ne prend en charge que le tracé de la première ligne de texte.

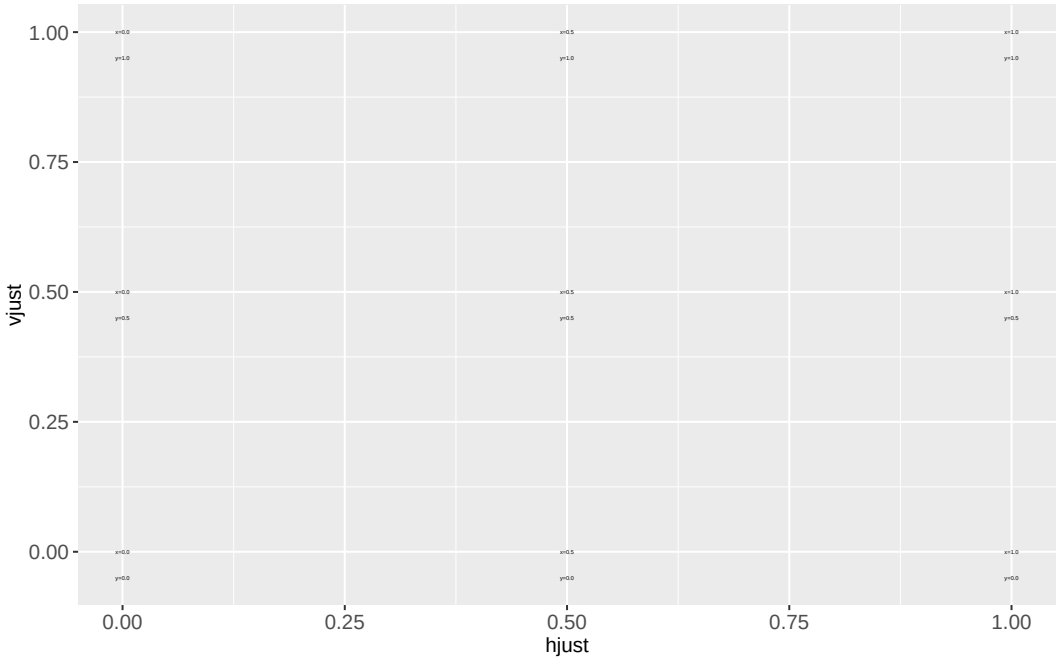
```
animint(gg.return)
```



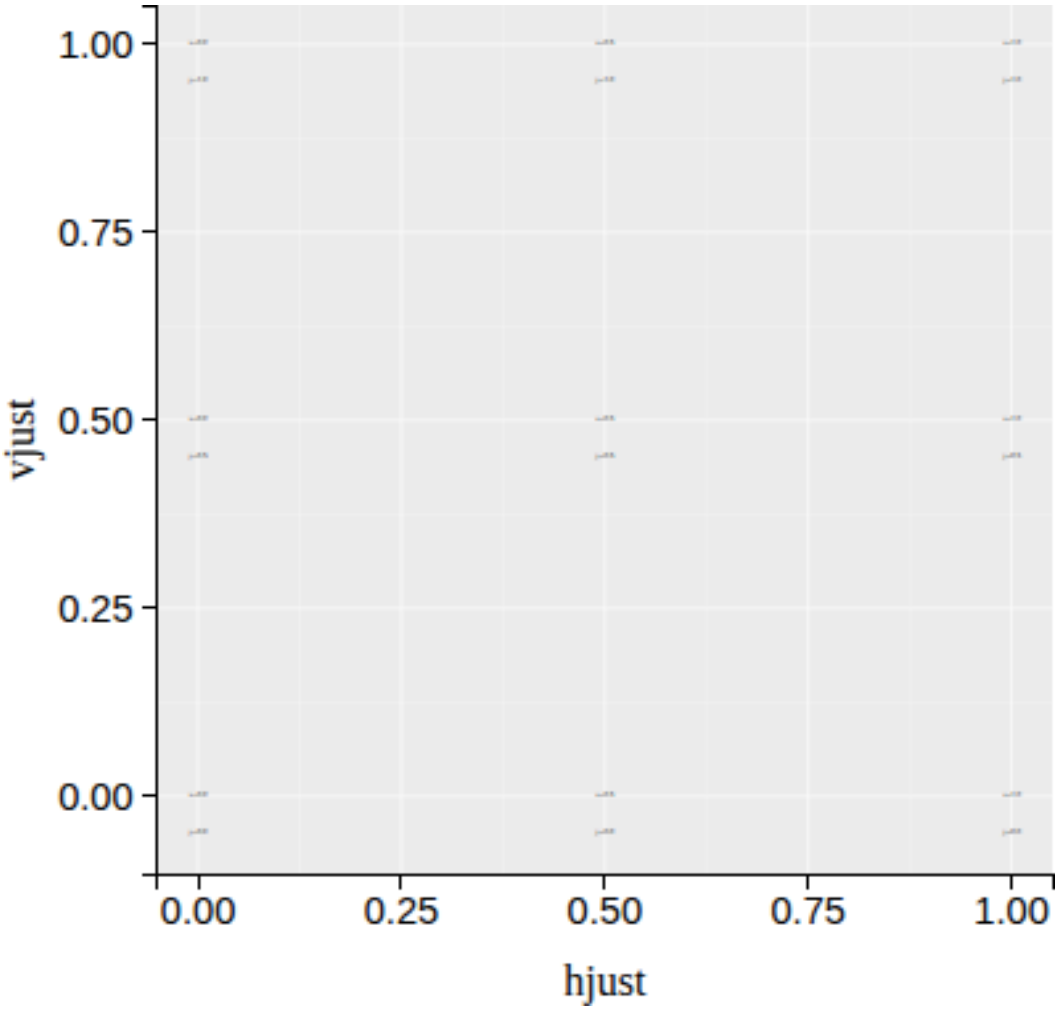
La prise en charge de plusieurs lignes dans les étiquettes `geom_text` serait utile, mais nous n'avons pas encore eu le temps de l'implémenter. Nous avons créé une issue à ce sujet, et toute Pull Request qui implémente cette fonctionnalité sera acceptée.

En attendant, la solution de contournement consiste à utiliser un seul `geom_text` pour chaque ligne de texte que vous souhaitez afficher :

```
gg.two.lines <- ggplot()+
  geom_text(aes(
    hjust, vjust, label=sprintf("x=%.1f", hjust)),
    size=3,
    data=text.df)+
  geom_text(aes(
    hjust, vjust-0.05, label=sprintf("y=%.1f", vjust)),
    size=3,
    data=text.df)
gg.two.lines
```




```
animint(gg.two.lines)
```



7.7 Éviter certaines options du thème ggplot

L'un des objectifs d'`animint2` est de prendre en charge toutes les options de thème, mais nous n'avons pas encore eu le temps de toutes les implémenter. Si vous utilisez une option de thème qu'`animint2` ne prend pas encore en charge, n'hésitez pas à nous envoyer une Pull Request. La liste suivante répertorie les options `theme` déjà prises en charge par `animint2`.

- `panel.margin` désigne la distance entre les panneaux, et est utilisé dans l' idiome des facettes d'économie d'espace pour l'éliminer.
- `panel.grid.major` est utilisé pour dessiner à l'échelle, les lignes de la grille qui sont désignées par l'argument `breaks`.
- `panel.grid.minor` est utilisé pour dessiner les lignes de la grille entre ses lignes principales.
- `panel.background` est utilisé pour le `<rect>` en arrière-plan de chaque panneau.
- `panel.border` est utilisé pour la bordure `<rect>` de chaque panneau (sur l'arrière-plan `<rect>`).

- `legend.position="none"` fonctionne pour cacher toutes les légendes, mais aucune des autres positions de légende n'est prise en charge (les légendes apparaissent toujours à droite du graphique).

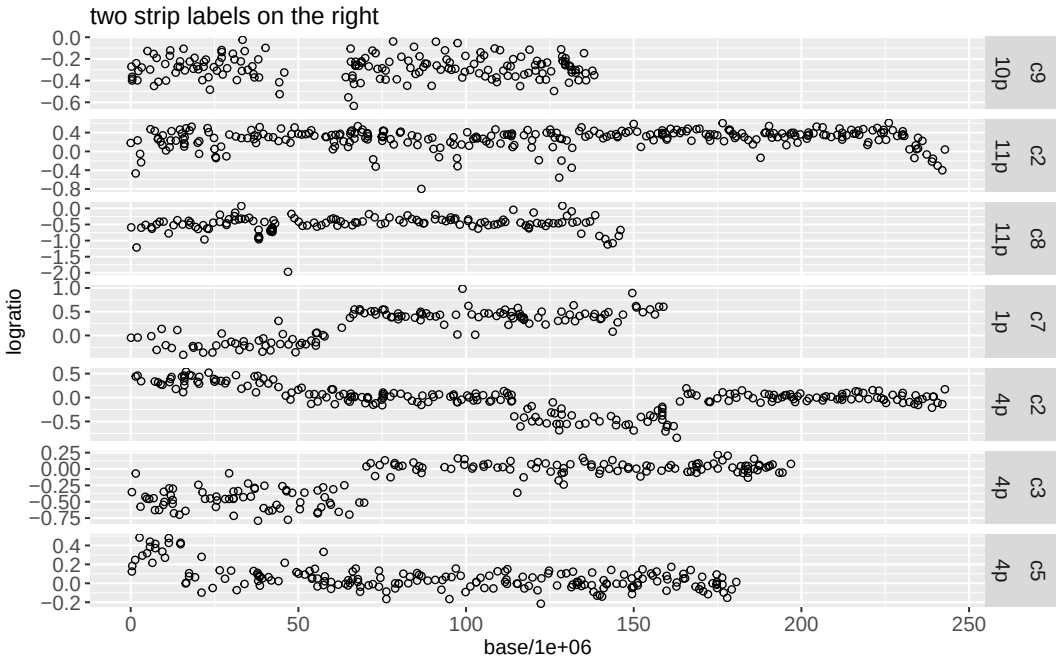
Les options de thème suivantes peuvent être définies par `element_blank` pour masquer les axes.

- `axis.title`, `axis.title.x`, `axis.title.y` désignent le titre de l'axe.
- `axis.ticks`, `axis.ticks.x`, `axis.ticks.y` désignent les tics de l'axe.
- `axis.line`, `axis.line.x`, `axis.line.y` désignent la ligne d'axe.
- `axis.text`, `axis.text.x`, `axis.text.y` désignent l'étiquette de texte de l'axe, et prennent en charge les arguments `angle` et `hjust` de `element_text`.

7.8 Facettes avec plusieurs variables par axe

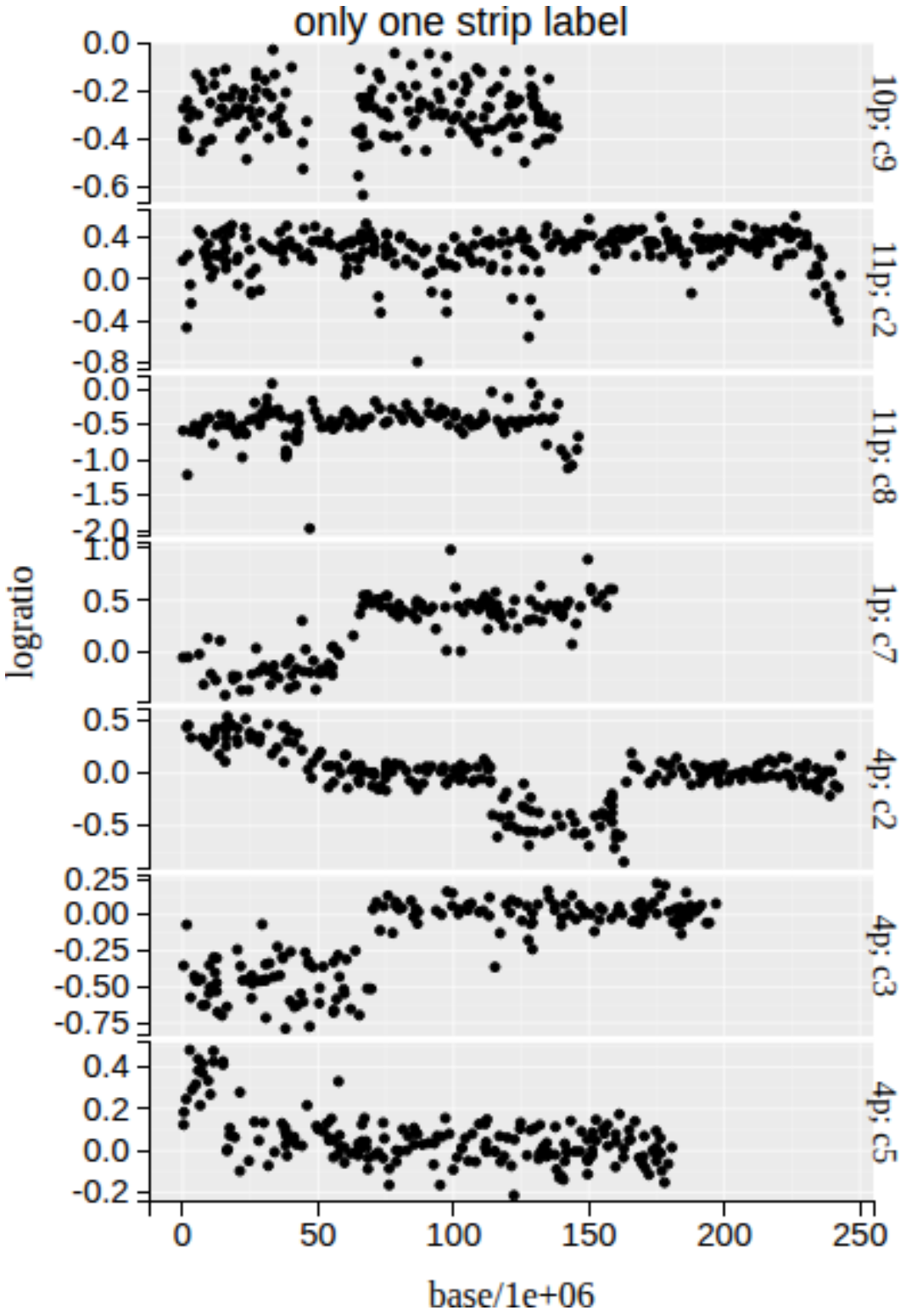
`animint2` prend en charge `facet_grid` pour créer des visualisations de données à plusieurs panneaux, mais ne prend en charge que de manière limitée les variables multiples par axe. Par exemple, le ggplot ci-dessous utilise deux variables pour créer des facettes verticales, ce qui se traduit par deux étiquettes de bande lorsqu'elles sont générées avec `ggplot2`.

```
data(intreg)
signals.df <- transform(
  intreg$signals,
  person=sub("[.].*", "p", signal),
  chromosome=sub(".*[.]", "c", signal))
two.strips <- ggplot()+
  theme_animint(height=600)+
  facet_grid(person + chromosome ~ ., scales="free")+
  geom_point(aes(base/1e6, logratio), data=signals.df)
two.strips+ggtitle("two strip labels on the right")
```



En comparaison, `animint2` affiche le même `ggplot` ci-dessous en n'utilisant qu'une seule étiquette de bande.

```
animint(two.strips+ggtitle("only one strip label"))
```



Il serait intéressant de prendre en charge plusieurs étiquettes de bande par axe, mais nous n'avons pas encore eu le temps de l'implémenter. Si vous souhaitez le faire, nous serions

heureux d'accepter votre Pull Request.

7.9 Définition interactive de mappings `aes()` à l'aide de shiny

Normalement, les mappings `aes()` sont définis une fois dans le code R, et ne peuvent pas être modifiés après l'affichage de l'`animint`. Une façon de surmonter cette limite est de définir des entrées `shiny` qui sont utilisées comme `aes()` d'`animint`, comme dans l'exemple suivant.

```
shiny::runApp(system.file(
  "examples", "shiny-WorldBank", package="animint2"))
```

7.10 Calcul interactif

Une autre limitation d'`animint2` est de ne pouvoir afficher que des données préalablement calculées et stockées dans un tableau. Cela signifie qu'`animint2` n'est pas approprié lorsqu'il y trop de sous-ensembles de données à tracer pour pouvoir les calculer tous. Dans ce cas, il serait préférable d'utiliser shiny.

7.11 Résumé du chapitre et exercices

Nous avons discuté des limites de l'implémentation actuelle de `animint2` et présenté plusieurs solutions de contournement.

Exercices :

- Réalisez un ggplot à facettes avec `stat_bin` qui ne fonctionnera pas avec `animint2` lorsque la variable facette est utilisée en tant que variable `showSelected`. Calculez le statut de chaque facette, et utilisez `stat_identity` pour que votre calcul fonctionne avec `animint2`.
- Faites un ggplot qui s'affiche bien en utilisant `facet_grid(. ~ var, scales="free")` mais ne s'affiche pas correctement avec `showSelected=var`. Pour résoudre le problème, calculez une version normalisée de `var` et utilisez-la dans `showSelected`.

Dans le chapitre 8, nous vous expliquerons comment créer une visualisation multi-panneaux des données de la Banque mondiale.

Part II

Exemples avancés

8

Visualisation des données de la Banque mondiale

Traduction de l'anglais Ch08-WorldBank-facets

Dans ce chapitre, nous allons explorer plusieurs visualisations des données de la Banque mondiale.

Plan du chapitre :

- Nous commençons par charger le jeu de données de la Banque mondiale et par définir quelques fonctions d'aide pour créer un ggplot multi-panneaux avec plusieurs geoms.
- Nous créons ensuite un graphique de série temporelle pour l'espérance de vie.
- Nous ajoutons ensuite un nuage de points de l'espérance de vie en fonction du taux de fertilité dans un deuxième panneau.
- Nous ajoutons ensuite un troisième panneau avec une série temporelle pour le taux de fertilité.

8.1 Chargement des données et définition des fonctions d'aide

Tout d'abord, nous chargeons l'ensemble des données de la Banque mondiale, et nous ne considérons que le sous-ensemble qui a des valeurs non manquantes à la fois pour `espérance.de.vie` et `taux.de.fertilité`.

```
library(animint2)
data(BanqueMondiale, package="animint2fr")
BanqueMondiale$Region <- sub(" (all income levels)", "", BanqueMondiale$region, fixed=TRUE)
library(data.table)
not.na <- data.table(
  BanqueMondiale)[!(is.na(espérance.de.vie) | is.na(taux.de.fertilité))]
```

Nous allons également tracer la variable population à l'aide d'une légende de taille. Avant de tracer le graphique, nous nous assurerons qu'aucune valeur n'est manquante.

```
not.na[is.na(not.na$population)]
```

```
   iso2c country year fertility.rate life.expectancy population
1:    KW  Kuwait 1992          2.338        72.95266         NA
2:    KW  Kuwait 1993          2.341        73.07373         NA
3:    KW  Kuwait 1994          2.413        73.18724         NA
```

```
   GDP.per.capita.Current.USD 15.to.25.yr.female.literacy iso3c
1:                         NA                           NA   KWT
2:                         NA                           NA   KWT
3:                         NA                           NA   KWT
                                       region     capital longitude
1: Middle East & North Africa (all income levels) Kuwait City   47.9824
2: Middle East & North Africa (all income levels) Kuwait City   47.9824
3: Middle East & North Africa (all income levels) Kuwait City   47.9824
   latitude              income        lending                    Region
1:  29.3721 High income: nonOECD Not classified Middle East & North Africa
2:  29.3721 High income: nonOECD Not classified Middle East & North Africa
3:  29.3721 High income: nonOECD Not classified Middle East & North Africa
                          région espérance.de.vie taux.de.fertilité année
1: Moyen-Orient et Afrique du Nord         72.95266             2.338  1992
2: Moyen-Orient et Afrique du Nord         73.07373             2.341  1993
3: Moyen-Orient et Afrique du Nord         73.18724             2.413  1994
     pays PIB.par.habitant.USD alphabétisation                 revenu
1: Koweït                   NA              NA Revenu élevé : nonOCDE
2: Koweït                   NA              NA Revenu élevé : nonOCDE
3: Koweït                   NA              NA Revenu élevé : nonOCDE
```

Le tableau ci-dessus montre que trois lignes ont des valeurs manquantes pour la variable population. concernant le Koweït entre 1992 et 1994. Le tableau ci-dessous présente les données des années voisines, de 1991 à 1995.

```
not.na[pays == "Kuwait" & 1991 <= année & année <= 1995]
```

```
Empty data.table (0 rows and 24 cols): iso2c,country,year,fertility.rate,life.expectancy,populatio
```

Le tableau ci-dessus montre que la population du Koweït a diminué au cours de la période 1991-1995, ce qui concorde avec la guerre du Golfe survenue à cette époque. Nous remplissons les valeurs manquantes ci-dessous.

```
not.na[is.na(population), population := 1700000]
not.na[pays == "Kuwait" & 1991 <= année & année <= 1995]
```

```
Empty data.table (0 rows and 24 cols): iso2c,country,year,fertility.rate,life.expectancy,populatio
```

Ensuite, nous définissons la fonction d'aide suivante, qui sera utilisée pour ajouter des colonnes aux ensembles de données afin d'affecter des geoms aux facettes.

```
FACETS <- function(df, top, side){
  data.frame(df,
             top=factor(top, c("Taux de fertilité", "Années")),
             side=factor(side, c("Années", "Espérance de vie")))
}
```

Notez que les niveaux des facteurs spécifieront l'ordre des facettes dans le ggplot. Ceci est un exemple de l'idiome addColumn then facet . Nous définissons ci-dessous trois fonctions d'aide, une pour chaque facette.

```
TS.RIGHT <- function(df)FACETS(df, "Années", "Espérance de vie")
SCATTER <- function(df)FACETS(df, "Taux de fertilité", "Espérance de vie")
TS.ABOVE <- function(df)FACETS(df, "Taux de fertilité", "Années")
```

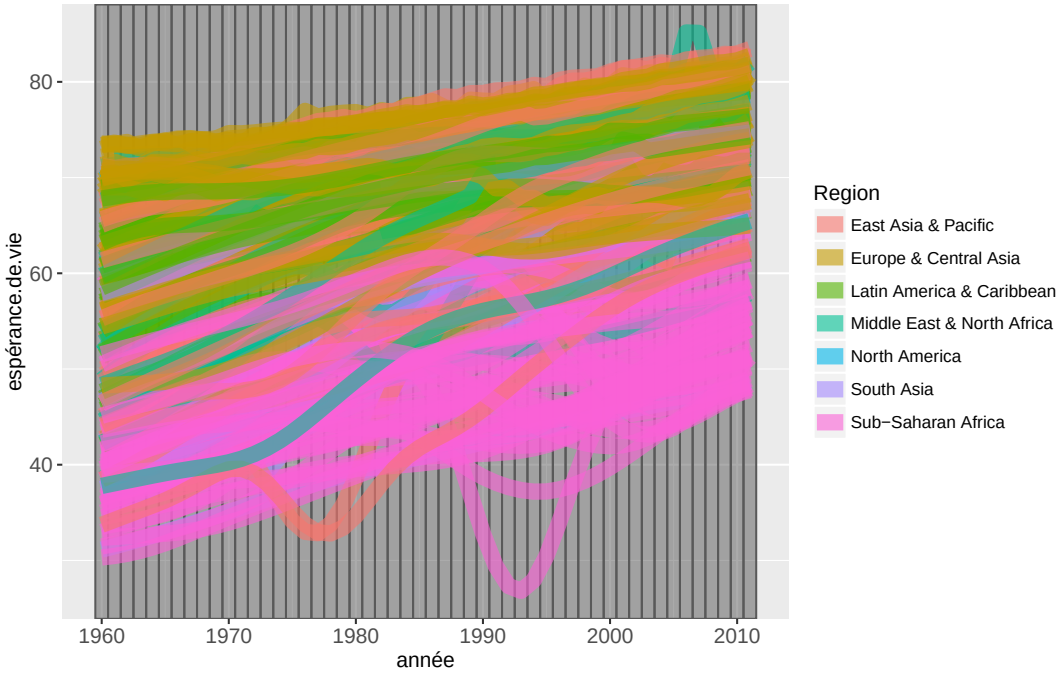
8.2 Premier graphique de la série temporelle

Tout d'abord, nous définissons un ensemble de données avec une ligne pour chaque année, que nous utiliserons pour sélectionner les années à l'aide d'un `geom_tallrect` en arrière-plan.

```
années <- unique(not.na[, .(année)])
```

Nous définissons le ggplot avec un `geom_tallrect` en arrière-plan, et un `geom_line` pour les séries temporelles.

```
ts.right <- ggplot()+
  geom_tallrect(aes(
    xmin=année-1/2, xmax=année+1/2),
    clickSelects="année",
    data=TS.RIGHT(années), alpha=1/2)+
  geom_line(aes(
    année, espérance.de.vie, group=pays, colour=Region),
    clickSelects="pays",
    data=TS.RIGHT(not.na), size=4, alpha=3/5)
ts.right
```

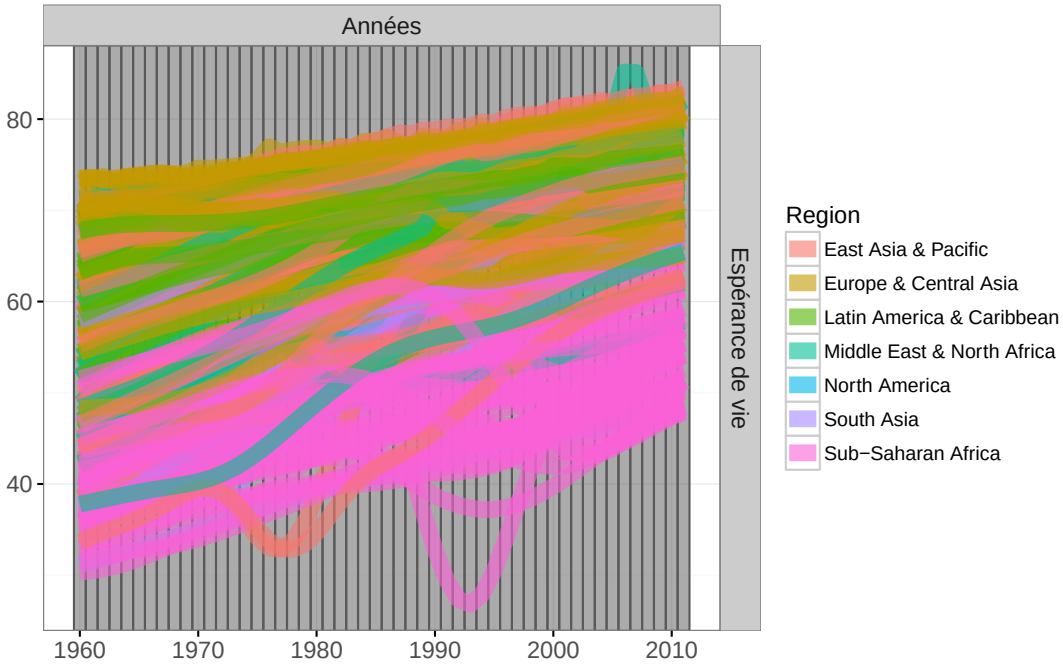


Remarquez que nous avons spécifié `clickSelects=année` pour qu'un clic sur un tallrect modifie l'année sélectionnée, et `clickSelects=pays` pour que le fait de cliquer sur une ligne sélectionne ou désélectionne un pays. Notez également que nous avons utilisé `TS.RIGHT` pour identifier les colonnes que nous utiliserons dans la spécification des facettes (section suivante).

8.3 Ajouter une facette de nuage de points

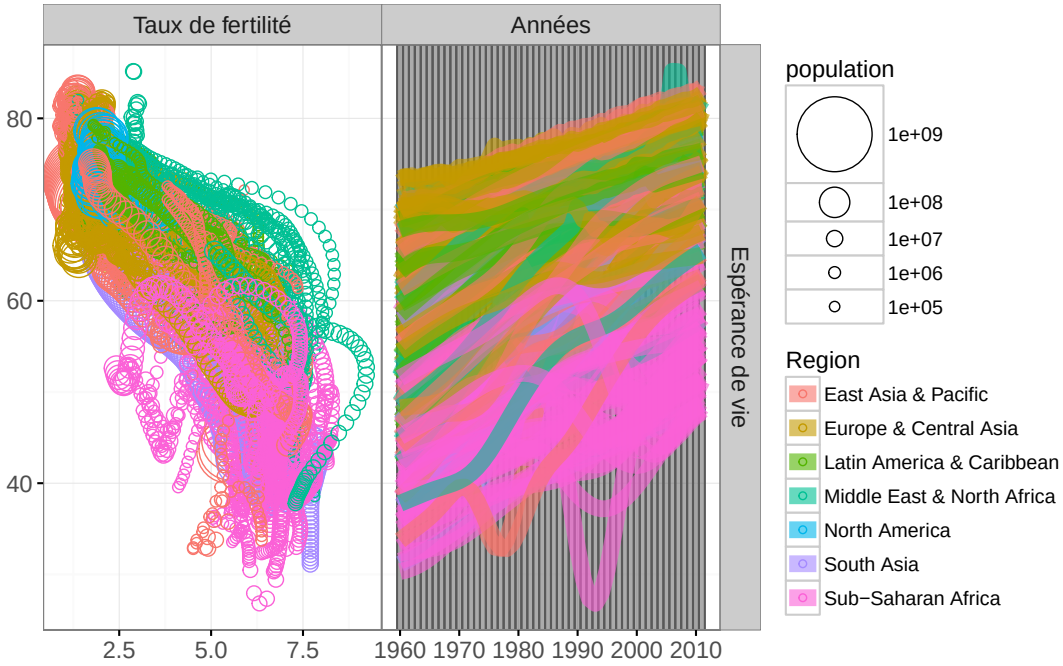
Nous commençons par ajouter simplement des facettes au graphique de la série temporelle précédente.

```
ts.facet <- ts.right+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(side ~ top, scales="free")+
  xlab("")+
  ylab("")
ts.facet
```



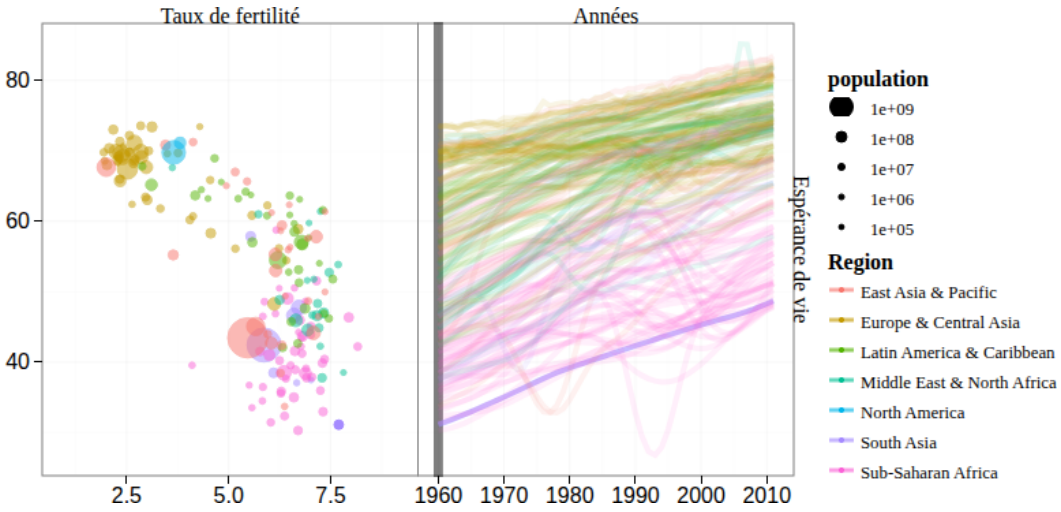
Nous définissons `panel.margin` à 0, ce qui est toujours une bonne idée pour économiser de l'espace dans un ggplot avec des facettes. Dans un des exemples tiré de, dans un exemple de l'idiome addColumn then facet, nous utilisons `scales="free"` et nous masquons les étiquettes des axes . Pour les remplacer, nous utilisons le libellé de la facette pour indiquer la variable encodée sur chaque axe. Ci-dessous, nous ajoutons une facette de nuage de points avec un point pour chaque année et chaque pays.

```
ts.scatter <- ts.facet+
  theme_animint(width=600)+
  geom_point(aes(
    taux.de.fertilité, espérance.de.vie,
    colour=Region, size=population,
    key=pays), # key aesthetic for animated transitions!
    clickSelects="pays",
    showSelected="année",
    data=SCATTER(not.na))+
  scale_size_animint(pixel.range=c(2, 20), breaks=10^(9:5))
ts.scatter
```



Remarquez comment nous utilisons `scale_size_animint` pour spécifier l'échelle des tailles en pixels, et les intervalles (`breaks`) dans la légende. Notez également que nous utilisons `SCATTER` pour spécifier les colonnes `top` et `side` qui sont utilisées dans la spécification de la facette. Nous affichons également ce ggplot de manière interactive ci-dessous.

```
animint(ts.scatter)
```

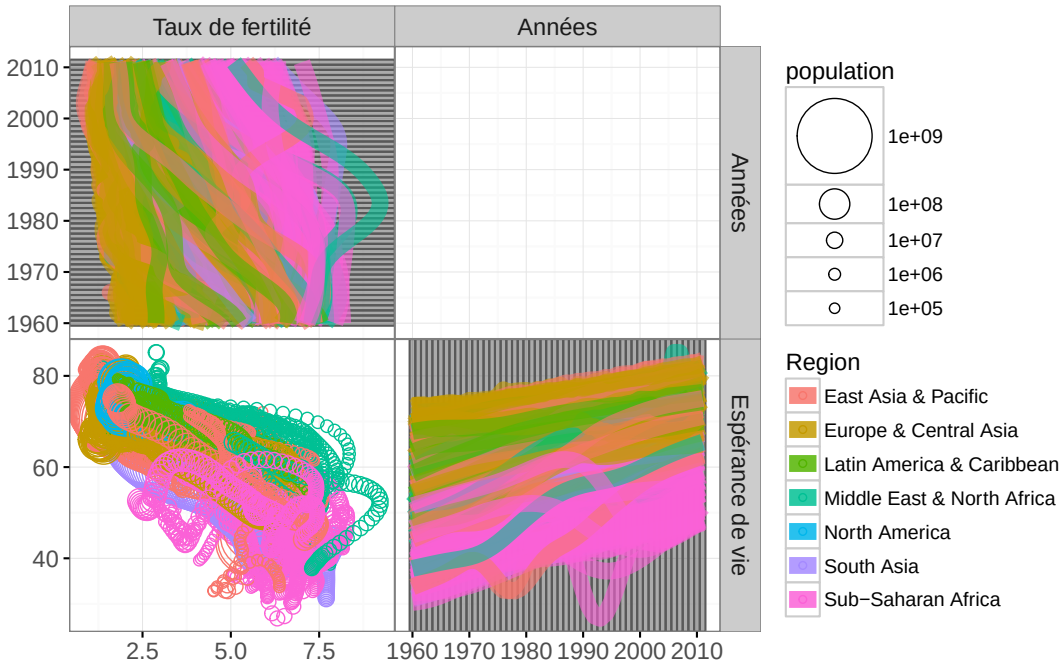


Notez que la sélection unique est utilisée par défaut pour l'année et le pays.

8.4 Ajout d'une autre facette pour les séries temporelles

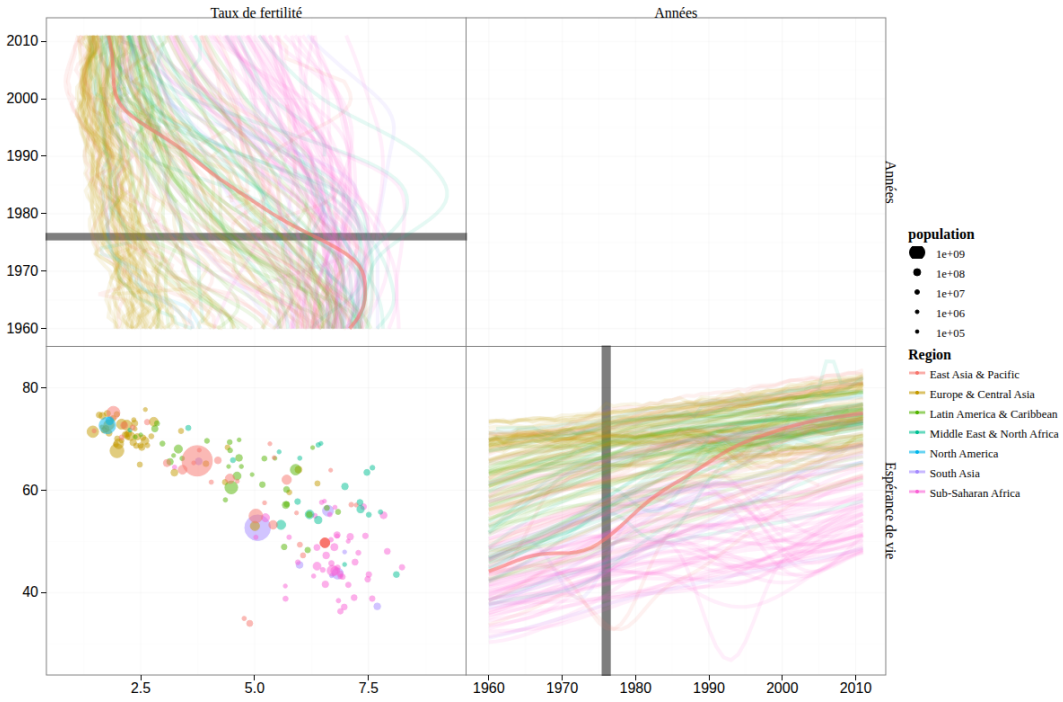
Ci-dessous, nous ajoutons des `widerects` pour sélectionner les années, et des chemins d'accès ("path") pour afficher le taux de fertilité.

```
scatter.both <- ts.scatter+
  geom_widerect(aes(
    ymin=année-1/2, ymax=année+1/2),
    clickSelects="année",
    data=TS.ABOVE(années), alpha=1/2)+
  geom_path(aes(
    taux.de.fertilité, année, group=pays, colour=Region),
    clickSelects="pays",
    data=TS.ABOVE(not.na), size=4, alpha=3/5)
scatter.both
```



Notez que `TS.ABOVE` a été utilisé pour spécifier les colonnes des facettes `top` et `side`. Nous affichons une version interactive ci-dessous.

```
viz.scatter.both <- animint(
  title="Données de la Banque mondiale (multiple selection, facets)",
  scatterBoth=scatter.both+
    theme_animint(width=1000, height=800),
  duration=list(année=1000),
  time=list(variable="année", ms=3000),
  first=list(année=1975, pays=c("United States", "Vietnam")),
  selector.types=list(pays="multiple"))
viz.scatter.both
```



8.5 Résumé du chapitre et exercices

Nous avons montré comment créer une visualisation multicouche et multipanneaux (mais à graphique unique) des données de la Banque mondiale.

Exercices :

- Sur chaque graphique de série temporelle, ajoutez un point dont la taille est proportionnelle à la population, comme dans le nuage de points. Les points ne doivent apparaître que lorsque le pays est sélectionné, et un clic sur les points doit désélectionner ce pays.
- Ajoutez des étiquettes de texte au graphique de la série temporelle situé à droite, avec les noms de chaque pays. Chaque étiquette ne doit apparaître que lorsque le pays est sélectionné, et doit disparaître lorsqu'on clique sur l'étiquette.
- Ajoutez une étiquette de texte au nuage de points pour indiquer l'année sélectionnée.
- Ajoutez des étiquettes de texte au nuage de points, avec des noms pour chaque pays. Chaque étiquette ne doit apparaître que lorsque le pays est sélectionné, et doit disparaître lorsqu'on clique sur l'étiquette.

Le chapitre 9 présente comment visualiser les données sur les vélos à Montréal.

9

Vélos de Montréal

Dans ce chapitre, nous explorerons plusieurs visualisations des données pour un ensemble de données portant sur les vélos de Montréal.

Plan du chapitre :

- Nous commençons par quelques séries temporelles non interactives.
- Nous créons une visualisation interactive de la fréquence des accidents dans le temps.
- Nous créons une visualisation interactive des données avec quatre graphiques, montrant les tendances mensuelles des accidents, les détails quotidiens et une carte des emplacements des compteurs.

9.1 Graphiques statiques

Nous commençons par charger le fichier `montreal.bikes` qui n'est pas disponible dans la version CRAN de `animint2`, afin d'économiser de l'espace sur le CRAN. Par conséquent, pour accéder à ce jeu de données, vous devrez installer `animint2` depuis GitHub :

```
tryCatch({
  data(montreal.bikes, package="animint2")
}, warning=function(w){
  devtools::install_github("animint/animint2")
})
```

Les données sont deux séries temporelles :

- `montreal.bikes$counter.counts` : les passages de vélos sur les compteurs, dans un tableau avec une ligne pour chaque combinaison de lieu et jour.
- `montreal.bikes$accidents` : les accidents, dans un tableau avec une ligne par accident.

Nous allons calculer des résumés par mois dans ces deux séries temporelles.

9.1.1 Compteurs

Pour commencer, nous affichons le tableau de données des compteurs de vélos.

```
mois_str <- function(POSIXct)strftime(POSIXct, "%Y-%m")
library(data.table)

(passages_dt <- data.table(montreal.bikes$counter.counts)[, .(
```

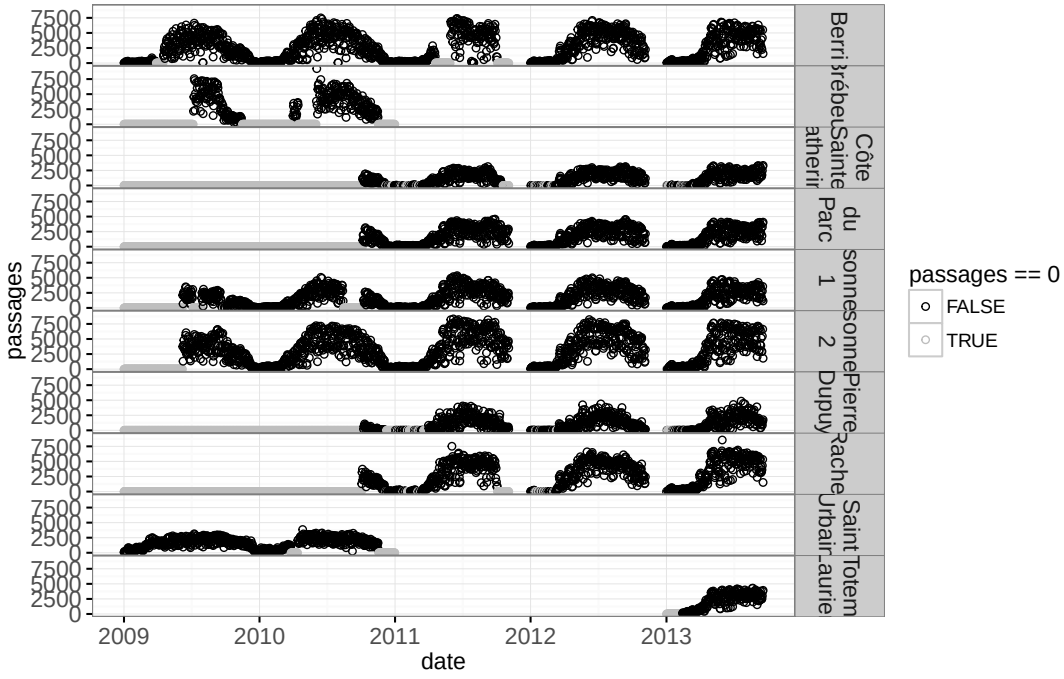
```
    lieu = location,
    date,
    mois.str = mois_str(date),
    passages=count)])
```

```
                lieu                date mois.str passages
    1:          Berri 2009-01-01 05:00:00  2009-01       29
    2:          Berri 2009-01-02 05:00:00  2009-01       19
   ---
13382: Totem_Laurier 2013-09-17 04:00:00  2013-09     3745
13383: Totem_Laurier 2013-09-18 04:00:00  2013-09     3921
```

Ci-dessus, nous voyons une ligne pour chaque combinaison de lieu et jour. Le comptage de vélos présente des données de séries temporelles que nous visualisons ci-dessous.

```
passages_dt[, lieu.lines := gsub("[- _]", "\n", lieu)]
library(animint2)
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(lieu.lines ~ .)+
  geom_point(aes(
    date, passages, color=passages==0),
    shape=21,
    data=passages_dt)+
  scale_color_manual(values=c("TRUE"="grey", "FALSE"="black"))
```

```
Warning: Removed 407 rows containing missing values (geom_point).
```



Le graphique ci-dessus données permet de voir facilement la différence entre les zéros (en gris) et les valeurs manquantes. On voit bien la régularité au fil des saisons (moins de vélos en hiver).

9.1.2 Accidents

Pour commencer avec les données d'accidents, nous affichons une ligne :

```
  montreal.bikes$accidents[1,]
```

```
    date.str time.str deaths people.severely.injured people.slightly.injured
1 2012-01-02    18:35      0                       0                       1
  street.number             street cross.street location.int position.int
1            NA ST JEAN BAPTISTE O   AV ROULEAU           32            6
             position                              location
1 Voie de circulation En intersection (moins de 5 mètres)
```

Pour chaque accident il y a des données sur la date, l'heure, la localisation et le nombre de morts et de blessés. Certaines valeurs sont en français (par exemple : `Voie de circulation`, `En intersection`, etc). Pour les colonnes avec noms en anglais, nous allons faire des copies en français :

```
  gravité <- c(
    décès="deaths",
    grave="people.severely.injured",
    mineure="people.slightly.injured")
  montreal.bikes$accidents[, names(gravité)] <-
    montreal.bikes$accidents[, gravité]
  accidents_dt <- data.table(montreal.bikes$accidents[, c(
    "date.str", "time.str", names(gravité),
    "street", "street.number", "cross.street")])
```

Dans le code ci-dessous, nous rajoutons une colonne pour le mois.

```
  ymd2POSIXct <- function(date.str){
    as.POSIXct(strptime(date.str, "%Y-%m-%d"))
  }
  (accidents_dt[
  , date := ymd2POSIXct(date.str)
  ][
  , mois.str := mois_str(date)
  ][])
```

```
        date.str time.str décès grave mineure             street street.number
   1: 2012-01-02    18:35     0     0       1 ST JEAN BAPTISTE O            NA
   2: 2012-01-05    21:50     0     0       1             FOSTER            NA
  ---
5594: 2014-12-27    12:35     0     0       1   CH DES PATRIOTES            NA
5595: 2014-12-30    11:55     0     0       1     PIERREFONDS BD         14965
        cross.street       date mois.str
   1:     AV ROULEAU 2012-01-02  2012-01
```

```
   2:        JANELLE 2012-01-05  2012-01
  ---
5594:        1RE RUE 2014-12-27  2014-12
5595: JACQUES BIZARD 2014-12-30  2014-12
```

Dans la sortie ci-dessus, on voit que les derniers mois pour les accidents ne sont pas les mêmes que les compteurs. On compare les intervalles dans le code ci-dessous :

```
data.list <- list(accidents=accidents_dt, passages=passages_dt)
sapply(data.list, function(DT)range(DT$mois.str))
```

```
     accidents passages
[1,] "2012-01" "2009-01"
[2,] "2014-12" "2013-09"
```

Dans la sortie ci-dessus, on voit que les passages et les accidents se chevauchent. Nous allons compiler des résumés pour tous les mois, c'est pourquoi nous faisons un tableau de données pour chaque mois ci-dessous.

```
uniq.mois.vec <- sort(unique(unlist(lapply(
  data.list, "[[", "mois.str"))))
mois_01 <- function(mois)ymd2POSIXct(paste0(mois, "-01"))
mois_dt <- data.table(mois.01 = mois_01(uniq.mois.vec))
```

Le code ci-dessous définit l'environnement linguistique (locale) pour avoir les noms de mois en français.

```
old.locale <- Sys.setlocale(locale="fr_CA.UTF-8")
mois_français_str <- function(POSIXct)strftime(POSIXct, "%B %Y")
mois.levs <- mois_français_str(mois_dt$mois.01)
mois_français <- function(POSIXct)factor(
  mois_français_str(POSIXct), mois.levs)
mois_dt[, mois.français := mois_français(mois.01)][]
```

```
       mois.01 mois.français
 1: 2009-01-01  janvier 2009
 2: 2009-02-01  février 2009
---
71: 2014-11-01 novembre 2014
72: 2014-12-01 décembre 2014
```

La sortie ci-dessus comprend une ligne pour chaque mois. Notez que nous avons créé une colonne `mois.français` qui sera utilisé comme variable de sélection pour les mois. Dans le code ci-dessous, nous calculons la somme des accidents par mois.

```
(accidents.par.mois <- dcast(
  accidents_dt,
  mois.str ~ .,
  sum,
  value.var=names(gravité)))
```

```
    mois.str décès grave mineure
 1:  2012-01     1     0      10
 2:  2012-02     0     0      20
---
35:  2014-11     1     2      69
36:  2014-12     0     0      10
```

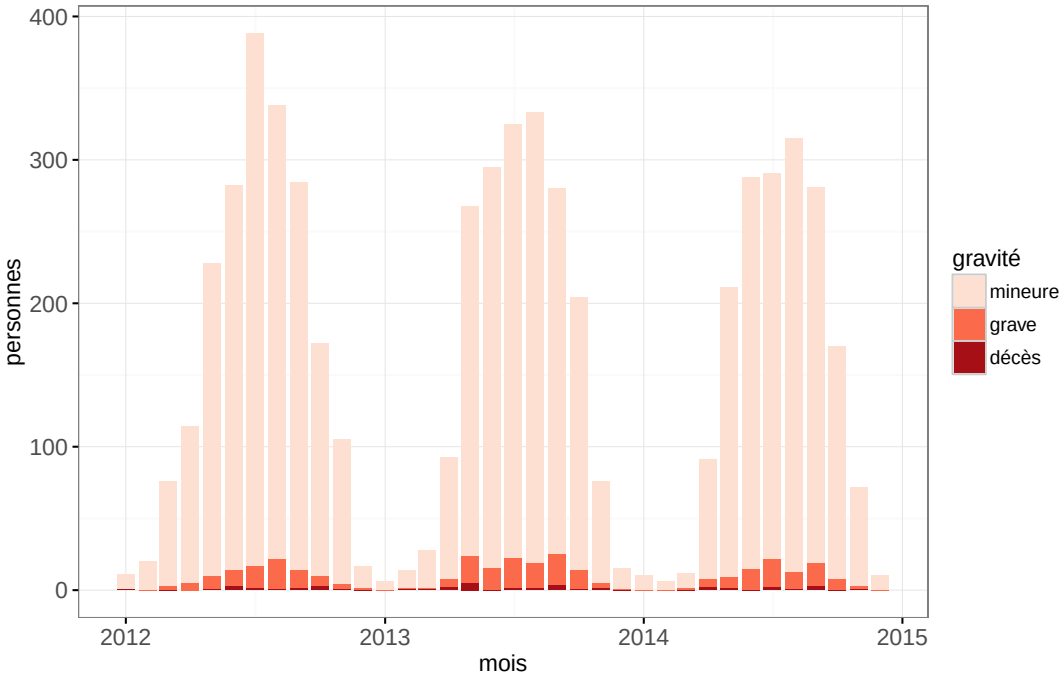
Ci-dessus on voit une ligne pour chaque mois, avec différentes colonnes pour chaque niveau de gravité. Nous convertissons ces colonnes en lignes avec le code ci-dessous :

```
(accidents.tall <- melt(
  accidents.par.mois,
  measure.vars=names(gravité),
  variable.name="gravité",
  value.name="personnes"))
```

```
    mois.str gravité personnes
  1:  2012-01   décès         1
  2:  2012-02   décès         0
 ---
107:  2014-11 mineure        69
108:  2014-12 mineure        10
```

Ci-dessus on voit une ligne pour chaque combinaison de mois et gravité. Dans le code ci-dessous, nous utilisons ces données pour créer un graphique montrant le nombre d'accidents par mois.

```
gravité.colors <- c(
  mineure="#FEE0D2",#lite red
  grave="#FB6A4A",
  décès="#A50F15")#dark red
ggplot()+
  theme_bw()+
  geom_bar(aes(
    mois_01(mois.str), personnes, fill=gravité),
    stat="identity",
    data=accidents.tall)+
  scale_fill_manual(
    values=gravité.colors, breaks=names(gravité.colors))+
  scale_x_datetime("mois")
```



Ci-dessus, on voit le nombre de personnes décédées et blessées au fil du temps.

9.2 Visualisation interactive de la fréquence des accidents

Maintenant nous voulons comparer, pour chaque mois, les données de compteurs et d'accidents. Est-ce que nous avons plus d'accidents quand il y a plus de passages en vélo ? Pour vérifier, nous devons calculer un résumé des passages par mois avec le code ci-dessous.

```
(passages.par.mois <- dcast(
  passages_dt[!is.na(passages)],
  lieu + mois.str ~ .,
  list(length, mean, sum),
  value.var="passages"))
```

```
             lieu mois.str passages_length passages_mean passages_sum
  1:         Berri  2009-01              31      100.3226         3110
  2:         Berri  2009-02              28      159.6786         4471
 ---
441: Totem_Laurier  2013-08              31     3162.7097        98044
442: Totem_Laurier  2013-09              18     2888.7778        51998
```

La sortie ci-dessus contient une ligne pour chaque combinaison de lieu et mois, avec des colonnes pour:

- `passages_length` : le nombre de jours.
- `passages_mean` : le moyenne de passages par jour.

- `passages_sum` : le total nombre de passages.

On remarque que certains mois contiennent des journées manquantes. Par exemple, il n'y a que 29 jours pour `Berri` en avril 2009, et 18 jours pour `Totem_Laurier` en septembre 2013. Pour modeliser seulement les mois entiers, nous voulons enlever les mois avec jours manquants. Alors on utilise le code ci-dessous pour calculer le nombre de jours dans chaque mois :

```
un.jour <- 60 * 60 * 24
mois_suivant <- function(POSIXct)mois_01(POSIXct + un.jour * 31)
passages.par.mois[, jours.dans.mois := as.integer(round(difftime(
  mois_01(mois_str(mois_suivant(mois_01(mois.str)))),
  mois_01(mois.str),
  units="days"
)))][]
```

```
             lieu mois.str passages_length passages_mean passages_sum
  1:         Berri  2009-01              31      100.3226         3110
  2:         Berri  2009-02              28      159.6786         4471
 ---
441: Totem_Laurier  2013-08              31     3162.7097        98044
442: Totem_Laurier  2013-09              18     2888.7778        51998
     jours.dans.mois
  1:              31
  2:              28
 ---
441:              31
442:              30
```

Dans la sortie ci-dessus, on voit la nouvelle colonne `jours.dans.mois`. Avec le code ci-dessous, on affiche seulement les mois avec jours manquants :

```
passages.par.mois[
  passages_length < jours.dans.mois,
  .(lieu, mois.str, passages_length, jours.dans.mois)]
```

```
             lieu mois.str passages_length jours.dans.mois
 1:         Berri  2009-04              29              30
 2:         Berri  2011-11               3              30
---
22:        Rachel  2013-09              18              30
23: Totem_Laurier  2013-09              18              30
```

Nous allons exclure les données ci-dessus, en utilisant le code ci-dessous :

```
mois.complets <- passages.par.mois[passages_length == jours.dans.mois]
```

Ensuite, nous faisons un tableau avec les passages et les accidents dans différents colonnes :

```
city.wide.complete <- mois.complets[passages_sum>0, .(
  lieux=.N,
```

```
    total.passages=sum(passages_sum)
  ), keyby=mois.str]
  city.wide.accidents <- accidents_dt[, .(
    total.accidents=.N
  ), keyby=mois.str]
  (scatter.not.na <- city.wide.accidents[
    city.wide.complete, nomatch=0L
  ][, mois.01 := mois_01(mois.str)][])
```

```
    mois.str total.accidents lieux total.passages    mois.01
 1:  2012-01              11     7          20386 2012-01-01
 2:  2012-02              19     7          26727 2012-02-01
---
17:  2013-07             315     8         916662 2013-07-01
18:  2013-08             326     8         856066 2013-08-01
```

Dans la sortie ci-dessus, on voit une ligne pour chaque mois avec les données de compteurs et accidents. Ensuite, nous faisons un modèle linéaire pour prédire les accidents à partir des passages :

```
  (fit <- lm(total.accidents ~ total.passages - 1, scatter.not.na))
```

```
Call:
lm(formula = total.accidents ~ total.passages - 1, data = scatter.not.na)

Coefficients:
total.passages
     0.0003723
```

```
  scatter.not.na[, mean(total.accidents/total.passages)]
```

```
[1] 0.0003847625
```

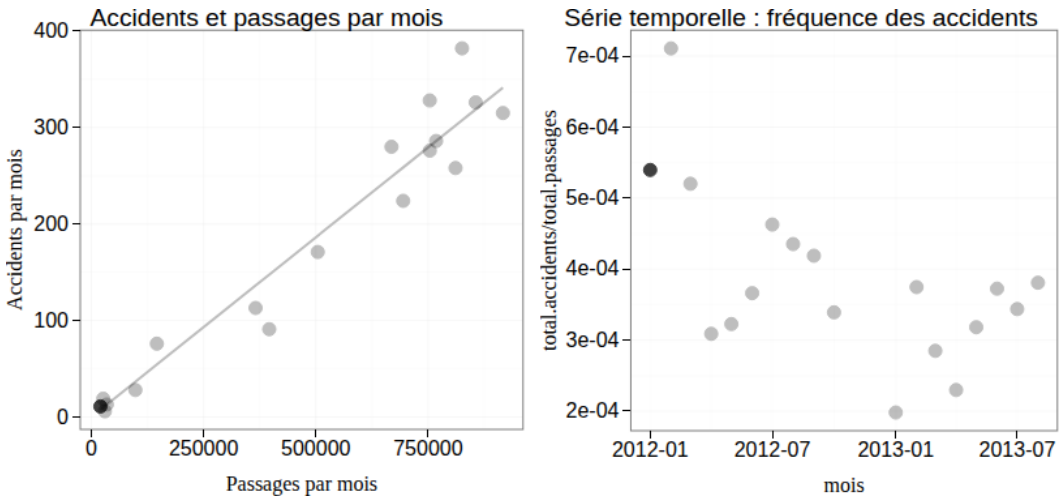
```
  scatter.not.na[, sum(total.accidents)/sum(total.passages)]
```

```
[1] 0.0003693805
```

Dans la sortie ci-dessus, on voit que le coefficient du modèle est proche des moyennes empiriques. Enfin, nous faisons un graphique interactif avec le code ci-dessous.

```
  scatter.not.na[, let(
    pred.accidents = predict(fit),
    mois.français = mois_français(mois.01)
  )]
  animint(
    regression=ggplot()+
      theme_bw()+
      ggtitle("Accidents et passages par mois")+
      geom_line(aes(
        total.passages, pred.accidents),
```

```
      color="grey",
      data=scatter.not.na)+
    geom_point(aes(
      total.passages, total.accidents),
      clickSelects="mois.français",
      size=5,
      alpha=0.75,
      data=scatter.not.na)+
    ylab("Accidents par mois")+
    xlab("Passages par mois"),
  timeSeries=ggplot()+
    theme_bw()+
    ggtitle("Série temporelle : fréquence des accidents")+
    xlab("mois")+
    geom_point(aes(
      mois.01, total.accidents/total.passages),
      clickSelects="mois.français",
      size=5,
      alpha=0.75,
      data=scatter.not.na))
```



La visualisation des données ci-dessus contient deux graphiques réliés. Le graphique de gauche montre que le nombre d'accidents augmente avec le nombre de cyclistes. Le graphique de droite montre la fréquence des accidents au fil du temps.

9.3 Visualisation interactive avec carte et détails

Dans cette partie, nous allons faire une visualisation avec plusieurs composants :

- Résumé des compteurs : plan des compteurs ou min/max des mois pour chaque compteur, pour sélectionner un compteur.
- Détails d'un compteur, résumé des mois : séries temporelles par mois, pour les accidents

et les données d'un compteur. Cliquer pour sélectionner un mois.
- Détails d'un compteur et d'un mois : séries temporelles par jour, pour le mois sélectionné.

9.3.1 Résumé des compteurs avec plan

Les données `counter.locations` contient l'emplacement géographique de chaque compteur. Pour examiner ces données, il faut d'abord convertir le nom en unicode :

```
(counter.locations <- data.table(montreal.bikes$counter.locations)[, .(
  lon = coord_X, lat = coord_Y,
  nom_comptage=iconv(nom_comptage, "latin1", "UTF-8"))])
```

```
          lon      lat              nom_comptage
 1: -73.58888 45.51955              Saint-Urbain
 2: -73.57398 45.52741                   Brebeuf
---
20: -73.58221 45.51370          Parc U-Zelt Test
21: -73.60311 45.52782 Saint-Laurent U-Zelt Test
```

Dans la sortie ci-dessus, on voit que la colonne `nom_comptage` indique l'emplacement, mais ce ne sont pas exactement les mêmes valeurs que la colonne `lieu` dans les données de passages. On utilise le code ci-dessous pour établir une correspondence entre les tableaux :

```
loc.name.code <- c(
  Berri1="Berri",
  Brebeuf="Brébeuf",
  CSC="Côte-Sainte-Catherine",
  Maisonneuve_1="Maisonneuve 1",
  Maisonneuve_2="Maisonneuve 2",
  Parc="du Parc",
  PierDup="Pierre-Dupuy",
  "Rachel/Papineau"="Rachel",
  "Saint-Urbain"="Saint-Urbain",
  Totem_Laurier="Totem_Laurier")
(show.locations <- counter.locations[
, lieu := loc.name.code[nom_comptage]
][!is.na(lieu)])
```

```
          lon      lat  nom_comptage          lieu
 1: -73.58888 45.51955  Saint-Urbain  Saint-Urbain
 2: -73.57398 45.52741       Brebeuf       Brébeuf
---
 9: -73.58883 45.52777 Totem_Laurier Totem_Laurier
10: -73.56284 45.51613        Berri1         Berri
```

La sortie ci-dessus contient l'emplacement géographique pour chaque compteur. L'emplacement des compteurs est tracé ci-dessous.

```
map.lim <- show.locations[, lapply(.SD, range), .SDcols=c("lat","lon")]
diff.vec <- sapply(map.lim, diff)
```

```
diff.mat <- c(-1, 1) * matrix(diff.vec, 2, 2, byrow=TRUE)
scale.mat <- as.matrix(map.lim) + diff.mat
bike.paths <- data.table(montreal.bikes$path.locations)
show.paths <- bike.paths[(
  lat %between% scale.mat[, "lat"]
) & (
  lon %between% scale.mat[, "lon"]
)]
(mtl.map <- ggplot()+
   theme_bw()+
   theme(
     panel.margin=grid::unit(0, "lines"),
     axis.line=element_blank(), axis.text=element_blank(),
     axis.ticks=element_blank(), axis.title=element_blank(),
     panel.background = element_blank(),
     panel.border = element_blank())+
   coord_cartesian(xlim=map.lim$lon, ylim=map.lim$lat)+
   scale_x_continuous(limits=map.lim$lon)+
   scale_y_continuous(limits=map.lim$lat)+
   geom_path(aes(
     lon, lat,
     tooltip=TYPE_VOIE,
     group=paste(feature.i, path.i)),
     color="grey",
     data=show.paths)+
   geom_text(aes(
     lon, lat,
     label=lieu),
     clickSelects="lieu",
     data=show.locations))
```

```
Warning: Removed 96 rows containing missing values (geom_path).
```

La sortie ci-dessus est un plan de Montréal, avec texte pour chacun des dix compteurs.

9.3.2 Résumé des dates extrêmes pour chaque compteur

Maintenant on calcule les dates extrêmes pour chaque compteur :

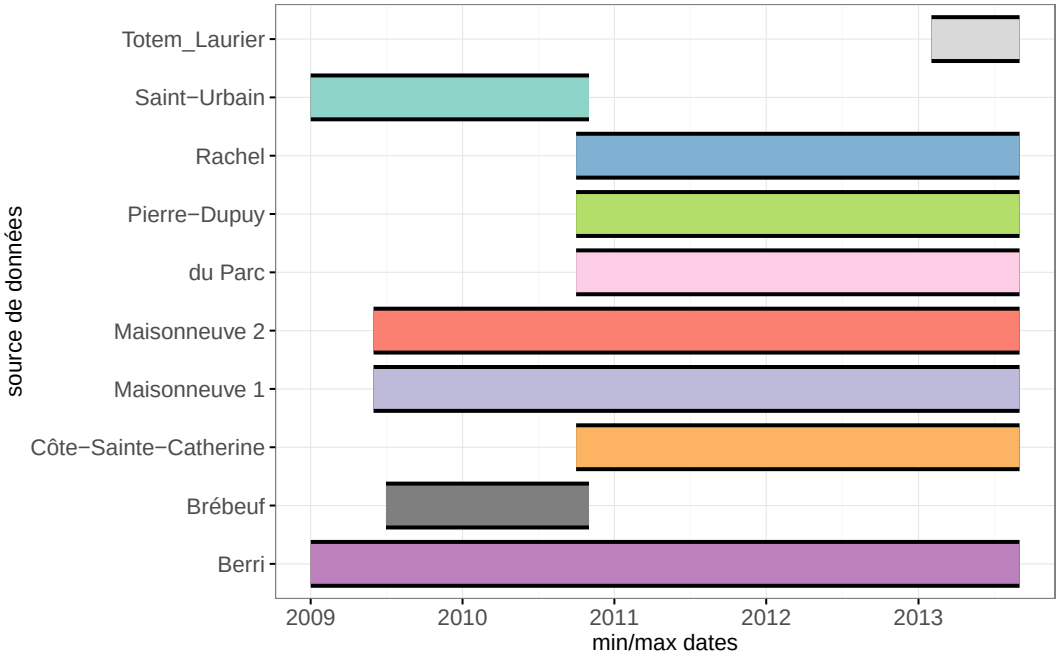
```
(location.ranges <- dcast(
  passages.par.mois[0 < passages_sum][
  , mois.01 := mois_01(mois.str)],
  lieu ~ .,
  list(min, max),
  value.var="mois.01"))
```

```
            lieu mois.01_min mois.01_max
 1:         Berri  2009-01-01  2013-09-01
 2:       Brébeuf  2009-07-01  2010-11-01
---
 9:  Saint-Urbain  2009-01-01  2010-11-01
10: Totem_Laurier  2013-02-01  2013-09-01
```

Dans la sortie ci-dessus, on voit une ligne pour chaque compteur, avec des colonnes pour les dates extrêmes. Le graphique ci-dessous montre la période pendant laquelle chaque compteur a fonctionné.

```
location.colors <- c(#dput(RColorBrewer::brewer.pal(12, "Set3"))
  "#8DD3C7", "grey50", "#BEBADA", "#FB8072", "#80B1D3", "#FDB462",
  "#B3DE69", "#FCCDE5", "#D9D9D9", "#BC80BD", "#CCEBC5", "#FFED6F")
names(location.colors) <- show.locations$lieu
seg.size <- 10
```

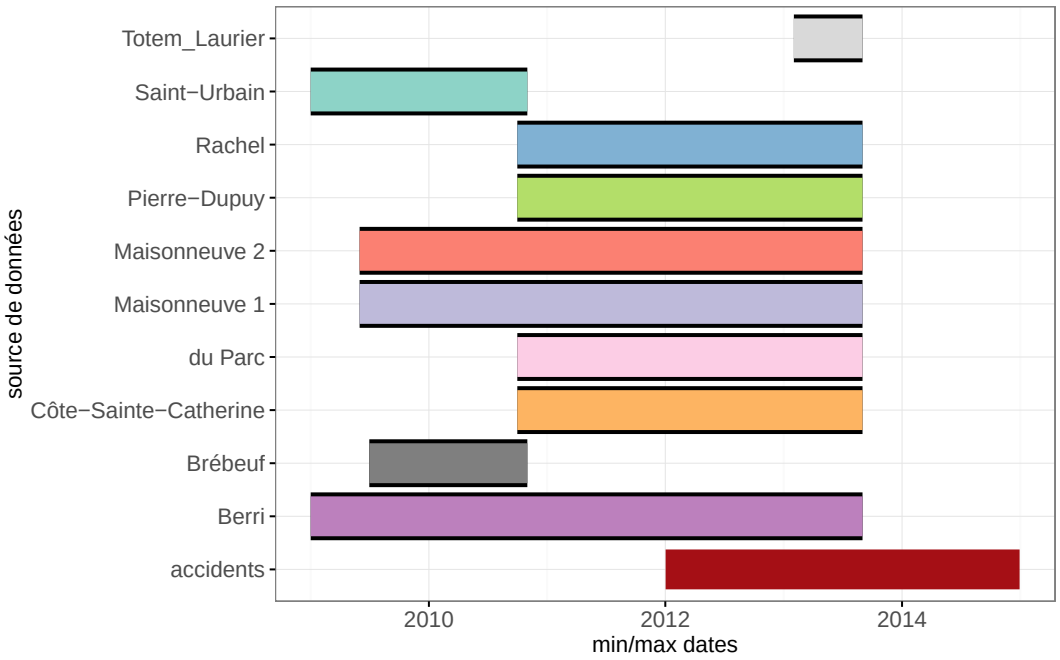
```
(CounterRanges <- ggplot()+
  theme_bw()+
  xlab("min/max dates")+
  ylab("source de données")+
  scale_color_manual(values=location.colors)+
  guides(color="none")+
  geom_segment(aes(
    mois.01_min, lieu,
    xend=mois.01_max, yend=lieu),
    showSelected="lieu",
    data=location.ranges,
    size=seg.size+2)+
  geom_segment(aes(
    mois.01_min, lieu,
    xend=mois.01_max, yend=lieu,
    color=lieu),
    clickSelects="lieu",
    data=location.ranges,
    size=seg.size))
```



La sortie ci-dessus contient un segment pour chaque compteur. Avec le code ci-dessous, nous rajoutons un segment pour les accidents.

```
accidents.range <- dcast(
  data.table(lieu="accidents", accidents_dt),
  lieu ~ .,
  list(min, max),
  value.var="date")
```

```
(MonthSummary <- CounterRanges+
  geom_segment(aes(
    date_min, lieu,
    xend=date_max, yend=lieu),
    color=gravité.colors[["décès"]],
    data=accidents.range,
    size=seg.size))
```



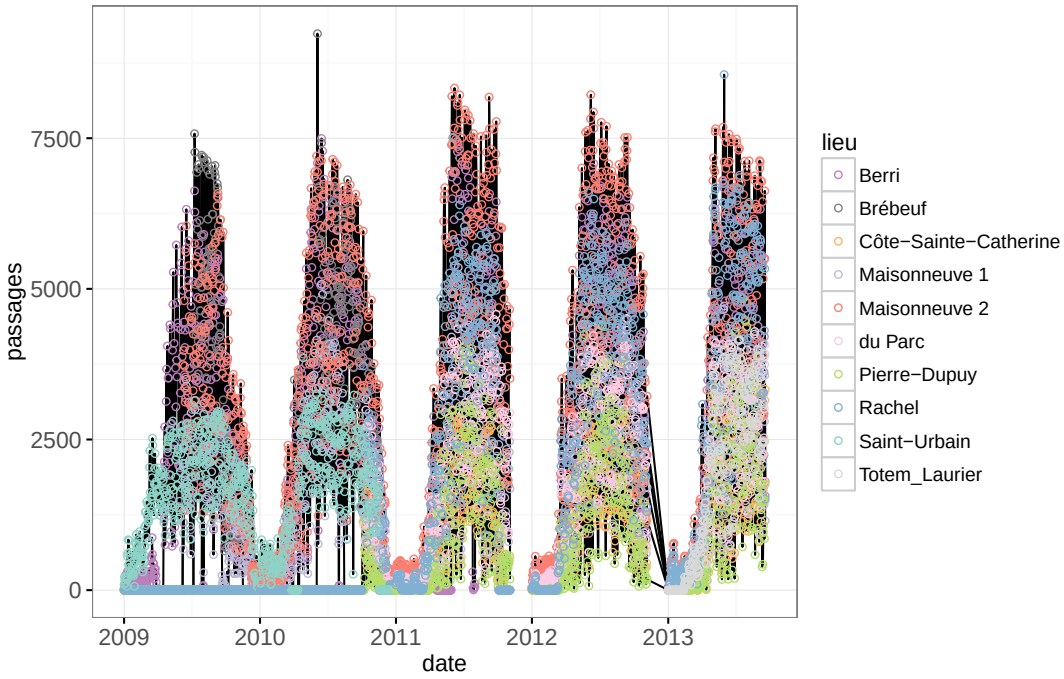
Dans la sortie ci-dessus, on voit un segment de plus (pour les accidents en bas).

9.3.3 Séries temporelles par mois

Le graphique ci-dessous présente le comptage de vélos à chaque localisation, chaque jour.

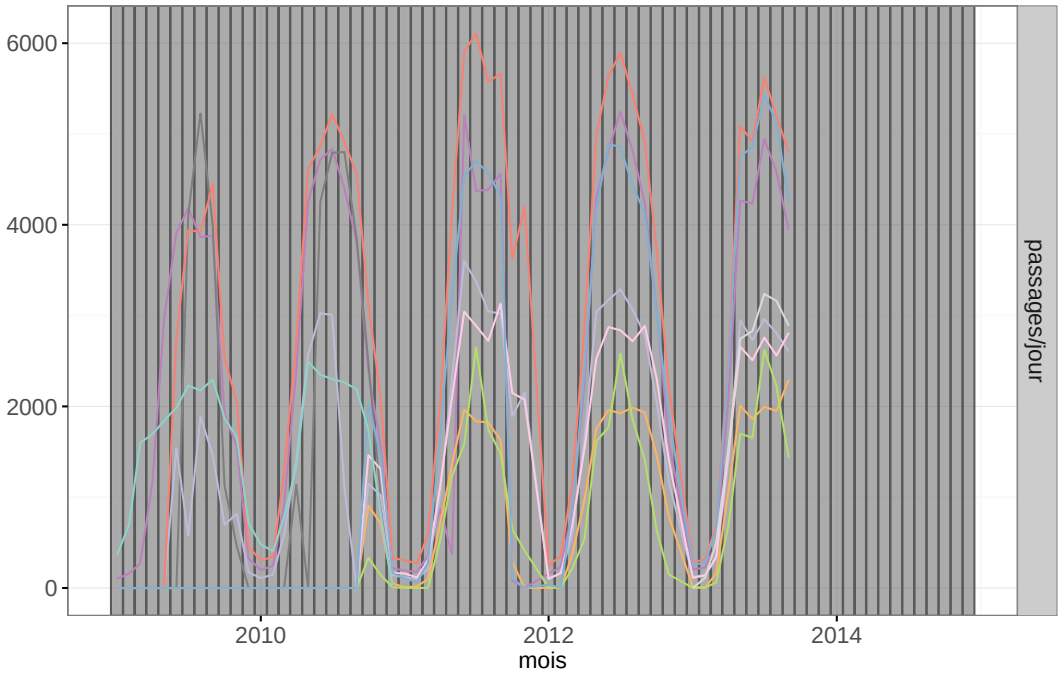
```
ggplot()+
  theme_bw()+
  geom_line(aes(
    date, passages, group=lieu),
    data=passages_dt)+
  scale_color_manual(values=location.colors)+
  geom_point(aes(
    date, passages, color=lieu),
    data=passages_dt)
```

```
Warning: Removed 407 rows containing missing values (geom_point).
```



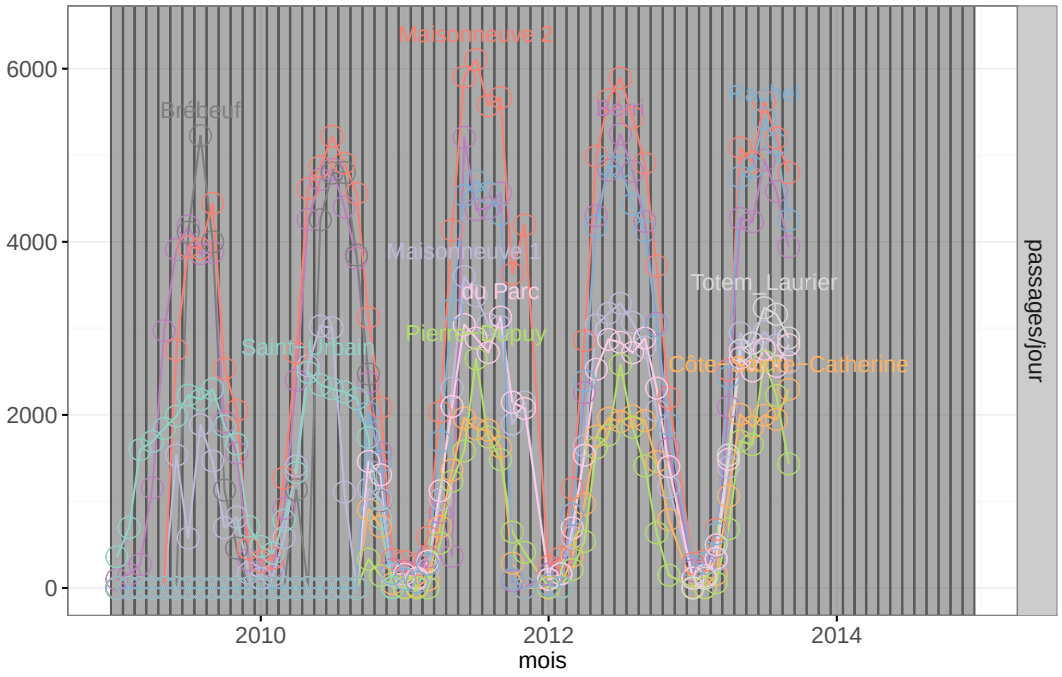
Le graphique ci-dessous reprend les mêmes données mais pour chaque mois.

```
FACET <- function(DT, facet)data.table(DT, facet)
COMPTEURS <- function(DT)FACET(DT, "passages/jour")
(MonthSeries <- ggplot()+
  guides(color="none")+
  theme_bw()+
  facet_grid(facet ~ ., scales="free")+
  geom_tallrect(aes(
    xmin=mois.01-15*un.jour, xmax=mois.01+15*un.jour),
    clickSelects="mois.français",
    data=mois_dt,
    alpha=1/2)+
  geom_line(aes(
    mois_01(mois.str), passages_mean, group=lieu,
    color=lieu),
    showSelected="lieu",
    clickSelects="lieu",
    data=COMPTEURS(passages.par.mois))+
  scale_color_manual(values=location.colors)+
  xlab("mois")+
  ylab(""))
```



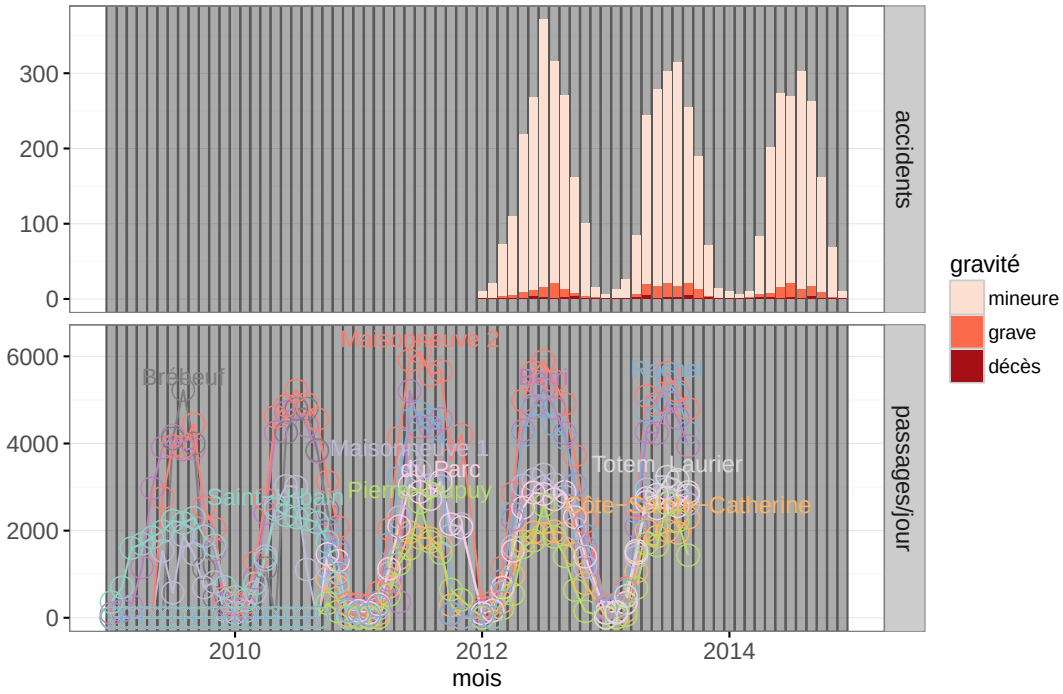
Le graphique ci-dessus contient une ligne pour chaque compteur. Dans le code ci-dessous, on rajoute deux geoms.

```
mois.text <- passages.par.mois[
, .SD[which.max(passages_mean)]
, by=lieu]
(MonthText <- MonthSeries+
  geom_point(aes(
    mois_01(mois.str), passages_mean, color=lieu,
    tooltip=paste(
      passages_mean, "vélos à",
      lieu, "en", mois_français(mois_01(mois.str)))),
    showSelected="lieu",
    clickSelects="lieu",
    size=5,
    data=COMPTEURS(passages.par.mois))+
  geom_text(aes(
    mois_01(mois.str), passages_mean+300,
    color=lieu, label=lieu),
    showSelected="lieu",
    clickSelects="lieu",
    data=COMPTEURS(mois.text)))
```



Le graphique ci-dessous rajoute les accidents.

```
ACCIDENTS <- function(DT)FACET(DT, "accidents")
(MonthFacet <- MonthText+
   facet_grid(facet ~ ., scales="free")+
   scale_fill_manual(
     values=gravité.colors, breaks=names(gravité.colors))+
   geom_bar(aes(
     mois_01(mois.str), personnes,
     fill=gravité),
     showSelected="gravité",
     stat="identity",
     position="identity",
     color=NA,
     data=ACCIDENTS(accidents.tall[order(-gravité)])))
```



Dans la sortie ci-dessus, on voit les deux sources de données (accidents et compteurs).

9.3.4 Détails pour un mois

Dans cette partie, nous voulons faire un graphique des accidents pour chaque mois, avec un point pour chaque personne qui a eu un accident. Ci-dessous, nous classons la gravité de chaque accident en fonction du résultat le plus grave pour les personnes touchées.

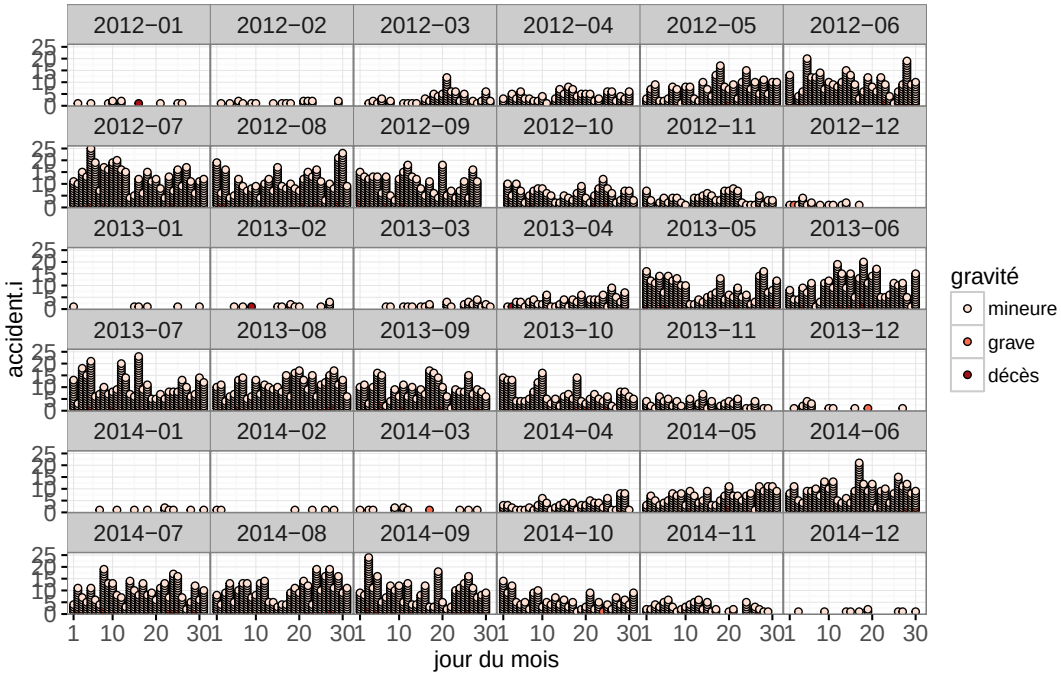
```
accidents_dt[, gravité.str := fcase(
  0 < décès, "décès",
  0 < grave, "grave",
  default="mineure"
)][
, gravité := factor(gravité.str, names(gravité.colors))
][, table(gravité)]
```

```
gravité
mineure   grave   décès
   5262     289      44
```

Le résultat ci-dessus montre que les accidents avec des blessures mineures sont les plus fréquents et que les accidents avec au moins un décès sont les moins fréquents. Dans le code ci-dessous, nous faisons une colonne `accident.i` qui donne un numéro unique pour chaque accident dans chaque jour.

```
jour_du_mois <- function(POSIXct)as.integer(strftime(POSIXct, "%d"))
add_jour_mois <- function(DT)DT[, let(
  jour.du.mois = jour_du_mois(date),
```

```
  mois.français = mois_français(date))]
accidents.cumsum <- add_jour_mois(accidents_dt[
  order(date, -gravité)
][
, accident.i := seq_len(.N)
, by=date
])
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_wrap("mois.str")+
  scale_fill_manual(values=gravité.colors)+
  scale_x_continuous("jour du mois", breaks=c(1, 10, 20, 30))+
  geom_point(aes(
    jour.du.mois, accident.i, fill=gravité),
    data=accidents.cumsum)
```



Dans la sortie ci-dessus, on voit un point pour chaque accident. Ensuite, on fait une grille de jours dans le code ci-dessous.

```
(days.dt <- mois_dt[, .(date=seq(
  min(mois.01),
  max(mois_suivant(mois.01)),
  by="day"
))][#jour de la semaine :
, jds := strftime(date, "%a")
][])
```

```
           date jds
   1: 2009-01-01 jeu
   2: 2009-01-02 ven
  ---
2191: 2014-12-31 mer
2192: 2015-01-01 jeu
```

La sortie ci-dessus contient une ligne pour chaque jour dans la période où nous avons des données. Dans le code ci-dessous, on fait un tableau pour souligner les fins de semaine.

```
(weekend.dt <- add_jour_mois(days.dt[
  grepl("sam|dim", jds)#windows="sam." ubuntu="sam"
])[])
```

```
          date jds jour.du.mois mois.français
  1: 2009-01-03 sam            3  janvier 2009
  2: 2009-01-04 dim            4  janvier 2009
 ---
625: 2014-12-27 sam           27 décembre 2014
626: 2014-12-28 dim           28 décembre 2014
```

La sortie ci-dessus contient une ligne pour chaque jour de fin de semaine. Ensuite, on fait un tableau qu'on va utiliser pour afficher le nom de chaque lieu.

```
add_jour_mois(passages_dt)
(jour.text <- passages_dt[
, .SD[which.max(passages)]
, by=.(lieu, mois.français)])
```

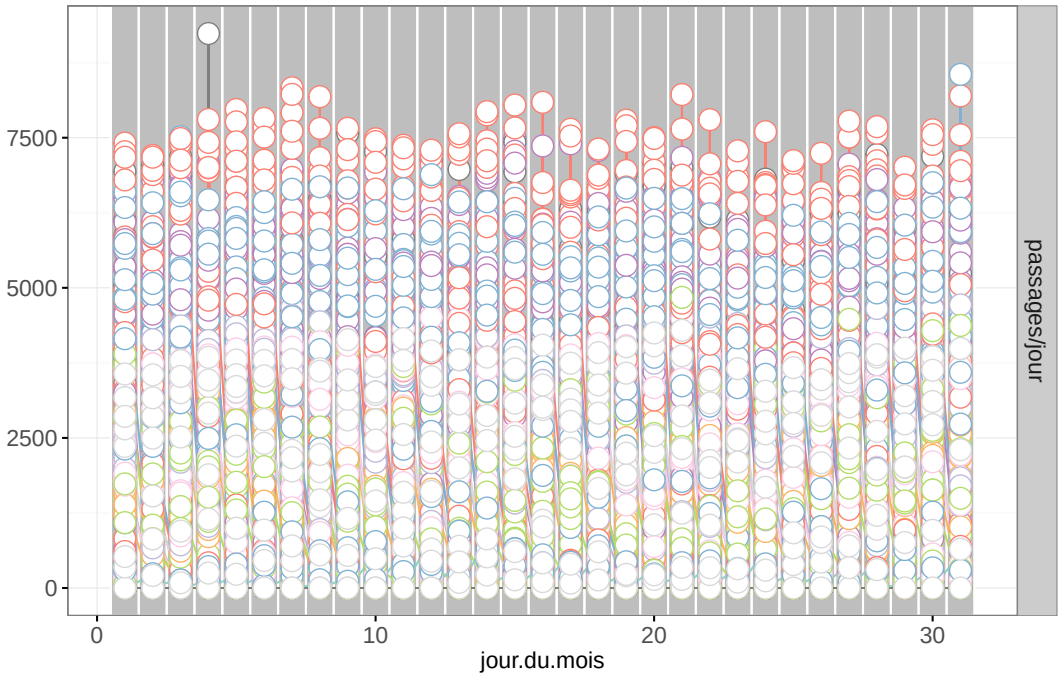
```
             lieu  mois.français                date mois.str passages
  1:         Berri   janvier 2009 2009-01-11 05:00:00  2009-01      318
  2:         Berri   février 2009 2009-02-18 05:00:00  2009-02      326
 ---
441: Totem_Laurier      août 2013 2013-08-21 04:00:00  2013-08     4293
442: Totem_Laurier septembre 2013 2013-09-18 04:00:00  2013-09     3921
         lieu.lines jour.du.mois
  1:          Berri           11
  2:          Berri           18
 ---
441: Totem\nLaurier           21
442: Totem\nLaurier           18
```

La sortie ci-dessus contient le jour avec le plus de passages, pour chaque lieu et mois. Ensuite, le code ci-dessous fait un graphique de passages par jour sur les compteurs.

```
(DaysCompteurs <- ggplot()+
  geom_tallrect(aes(
    xmin=jour.du.mois-0.5, xmax=jour.du.mois+0.5,
    key=paste(date)),
    showSelected="mois.français",
    fill="grey",
```

```
      color="white",
      data=weekend.dt)+
    guides(color="none", fill="none")+
    theme_bw()+
    facet_grid(facet ~ ., scales="free")+
    geom_line(aes(
      jour.du.mois, passages, group=lieu,
      key=lieu, color=lieu),
      showSelected=c("lieu", "mois.français"),
      clickSelects="lieu",
      chunk_vars=c("mois.français"),
      data=COMPTEURS(passages_dt))+
    scale_color_manual(values=location.colors)+
    ylab("")+
    geom_point(aes(
      jour.du.mois, passages, color=lieu,
      key=paste(jour.du.mois, lieu),
      tooltip=paste(
        passages, "cyclistes à",
        lieu, "en",
        date)),
      showSelected=c("lieu", "mois.français"),
      clickSelects="lieu",
      size=5,
      chunk_vars=c("mois.français"),
      fill="white",
      data=COMPTEURS(passages_dt)))
```

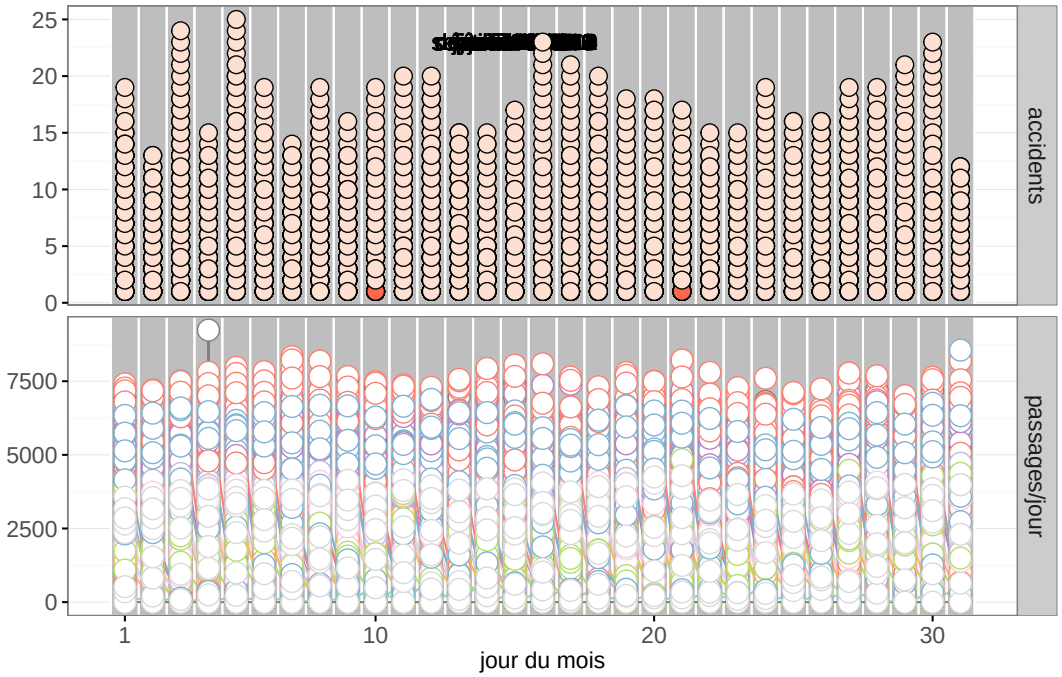
```
Warning: Removed 407 rows containing missing values (geom_point).
```



La sortie ci-dessus affiche les données des compteurs, avec trop de données car le graphique statique ne prend pas en compte du mot-clé `showSelected`. Le code ci-dessous rajoute les données d'accidents.

```
(DaysFacet <- DaysCompteurs+
   scale_fill_manual(
     values=gravité.colors, breaks=names(gravité.colors))+
   geom_text(aes(
     15, 23, label=mois.français, key=1),
     showSelected="mois.français",
     data=ACCIDENTS(mois_dt))+
   scale_x_continuous("jour du mois", breaks=c(1, 10, 20, 30))+
   geom_point(aes(
     jour.du.mois, accident.i,
     key=paste(date.str, accident.i),
     fill=gravité),
     showSelected=c("gravité","mois.français"),
     size=4,
     chunk_vars=c("mois.français"),
     data=ACCIDENTS(accidents.cumsum)))
```

```
Warning: Removed 407 rows containing missing values (geom_point).
```

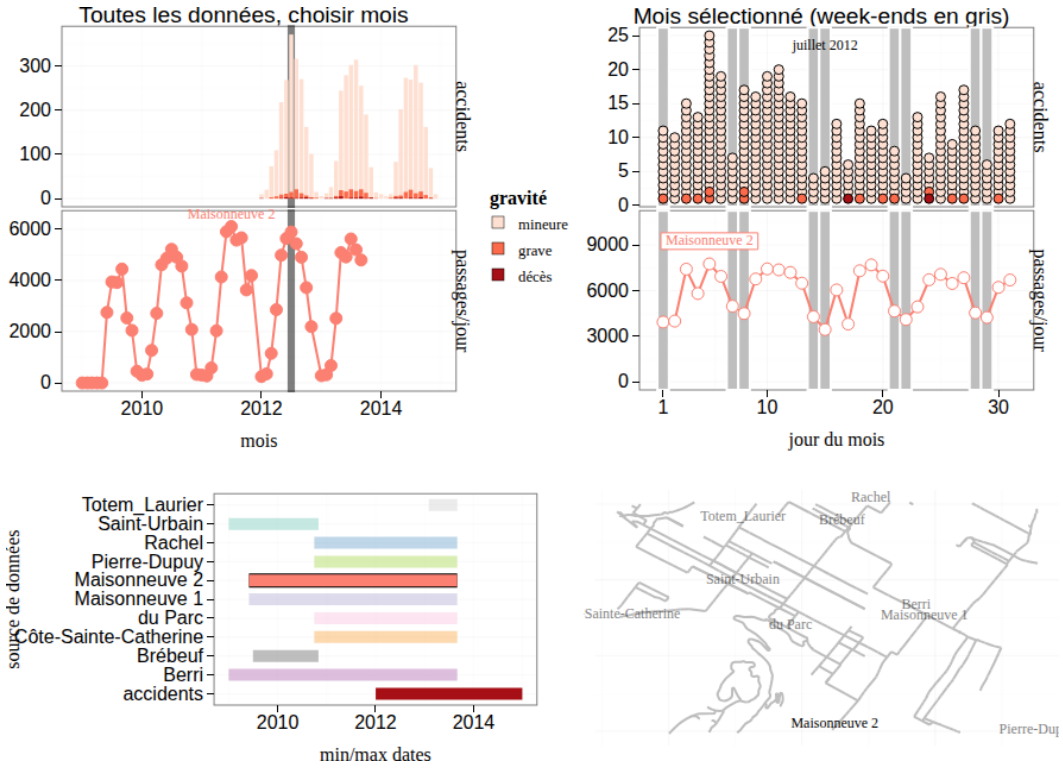


La sortie ci-dessus contient les deux séries temporelles (accidents et compteurs).

9.3.5 Graphique interactif

Enfin, le code ci-dessous combine tous les graphiques.

```
animint(
  MonthFacet+
    ggtitle("Toutes les données, choisir mois"),
  DaysFacet+
    ggtitle("Mois sélectionné (week-ends en gris)")+
    geom_label_aligned(aes(
      jour.du.mois, passages+1500, color=lieu, label=lieu,
      key=lieu),
      showSelected=c("lieu", "mois.français"),
      clickSelects="lieu",
      data=COMPTEURS(jour.text))+
    theme_animint(last_in_row=TRUE),
  MonthSummary+theme_animint(width=450, height=250),
  mtl.map+theme_animint(height=250),
  selector.types=list(severity="multiple"),
  duration=list(mois.français=2000),
  first=list(
    lieu="Maisonneuve 2",
    mois.français="juillet 2012"))
```



La sortie ci-dessus contient 4 graphiques :

- En haut à gauche : séries temporelles avec résumé pour chaque mois.
- En haut à droite : séries temporelles pour le mois sélectionné.
- En bas à gauche : le min et max dates pour chaque source de données.
- En bas à droite : lieux des compteurs sur le plan de Montréal.

9.4 Résumé du chapitre et exercices

Nous avons vu plusieurs graphiques pour visualiser les séries temporelles en rapport avec les vélos de Montréal.

Exercices :

- Faire `lieu` une variable à sélection multiple.
- Sur la carte, dessinez un cercle avec `aes(color=lieu)` pour chaque compteur, dont la taille varie en fonction des passages dans le mois sélectionné.
- Sur le graphique `MonthSummary`, ajoutez rectangles en arrière-plan pour sélectionner le mois.
- Supprimez le graphique `MonthSummary` et ajoutez une visualisation similaire dans un troisième panneau de l'écran dans le graphique `MonthFacet`. Astuce : utilise `theme_animint(rowspan=2)`.
- Dans `DaysFacet`, utilisez `aes(tooltip)` avec quelques détails pour chaque accident (addresse, nombre de personnes, etc).

Dans le chapitre 10, nous vous expliquerons comment visualiser le modèle d'apprentissage automatique des K-voisins les plus proches.

10

K plus proches voisins

Dans ce chapitre, nous allons explorer plusieurs visualisations des données du classificateur des K plus proches voisins (KPPV).

Plan du chapitre :

- Nous commencerons par la visualisation statique originale des données, repensée sous la forme de deux ggplots affichés par `animint2`. Voici un graphique de l'erreur de validation à 10 divisions et un graphique des prédictions du classificateur 7 plus proches voisins.
- Nous proposons deux refontes. Grâce à la première vous pourrez sélectionner le nombre de voisins utilisés pour les prédictions du modèle
- et grâce à la seconde, le nombre de divisions utilisés pour le calcul de l'erreur de validation croisée.

10.1 Figure statique originale

Notre but dans ce chapitre est de reproduire une version interactive de la figure 13.4 dans le livre Elements of Statistical Learning (**Hastie2009?**). Cette figure se compose de deux graphiques :

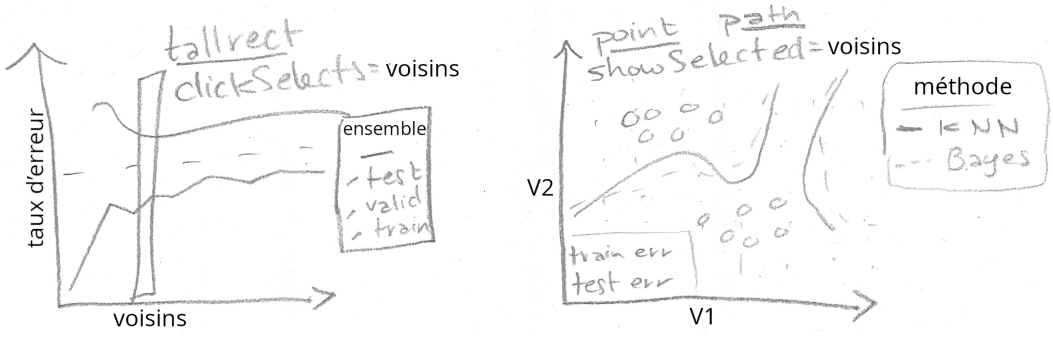


Figure 10.1: Esquisse pour K-Plus-Proches-Voisins

A gauche : courbes d'erreur de mauvaise classification, en fonction du nombre de voisins.

- `geom_line` et `geom_point` pour les courbes d'erreur.
- `geom_linerange` pour les barres d'erreur de la courbe d'erreur de validation.
- `geom_hline` pour l'erreur de Bayes.
- x = voisins.

- y = pourcentage d'erreur.
- couleur = type d'erreur.

À droite : les données et les limites de décision dans l'espace bidimensionnel des variables d'entrée.

- `geom_point` pour les points de données.
- `geom_point` pour les prédictions de classification sur la grille en arrière-plan.
- `geom_path` pour les limites de décision.
- `geom_text` pour les taux d'erreur train/test/Bayes.

10.1.1 Graphique des courbes d'erreurs de mauvaise classification

Nous commençons par charger l'ensemble des données.

```
if(!file.exists("ESL.mixture.rda")){
  curl::curl_download(
    "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda",
    "ESL.mixture.rda")
}
load("ESL.mixture.rda")
str(ESL.mixture)
```

```
List of 8
 $ x       : num [1:200, 1:2] 2.5261 0.367 0.7682 0.6934 -0.0198 ...
 $ y       : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
 $ xnew    : 'matrix' num [1:6831, 1:2] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -
1.9 -1.8 -1.7 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:6831] "1" "2" "3" "4" ...
  .. ..$ : chr [1:2] "x1" "x2"
 $ prob    : num [1:6831] 3.55e-05 3.05e-05 2.63e-05 2.27e-05 1.96e-05 ...
  ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
 $ marginal: num [1:6831] 6.65e-15 2.31e-14 7.62e-14 2.39e-13 7.15e-13 ...
  ..- attr(*, ".Names")= chr [1:6831] "1" "2" "3" "4" ...
 $ px1     : num [1:69] -2.6 -2.5 -2.4 -2.3 -2.2 -2.1 -2 -1.9 -1.8 -1.7 ...
 $ px2     : num [1:99] -2 -1.95 -1.9 -1.85 -1.8 -1.75 -1.7 -1.65 -1.6 -
1.55 ...
 $ means   : num [1:20, 1:2] -0.2534 0.2667 2.0965 -0.0613 2.7035 ...
```

Nous utiliserons les éléments suivants de cet ensemble de données :

- `x`, la matrice des variables d'entrée, de l'ensemble de données d'apprentissage (200 observations sur les lignes x 2 variables numériques sur les colonnes).
- `y`, le vecteur de sortie, de l'ensemble de données d'apprentissage (200 étiquettes de classe, soit 0 ou 1).
- `xnew`, la matrice représentant la grille de points dans l'espace des variables d'entrée, où seront affichées les prédictions de la fonction de classification (6831 points de la grille sur les lignes x 2 variables numériques sur les colonnes).
- `prob`, la probabilité de la classe 1 à chacun des points de la grille (6831 valeurs numériques comprises entre 0 et 1).
- `px1`, la grille de points pour le premier variable d'entrée (69 valeurs numériques comprises entre -2,6 et 4,2). Ces points seront utilisés pour calculer la limite de décision de Bayes à

l'aide de la fonction `contourLines`.

- `px2`, la grille de points pour la deuxième variable d'entrée (99 valeurs numériques comprises entre -2 et 2,9).
- `means`, les 20 centres des distributions normales dans le modèle de simulation (20 centres sur les lignes x 2 variables sur les colonnes).

Tout d'abord, nous créons un ensemble de tests, en suivant le code d'exemple de `help(ESL.mixture)`. Notez que nous utilisons un `data.table` plutôt qu'un `data.frame` pour stocker ces données volumineuses, puisque `data.table` est souvent plus rapide et plus économe en mémoire pour les ensembles de données volumineux.

```
library(MASS)
library(data.table)
set.seed(123)
centers <- c(
  sample(1:10, 5000, replace=TRUE),
  sample(11:20, 5000, replace=TRUE))
mix.test <- mvrnorm(10000, c(0,0), 0.2*diag(2))
test.points <- data.table(
  mix.test + ESL.mixture$means[centers,],
  label=factor(c(rep(0, 5000), rep(1, 5000))))
test.points
```

```
             V1        V2 label
   1:  2.0210959 1.3905124     0
   2:  2.7488414 1.0327241     0
   ---
 9999: -1.9089417 1.6135246     1
10000:  0.7678115 0.3154265     1
```

Nous créons ensuite un tableau de données qui comprend tous les points de test et les points de la grille, que nous utiliserons dans l'argument de test de la fonction KPPV.

```
pred.grid <- data.table(ESL.mixture$xnew, label=NA)
input.cols <- c("V1", "V2")
names(pred.grid)[1:2] <- input.cols
test.and.grid <- rbind(
  data.table(test.points, set="test"),
  data.table(pred.grid, set="grid"))
test.and.grid$fold <- NA
test.and.grid
```

```
           V1       V2 label  set fold
   1: 2.021096 1.390512     0 test   NA
   2: 2.748841 1.032724     0 test   NA
   ---
16830: 4.100000 2.900000  <NA> grid   NA
16831: 4.200000 2.900000  <NA> grid   NA
```

Nous assignons aléatoirement chaque observation de l'ensemble de données d'apprentissage à l'un des 10 divisions.

```
n.folds <- 10
set.seed(2)
mixture <- with(ESL.mixture, data.table(x, label=factor(y)))
mixture$fold <- sample(rep(1:n.folds, l=nrow(mixture)))
mixture
```

```
              V1        V2 label fold
  1:  2.526092968 0.3210504     0    5
  2:  0.366954472 0.0314621     0    8
 ---
199:  0.008130556 2.2422639     1    4
200: -0.196246334 0.5514036     1    8
```

Nous définissons la fonction `OneFold` pour diviser les 200 observations en un ensemble d'entraînement et un ensemble de validation. Elle calcule ensuite la probabilité prédite par le classificateur des K Plus Proches Voisins pour chacun des points de données dans tous les ensembles (entraînement, validation, test et grille).

```
OneFold <- function(validation.fold){
  set <- ifelse(mixture$fold == validation.fold, "validation", "train")
  fold.data <- rbind(test.and.grid, data.table(mixture, set))
  fold.data$data.i <- 1:nrow(fold.data)
  only.train <- subset(fold.data, set == "train")
  data.by.neighbors <- list()
  for(neighbors in seq(1, 30, by=2)){
    if(interactive())cat(sprintf(
      "n.folds=%4d validation.fold=%d neighbors=%d\n",
      n.folds, validation.fold, neighbors))
    set.seed(1)
    pred.label <- class::knn( # random tie-breaking.
      only.train[, input.cols, with=FALSE],
      fold.data[, input.cols, with=FALSE],
      only.train$label,
      k=neighbors,
      prob=TRUE)
    prob.winning.class <- attr(pred.label, "prob")
    fold.data$probability <- ifelse(
      pred.label=="1", prob.winning.class, 1-prob.winning.class)
    fold.data[, pred.label := ifelse(0.5 < probability, "1", "0")]
    fold.data[, is.error := label != pred.label]
    fold.data[, prediction := ifelse(is.error, "erronée", "correcte")]
    data.by.neighbors[[paste(neighbors)]] <-
      data.table(neighbors, fold.data)
  }#for(neighbors
  do.call(rbind, data.by.neighbors)
}#for(validation.fold
```

Ci-dessous, nous exécutons la fonction `OneFold` en parallèle à l'aide du package `future`. Pour `validation.fold` de 1 à 10, on calcule l'erreur de l'ensemble de validation. Pour `validation.fold=0`, on traite l'ensemble des 200 observations comme un ensemble d'entraînement, qui sera utilisé pour visualiser les frontières de décision apprises avec

K Plus Proches Voisins.

```
future::plan("multisession")
data.all.folds.list <- future.apply::future_lapply(
  0:n.folds, function(validation.fold){
    one.fold <- OneFold(validation.fold)
    data.table(validation.fold, one.fold)
  }, future.seed = NULL)
(data.all.folds <- do.call(rbind, data.all.folds.list))
```

```
        validation.fold neighbors           V1        V2 label   set fold
     1:               0         1  2.021095933 1.3905124     0  test   NA
     2:               0         1  2.748841354 1.0327241     0  test   NA
    ---
2810114:              10        29  0.008130556 2.2422639     1 train    4
2810115:              10        29 -0.196246334 0.5514036     1 train    8
        data.i probability pred.label is.error prediction
     1:      1   0.0000000          0    FALSE   correcte
     2:      2   0.0000000          0    FALSE   correcte
    ---
2810114:  17030   0.7586207          1    FALSE   correcte
2810115:  17031   0.4137931          0     TRUE    erronée
```

Le tableau de données des prédictions contient près de 3 millions d'observations ! Lorsqu'il y a autant de données, les visualiser toutes en même temps n'est ni pratique ni informatif. Au lieu de les visualiser simultanément, nous allons calculer et tracer des statistiques sommaires. Dans le code ci-dessous, nous calculons la moyenne et l'erreur standard de l'erreur de mauvaise classification pour chaque modèle (sur les 10 divisions dans la validation croisée). Ceci est un exemple de l'idiome summarize data table qui est généralement utile pour calculer des statistiques sommaires pour un tableau de données unique.

```
labeled.data <- data.all.folds[!is.na(label),]
error.stats <- labeled.data[, list(
  error.prop=mean(is.error)
  ), by=.(set, validation.fold, neighbors)]
validation.error <- error.stats[set=="validation", list(
  mean=mean(error.prop),
  sd=sd(error.prop)/sqrt(.N)
  ), by=.(set, neighbors)]
validation.error
```

```
          set neighbors  mean         sd
 1: validation         1 0.240 0.01943651
 2: validation         3 0.165 0.02362908
---
14: validation        27 0.195 0.02034426
15: validation        29 0.205 0.02291288
```

Nous construisons ci-dessous des tableaux de données pour l'erreur d'entraînement et pour l'erreur de Bayes (nous savons qu'elle est de 0,21 pour les données de l'exemple de mélange).

```
  Bayes.error <- data.table(
    set="Bayes",
    validation.fold=NA,
    neighbors=NA,
    error.prop=0.21)
  Bayes.error
```

```
    set validation.fold neighbors error.prop
1: Bayes              NA        NA       0.21
```

```
  other.error <- error.stats[validation.fold==0,]
  head(other.error)
```

```
     set validation.fold neighbors error.prop
 1:  test               0         1     0.2938
 2: train               0         1     0.0000
---
 5:  test               0         5     0.2273
 6: train               0         5     0.1300
```

Ci-dessous, nous construisons une palette de couleurs à partir de `dput(RColorBrewer::brewer.pal(Inf, "Set1"))` et des palettes de types de lignes (`linetype`).

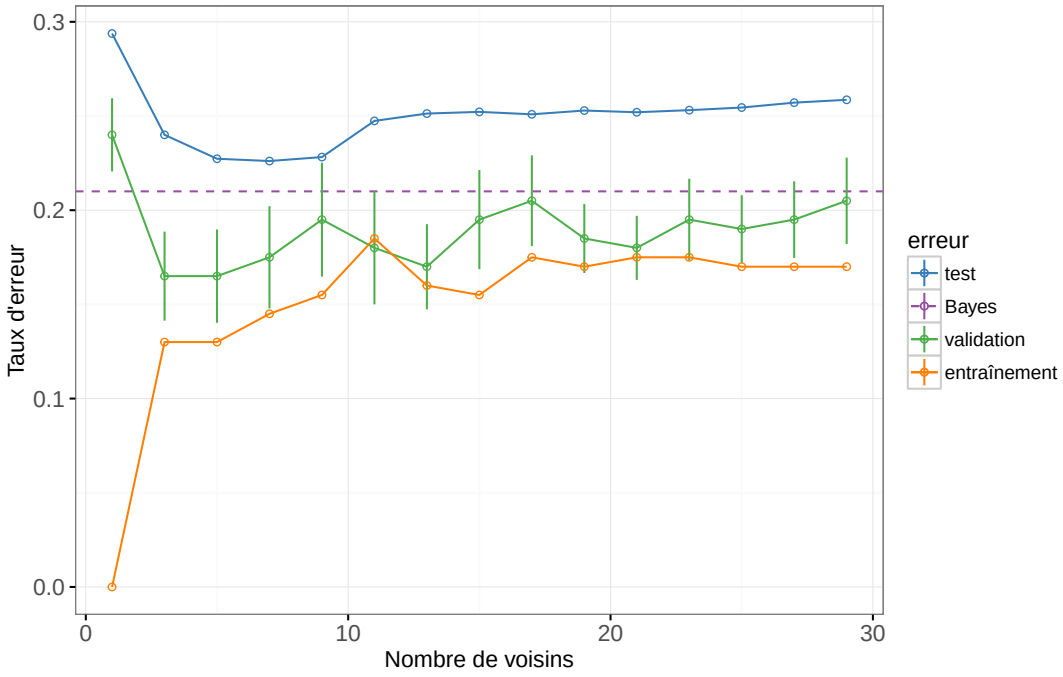
```
  set.colors <- c(
    test="#377EB8", #blue
    Bayes="#984EA3",#purple
    validation="#4DAF4A",#green
    entraînement="#FF7F00")#orange
  classifier.linetypes <- c(
    Bayes="dashed",
    KPPV="solid")
  set.linetypes <- set.colors
  set.linetypes[] <- classifier.linetypes[["KPPV"]]
  set.linetypes["Bayes"] <- classifier.linetypes[["Bayes"]]
  cbind(set.linetypes, set.colors)
```

```
            set.linetypes set.colors
test         "solid"       "#377EB8"
Bayes        "dashed"      "#984EA3"
validation   "solid"       "#4DAF4A"
entraînement "solid"       "#FF7F00"
```

Le code ci-dessous reproduit le graphique des courbes d'erreur de la figure originale.

```
  library(animint2)
  add_set_fr <- function(DT)DT[
  , set_fr := ifelse(set=="train", "entraînement", set)]
  add_set_fr(other.error)
  add_set_fr(validation.error)
```

```
add_set_fr(Bayes.error)
legend.name.type <- "erreur"
errorPlotStatic <- ggplot()+
  theme_bw()+
  geom_hline(aes(
    yintercept=error.prop, color=set_fr, linetype=set_fr),
    data=Bayes.error)+
  scale_color_manual(
    legend.name.type, values=set.colors, breaks=names(set.colors))+
  scale_linetype_manual(
    legend.name.type, values=set.linetypes, breaks=names(set.linetypes))+
  ylab("Taux d'erreur")+
  xlab("Nombre de voisins")+
  geom_linerange(aes(
    neighbors, ymin=mean-sd, ymax=mean+sd,
    color=set_fr),
    data=validation.error)+
  geom_line(aes(
    neighbors, mean, linetype=set_fr, color=set_fr),
    data=validation.error)+
  geom_line(aes(
    neighbors, error.prop,
    group=set_fr, linetype=set_fr, color=set_fr),
    data=other.error)+
  geom_point(aes(
    neighbors, mean, color=set_fr),
    data=validation.error)+
  geom_point(aes(
    neighbors, error.prop, color=set_fr),
    data=other.error)
errorPlotStatic
```



10.1.2 Graphique des limites de décision dans l'espace des variables d'entrée.

Pour la visualisation statique des données de l'espace des variables, nous ne montrons que le modèle avec 7 voisins.

```
show.neighbors <- 7
show.data <- data.all.folds[validation.fold==0 & neighbors==show.neighbors,]
show.points <- show.data[set=="train",]
show.points
```

```
     validation.fold neighbors           V1        V2 label   set fold data.i
  1:               0         7  2.526092968 0.3210504     0 train    5  16832
  2:               0         7  0.366954472 0.0314621     0 train    8  16833
 ---
199:               0         7  0.008130556 2.2422639     1 train    4  17030
200:               0         7 -0.196246334 0.5514036     1 train    8  17031
     probability pred.label is.error prediction
  1:   0.1428571          0    FALSE   correcte
  2:   0.1428571          0    FALSE   correcte
 ---
199:   0.8571429          1    FALSE   correcte
200:   0.2857143          0     TRUE    erronée
```

Ensuite, nous calculons les taux d'erreur de mauvaise classification, que nous afficherons en bas à gauche du graphique de l'espace des variables.

```
  text.height <- 0.25
  text.V1.prop <- 0
  text.V2.bottom <- -2
  text.V1.error <- -2.6
  error.text <- rbind(
    Bayes.error,
    other.error[neighbors==show.neighbors,])
  error.text[, V2.top := text.V2.bottom + text.height * (1:.N)]
  error.text[, V2.bottom := V2.top - text.height]
  error.text
```

```
     set validation.fold neighbors error.prop       set_fr V2.top V2.bottom
1: Bayes              NA        NA     0.2100        Bayes  -1.75     -2.00
2:  test               0         7     0.2261         test  -1.50     -1.75
3: train               0         7     0.1450 entraînement  -1.25     -1.50
```

Nous définissons la fonction suivante que nous utiliserons pour calculer les limites de décision.

```
  getBoundaryDF <- function(prob.vec){
    stopifnot(length(prob.vec) == 6831)
    several.paths <- with(ESL.mixture, contourLines(
      px1, px2,
      matrix(prob.vec, length(px1), length(px2)),
      levels=0.5))
    contour.list <- list()
    for(path.i in seq_along(several.paths)){
      contour.list[[path.i]] <- with(several.paths[[path.i]], data.table(
        path.i, V1=x, V2=y))
    }
    do.call(rbind, contour.list)
  }
```

Nous utilisons cette fonction pour calculer les limites de décision pour les 7 plus proches voisins et pour la fonction optimale de Bayes.

```
  boundary.grid <- show.data[set=="grid",]
  boundary.grid[, label := pred.label]
  pred.boundary <- getBoundaryDF(boundary.grid$probability)
  pred.boundary$classifier <- "KPPV"
  Bayes.boundary <- getBoundaryDF(ESL.mixture$prob)
  Bayes.boundary$classifier <- "Bayes"
  Bayes.boundary
```

```
     path.i        V1        V2 classifier
  1:      1 -2.600000 -0.528615      Bayes
  2:      1 -2.557084 -0.500000      Bayes
 ---
249:      2  3.022480  2.850000      Bayes
250:      2  3.028586  2.900000      Bayes
```

Ci-dessous, nous ne considérons que les points de la grille qui ne chevauchent pas les étiquettes

de texte.

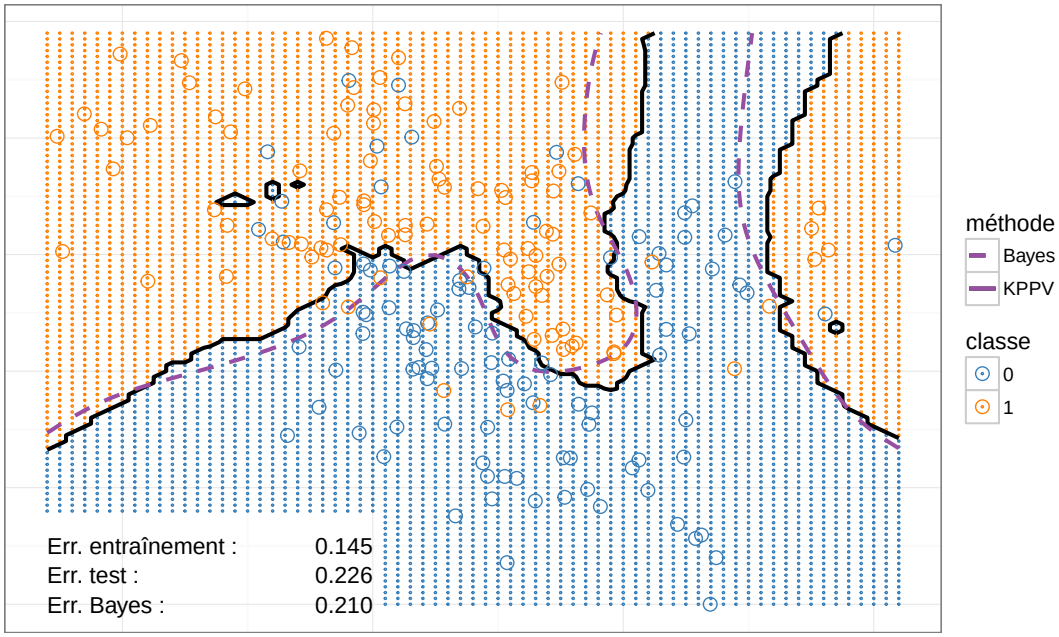
```
on.text <- function(V1, V2){
  V2 <= max(error.text$V2.top) & V1 <= text.V1.prop
}
show.grid <- boundary.grid[!on.text(V1, V2),]
show.grid
```

```
     validation.fold neighbors  V1   V2 label  set fold data.i probability
   1:               0         7 0.1 -2.0     0 grid   NA  10028   0.0000000
   2:               0         7 0.2 -2.0     0 grid   NA  10029   0.0000000
  ---
6398:               0         7 4.1  2.9     1 grid   NA  16830   0.5714286
6399:               0         7 4.2  2.9     1 grid   NA  16831   0.5714286
     pred.label is.error prediction
   1:          0       NA       <NA>
   2:          0       NA       <NA>
  ---
6398:          1       NA       <NA>
6399:          1       NA       <NA>
```

Le nuage de points ci-dessous reproduit le classificateur des 7 Plus Proches Voisins de la figure originale.

```
label.colors <- c(
  "0"="#377EB8",
  "1"="#FF7F00")
scatterPlotStatic <- ggplot()+
  theme_bw()+
  theme(axis.text=element_blank(),
        axis.ticks=element_blank(),
        axis.title=element_blank())+
  ggtitle("7-Plus proches voisins")+
  scale_color_manual(
    "classe",
    values=label.colors)+
  scale_linetype_manual(
    "méthode",
    values=classifier.linetypes)+
  geom_point(aes(
    V1, V2, color=label),
    size=0.2,
    data=show.grid)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    size=1,
    data=pred.boundary)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    color=set.colors[["Bayes"]],
```

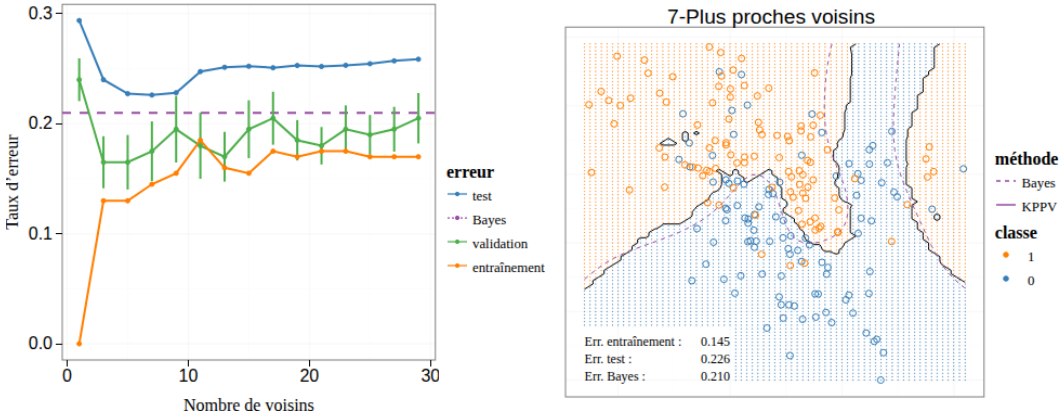
```
    size=1,
    data=Bayes.boundary)+
  geom_point(aes(
    V1, V2, color=label),
    fill=NA,
    size=3,
    shape=21,
    data=show.points)+
  geom_text(aes(
    text.V1.error, V2.bottom,
    label=paste("Err.", set_fr, ":")),
    data=error.text,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, V2.bottom, label=sprintf("%.3f", error.prop)),
    data=error.text,
    hjust=1)
scatterPlotStatic
```



10.1.3 Graphiques combinés

Enfin, nous combinons les deux `ggplots` et les affichons sous forme d'`animint2`.

```
animint(errorPlotStatic, scatterPlotStatic)
```



Bien que cette visualisation des données comporte trois légendes interactives, elle est statique dans le sens où elle n'affiche que les prédictions du modèle des 7 Plus Proches Voisins.

10.2 Sélectionner le nombre de voisins à l'aide de l'interactivité

Dans cette section, nous proposons une visualisation interactive qui permet à l'utilisateur de sélectionner K, le nombre de voisins.

10.2.1 Graphique interactif des courbes d'erreur

Examinons d'abord la refonte du graphique des courbes d'erreur.

Notez les changements suivants :

- ajout d'un sélecteur pour le nombre de voisins (`geom_tallrect`).
- changement de la limite de décision de Bayes de `geom_hline` avec une entrée de légende, vers un `geom_segment` avec une étiquette de texte.
- ajout d'une légende de type de ligne pour distinguer les taux d'erreur des modèles de Bayes et de KPPV.
- changement des barres d'erreur (`geom_linerange`) en bandes d'erreur (`geom_ribbon`).

Les seules nouvelles données que nous devons définir sont les points d'extrémité du segment que nous utiliserons pour tracer la frontière de décision de Bayes. À noter que nous redéfinissons également l'ensemble "test" pour souligner que l'erreur de Bayes représente le meilleur taux d'erreur atteignable pour les données de test.

```
Bayes.segment <- data.table(
  Bayes.error,
  classifier="Bayes",
  min.neighbors=1,
  max.neighbors=29)
Bayes.segment$set_fr <- "test"
```

Nous ajoutons également aux tableaux de données une variable d'erreur qui contient l'erreur de prédiction des modèles de type KPPV. Cette variable d'erreur sera utilisée pour la légende du type de ligne.

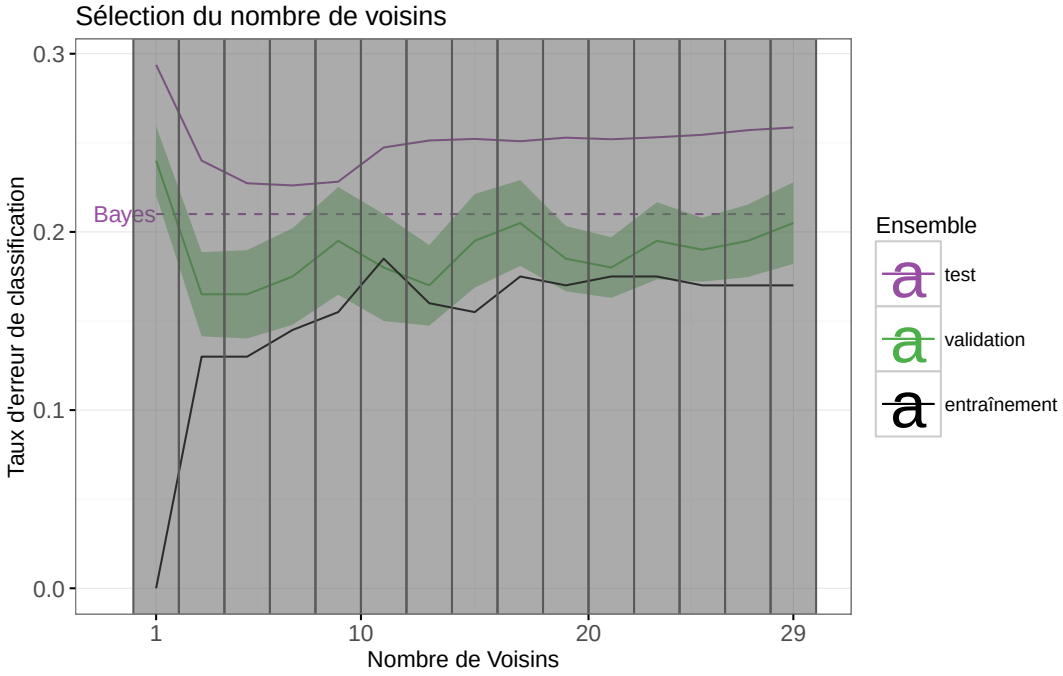
```
validation.error$classifier <- "KPPV"
other.error$classifier <- "KPPV"
```

Nous redéfinissons le graphique des courbes d'erreur ci-dessous. À noter que :

- Nous utilisons `showSelected` dans `geom_text` et `geom_ribbon` afin qu'ils soient masqués lorsque l'on clique sur les légendes interactives.
- Nous utilisons `clickSelects` dans `geom_tallrect` pour sélectionner le nombre de voisins. Les geoms cliquables doivent être placés en dernier (couche supérieure) afin de ne pas être masqués par les geoms non cliquables (couches inférieures).

```
set.colors <- c(
  test="#984EA3",#purple
  validation="#4DAF4A",#green
  Bayes="#984EA3",#purple
  entraînement="black")
legend.name <- "Ensemble"
errorPlot <- ggplot()+
  ggtitle("Sélection du nombre de voisins")+
  theme_bw()+
  theme_animint(height=500)+
  geom_text(aes(
    min.neighbors, error.prop,
    color=set_fr, label="Bayes"),
    showSelected="classifier",
    hjust=1,
    data=Bayes.segment)+
  geom_segment(aes(
    min.neighbors, error.prop,
    xend=max.neighbors, yend=error.prop,
    color=set_fr,
    linetype=classifier),
    showSelected="classifier",
    data=Bayes.segment)+
  scale_color_manual(
    legend.name,
    values=set.colors, breaks=names(set.colors))+
  scale_fill_manual(
    legend.name,
    values=set.colors)+
  scale_linetype_manual(
    legend.name,
    values=classifier.linetypes)+
  guides(fill="none", linetype="none")+
  ylab("Taux d'erreur de classification")+
  scale_x_continuous(
    "Nombre de Voisins",
    limits=c(-1, 30),
    breaks=c(1, 10, 20, 29))+
  geom_ribbon(aes(
```

```
      neighbors, ymin=mean-sd, ymax=mean+sd,
      fill=set_fr),
      showSelected=c("classifier", "set_fr"),
      alpha=0.5,
      color="transparent",
      data=validation.error)+
    geom_line(aes(
      neighbors, mean, color=set_fr,
      linetype=classifier),
      showSelected="classifier",
      data=validation.error)+
    geom_line(aes(
      neighbors, error.prop, group=set_fr, color=set_fr,
      linetype=classifier),
      showSelected="classifier",
      data=other.error)+
    geom_tallrect(aes(
      xmin=neighbors-1, xmax=neighbors+1),
      clickSelects="neighbors",
      alpha=0.5,
      data=validation.error)
  errorPlot
```



10.2.2 Graphique de l'espace des éléments montrant le nombre de voisins sélectionnés.

Concentrons nous ensuite sur la refonte du graphique de l'espace des caractéristiques. Dans la section précédente, nous n'avons considéré que le sous-ensemble de données du modèle à 7

voisins. Notre refonte comprend les changements suivants :

- Nous utilisons les voisins comme variable `showSelected`.
- Nous ajoutons une légende pour indiquer les points de données d'entraînement mal classés.
- Nous utilisons des coordonnées à espacement égal afin que la distance visuelle (pixels) soit la même que la distance euclidienne dans l'espace des variables.

```
show.data <- data.all.folds[validation.fold==0,]
show.points <- show.data[set=="train",]
show.points
```

```
      validation.fold neighbors           V1        V2 label   set fold data.i
   1:               0         1  2.526092968 0.3210504     0 train    5  16832
   2:               0         1  0.366954472 0.0314621     0 train    8  16833
  ---
2999:               0        29  0.008130556 2.2422639     1 train    4  17030
3000:               0        29 -0.196246334 0.5514036     1 train    8  17031
      probability pred.label is.error prediction
   1:   0.0000000          0    FALSE   correcte
   2:   0.0000000          0    FALSE   correcte
  ---
2999:   0.7586207          1    FALSE   correcte
3000:   0.3793103          0     TRUE    erronée
```

Ci-dessous, nous calculons les limites de décision prédites séparément pour chaque modèle de K Plus Proches Voisins.

```
boundary.grid <- show.data[set=="grid",]
boundary.grid[, label := pred.label]
show.grid <- boundary.grid[!on.text(V1, V2),]
pred.boundary <- boundary.grid[, getBoundaryDF(probability), by=neighbors]
pred.boundary$classifier <- "KPPV"
pred.boundary
```

```
      neighbors path.i       V1        V2 classifier
   1:         1      1 -2.60000 -1.025000       KPPV
   2:         1      1 -2.55000 -1.000000       KPPV
  ---
4491:        29      2  2.80099  1.900000       KPPV
4492:        29      2  2.80000  1.897619       KPPV
```

Au lieu d'afficher le nombre de voisins dans le titre du graphique, nous créons ci-dessous un élément `geom_text` qui sera mis à jour en fonction du nombre de voisins sélectionnés.

```
show.text <- show.grid[, list(
  V1=mean(range(V1)), V2=3.05), by=neighbors]
```

Nous calculons ci-dessous la position du texte qui affichera, en bas à gauche, le taux d'erreur du modèle sélectionné.

```
other.error[, V2.bottom := rep(
  text.V2.bottom + text.height * 1:2, l=.N)]
```

Ci-dessous, nous redéfinissons les données de l'erreur de Bayes sans colonne de voisins, afin qu'elles apparaissent dans chaque sous-ensemble `showSelected`.

```
Bayes.error <- data.table(
  set_fr="Bayes",
  error.prop=0.21)
```

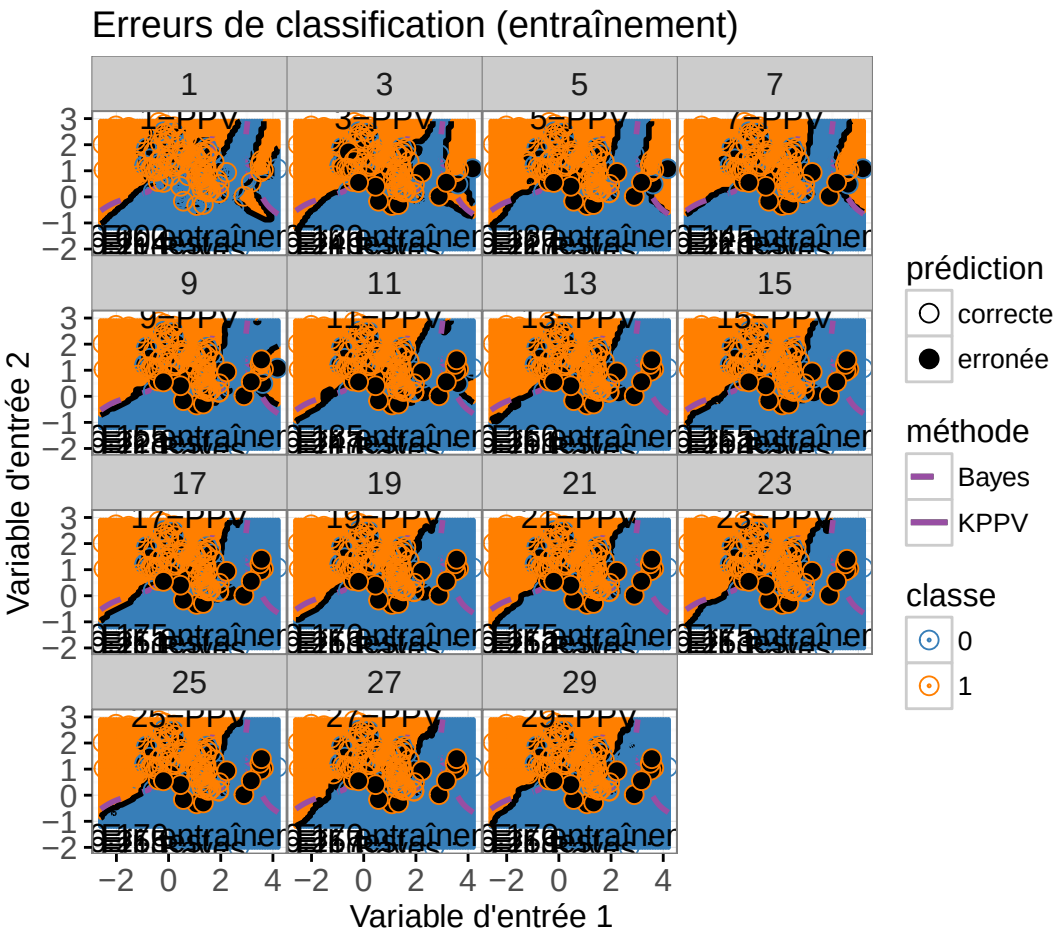
Enfin, nous redéfinissons le `ggplot`, en utilisant les voisins ("neighbors") comme variable `showSelected` dans les geoms point, "path" et texte.

```
scatterPlot <- ggplot()+
  ggtitle("Erreurs de classification (entraînement)")+
  theme_bw()+
  theme_animint(width=500, height=500)+
  xlab("Variable d'entrée 1")+
  ylab("Variable d'entrée 2")+
  coord_equal()+
  scale_linetype_manual(
    "méthode", values=classifier.linetypes)+
  scale_fill_manual(
    "prédiction",
    values=c(erronée="black", correcte="transparent"))+
  scale_color_manual("classe", values=label.colors)+
  geom_point(aes(
    V1, V2, color=label),
    showSelected="neighbors",
    size=0.2,
    data=show.grid)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    showSelected="neighbors",
    size=1,
    data=pred.boundary)+
  geom_path(aes(
    V1, V2, group=path.i, linetype=classifier),
    color=set.colors[["test"]],
    size=1,
    data=Bayes.boundary)+
  geom_point(aes(
    V1, V2, color=label,
    fill=prediction),
    showSelected="neighbors",
    size=3,
    shape=21,
    data=show.points)+
  geom_text(aes(
```

```
    text.V1.error, text.V2.bottom, label=paste("Err.", set_fr, ":")),
    data=Bayes.error,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, text.V2.bottom, label=sprintf("%.3f", error.prop)),
    data=Bayes.error,
    hjust=1)+
  geom_text(aes(
    text.V1.error, V2.bottom, label=paste("Err.", set_fr, ":")),
    showSelected="neighbors",
    data=other.error,
    hjust=0)+
  geom_text(aes(
    text.V1.prop, V2.bottom, label=sprintf("%.3f", error.prop)),
    showSelected="neighbors",
    data=other.error,
    hjust=1)+
  geom_text(aes(
    V1, V2,
    label=paste0(neighbors, "-PPV")),
    showSelected="neighbors",
    data=show.text)
```

Avant de compiler la visualisation des données interactive, nous imprimons un `ggplot` statique avec une facette pour chaque valeur de voisins.

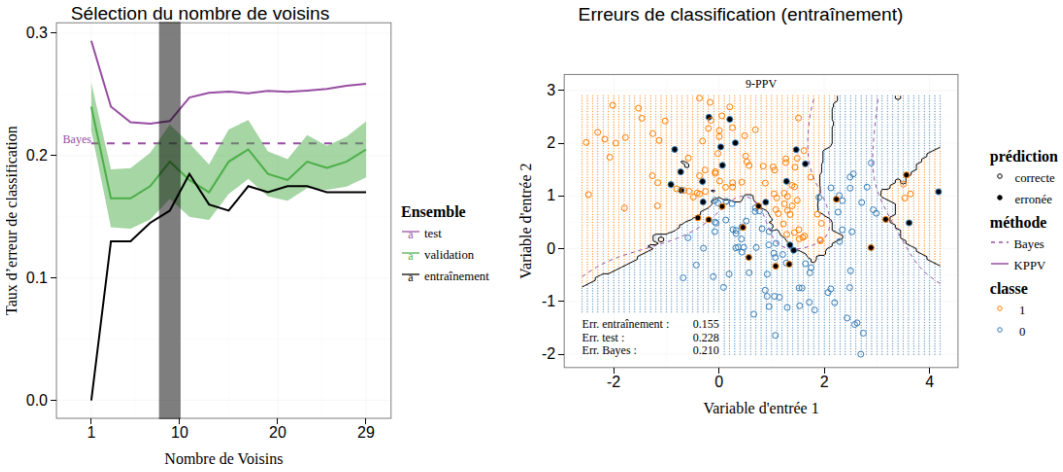
```
scatterPlot+
  facet_wrap("neighbors")+
  theme(panel.margin=grid::unit(0, "lines"))
```



10.2.3 Visualisation des données interactive combinée

Enfin, nous combinons les deux graphiques dans une visualisation des données unique avec les voisins comme variable de sélection.

```
animint(
  errorPlot,
  scatterPlot,
  first=list(neighbors=7),
  time=list(variable="neighbors", ms=3000))
```



Notez que les voisins ("neighbors") sont utilisés comme variable de temps, de sorte que l'animation montre les prédictions des différents modèles.

10.3 Sélectionner le nombre de divisions dans la validation croisée

Dans cette section, nous faisons une visualisation qui permet à l'utilisateur de sélectionner le nombre de divisions utilisés pour calculer la courbe d'erreur de validation.

La boucle `for` ci-dessous calcule la courbe d'erreur de validation pour plusieurs valeurs différentes de `n.folds`.

```
error.by.folds <- list()
error.by.folds[["10"]] <- data.table(n.folds=10, validation.error)
for(n.folds in c(3, 5, 15)){
  set.seed(2)
  mixture <- with(ESL.mixture, data.table(x, label=factor(y)))
  mixture$fold <- sample(rep(1:n.folds, l=nrow(mixture)))
  only.validation.list <- future.apply::future_lapply(
    1:n.folds, function(validation.fold){
      one.fold <- OneFold(validation.fold)
      data.table(validation.fold, one.fold[set=="validation"])
    }, future.seed=NULL)
  only.validation <- do.call(rbind, only.validation.list)
  only.validation.error <- only.validation[, list(
    error.prop=mean(is.error)
  ), by=.(set, set_fr=set, validation.fold, neighbors)]
  only.validation.stats <- only.validation.error[, list(
    mean=mean(error.prop),
    sd=sd(error.prop)/sqrt(.N)
  ), by=.(set, set_fr=set, neighbors)]
  error.by.folds[[paste(n.folds)]] <-
    data.table(n.folds, only.validation.stats, classifier="KPPV")
```

```
}
validation.error.several <- do.call(rbind, error.by.folds)
```

Le code ci-dessous calcule le minimum de la courbe d'erreur pour chaque valeur de `n.folds`.

```
min.validation <- validation.error.several[, .SD[which.min(mean),], by=n.folds]
```

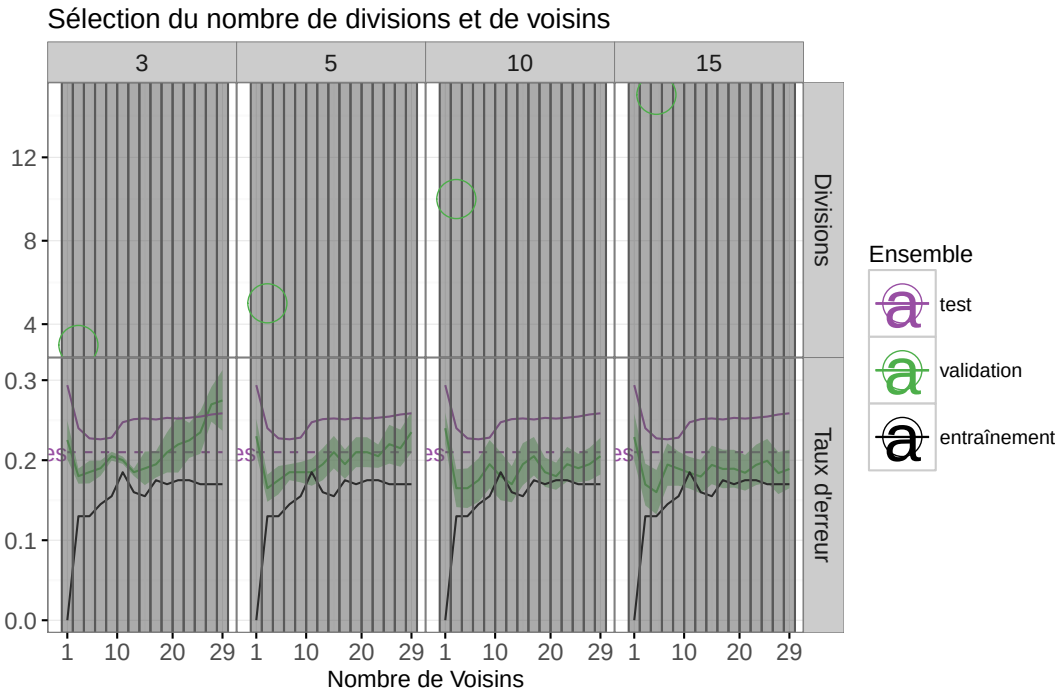
Le code ci-dessous crée un nouveau graphique de courbe d'erreur à deux facettes.

```
facets <- function(df, facet){
  data.frame(df, facet=factor(facet, c("Divisions", "Taux d'erreur")))
}
errorPlotNew <- ggplot()+
  ggtitle("Sélection du nombre de divisions et de voisins")+
  theme_bw()+
  theme_animint(height=500)+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_grid(facet ~ ., scales="free")+
  geom_text(aes(
    min.neighbors, error.prop,
    color=set_fr, label="Bayes"),
    showSelected="classifier",
    hjust=1,
    data=facets(Bayes.segment, "Taux d'erreur"))+
  geom_segment(aes(
    min.neighbors, error.prop,
    xend=max.neighbors, yend=error.prop,
    color=set_fr,
    linetype=classifier),
    showSelected="classifier",
    data=facets(Bayes.segment, "Taux d'erreur"))+
  scale_color_manual(
    legend.name, values=set.colors, breaks=names(set.colors))+
  scale_fill_manual(
    legend.name, values=set.colors, breaks=names(set.colors))+
  scale_linetype_manual(
    legend.name, values=classifier.linetypes)+
  guides(fill="none", linetype="none")+
  ylab("")+
  scale_x_continuous(
    "Nombre de Voisins",
    limits=c(-1, 30),
    breaks=c(1, 10, 20, 29))+
  geom_ribbon(aes(
    neighbors, ymin=mean-sd, ymax=mean+sd,
    fill=set_fr),
    showSelected=c("classifier", "set_fr", "n.folds"),
    alpha=0.5,
    color="transparent",
```

```
    data=facets(validation.error.several, "Taux d'erreur"))+
  geom_line(aes(
    neighbors, mean, color=set_fr,
    linetype=classifier),
    showSelected=c("classifier", "n.folds"),
    data=facets(validation.error.several, "Taux d'erreur"))+
  geom_line(aes(
    neighbors, error.prop, group=set_fr, color=set_fr,
    linetype=classifier),
    showSelected="classifier",
    data=facets(other.error, "Taux d'erreur"))+
  geom_tallrect(aes(
    xmin=neighbors-1, xmax=neighbors+1),
    clickSelects="neighbors",
    alpha=0.5,
    data=validation.error)+
  geom_point(aes(
    neighbors, n.folds, color=set_fr),
    clickSelects="n.folds",
    size=9,
    data=facets(min.validation, "Divisions"))
```

Le code ci-dessous prévisualise le nouveau graphique de la courbe d'erreur, en ajoutant une facette supplémentaire pour la variable `showSelected`.

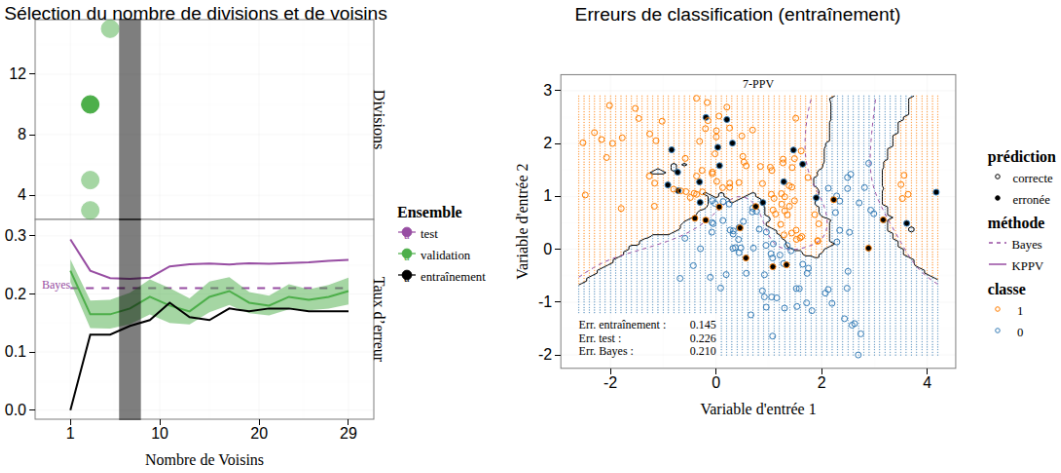
```
errorPlotNew+facet_grid(facet ~ n.folds, scales="free")
```



Le code ci-dessous crée une visualisation des données interactive à l'aide du nouveau graphique

de la courbe d'erreur.

```
animint(
  errorPlotNew,
  scatterPlot,
  first=list(neighbors=7, n.folds=10))
```



10.4 Résumé du chapitre et exercices

Nous avons montré comment ajouter deux fonctionnalités interactives à une visualisation des données des prédictions du modèle des K Plus Proches Voisins ("K-Nearest-Neighbors"). Nous avons commencé par une visualisation des données statique qui ne montrait que les prédictions du modèle des 7 Plus Proches Voisins. Ensuite, nous avons créé une refonte interactive qui permettait de sélectionner K, le nombre de voisins. Nous avons procédé à une autre refonte en ajoutant une facette permettant de sélectionner le nombre de divsions dans la validation croisée.

Exercices :

- Faites en sorte que les taux d'erreur du texte affichés en bas à gauche du deuxième graphique soient masqués quand on clique sur les entrées de la légende pour Bayes, train, test. Conseil : vous pouvez soit utiliser un `geom_text` avec `showSelected=c(selectorNameColumn="selectorValueColumn")` (comme expliqué dans le chapitre 14) ou deux `geom_text` chacun avec un paramètre `showSelected` différent.
- La colonne de probabilité ("probability") du tableau de données `show.grid` est la probabilité prédite de la classe 1. Comment feriez-vous la refonte de la visualisation pour montrer la probabilité prédite plutôt que la classe prédite à chaque point de la grille ? La difficulté principale est que la probabilité est une variable numérique, mais que `ggplot2` impose que chaque échelle soit exclusivement continue ou discrète (pas les deux). Vous pourriez utiliser une échelle de remplissage continue ("continuous fill scale"), mais vous devrez alors utiliser une échelle différente pour montrer la variable de prédiction.
- Ajoutez un nouveau graphique qui montre les tailles relatives des ensembles

d'entrainement ("train"), de validation et de test. Assurez-vous que la taille tracée des ensembles de validation et d'entraînement change en fonction de la valeur sélectionnée de `n.folds`.

- Jusqu'à présent, les graphiques de l'espace des variables ne montraient que les prédictions et les erreurs du modèle pour l'ensemble des données d'entrainement (`validation.fold==0`). Créez une visualisation qui inclut un nouveau graphique ou une nouvelle facette pour sélectionner `validation.fold`, et un graphique de l'espace des variables avec facettes (une facette pour l'ensemble de données d'entrainement, une facette pour l'ensemble de données de validation).

Dans le chapitre 11, nous vous expliquerons comment visualiser le `Lasso`, un modèle d'apprentissage automatique.

11

Lasso

Traduction de l'anglais Ch11-lasso

L'objectif de ce chapitre est de créer une visualisation des données interactive qui explique le Lasso, un modèle d'apprentissage automatique pour la régression linéaire régularisée.

Plan du chapitre :

- Nous commençons par plusieurs visualisations de données statiques du chemin ("path") du lasso.
- Nous créons ensuite une version interactive avec une facette et un graphique montrant l'erreur d'entraînement/validation et les résidus.
- Enfin, nous remanions la visualisation interactive des données avec des légendes simplifiées et des `tallrects` mobiles.

11.1 Graphiques statiques du chemin ("path") de la régularisation du coefficient

Nous commençons par charger l'ensemble de données sur le cancer de la prostate.

```
if(!file.exists("prostate.data")){
  curl::curl_download(
    "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data",
    "prostate.data")
}
prostate <- data.table::fread("prostate.data")
head(prostate)
```

```
   V1     lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
 1:  1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -
0.4307829
 2:  2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -
0.1625189
---
 5:  5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
 6:  6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
    train
 1:     T
 2:     T
---
```

```
5:     T
6:     T
```

Nous construisons un entrainement d'entrées `x` et des sorties `y` à l'aide du code ci-dessous.

```
input.cols <- c(
  "lcavol", "lweight", "age", "lbph", "svi", "lcp", "gleason",
  "pgg45")
prostate.inputs <- prostate[, ..input.cols]
is.train <- prostate$train == "T"
x <- as.matrix(prostate.inputs[is.train])
head(x)
```

```
        lcavol  lweight age      lbph svi       lcp gleason pgg45
[1,] -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0
[2,] -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0
[3,] -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20
[4,] -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0
[5,]  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0
[6,] -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0
```

```
y <- prostate[is.train, lpsa]
head(y)
```

```
[1] -0.4307829 -0.1625189 -0.1625189 -0.1625189  0.3715636  0.7654678
```

Ci-dessous, nous procédons à l'ajustement du chemin complet des solutions lasso à l'aide du package `lars`.

```
if(!requireNamespace("lars"))install.packages("lars")
```

```
Loading required namespace: lars
```

```
library(lars)
```

```
Loaded lars 1.3
```

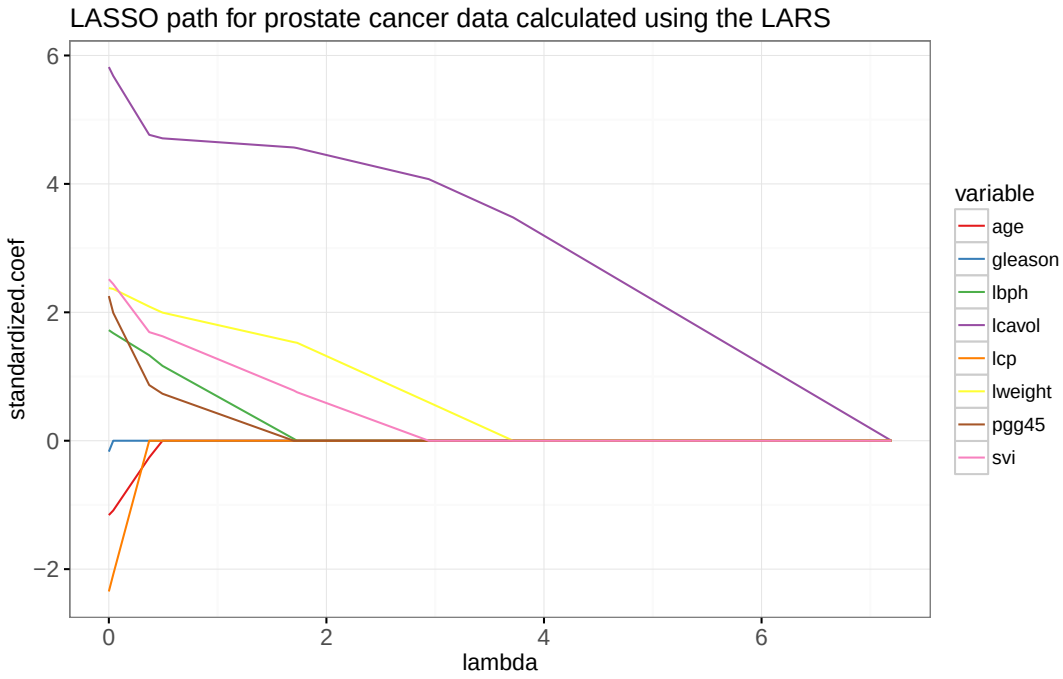
```
fit <- lars(x,y,type="lasso")
fit$lambda
```

```
[1] 7.1939462 3.7172742 2.9403866 1.7305064 1.7002813 0.4933166 0.3711651
[8] 0.0403451
```

Les chemins des valeurs `lambda` ne sont pas uniformément espacés.

```
pred.nox <- predict(fit, type="coef")
beta <- scale(pred.nox$coefficients, FALSE, 1/fit$normx)
arclength <- rowSums(abs(beta))
path.list <- list()
for(variable in colnames(beta)){
  standardized.coef <- beta[, variable]
```

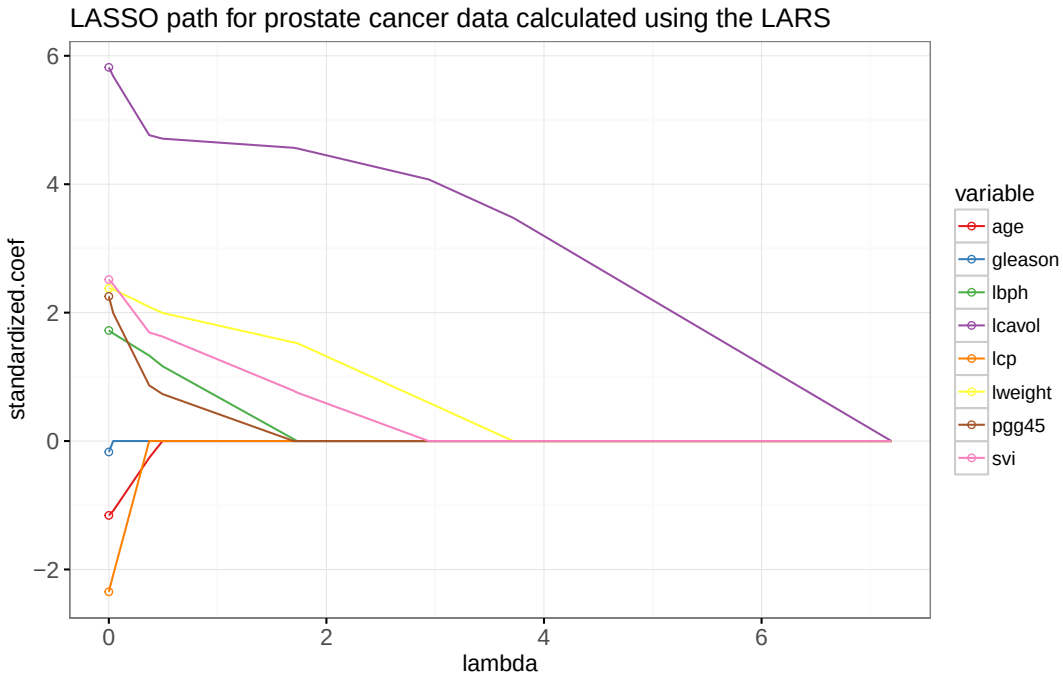
```
  path.list[[variable]] <- data.table::data.table(
    step=seq_along(standardized.coef),
    lambda=c(fit$lambda, 0),
    variable,
    standardized.coef,
    fraction=pred.nox$fraction,
    arclength)
}
path <- do.call(rbind, path.list)
variable.colors <- c(
  "#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
  "#A65628", "#F781BF", "#999999")
library(animint2)
gg.lambda <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    lambda, standardized.coef, color=variable, group=variable),
    data=path)+
  ggtitle("LASSO path for prostate cancer data calculated using the LARS")
gg.lambda
```



Le graphique ci-dessus montre l'ensemble du chemin de lasso, les pondérations optimales dans le problème de régression par moindres carrés régularisés L1, pour chaque paramètre de régularisation lambda. Le chemin commence à la solution des moindres carrés, `lambda=0` à gauche. Il se termine par le modèle à ordonnée à l'origine complètement régularisé à droite. Pour voir l'équivalence avec la solution des moindres carrés ordinaires, nous ajoutons des

points dans le graphique ci-dessous.

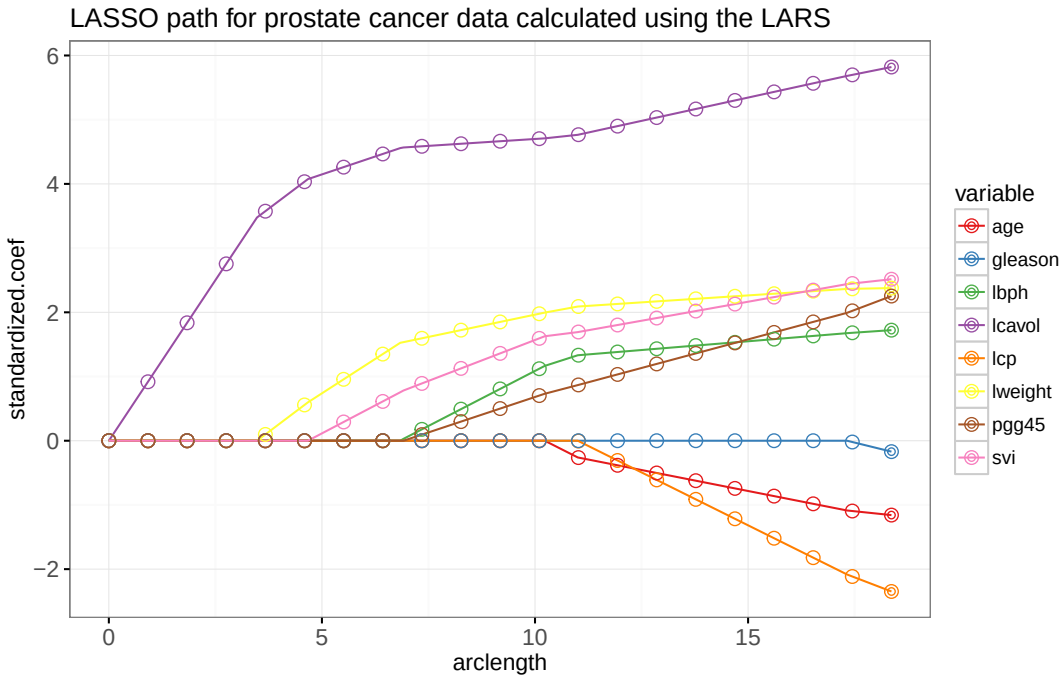
```
x.scaled <- with(fit, scale(x, meanx, normx))
lfit <- lm.fit(x.scaled, y)
lpoints <- data.table::data.table(
  variable=colnames(x),
  standardized.coef=lfit$coefficients,
  arclength=sum(abs(lfit$coefficients)))
gg.lambda+
  geom_point(aes(
    0, standardized.coef, color=variable),
    data=lpoints)
```



Dans le prochain graphique ci-dessous, nous montrons le chemin en fonction de la norme L1 (arclength), avec quelques points supplémentaires sur une grille régulièrement espacée que nous utiliserons plus tard pour l'animation.

```
fraction <- sort(unique(c(
  seq(0, 1, l=21))))
pred.fraction <- predict(
  fit, prostate.inputs,
  type="coef", mode="fraction", s=fraction)
coef.grid.list <- list()
coef.grid.mat <- scale(pred.fraction$coefficients, FALSE, 1/fit$normx)
for(fraction.i in seq_along(fraction)){
  standardized.coef <- coef.grid.mat[fraction.i,]
  coef.grid.list[[fraction.i]] <- data.table::data.table(
    fraction=fraction[[fraction.i]],
```

```
    variable=colnames(x),
    standardized.coef,
    arclength=sum(abs(standardized.coef)))
}
coef.grid <- do.call(rbind, coef.grid.list)
ggplot()+
  ggtitle("LASSO path for prostate cancer data calculated using the LARS")+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    arclength, standardized.coef, color=variable, group=variable),
    data=path)+
  geom_point(aes(
    arclength, standardized.coef, color=variable),
    data=lpoints)+
  geom_point(aes(
    arclength, standardized.coef, color=variable),
    shape=21,
    fill=NA,
    size=3,
    data=coef.grid)
```



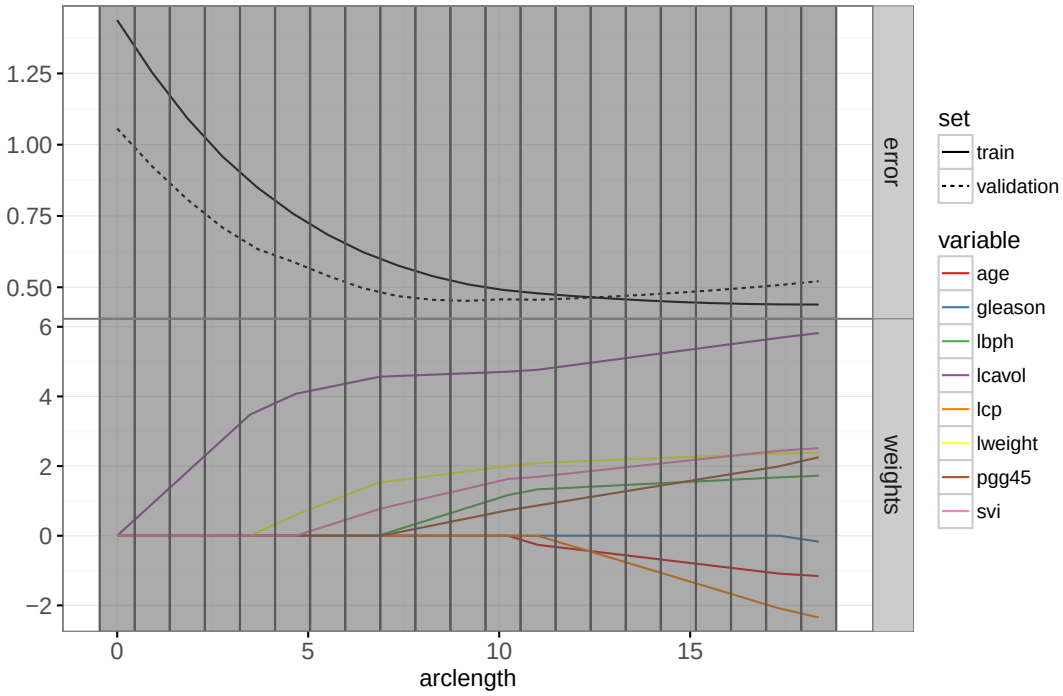
Le graphique ci-dessus montre que les pondérations aux points de la grille sont cohérentes avec les lignes qui représentent l'ensemble du chemin des solutions. L'algorithme LARS fournit rapidement des solutions Lasso pour autant de points de grille que vous le souhaitez. Plus précisément, étant donné que l'algorithme LARS ne calcule que les points de changement dans le chemin linéaire par morceaux, sa complexité temporelle ne dépend que du nombre

de points de changement (et non du nombre de points de grille).

11.2 Visualisation interactive du chemin (path) de la régularisation

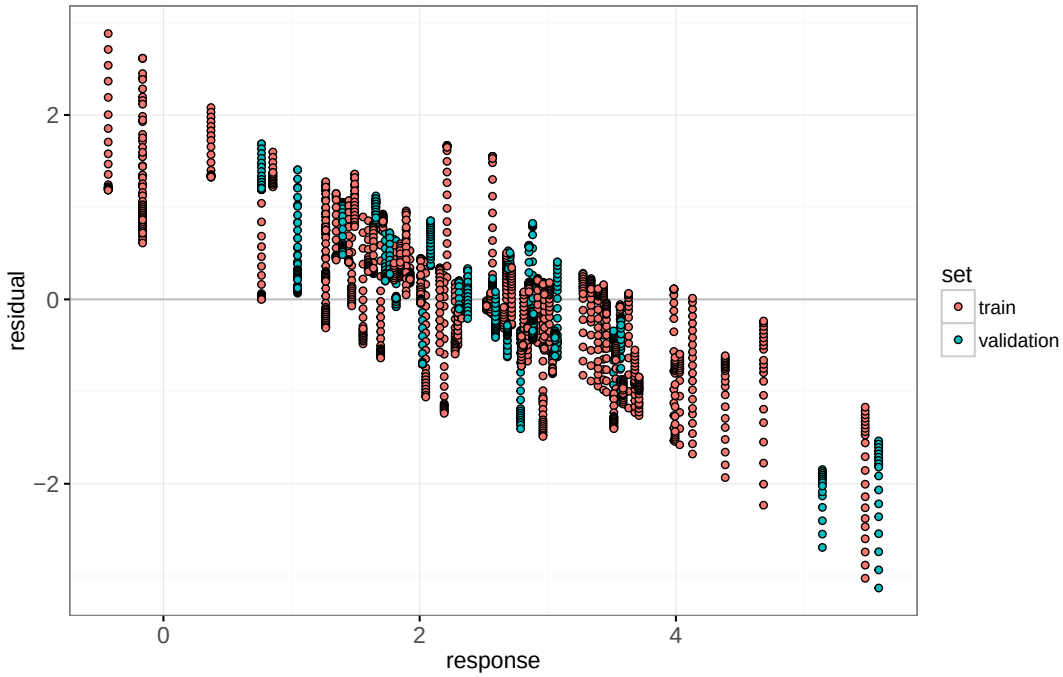
Le graphique ci-dessous combine le chemin des pondérations du lasso avec le graphique des erreurs d'entraînement/test.

```
pred.list <- predict(
  fit, prostate.inputs,
  mode="fraction", s=fraction)
residual.mat <- pred.list$fit - prostate$lpsa
squares.mat <- residual.mat * residual.mat
mean.error.list <- list()
for(set in c("train", "validation")){
  val <- if(set=="train")TRUE else FALSE
  is.set <- is.train == val
  mse <- colMeans(squares.mat[is.set, ])
  mean.error.list[[paste(set)]] <- data.table::data.table(
    set, mse, fraction,
    arclength=rowSums(abs(coef.grid.mat)))
}
mean.error <- do.call(rbind, mean.error.list)
rect.width <- diff(mean.error$arclength[1:2])/2
addY <- function(dt, y){
  data.table::data.table(dt, y.var=factor(y, c("error", "weights")))
}
tallrect.dt <- coef.grid[variable==variable[1],]
gg.path <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(y.var ~ ., scales="free")+
  ylab("")+
  scale_color_manual(values=variable.colors)+
  geom_line(aes(
    arclength, standardized.coef, color=variable, group=variable),
    data=addY(path, "weights"))+
  geom_line(aes(
    arclength, mse, linetype=set, group=set),
    data=addY(mean.error, "error"))+
  geom_tallrect(aes(
    xmin=arclength-rect.width,
    xmax=arclength+rect.width),
    clickSelects="arclength",
    alpha=0.5,
    data=tallrect.dt)
print(gg.path)
```



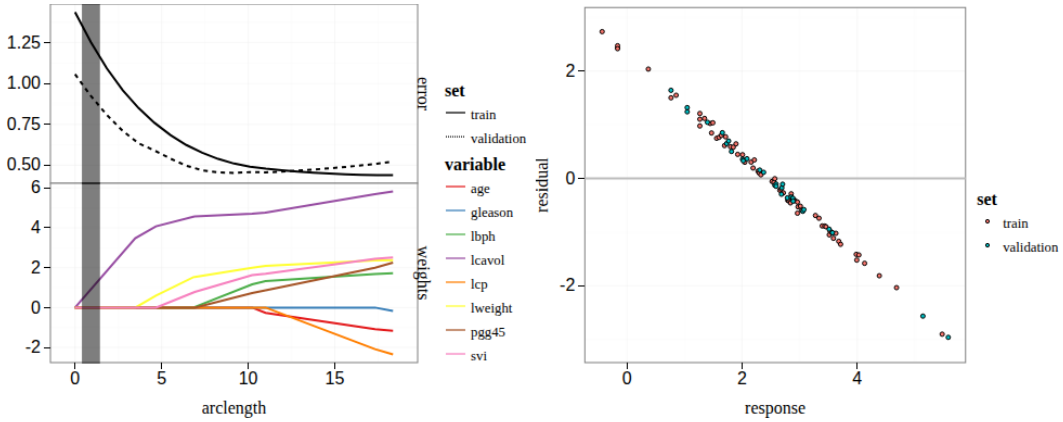
Enfin, nous ajoutons un graphique des résidus par rapport aux valeurs réelles.

```
lasso.res.list <- list()
for(fraction.i in seq_along(fraction)){
  lasso.res.list[[fraction.i]] <- data.table::data.table(
    observation.i=1:nrow(prostate),
    fraction=fraction[[fraction.i]],
    residual=residual.mat[, fraction.i],
    response=prostate$lpsa,
    arclength=sum(abs(coef.grid.mat[fraction.i,])),
    set=ifelse(prostate$train, "train","validation"))
}
lasso.res <- do.call(rbind, lasso.res.list)
hline.dt <- data.table::data.table(residual=0)
gg.res <- ggplot()+
  theme_bw()+
  geom_hline(aes(
    yintercept=residual),
    data=hline.dt,
    color="grey")+
  geom_point(aes(
    response, residual, fill=set,
    key=observation.i),
    showSelected="arclength",
    shape=21,
    data=lasso.res)
print(gg.res)
```



Ci-dessous, nous combinons les ggplots ci-dessus en un seul `animint2`. En cliquant sur le premier graphique, on modifie le paramètre de régularisation et les résidus qui sont affichés dans le second graphique.

```
animint(
  gg.path,
  gg.res,
  duration=list(arclength=2000),
  time=list(variable="arclength", ms=2000))
```



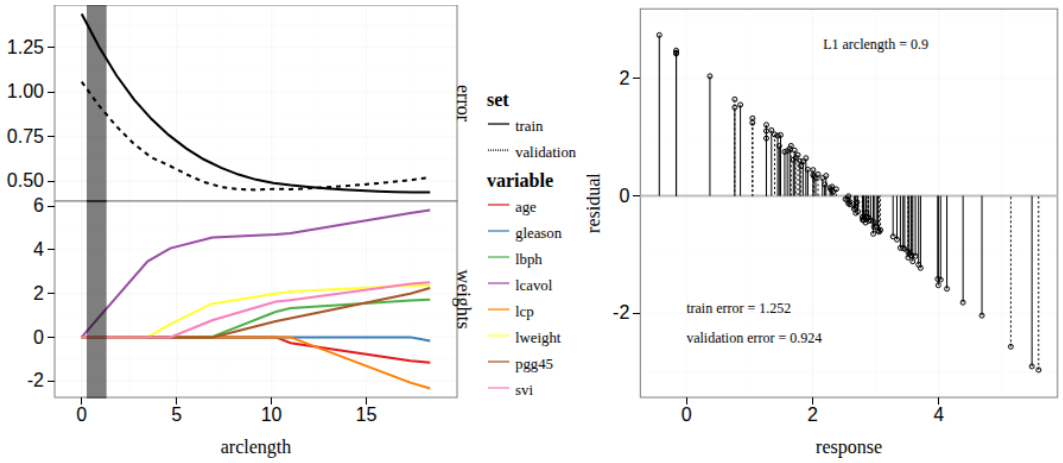
11.3 Refonte avec des `tallrects` mobiles

Le refonte ci-dessous comporte deux changements. Tout d'abord, vous avez peut-être remarqué qu'il y a deux légendes de "set" différentes dans l'`animint2` précédent (`linetype="set"` dans le premier graphique de chemin et `color="set"` dans le second graphique de résidus). Il serait plus facile pour le lecteur de décoder si la variable "set" n'est mappée qu'une seule fois. Ainsi, dans la refonte ci-dessous, nous remplaçons le `geom_point` dans le deuxième graphique par un `geom_segment` avec `linetype=set`.

Deuxièmement, nous avons remplacé le `tallrect` unique du premier graphique par deux `tallrects`. Le premier `tallrect` a `showSelected=arclength` et est utilisé pour afficher la longueur d'arc ("arclength") sélectionnée à l'aide d'un rectangle gris. Puisque nous spécifions une durée `duration` pour la variable `arclength` et la même valeur `key=1`, nous observerons une transition graduelle du `tallrect` gris sélectionné. Le deuxième tallrect a `clickSelects=arclength` et le fait de cliquer dessus a pour effet de modifier la valeur sélectionnée de `arclength`. Nous spécifions un autre ensemble de données avec plus de lignes, et utilisons les variables clickSelects/showSelected nommées pour indiquer que `arclength` doit également être utilisé comme une variable `showSelected`.

```
tallrect.show.list <- list()
for(a in tallrect.dt$arclength){
  is.selected <- tallrect.dt$arclength == a
  not.selected <- tallrect.dt[!is.selected]
  tallrect.show.list[[paste(a)]] <- data.table::data.table(
    not.selected, show.val=a, show.var="arclength")
}
tallrect.show <- do.call(rbind, tallrect.show.list)
animint(
  path=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    ylab("")+
    scale_color_manual(values=variable.colors)+
    geom_line(aes(
      arclength, standardized.coef, color=variable, group=variable),
      data=addY(path, "weights"))+
    geom_line(aes(
      arclength, mse, linetype=set, group=set),
      data=addY(mean.error, "error"))+
    geom_tallrect(aes(
      xmin=arclength-rect.width,
      xmax=arclength+rect.width,
      key=1),
      showSelected="arclength",
      alpha=0.5,
      data=tallrect.dt)+
    geom_tallrect(aes(
```

```
      xmin=arclength-rect.width,
      xmax=arclength+rect.width,
      key=paste(arclength, show.val)),
      clickSelects="arclength",
      showSelected=c("show.var"="show.val"),
      alpha=0.5,
      data=tallrect.show),
  res=ggplot()+
    theme_bw()+
    geom_hline(aes(
      yintercept=residual),
      data=hline.dt,
      color="grey")+
    guides(linetype="none")+
    geom_point(aes(
      response, residual,
      key=observation.i),
      showSelected=c("set", "arclength"),
      shape=21,
      fill=NA,
      color="black",
      data=lasso.res)+
    geom_text(aes(
      3, 2.5, label=sprintf("L1 arclength = %.1f", arclength),
      key=1),
      showSelected="arclength",
      data=tallrect.dt)+
    geom_text(aes(
      0, -2, label=sprintf("train error = %.3f", mse),
      key=1),
      showSelected=c("set", "arclength"),
      hjust=0,
      data=mean.error[set=="train"])+
    geom_text(aes(
      0, -2.5, label=sprintf("validation error = %.3f", mse),
      key=1),
      showSelected=c("set", "arclength"),
      hjust=0,
      data=mean.error[set=="validation"])+
    geom_segment(aes(
      response, residual,
      xend=response, yend=0,
      linetype=set,
      key=observation.i),
      showSelected=c("set", "arclength"),
      size=1,
      data=lasso.res),
  duration=list(arclength=2000),
  time=list(variable="arclength", ms=2000))
```



11.4 Résumé du chapitre et exercices

Nous avons créé une visualisation du modèle d'apprentissage automatique Lasso, qui montre de manière simultanée le chemin de régularisation et les courbes d'erreur. L'interactivité a été utilisée pour montrer les détails pour différentes valeurs du paramètre de régularisation.

Exercices :

- Refaites cette visualisation des données, en incluant le même effet visuel pour les `tallrects`, en utilisant un seul `geom_tallrect`. Conseil : créez un autre ensemble de données avec `expand.grid(arclength.click=arclength, arclength.show=arclength)` comme dans la définition de la fonction `make_tallrect_or_widerect`.
- Ajoutez un autre nuage de points qui montre les valeurs prédites contre la réponse, avec un `geom_abline` en arrière-plan pour indiquer une prédiction parfaite.
- À quoi ressembleraient les courbes d'erreur si l'on choisissait d'autres répartitions entraînement/validation ? Effectuez une validation croisée 4-plis et ajoutez un graphique qui peut être utilisé pour sélectionner le pli de test.

Dans le chapitre 12, nous vous expliquerons comment visualiser la machine à vecteurs de support.

12

Support Vector Machines

This goal of this chapter is to create an interactive data visualization that explains the Support Vector Machine, a machine learning model for binary classification.

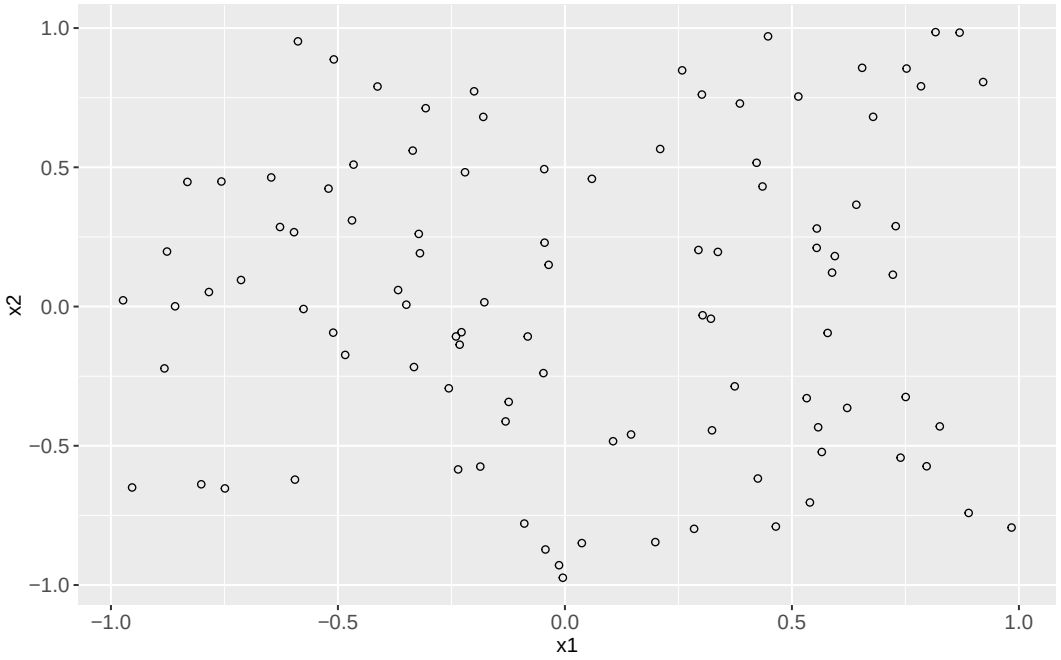
Chapter summary:

- We begin by simulating some data for binary classification in two dimensions, and making some static plots.
- In the second section, we make an interactive data visualization to show how the linear Support Vector Machine decision boundary changes as a function of the cost hyper-parameter.
- In the last section, we make an interactive data visualization to show how the decision boundary of the polynomial kernel Support Vector Machine changes as a function of the two hyper-parameters (cost and degree).

12.1 Generate and plot some data

We begin by generating two input features, `x1` and `x2`.

```
library(data.table)
N <- 50
set.seed(1)
getInput <- function(){
  c(#rnorm(N, sd=0.3),
    runif(N, -1, 1),
    runif(N, -1, 1)
    )
}
data.dt <- data.table(
  x1=getInput(),
  x2=getInput())
library(animint2)
ggplot()+
  geom_point(aes(
    x1, x2),
    data=data.dt)
```

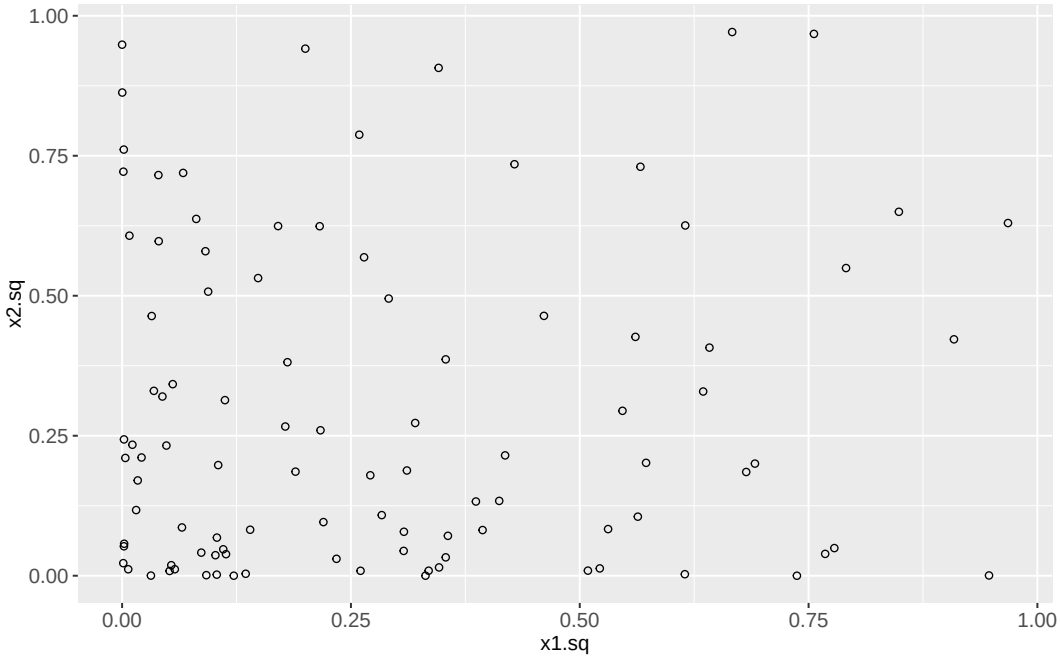


The plot below shows the same data, after computing two additional input features (the squares of the original two inputs).

```
data.dt[, let(
  x1.sq = x1^2,
  x2.sq = x2^2)][]
```

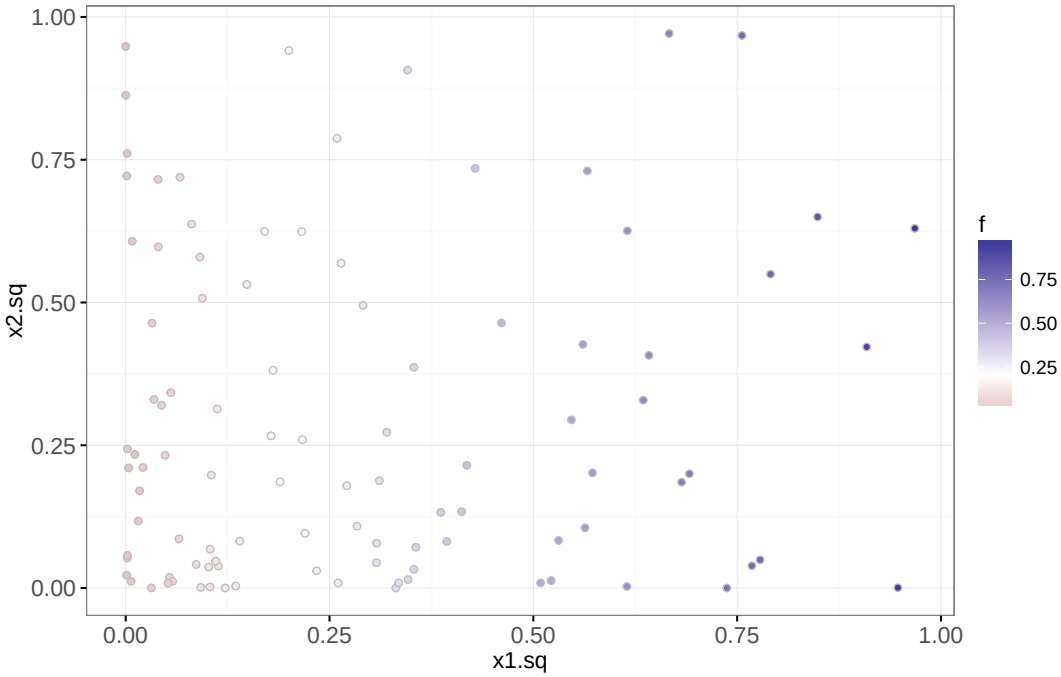
```
            x1         x2      x1.sq      x2.sq
  1: -0.4689827  0.3094479 0.21994475 0.09575798
  2: -0.2557522 -0.2936055 0.06540919 0.08620416
 ---
 99:  0.6217405 -0.3640726 0.38656123 0.13254888
100:  0.2098666  0.5657027 0.04404398 0.32001952
```

```
ggplot()+
  geom_point(aes(
    x1.sq, x2.sq),
    data=data.dt)
```



In our simulation, we assume that the output score `f` is a linear function of `x1.sq`, and ignores `x2.sq`. The plot below visualizes the output scores using the point fill aesthetic.

```
data.dt[, f := x1.sq]
true.decision.boundary <- 0.2
ggplot()+
  theme_bw()+
  scale_fill_gradient2(midpoint=true.decision.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, fill=f),
    shape=21,
    color="grey",
    data=data.dt)
```



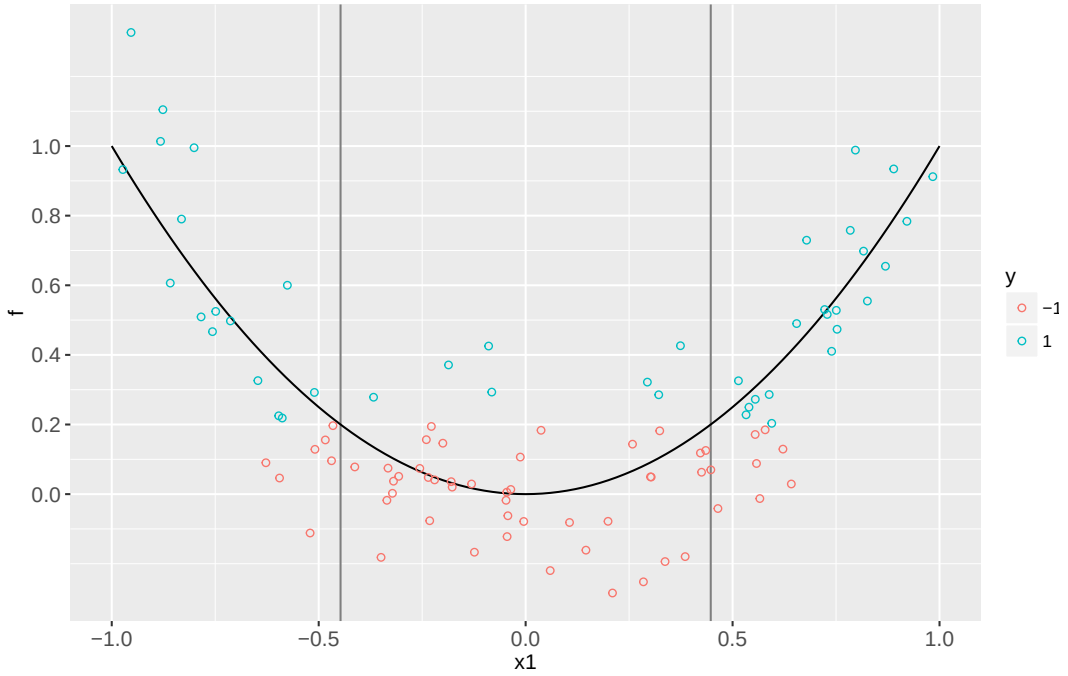
In particular, we assume that the label `y` is negative (-1) if `x1.sq + noise < threshold`, and positive (1) otherwise. The plot below visualizes the scores and labels, as a function of the input feature `x1`. It also shows the true score function in black. Of course, we would not be able to make this visualization with real data (only the labels are known in real data, not the scores).

```
data.dt[
, f.noise := f+rnorm(N, 0, 0.2)
][
, y.num := ifelse(f.noise<true.decision.boundary, -1, 1)
][
, y := factor(y.num)
]
table(data.dt$y)
```

```
-1  1
56 44
```

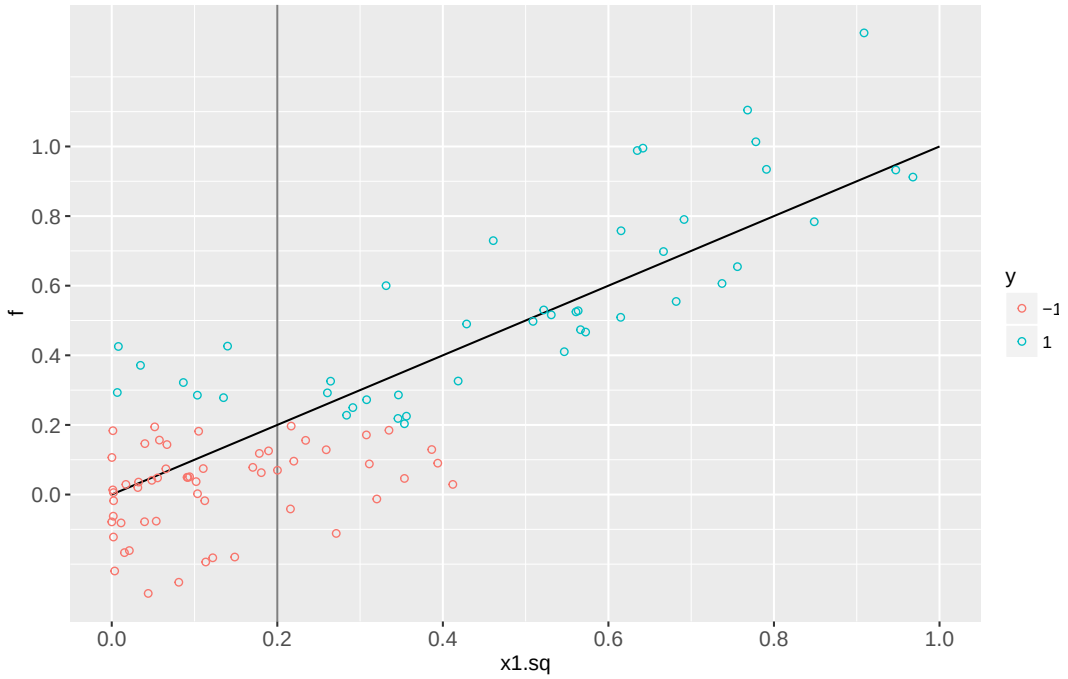
```
scores <- data.table(x1=seq(-1, 1, l=101))[
, x1.sq := x1^2
][
, f := x1.sq ]
x1.boundaries <- data.table(
  boundary=c(1, -1)*sqrt(true.decision.boundary))
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
```

```
    color="grey50",
    data=x1.boundaries)+
  geom_line(aes(
    x1, f),
    data=scores)+
  geom_point(aes(
    x1, f.noise, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



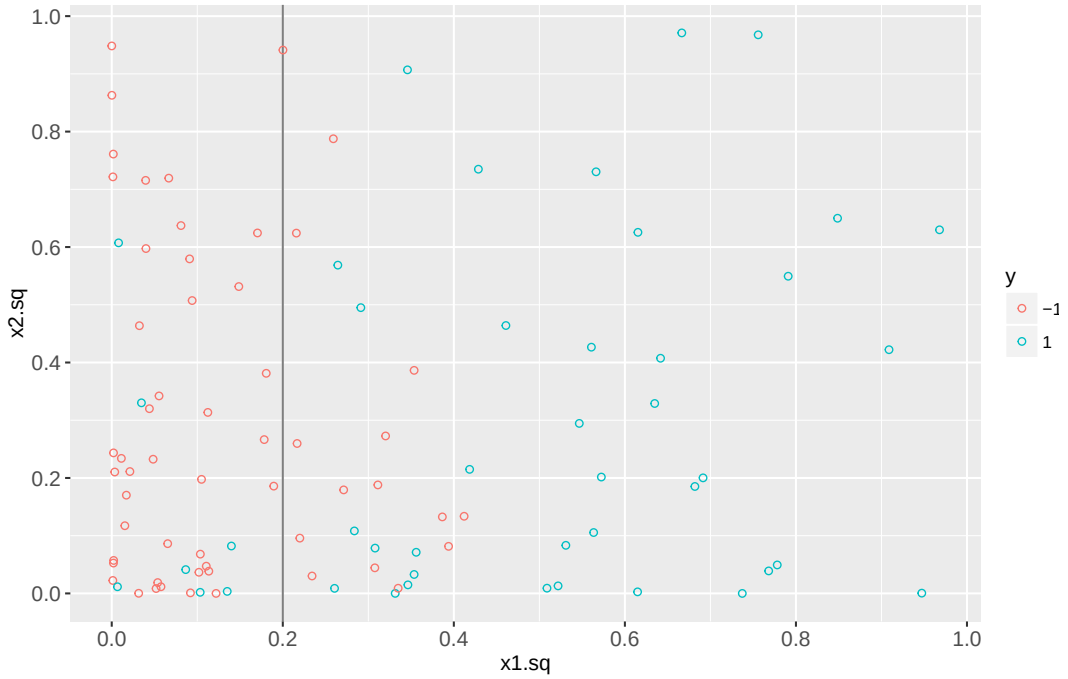
The plot below shows the scores and labels, as a function of the squared feature `x1.sq`. It is clear that the score function that we want to learn is linear in `x1.sq`.

```
x1sq.boundary <- data.table(boundary=true.decision.boundary)
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_line(aes(x1.sq, f), data=scores)+
  geom_point(aes(
    x1.sq, f.noise, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



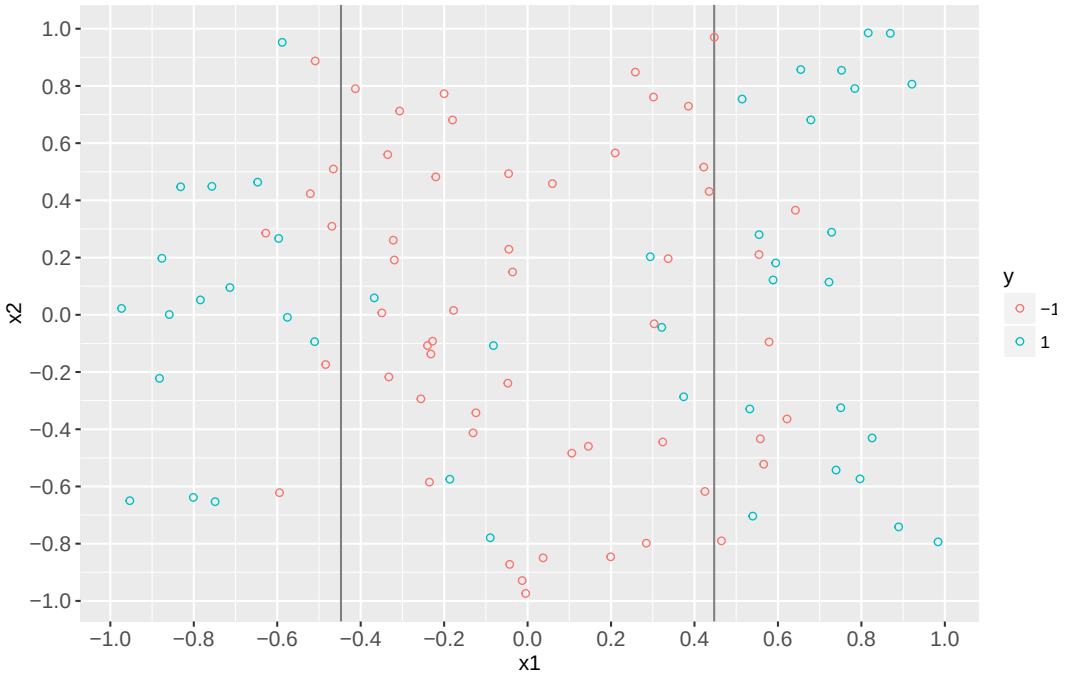
Next, we visualize the labels in the two-dimensional squared feature space. It is clear that the decision boundary is linear in this space.

```
ggplot()+
  scale_y_continuous(breaks=seq(0, 1, by=0.2))+
  scale_x_continuous(breaks=seq(0, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1sq.boundary)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



The plot below shows the input feature space (`x1` and `x2`). It is clear that the decision boundary is non-linear in `x1`.

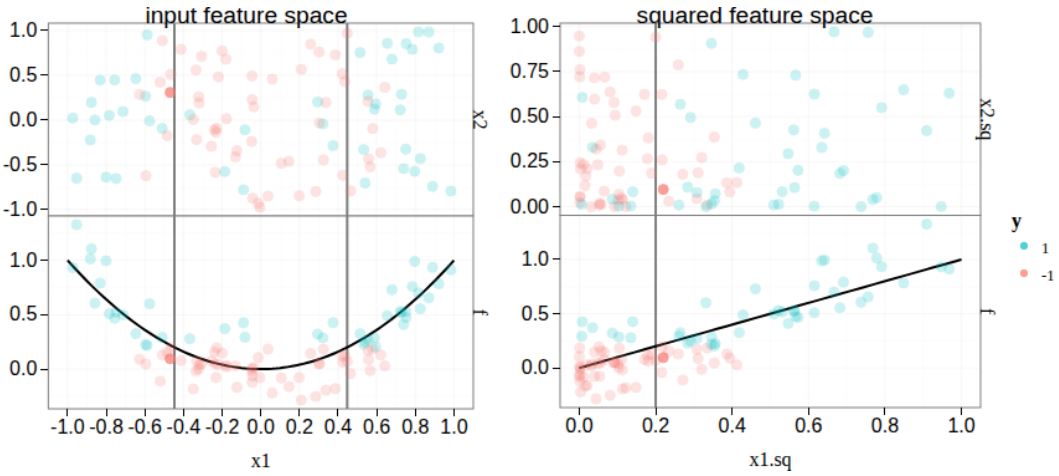
```
ggplot()+
  scale_y_continuous(breaks=seq(-1, 1, by=0.2))+
  scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1.boundaries)+
  geom_point(aes(
    x1, x2, color=y),
    shape=21,
    fill=NA,
    data=data.dt)
```



The animint below uses `clickSelects` to show which points in the input and squared space correspond. We just need to create an `data.i` variable that has a unique ID for each data point.

```
data.dt[, data.i := 1:.N]
YVAR <- function(dt, y.var){
  dt$y.var <- factor(y.var, c("x2", "x2.sq", "f"))
  dt
}
animint(
  input=ggplot()+
    ggtitle("input feature space")+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
    ylab("")+
    guides(color="none")+
    geom_vline(aes(
      xintercept=boundary),
      color="grey50",
      data=x1.boundaries)+
    geom_point(aes(
      x1, x2, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "x2"))+
```

```
    geom_line(aes(
      x1, f),
      data=YVAR(scores, "f"))+
    geom_point(aes(
      x1, f.noise, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "f")),
  square=ggplot()+
    ggtitle("squared feature space")+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    facet_grid(y.var ~ ., scales="free")+
    ylab("")+
    scale_x_continuous(breaks=seq(0, 1, by=0.2))+
    geom_vline(aes(
      xintercept=boundary),
      color="grey50",
      data=x1sq.boundary)+
    geom_point(aes(
      x1.sq, x2.sq, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "x2.sq"))+
    geom_line(aes(
      x1.sq, f),
      data=YVAR(scores, "f"))+
    geom_point(aes(
      x1.sq, f.noise, color=y),
      clickSelects="data.i",
      size=4,
      alpha=0.7,
      data=YVAR(data.dt, "f")))
```



Note how we used two multi-panel plots with the addColumn then facet idiom, rather than creating four separate plots. This emphasizes the fact that some plots/facets have a common `x1` or `x1.sq`axis. Note that we also hid the color legend in the first plot, since it is sufficient to just have one color legend.

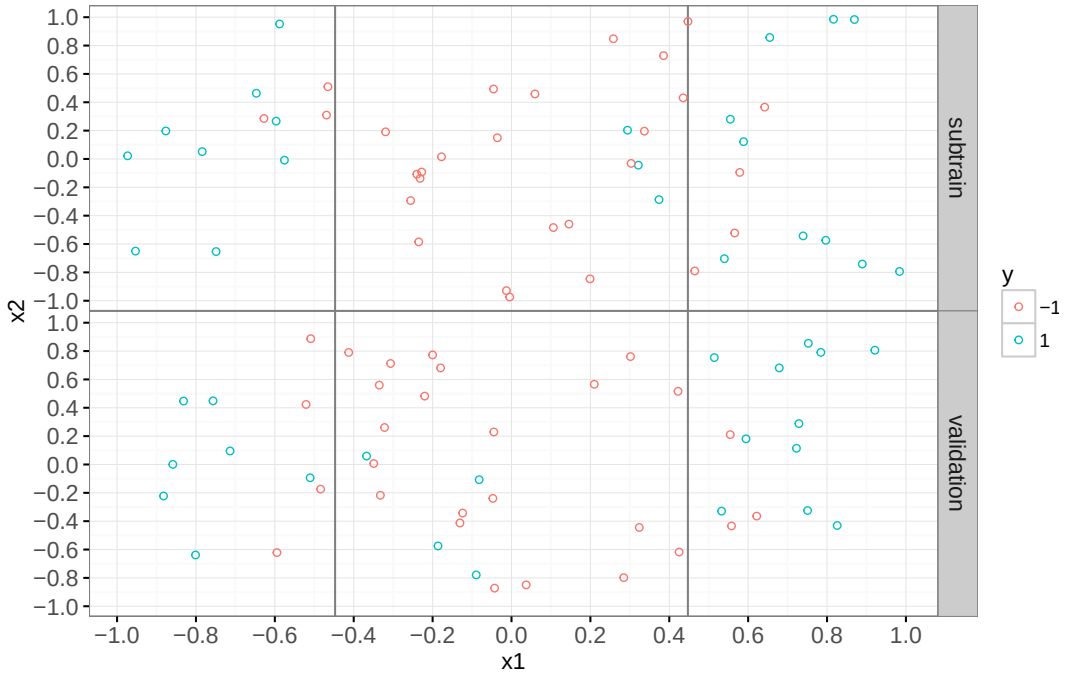
12.2 Linear SVM

```
train.i <- 1:N
data.dt[
, set := "validation"
][
  train.i, set := "subtrain"
]
table(data.dt$set)
```

```
subtrain validation
      50         50
```

```
subtrain.dt <- data.dt[set=="subtrain",]
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(set ~ .)+
  geom_vline(aes(
    xintercept=boundary),
    color="grey50",
    data=x1.boundaries)+
  scale_y_continuous(breaks=seq(-1, 1, by=0.2))+
  scale_x_continuous(breaks=seq(-1, 1, by=0.2))+
  geom_point(aes(
    x1, x2, color=y),
```

```
    data=data.dt)
```



We begin by fitting a linear SVM to the train data in the squared feature space, and visualizing the true labels `y` along with the predicted labels `pred.y`.

```
library(kernlab)
```

```
Attaching package: 'kernlab'
```

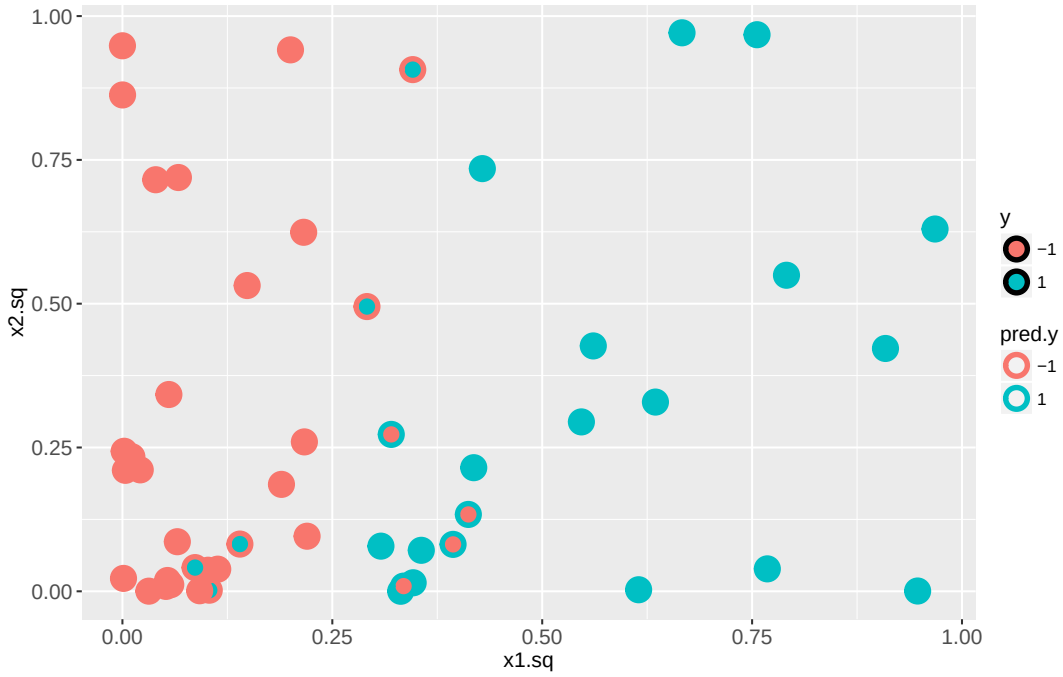
```
The following object is masked from 'package:animint2':

    alpha
```

```
squared.mat <- subtrain.dt[, cbind(x1.sq, x2.sq)]
y.vec <- subtrain.dt$y
fit <- ksvm(squared.mat, y.vec, kernel="vanilladot")
```

```
 Setting default kernel parameters
```

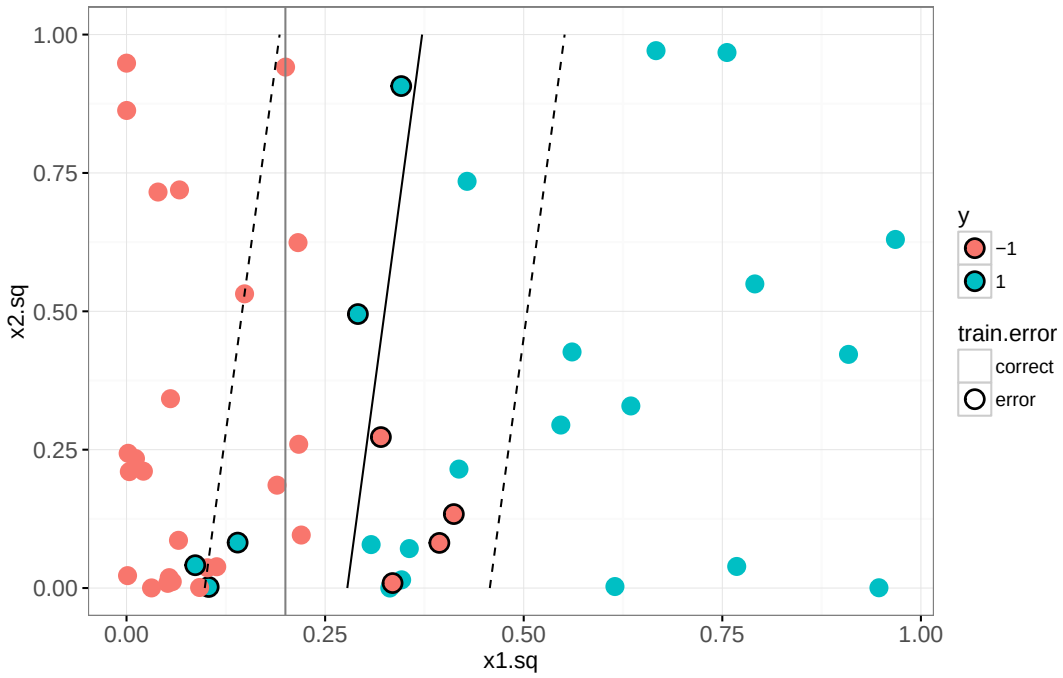
```
subtrain.dt$pred.y <- predict(fit)
ggplot()+
  geom_point(aes(
    x1.sq, x2.sq, color=pred.y, fill=y),
    shape=21,
    size=4,
    stroke=2,
    data=subtrain.dt)
```



It is clear from the plot above that there are several mis-classified train data points. In the plot below we visualize the decision boundary and margin.

```
predF <- function(fit, X){
  fit.sc <- scaling(fit)$x.scale
  if(is.null(fit.sc)){
    fit.sc <- list(
      "scaled:center"=c(0,0),
      "scaled:scale"=c(1,1))
  }
  mu <- fit.sc[["scaled:center"]]
  sigma <- fit.sc[["scaled:scale"]]
  X.sc <- scale(X, mu, sigma)
  kernelMult(
    kernelf(fit),
    X.sc,
    xmatrix(fit)[[1]],
    coef(fit)[[1]])-b(fit)
}
xsq.vec <- seq(0, 1, l=41)
grid.sq.dt <- data.table(expand.grid(
  x1.sq=xsq.vec,
  x2.sq=xsq.vec
))[
, pred.f := predF(fit, cbind(x1.sq, x2.sq))]
subtrain.dt[, train.error := ifelse(y==pred.y, "correct", "error")]
ggplot()+
  theme_bw()+
```

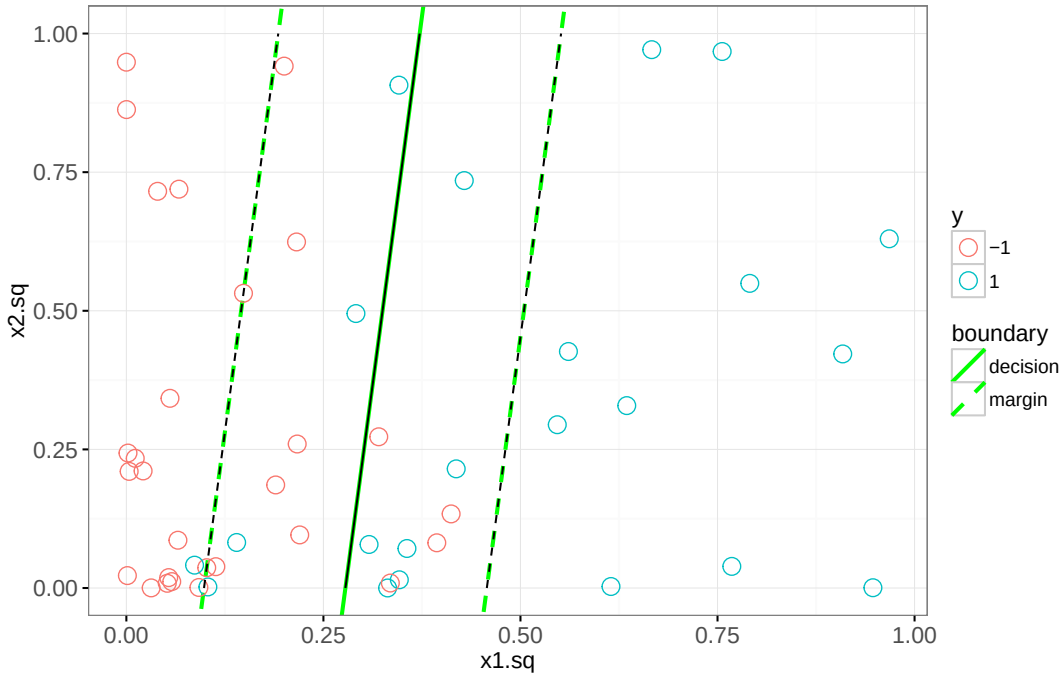
```
  scale_color_manual(values=c(error="black", correct=NA))+
  geom_point(aes(
    x1.sq, x2.sq, fill=y, color=train.error),
    shape=21,
    stroke=1,
    size=4,
    data=subtrain.dt)+
  geom_vline(aes(
    xintercept=boundary), color="grey50",
    data=x1sq.boundary)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=0,
    color="black",
    data=grid.sq.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=c(-1, 1),
    color="black",
    linetype="dashed",
    data=grid.sq.dt)
```



The plot above shows the true decision boundary using a grey `vline`. It also uses`geom_contour` to display the decision boundary (solid black line, predicted score 0) and the margin (dashed black line, predicted score -1 and 1). Since the decision boundary and margin are linear in this space, we can also use `geom_abline` to display them. To do that we need to do some math, and work out the equations for the slope and intercepts of those lines (as a function of the learned bias `b(fit)` and `weight.vec`, as well as the scale parameters

`mu` and `sigma`).

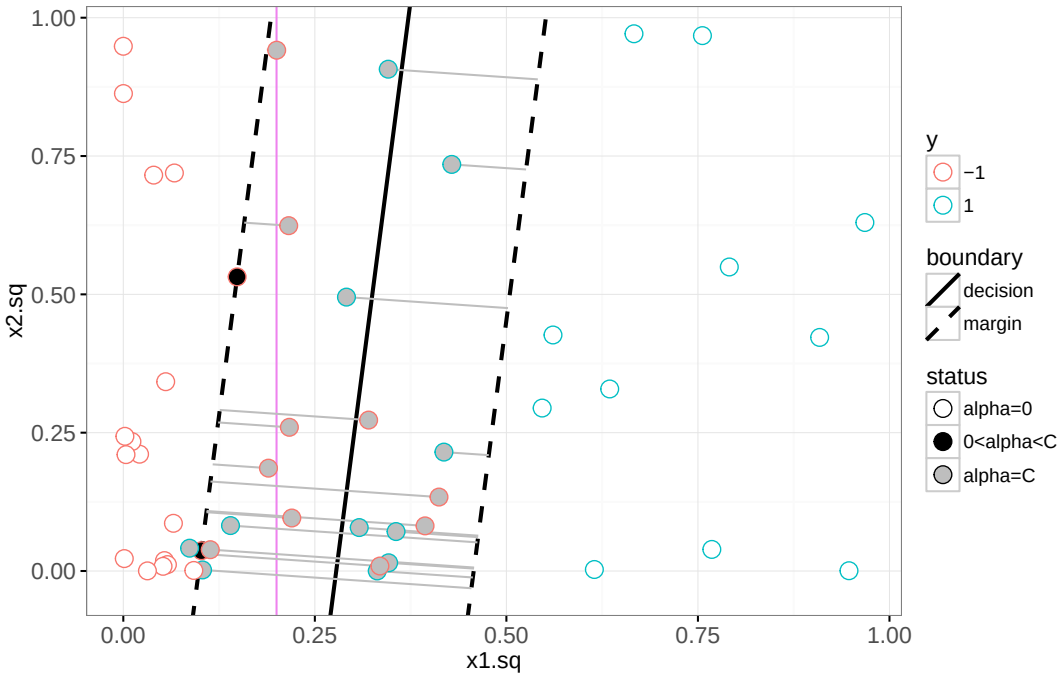
```
## The equation of the margin lines is x2 = m2 + s2/w2[c+b+w1*m1/s1]
## -s2*w1/(w2*s1)*x1 for c=1 and -1. x is input feature, m is mean, s
## is scale, w is learned weight.
fit.sc <- scaling(fit)$x.scale
if(is.null(fit.sc)){
  fit.sc <- list(
    "scaled:center"=c(0,0),
    "scaled:scale"=c(1,1))
}
mu <- fit.sc[["scaled:center"]]
sigma <- fit.sc[["scaled:scale"]]
weight.vec <- colSums(xmatrix(fit)[[1]]*coef(fit)[[1]])
predF.linear <- function(fit, X){
  X.sc <- scale(X, mu, sigma)
  X.sc %*% weight.vec - b(fit)
}
abline.dt <- data.table(
  y=factor(c(-1,0,1)),
  boundary=c("margin", "decision", "margin"),
  intercept=mu[2]+sigma[2]/weight.vec[2]*(
    c(-1, 0, 1)+b(fit)+weight.vec[1]*mu[1]/sigma[1]),
  slope=-weight.vec[1]*sigma[2]/(weight.vec[2]*sigma[1]))
ggplot()+
  theme_bw()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
    color="green",
    size=1,
    data=abline.dt)+
  geom_point(aes(
    x1.sq, x2.sq, color=y),
    shape=21,
    fill=NA,
    size=4,
    data=subtrain.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=0,
    color="black",
    data=grid.sq.dt)+
  geom_contour(aes(
    x1.sq, x2.sq, z=pred.f),
    breaks=c(-1, 1),
    color="black",
    linetype="dashed",
    data=grid.sq.dt)
```



The plot above confirms that our computation of the slope and intercepts (green lines) agrees with the contours (black lines). In the plot below, we show the learned `alpha` coefficients, and add a `geom_segment` to visualize the slack.

```
subtrain.dt[, alpha := 0]
train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
subtrain.dt[, status := ifelse(
  alpha==0, "alpha=0",
  ifelse(alpha==1, "alpha=C", "0<alpha<C"))]
slack.slope <- weight.vec[2]*sigma[1]/(weight.vec[1]*sigma[2])
slack.dt <- subtrain.dt[alpha==1,]
slack.join <- abline.dt[slack.dt, on=list(y)]
slack.join[, x1.sq.margin := (
  x2.sq-slack.slope*x1.sq-intercept)/(slope-slack.slope)]
slack.join[, x2.sq.margin := slope*x1.sq.margin + intercept]
sv.colors <- c(
  "alpha=0"="white",
  "0<alpha<C"="black",
  "alpha=C"="grey")
ggplot()+
  theme_bw()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_vline(aes(
    xintercept=boundary), color="violet",
    data=x1sq.boundary)+
  geom_abline(aes(
    slope=slope, intercept=intercept, linetype=boundary),
```

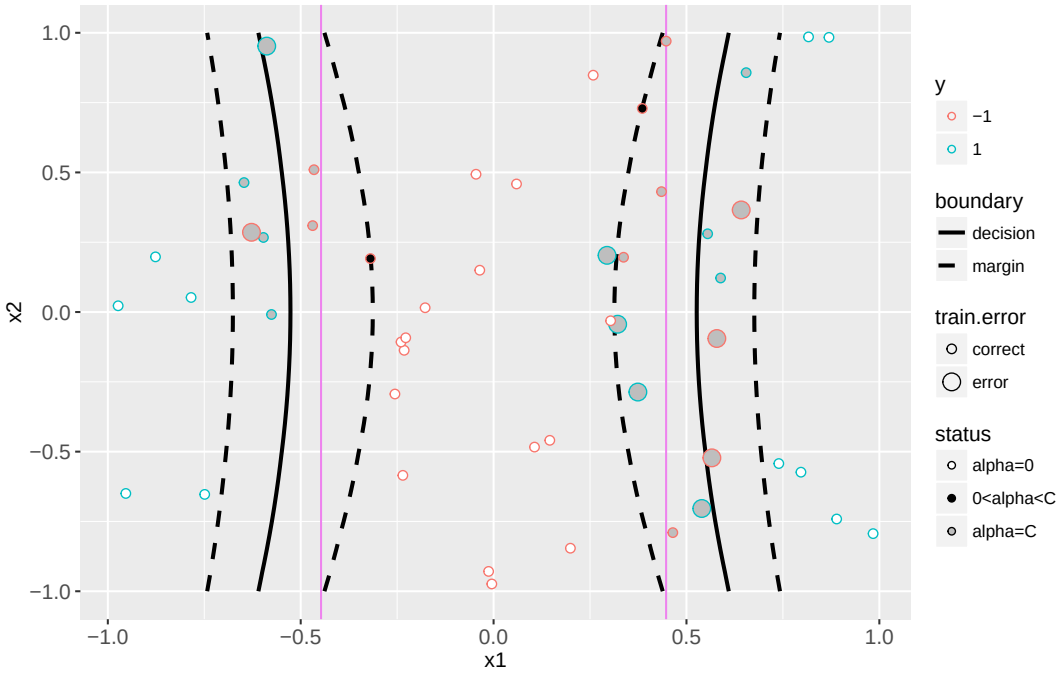
```
    size=1,
    data=abline.dt)+
  geom_segment(aes(
    x1.sq, x2.sq,
    xend=x1.sq.margin, yend=x2.sq.margin),
    color="grey",
    data=slack.join)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  geom_point(aes(
    x1.sq, x2.sq, color=y, fill=status),
    shape=21,
    size=4,
    data=subtrain.dt)
```



The plot above shows the slack in grey segments, and the decision and margin lines in black. The Bayes decision boundary is shown in the background as a vertical violet line. The support vectors are the points with non-zero alpha coefficients. Black filled support vectors are on the margin, and grey support vectors are on the wrong side of the margin (and have non-zero slack). The plot below shows the model that was learned in the original feature space,

```
n.grid <- 41
x.vec <- seq(-1, 1, l=n.grid)
grid.dt <- data.table(expand.grid(
  x1=x.vec,
  x2=x.vec))
getBoundaryDF <- function(score.vec, level.vec=c(-1, 0, 1)){
```

```
  stopifnot(length(score.vec) == n.grid * n.grid)
  several.paths <- contourLines(
    x.vec, x.vec,
    matrix(score.vec, n.grid, n.grid),
    levels=level.vec)
  contour.list <- list()
  for(path.i in seq_along(several.paths)){
    contour.list[[path.i]] <- with(several.paths[[path.i]], data.table(
      path.i,
      level.num=as.numeric(level),
      level.fac=factor(level, level.vec),
      boundary=ifelse(level==0, "decision", "margin"),
      x1=x, x2=y))
  }
  do.call(rbind, contour.list)
}
grid.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
boundaries <- grid.dt[, getBoundaryDF(pred.f)]
ggplot()+
  scale_linetype_manual(values=c(margin="dashed", decision="solid"))+
  geom_vline(aes(
    xintercept=boundary),
    color="violet",
    data=x1.boundaries)+
  geom_path(aes(
    x1, x2, group=path.i, linetype=boundary),
    size=1,
    data=boundaries)+
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  scale_size_manual(values=c(correct=2, error=4))+
  geom_point(aes(
    x1, x2, color=y,
    size=train.error,
    fill=status),
    shape=21,
    data=subtrain.dt)
```



The goal below will be to make an animint that shows how the decision boundary, margin, and slack change as a function of the cost parameter.

```
modelInfo.list <- list()
predictions.list <- list()
slackSegs.list <- list()
modelLines.list <- list()
inputBoundaries.list <- list()
setErrors.list <- list()
cost.by <- 0.2
for(cost.param in round(10^seq(-1, 1, by=cost.by),1)){
  fit <- ksvm(
    squared.mat, y.vec, kernel="vanilladot", scaled=FALSE, C=cost.param)
  fit.sc <- scaling(fit)$x.scale
  if(is.null(fit.sc)){
    fit.sc <- list(
      "scaled:center"=c(0,0),
      "scaled:scale"=c(1,1))
  }
  mu <- fit.sc[["scaled:center"]]
  sigma <- fit.sc[["scaled:scale"]]
  weight.vec <- colSums(xmatrix(fit)[[1]]*coef(fit)[[1]])
  grid.sq.dt[, pred.f := predF(fit, cbind(x1.sq, x2.sq))]
  data.dt[, pred.y := predict(fit, cbind(x1.sq, x2.sq))]
  one.error <- data.dt[, list(errors=sum(y!=pred.y)), by=set]
  setErrors.list[[paste(cost.param)]] <- data.table(
    cost.param, one.error)
  subtrain.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
```

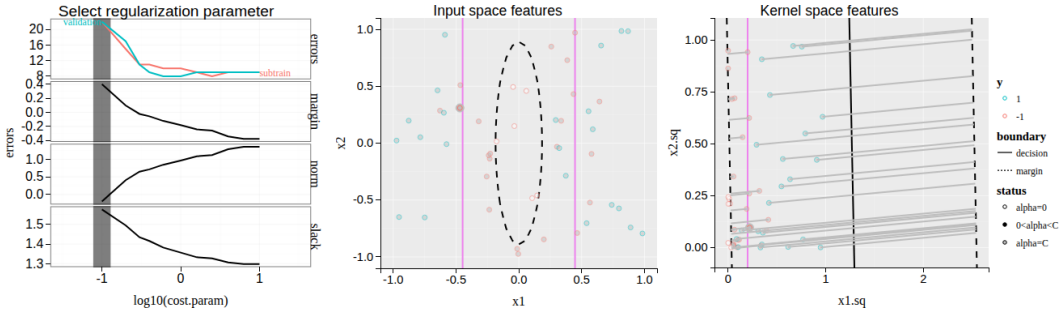
```
  grid.dt[, pred.f := predF(fit, cbind(x1^2, x2^2))]
  boundaries <- getBoundaryDF(grid.dt$pred.f)
  inputBoundaries.list[[paste(cost.param)]] <- data.table(
    cost.param, boundaries)
  subtrain.dt$alpha <- 0
  train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
  subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
  subtrain.dt[, status := ifelse(
    alpha==0, "alpha=0",
    ifelse(alpha==cost.param, "alpha=C", "0<alpha<C"))]
  ## The equation of the margin lines is x2 = m2 + s2/w2[c+b+w1*m1/s1]
  ## -s2*w1/(w2*s1)*x1 for c=1 and -1. x is input feature, m is mean, s
  ## is scale, w is learned weight.
  slack.slope <- weight.vec[2]*sigma[1]/(weight.vec[1]*sigma[2])
  abline.dt <- data.table(
    y=factor(c(-1,0,1)),
    boundary=c("margin", "decision", "margin"),
    intercept=mu[2]+sigma[2]/weight.vec[2]*(
      c(-1, 0, 1)+b(fit)+weight.vec[1]*mu[1]/sigma[1]),
    slope=-weight.vec[1]*sigma[2]/(weight.vec[2]*sigma[1]))
  slack.dt <- subtrain.dt[alpha==cost.param]
  slack.join <- abline.dt[slack.dt, on=list(y)]
  slack.join[, x1.sq.margin := (
    x2.sq-slack.slope*x1.sq-intercept)/(slope-slack.slope)]
  slack.join[, x2.sq.margin := slope*x1.sq.margin + intercept]
  norm.weights <- as.numeric(weight.vec %*% weight.vec)
  modelInfo.list[[paste(cost.param)]] <- data.table(
    cost.param,
    slack=slack.join[, sum(1-pred.f*y.num)],
    norm=norm.weights,
    margin=2/sqrt(norm.weights))
  predictions.list[[paste(cost.param)]] <- data.table(
    cost.param, subtrain.dt)
  slackSegs.list[[paste(cost.param)]] <- data.table(
    cost.param, slack.join)
  modelLines.list[[paste(cost.param)]] <- data.table(
    cost.param, abline.dt)
}
```

```
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
Setting default kernel parameters
```

```
inputBoundaries <- do.call(rbind, inputBoundaries.list)
predictions <- do.call(rbind, predictions.list)
slackSegs <- do.call(rbind, slackSegs.list)
modelLines <- do.call(rbind, modelLines.list)
modelInfo <- do.call(rbind, modelInfo.list)
setErrors <- do.call(rbind, setErrors.list)
modelInfo.tall <- melt(modelInfo, id.vars="cost.param")
grid.sq.dt$boundary <- "true"
setErrors$variable <- "errors"
inputBoundaries[, boundary := ifelse(level.num==0, "decision", "margin")]
slackSegs$boundary <- "margin"
set.label.select <- data.table(
  cost.param=range(setErrors$cost.param),
  set=c("validation", "subtrain"),
  hjust=c(1, 0))
set.labels <- setErrors[set.label.select, on=list(cost.param, set)]
viz.linear.svm <- animint(
  selectModel=ggplot()+
    ggtitle("Select regularization parameter")+
    scale_x_continuous(limits=c(-1.5, 1.5))+
    geom_tallrect(aes(
      xmin=log10(cost.param)-cost.by/2,
      xmax=log10(cost.param)+cost.by/2),
      clickSelects="cost.param",
      alpha=0.5,
      data=modelInfo)+
    theme_bw()+
    facet_grid(variable ~ ., scales="free")+
    geom_line(aes(
      log10(cost.param), errors,
      group=set, color=set),
      data=setErrors)+
    geom_text(aes(
      log10(cost.param), errors-1, label=set,
      hjust=hjust,
      color=set),
      data=set.labels)+
    guides(color="none")+
    geom_line(aes(
      log10(cost.param), log10(value)),
      data=modelInfo.tall),
  inputSpace=ggplot()+
    ggtitle("Input space features")+
    scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
    geom_vline(aes(
      xintercept=boundary),
      color="violet",
      data=x1.boundaries)+
    guides(color="none", fill="none", linetype="none")+
```

```
    scale_linetype_manual(values=c(
      "-1"="dashed",
      "0"="solid",
      "1"="dashed"))+
    geom_path(aes(
      x1, x2,
      group=path.i,
      linetype=level.fac),
      showSelected=c("boundary", "cost.param"),
      color="black",
      data=inputBoundaries)+
    geom_point(aes(
      x1, x2, fill=status),
      showSelected=c("status", "y", "data.i", "cost.param"),
      size=5,
      color="grey",
      data=predictions)+
    geom_point(aes(
      x1, x2, color=y, fill=status),
      showSelected=c("cost.param", "status", "y"),
      clickSelects="data.i",
      size=3,
      data=predictions),
  kernelSpace=ggplot()+
    ggtitle("Kernel space features")+
    geom_vline(aes(
      xintercept=boundary), color="violet",
      data=x1sq.boundary)+
    ##coord_cartesian(xlim=c(0, 1), ylim=c(0, 1))+
    geom_abline(aes(
      slope=slope, intercept=intercept, linetype=boundary),
      showSelected="cost.param",
      color="black",
      data=modelLines)+
    scale_linetype_manual(values=c(
      decision="solid",
      margin="dashed",
      true="solid"))+
    geom_point(aes(
      x1.sq, x2.sq, fill=status),
      showSelected=c("data.i", "cost.param"),
      size=5,
      color="grey",
      data=predictions)+
    geom_point(aes(
      x1.sq, x2.sq, color=y, fill=status),
      clickSelects="data.i",
      showSelected="cost.param",
      size=3,
```

```
      data=predictions)+
    scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
    geom_segment(aes(
      x1.sq, x2.sq,
      xend=x1.sq.margin, yend=x2.sq.margin),
      showSelected=c("cost.param", "boundary"),
      color="grey",
      data=slackSegs))
viz.linear.svm
```



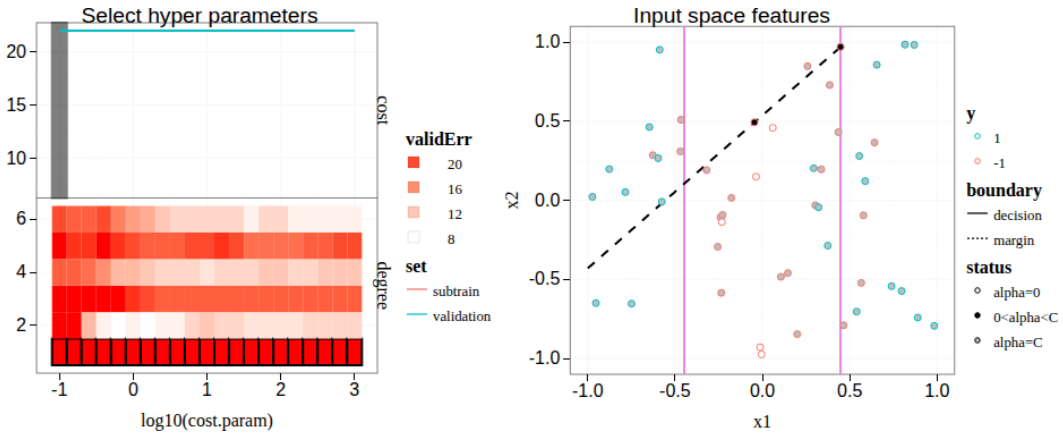
12.3 Non-linear polynomial kernel SVM

In the previous section we fit a linear kernel in the squared feature space, which resulted in learning a function which is non-linear in terms of the original feature space. In this section we directly fit a non-linear polynomial kernel in the original space.

```
predictions.list <- list()
inputBoundaries.list <- list()
setErrors.list <- list()
cost.by <- 0.2
orig.mat <- subtrain.dt[, cbind(x1, x2)]
for(cost.param in 10^seq(-1, 3, by=cost.by)){
  for(degree.num in seq(1, 6, by=1)){
    k <- polydot(degree.num, offset=0)
    fit <- ksvm(
      orig.mat, y.vec, kernel=k, scaled=FALSE, C=cost.param)
    grid.dt[, pred.f := predF(fit, cbind(x1, x2))]
    grid.dt[, pred.y := predict(fit, cbind(x1, x2))]
    grid.dt[, stopifnot(sign(pred.f) == pred.y)]
    data.dt[, pred.y := predict(fit, cbind(x1, x2))]
    one.error <- data.dt[, list(errors=sum(y != pred.y)), by=set]
    setErrors.list[[paste(cost.param, degree.num)]] <- data.table(
      cost.param, degree.num, one.error)
    boundaries <- getBoundaryDF(grid.dt$pred.f)
    if(is.data.frame(boundaries) && nrow(boundaries)){
```

```
      cost.deg <- paste(cost.param, degree.num)
      inputBoundaries.list[[cost.deg]] <- data.table(
        cost.param, degree.num, boundaries)
    }
    subtrain.dt[, alpha := 0]
    train.row.vec <- as.integer(rownames(xmatrix(fit)[[1]]))
    subtrain.dt[train.row.vec, alpha := kernlab::alpha(fit)[[1]] ]
    subtrain.dt[, status := ifelse(
      alpha==0, "alpha=0",
      ifelse(alpha==cost.param, "alpha=C", "0<alpha<C"))]
    predictions.list[[paste(cost.param, degree.num)]] <- data.table(
      cost.param, degree.num, subtrain.dt)
  }
}
inputBoundaries <- do.call(rbind, inputBoundaries.list)
predictions <- do.call(rbind, predictions.list)
setErrors <- do.call(rbind, setErrors.list)
validationErrors <- setErrors[set=="validation"]
validationErrors$select <- "degree"
setErrors$select <- "cost"
animint(
  selectModel=ggplot()+
    ggtitle("Select hyper parameters")+
    geom_tallrect(aes(
      xmin=log10(cost.param)-cost.by/2,
      xmax=log10(cost.param)+cost.by/2),
      clickSelects="cost.param",
      alpha=0.5,
      data=setErrors[degree.num==1 & set=="subtrain",])+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "lines"))+
    theme_animint(width=350, rowspan=1)+
    facet_grid(select ~ ., scales="free")+
    ylab("")+
    geom_line(aes(
      log10(cost.param), errors,
      key=set,
      group=set,
      color=set),
      showSelected="degree.num",
      data=setErrors)+
    scale_fill_gradient("validErr", low="white", high="red")+
    geom_tile(aes(
      log10(cost.param), degree.num, fill=errors),
      clickSelects="degree.num",
      data=validationErrors),
  inputSpace=ggplot()+
    theme_bw()+
    ggtitle("Input space features")+
```

```
  scale_fill_manual(values=sv.colors, breaks=names(sv.colors))+
  geom_vline(aes(
    xintercept=boundary),
    color="violet",
    data=x1.boundaries)+
  scale_linetype_manual(values=c(
    margin="dashed",
    decision="solid"))+
  geom_path(aes(
    x1, x2,
    group=path.i,
    linetype=boundary),
    showSelected=c("degree.num", "cost.param"),
    color="black",
    data=inputBoundaries)+
  geom_point(aes(
    x1, x2, color=y, fill=status),
    showSelected=c("cost.param", "degree.num"),
    size=3,
    data=predictions))
```



12.4 Chapter summary and exercises

We used ggplots to visualize the Support Vector Machine model for binary classification. We used animint and interactivity to show how the SVM decision boundary changes as a function of the model hyper-parameters.

Exercises:

- Use HTML table layout for `viz.linear.svm`, so that the two feature space plots appear beside each other, and the "Select regularization parameter" plot appears above or below.
- Use `rbfdot` as the kernel function. Compute subtrain and validation error, then add a new panel to the "select hyper parameters" plot.
- Default scales use the same two colors for the **y** and **set** legends, which could be confusing.

Change the colors in one of the two legends so that they are different.

- Use `color` and `color_off` parameters to change the appearance of the `geom_tile` when selected or not, as explained in Chapter 6, section Specifying how selection state is displayed.

Next, Chapter 13 explains how to visualize the Poisson regression model.

13

Poisson regression

This goal of this chapter is to create an interactive data visualization that explains Poisson regression, a machine learning model for predicting an integer-valued output from inputs that are real-valued vectors. This is a "linear regression" model since it learns a linear function from the inputs to the output. Like least squares regression, Poisson regression can be formulated as a maximum likelihood problem. However, it differs from least squares linear regression since it uses a Poisson distribution to model the output labels, instead of a Gaussian distribution. This modeling choice is appropriate when output labels are non-negative integers.

Chapter outline:

- We begin by creating a plot that shows the probability mass function for a Poisson distribution mean parameter that can be interactively selected.
- We then add a second panel that shows the cumulative distribution function.
- We then add a second plot which shows the Poisson loss, with a second selector for label value.

13.1 Plot the probability mass function and select the Poisson mean parameter

The goal of this section is to create a data visualization that shows the probability mass function for a selected Poisson mean parameter.

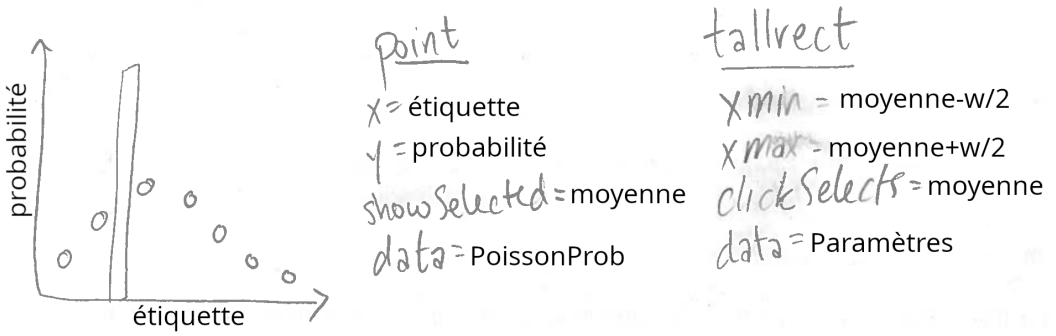


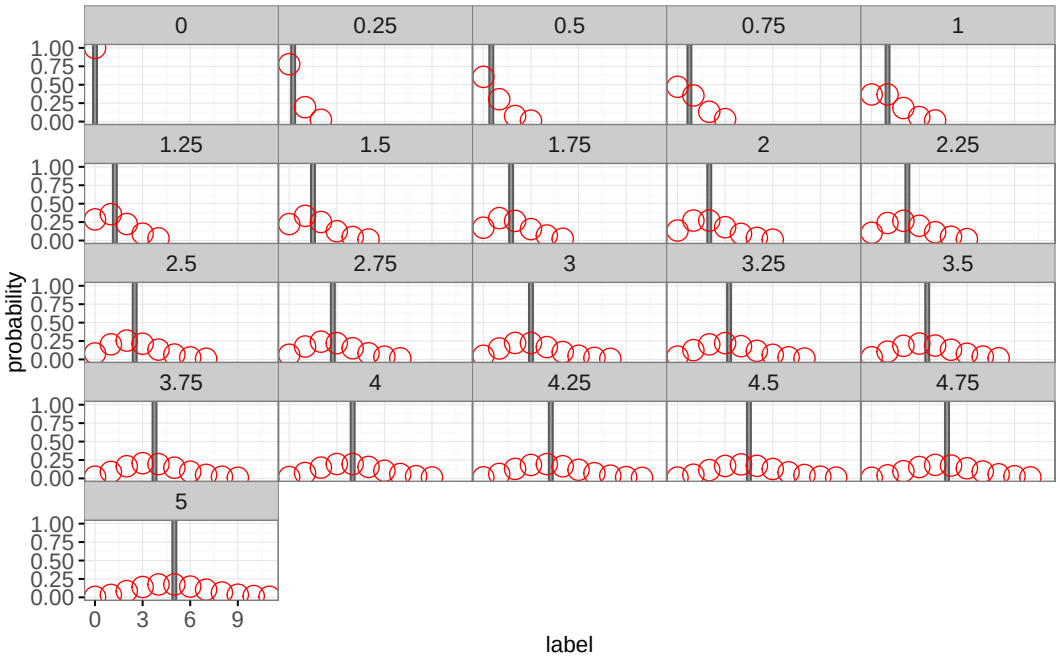
Figure 13.1: Poisson regression viz prob

```
library(data.table)
poisson.mean.diff <- 0.25
poisson.mean.vec <- seq(0, 5, by=poisson.mean.diff)
quantile.max <- 0.99
poisson.prob.list <- list()
for(poisson.mean in poisson.mean.vec){
  label.max <- qpois(quantile.max, poisson.mean)
  label <- 0:label.max
  probability <- dpois(label, poisson.mean)
  poisson.prob.list[[paste(poisson.mean)]] <- data.table(
    poisson.mean,
    label,
    probability,
    cum.prob=cumsum(probability))
}
poisson.prob <- do.call(rbind, poisson.prob.list)
poisson.prob
```

```
     poisson.mean label probability  cum.prob
  1:         0.00     0 1.000000000 1.0000000
  2:         0.25     0 0.778800783 0.7788008
 ---
155:         5.00    10 0.018132789 0.9863047
156:         5.00    11 0.008242177 0.9945469
```

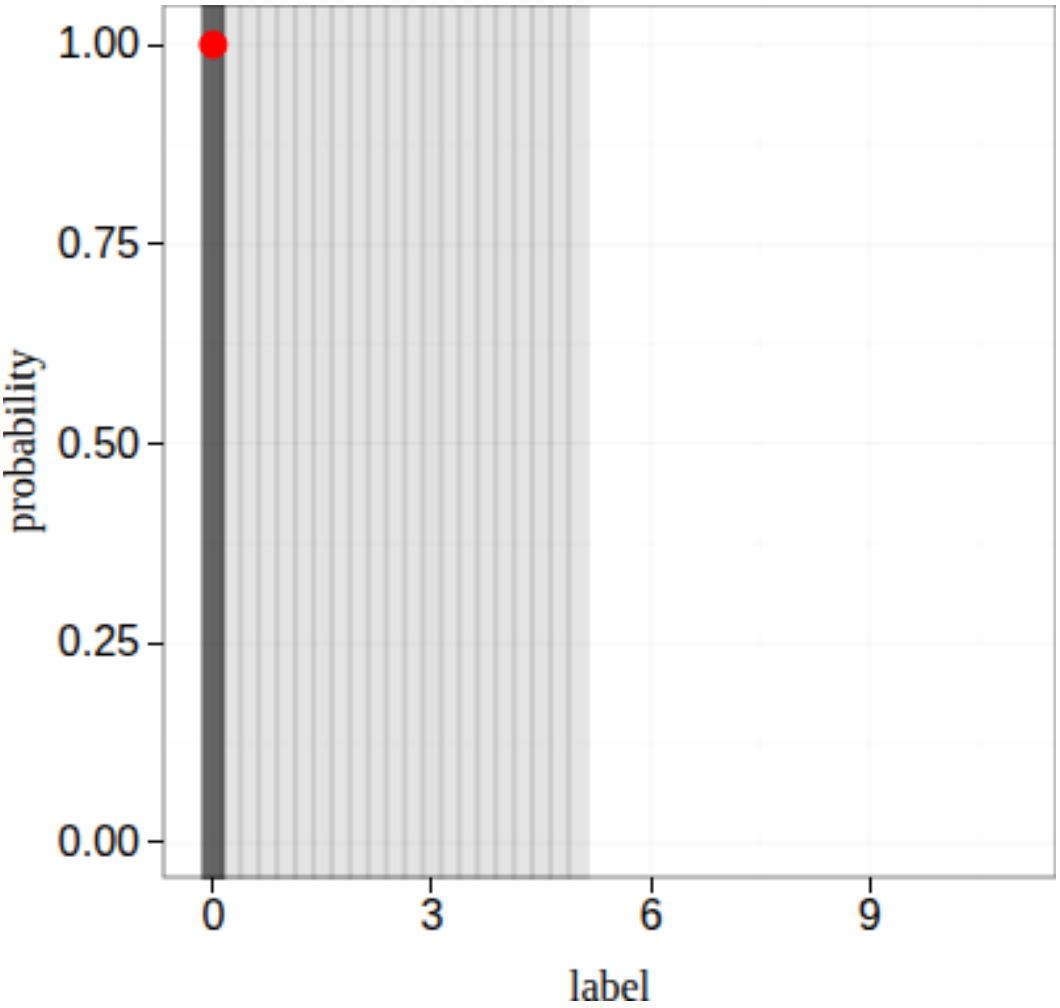
The static data viz below shows one facet for each Poisson distribution.

```
mean.tallrects <- data.table(
  poisson.mean=poisson.mean.vec,
  min=poisson.mean.vec - poisson.mean.diff/2,
  max=poisson.mean.vec + poisson.mean.diff/2)
library(animint2)
prob.mass <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "cm"))+
  geom_tallrect(aes(
    xmin=min, xmax=max),
    clickSelects="poisson.mean",
    alpha=0.6,
    data=mean.tallrects)+
  geom_point(aes(
    label, probability,
    tooltip=sprintf("prob(label = %d) = %f", label, probability)),
    color="red",
    showSelected="poisson.mean",
    size=5,
    data=poisson.prob)
prob.mass+
  facet_wrap("poisson.mean")
```



Note that we used `alpha=0.6` with `geom_tallrect`, which means that the tallrect for the selected mean has 0.6 opacity, and the other tallrects have 0.1 opacity. Note also that we use`color="red"` and `size=5` with `geom_point` so that it is easier to see the points on a grey background, and to hover the cursor over the points to see the tooltip. We next create an interactive version with animint.

```
animint(prob.mass)
```



You can click the viz above to change the mean of the Poisson distribution. You can also hover the cursor over a data point to see its probability. Note that for integer values of the Poisson mean, there are two labels that are the most probable (the mode of the Poisson distribution). For example the Poisson distribution with a mean of 3 attains its maximum probability of about 0.224 at label values of 2 and 3.

13.2 Add a panel for the cumulative distribution function

To add a panel for the cumulative distribution function, we will re-make the ggplot based on the sketch below.

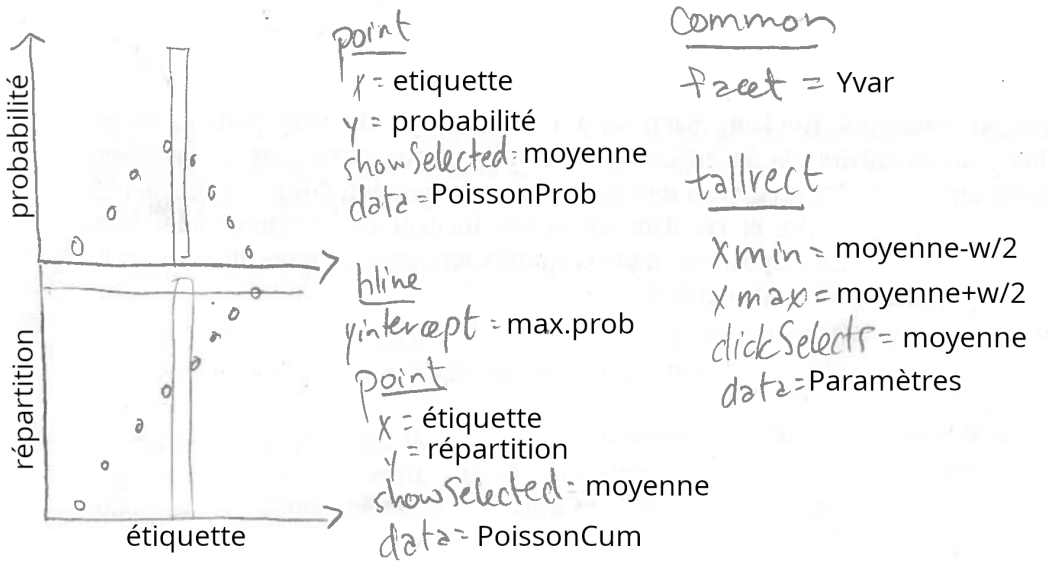
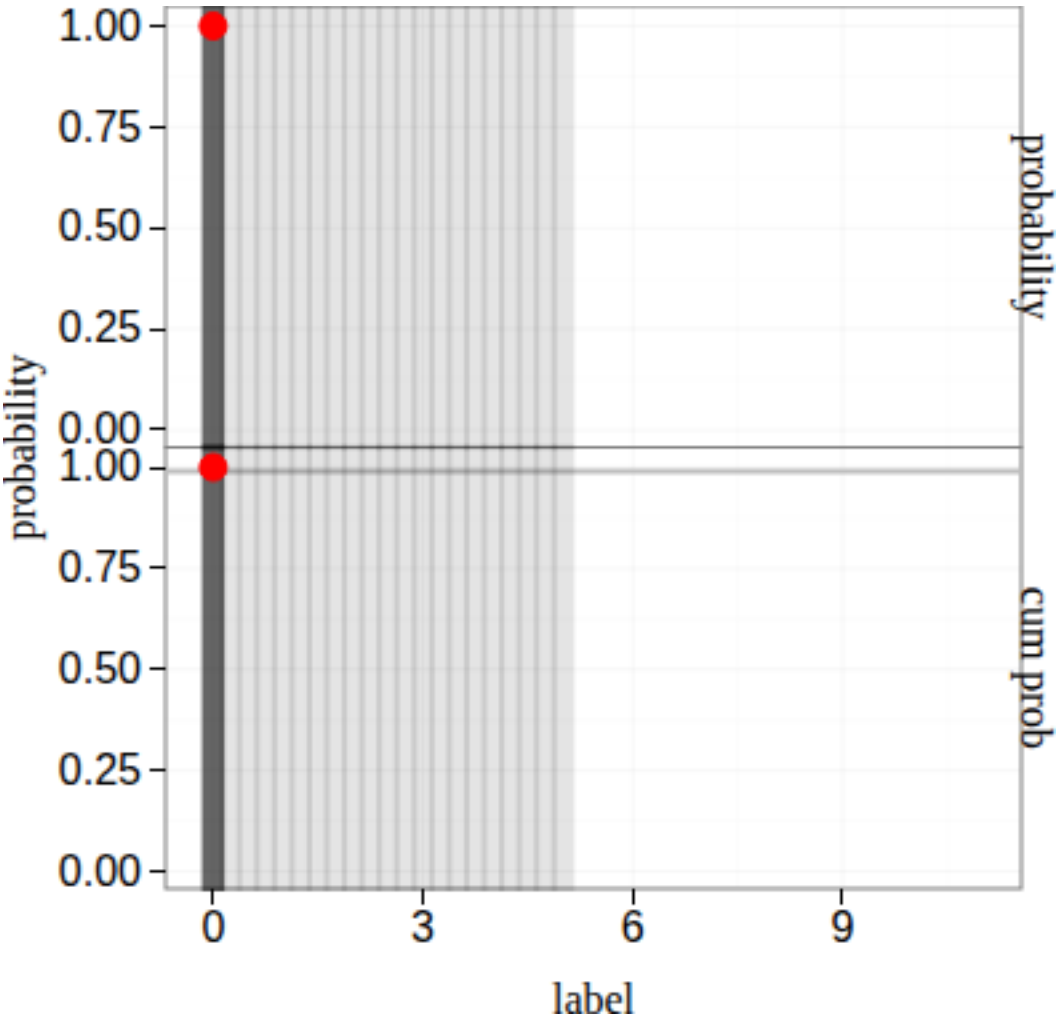


Figure 13.2: Poisson regression viz cum prob

When we specify the data sets, we will use the addColumn then facet idiom to add a `panel` variable.

```
addPanel <- function(dt, panel){
  data.table(dt, panel=factor(panel, c("probability", "cum prob")))
}
quantile.max.dt <- data.table(quantile.max)
animint(
  prob=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    facet_grid(panel ~ ., scales="free")+
    geom_hline(aes(
      yintercept=quantile.max),
      color="grey",
      data=addPanel(quantile.max.dt, "cum prob"))+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=mean.tallrects)+
    geom_point(aes(
      label, probability,
      tooltip=sprintf(
        "prob(label = %d) = %f", label, probability)),
      showSelected="poisson.mean",
      color="red",
      size=5,
      data=addPanel(poisson.prob, "probability"))+
```

```
      geom_point(aes(
        label, cum.prob,
        tooltip=sprintf(
          "prob(label <= %d) = %f", label, cum.prob)),
        showSelected="poisson.mean",
        color="red",
        size=5,
        data=addPanel(poisson.prob, "cum prob")))
```



Note how we used `addPanel` to add a `panel` variable to all the data sets for each geom except `geom_tallrect`. Using`panel` as a facet variable has the effect of drawing each geom in only one panel, except the `geom_tallrect` which is drawn in each panel.

Note that we also used a `geom_hline` to show 0.99, the cumulative distribution function threshold that was used to determine the set of points to plot for each Poisson distribution. This is an example of "show your arbitrary choices," one of the general principles of designing good interactive data visualizations.

13.3 Add a plot of the Poisson loss and a selector for label value

Next we will compute the Poisson loss for several values of the output label.

```
PoissonLoss <- function(label, seg.mean){
  stopifnot(is.numeric(label))
  stopifnot(is.numeric(seg.mean))
  if(any(seg.mean < 0)){
    stop("PoissonLoss undefined for negative segment mean")
  }
  if(length(seg.mean)==1)seg.mean <- rep(seg.mean, length(label))
  if(length(label)==1)label <- rep(label, length(seg.mean))
  stopifnot(length(seg.mean) == length(label))
  not.integer <- round(label) != label
  is.negative <- label < 0
  loss <- ifelse(
    not.integer | is.negative, Inf,
    ifelse(seg.mean == 0, ifelse(label == 0, 0, Inf),
           seg.mean - label * log(seg.mean)
           ## This term makes all the minima zero.
           -ifelse(label == 0, 0, label - label*log(label))))
  loss
}
```

Below we compute the loss for several label values, using the list of data tables idiom.

```
label.vec <- unique(poisson.prob$label)
label.range <- range(label.vec)
mean.vec <- seq(label.range[1], label.range[2], l=100)
loss.min.list <- list()
loss.fun.list <- list()
for(label in label.vec){
  loss <- PoissonLoss(label, mean.vec)
  loss.fun.list[[paste(label)]] <- data.table(
    label, poisson.mean=mean.vec, loss)
  loss.min.list[[paste(label)]] <- data.table(
    label, loss=0)
}
loss.fun <- do.call(rbind, loss.fun.list)
loss.min <- do.call(rbind, loss.min.list)
```

We also make a data table to display text labels for the selected mean and label values.

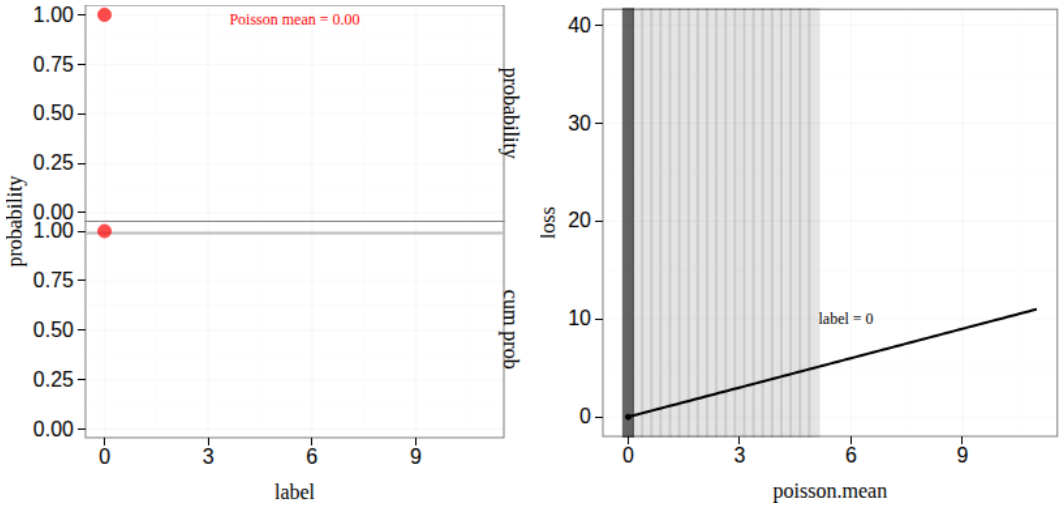
```
mean.text <- data.table(
  label=max(poisson.prob$label)/2,
  probability=0.95,
  poisson.mean=poisson.mean.vec)
loss.max <- 10
```

```
label.text <- data.table(
  poisson.mean=max(mean.tallrects$max),
  loss=loss.max*0.95,
  label=label.vec)
```

Next we make a data viz with an additional panel.

```
(viz.loss <- animint(
  prob=ggplot()+
    theme_bw()+
    theme(panel.margin=grid::unit(0, "cm"))+
    facet_grid(panel ~ ., scales="free")+
    geom_text(aes(
      label, probability, label=sprintf(
      "Poisson mean = %.2f", poisson.mean)),
      color="red",
      showSelected="poisson.mean",
      data=addPanel(mean.text, "probability"))+
    geom_hline(aes(
      yintercept=quantile.max),
      color="grey",
      data=addPanel(quantile.max.dt, "cum prob"))+
    geom_point(aes(
      label, probability,
      tooltip=sprintf(
        "prob(label = %d) = %f", label, probability)),
      showSelected="poisson.mean",
      clickSelects="label",
      color="red",
      size=5,
      alpha=0.7,
      data=addPanel(poisson.prob, "probability"))+
    geom_point(aes(
      label, cum.prob,
      tooltip=sprintf(
        "prob(label <= %d) = %f", label, cum.prob)),
      color="red",
      showSelected="poisson.mean",
      clickSelects="label",
      size=5,
      alpha=0.7,
      data=addPanel(poisson.prob, "cum prob")),
  loss=ggplot()+
    theme_bw()+
    geom_text(aes(
      poisson.mean, loss,
      label=sprintf("label = %d", label)),
      showSelected="label",
      hjust=0,
```

```
      data=label.text)+
    geom_line(aes(
      poisson.mean, loss),
      showSelected="label",
      data=loss.fun)+
    geom_point(aes(
      label, loss),
      showSelected="label",
      data=loss.min)+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=mean.tallrects)))
```



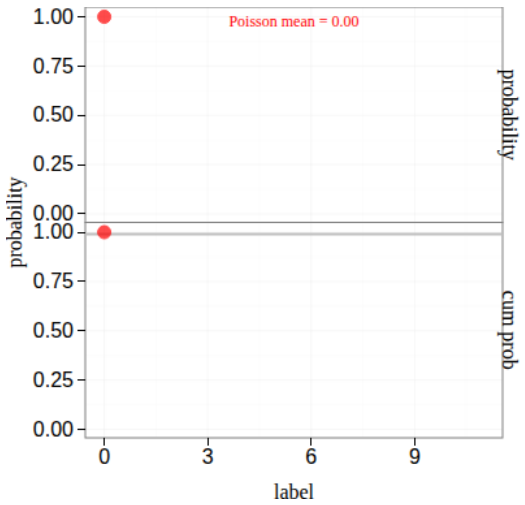
The data viz above shows the probability on the left and the Poisson loss on the right.

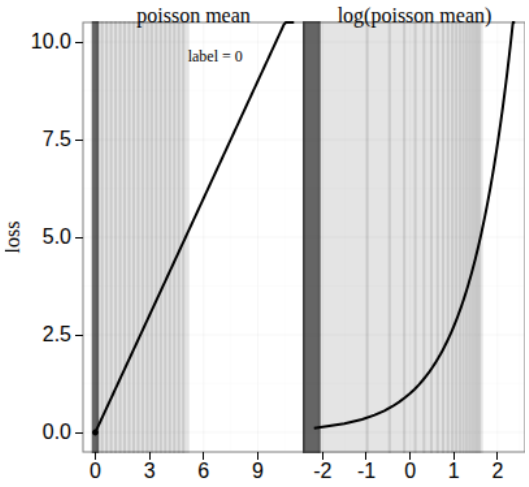
```
viz.log.loss <- viz.loss
addX <- function(dt, x.var)data.table(dt, x.var=factor(
  x.var, c("poisson mean", "log(poisson mean)")))
finite.loss <- loss.fun[is.finite(loss)]
finite.loss[, log.poisson.mean := log(poisson.mean)]
finite.log.loss <- finite.loss[is.finite(log.poisson.mean)]
mean.tallrects[, log.min := ifelse(min < 0, -Inf, log(min))]
```

```
Warning in log(min): NaNs produced
```

```
viz.log.loss$loss <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(. ~ x.var, scales="free")+
  xlab("")+
  coord_cartesian(ylim=c(0, loss.max))+
```

```
    geom_text(aes(
      poisson.mean, loss, label=sprintf(
                            "label = %d", label)),
      showSelected="label",
      hjust=0,
      data=addX(label.text, "poisson mean"))+
    geom_line(aes(
      poisson.mean, loss),
      showSelected="label",
      data=addX(finite.loss, "poisson mean"))+
    geom_point(aes(
      label, loss),
      showSelected="label",
      data=addX(loss.min, "poisson mean"))+
    geom_tallrect(aes(
      xmin=min, xmax=max),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=addX(mean.tallrects, "poisson mean"))+
    geom_line(aes(
      log.poisson.mean, loss),
      showSelected="label",
      data=addX(finite.log.loss, "log(poisson mean)"))+
    geom_point(aes(
      log(label), loss),
      showSelected="label",
      data=addX(loss.min[0<label,], "log(poisson mean)"))+
    geom_tallrect(aes(
      xmin=log.min, xmax=log(max)),
      clickSelects="poisson.mean",
      alpha=0.6,
      data=addX(mean.tallrects, "log(poisson mean)"))
  viz.log.loss
```





13.4 Chapter summary and exercises

We explained how to visualize the Poisson distribution and loss, which are used for the Poisson regression model.

Exercises:

- The code above used `addPanel` and `addX` helper functions with several geoms to create multi-panel plots, which results in repetition. To avoid that repetition, create a new data viz which uses a single geom with a larger data set. For example, the red points in the two panels of the first plot could be defined using one `geom_point` with a larger data set (Hint: use `data.table::melt` with `measure.vars=c("cum.prob", "probability")`.
- Create a similar sequence of data visualizations for the Binomial regression model.

Next, Chapter 14 explains how to use the named clickSelects/showSelected to visualize the PeakSegJoint machine learning model with data-driven selector variables.

14

Named `clickSelects` *and* `showSelected`

This chapter explains how to use named `clickSelects` and `showSelected` variables for creating data-driven selector names. This feature makes it easier to write animint code, and makes it faster to compile.

Chapter outline:

- We begin by attaching the `PSJ` data set and computing the data to plot.
- We show one method of defining an animint with many selectors, using for loops. This method is technically correct, but computationally inefficient.
- We then explain the preferred method for defining an animint with many selectors, using named `clickSelects` and `showSelected`. This method is more computationally efficient, and easier to code.

14.1 Download data set

The example data come from the PeakSegJoint package, which is for peak detection in genomic data sequences. The code below downloads the data set.

```
if(!requireNamespace("animint2data"))
  remotes::install_github("animint/animint2data")
```

```
Loading required namespace: animint2data
```

```
data(PSJ, package="animint2data")
sapply(PSJ, class)
```

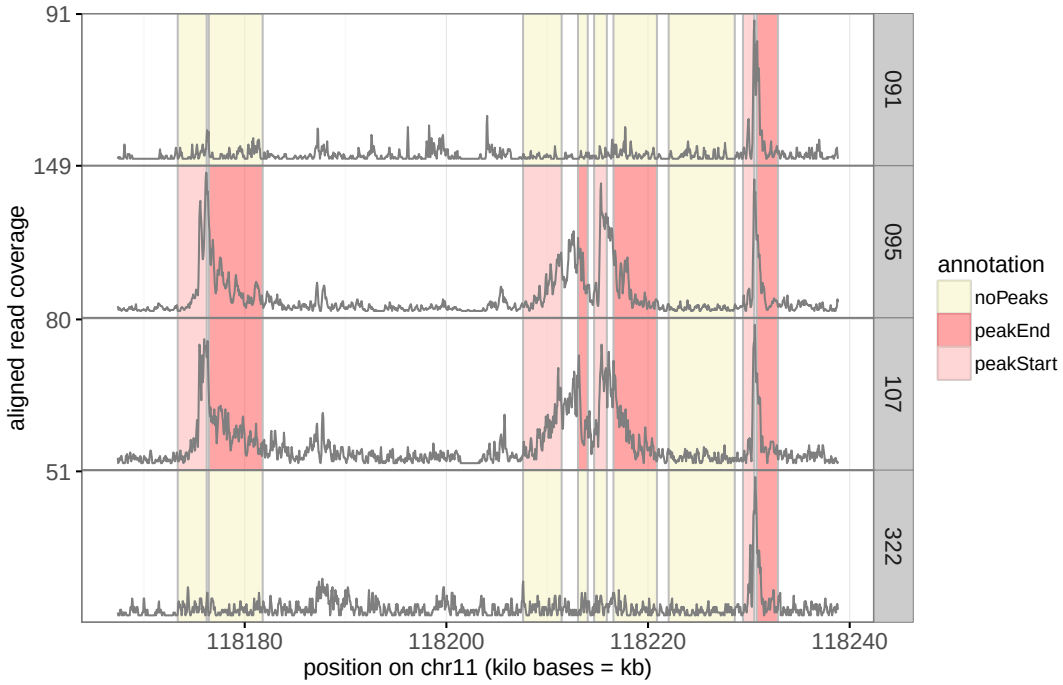
```
          problem.labels                 problems            filled.regions
            "data.frame"              "data.frame"              "data.frame"
                coverage                    first modelSelection.by.problem
            "data.frame"                   "list"                    "list"
     regions.by.problem         peaks.by.problem        error.total.chunk
                "list"                   "list"              "data.frame"
        error.total.all
            "data.frame"
```

Above we see that `PSJ` is a list of several lists and data frames.

14.2 Explore PSJ data with static ggplots

We begin by a plot of some genomic ChIP-seq data, which are sequential data that take large values when there is an active area (typically actively transcribed genes). In the code below we use show each sample in a separate panel.

```
library(animint2)
ann.colors <- c(
  noPeaks="#f6f4bf",
  peakStart="#ffafaf",
  peakEnd="#ff4c4c",
  peaks="#a445ee")
(gg.cov <- ggplot()+
  scale_y_continuous(
    "aligned read coverage",
    breaks=function(limits){
      floor(limits[2])
    })+
  scale_x_continuous(
    "position on chr11 (kilo bases = kb)")+
  coord_cartesian(xlim=c(118167.406, 118238.833))+
  geom_tallrect(aes(
    xmin=chromStart/1e3, xmax=chromEnd/1e3,
    fill=annotation),
    alpha=0.5,
    color="grey",
    data=PSJ$filled.regions)+
  scale_fill_manual(values=ann.colors)+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "cm"))+
  facet_grid(sample.id ~ ., labeller=function(df){
    df$sample.id <- sub("McGill0", "", sub(" ", "\n", df$sample.id))
    df
  }, scales="free")+
  geom_line(aes(
    base/1e3, count),
    data=PSJ$coverage,
    color="grey50"))
```



The figure above shows the raw noisy data as a grey `geom_line()`. Colored rectangles represent labels that indicate whether or not a peak start or end should be predicted in a given region and sample. Next, we add a panel for segmentation problems, in which an algorithm looked for a common peak across samples. The algorithm predicts start/end positions for a peak of large data values, in each problem. The code below computes a table with one row for each such problem.

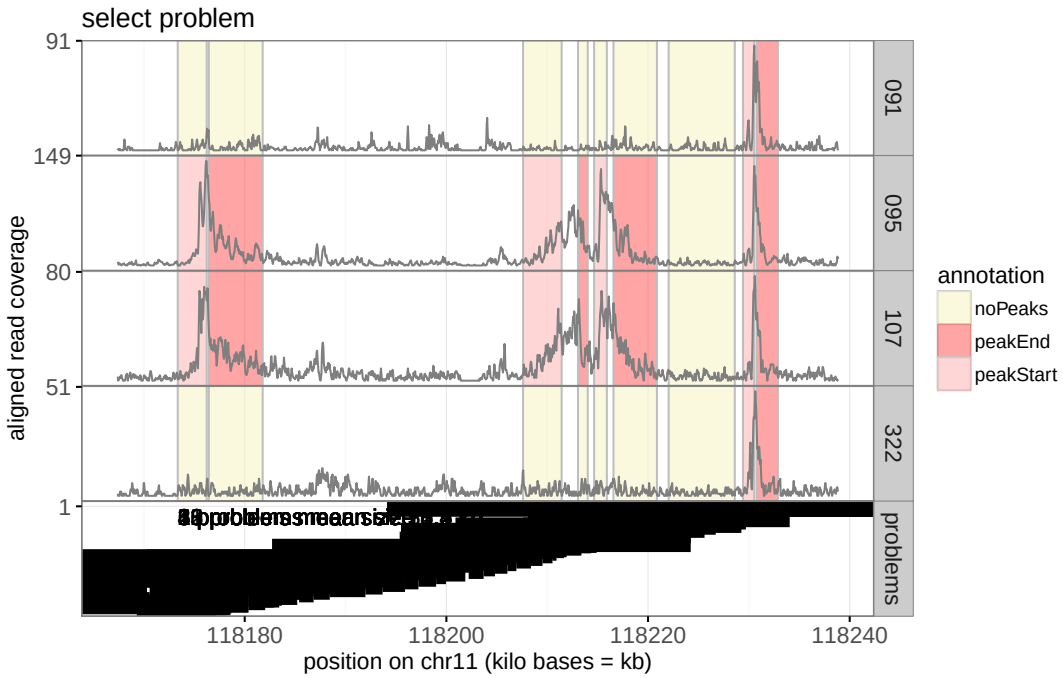
```
library(data.table)
(show.problems <- data.table(PSJ$problems)[
, y := problem.i/max(problem.i), by=bases.per.problem][])
```

```
     sample.id bases.per.problem problem.i              problem.name
  1:  problems              1629         1 chr11:118172963-118174591
  2:  problems              1629         2 chr11:118175144-118176570
 ---
203:  problems            104267         1 chr11:118125367-118194817
204:  problems            104267         2 chr11:118194818-118266077
     problemStart problemEnd          y
  1:    118172963  118174591 0.01818182
  2:    118175144  118176570 0.03636364
 ---
203:    118125367  118194817 0.50000000
204:    118194818  118266077 1.00000000
```

The code above added the `y` column which is used to display the problems in the code below.

```
(gg.cov.prob <- gg.cov+
  ggtitle("select problem")+
```

```
  geom_text(aes(
    chromStart/1e3, 0.9,
    label=sprintf(
      "%d problems mean size %.1f kb",
      problems, mean.bases/1e3)),
    showSelected="bases.per.problem",
    data=PSJ$problem.labels,
    hjust=0)+
  geom_segment(aes(
    problemStart/1e3, y,
    xend=problemEnd/1e3, yend=y),
    showSelected="bases.per.problem",
    clickSelects="problem.name",
    size=5,
    data=show.problems))
```



Above we see a bottom `problems` panel was added to the previous plot. The static graphic above is overplotted; the interactive version will be readable because it will only show one value of `bases.per.problem` at a time. To select the different values of `bases.per.problem` (problem size), we will use another plot, which shows the best error rate for each problem size, as in the data below.

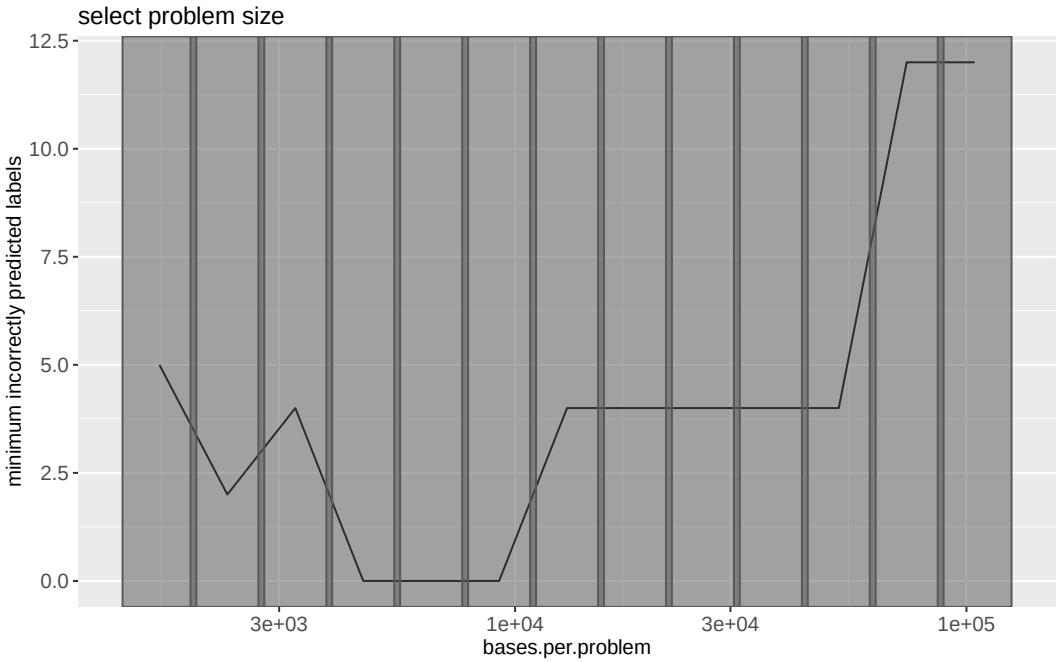
```
  (res.error <- data.table(PSJ$error.total.chunk))
```

```
   bases.per.problem fp fn errors regions min.bases.per.problem
1:              1629  1  4      5      36              1349.501
2:              2304  0  2      2      36              1908.686
---
```

```
12:             73728  0 12     12      36             61077.960
13:            104267  0 12     12      36             86377.166
    max.bases.per.problem
 1:              1966.387
 2:              2781.188
---
12:             88998.027
13:            125862.051
```

The table above has one row per value of `bases.per.problem`, which is a sliding window size parameter, that we will explore with interactivity. We use these data to draw the plot below.

```
(gg.res.error <- ggplot()+
  ggtitle("select problem size")+
  ylab("minimum incorrectly predicted labels")+
  geom_line(aes(
    bases.per.problem, errors),
    data=res.error)+
  geom_tallrect(aes(
    xmin=min.bases.per.problem,
    xmax=max.bases.per.problem),
    clickSelects="bases.per.problem",
    alpha=0.5,
    data=res.error)+
  scale_x_log10())
```



The figure above shows the minimum number of label errors as a function of problem size. Grey rectangles will be used to select the problem size.

There is a penalty parameter which controls the number of samples with a common peak, as defined in the model selection data table in the code below.

```
pdot <- function(L){
  out_list <- list()
  for(i in seq_along(L)){
    out_list[[i]] <- data.table(
      problem.dot=names(L)[[i]], L[[i]])
  }
  rbindlist(out_list)
}
(all.modelSelection <- pdot(PSJ$modelSelection.by.problem))
```

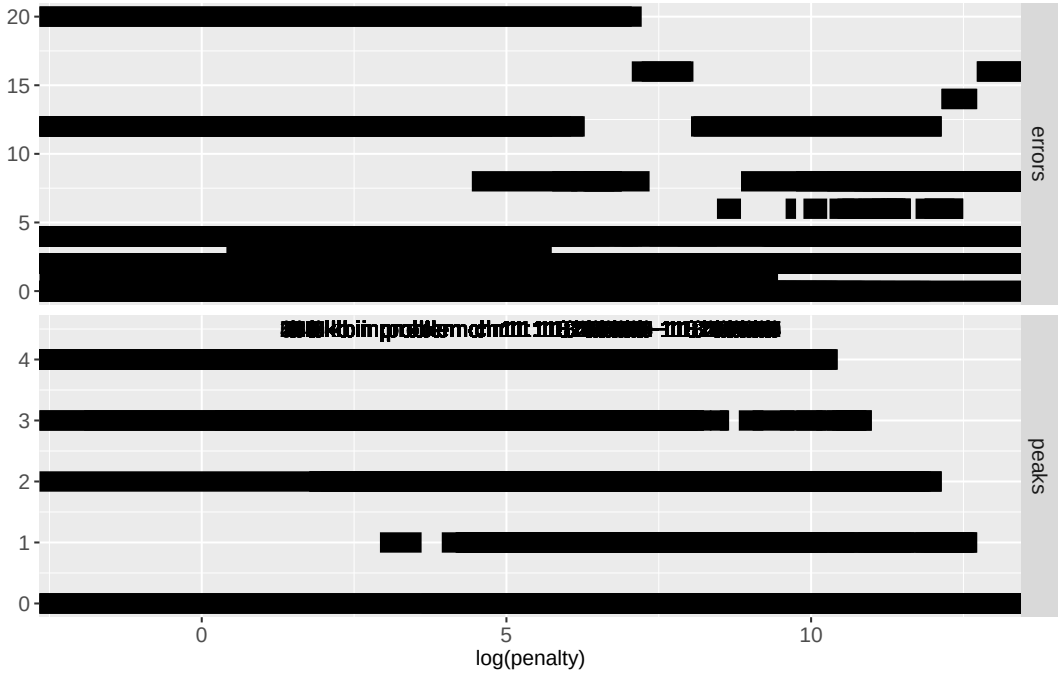
```
                      problem.dot bases.per.problem problem.i
  1: chr11.118172963.118174591peaks              1629         1
  2: chr11.118172963.118174591peaks              1629         1
 ---
979: chr11.118194818.118266077peaks            104267         2
980: chr11.118194818.118266077peaks            104267         2
                  problem.name problemStart problemEnd min.log.lambda
  1: chr11:118172963-118174591    118172963  118174591           -Inf
  2: chr11:118172963-118174591    118172963  118174591        4.56662
 ---
979: chr11:118194818-118266077    118194818  118266077       12.13670
980: chr11:118194818-118266077    118194818  118266077       12.71684
     max.log.lambda model.complexity peaks   min.lambda   max.lambda errors
  1:        4.56662                4     4      0.00000     96.21835     NA
  2:        4.68868                3     3     96.21835    108.70956     NA
 ---
979:       12.71684                1     1 186596.24134 333312.50992     14
980:            Inf                0     0 333312.50992          Inf     16
```

The code above uses the `pdot()` function, which uses the list of data tables idiom to add a column named `problem.dot`, which will be used below to define selectors in the interactive visualization. Below we plot the number of peaks and label errors, as a function of penalty parameter of the algorithm.

```
long.modelSelection <- melt(
  data.table(all.modelSelection)[, errors := as.numeric(errors)],
  measure.vars=c("peaks","errors"))
log.lambda.range <- all.modelSelection[, c(
  min(max.log.lambda), max(min.log.lambda))]
modelSelection.labels <- unique(all.modelSelection[, data.table(
  problem.name,
  bases.per.problem,
  problemStart,
  problemEnd,
  log.lambda=mean(log.lambda.range),
  peaks=max(peaks)+0.5)])
(gg.model.selection <- ggplot()+
```

```
  scale_x_continuous("log(penalty)")+
  geom_segment(aes(
    min.log.lambda, value,
    xend=max.log.lambda, yend=value),
    showSelected=c("bases.per.problem", "problem.name"),
    data=long.modelSelection,
    size=5)+
  geom_text(aes(
    log.lambda, peaks,
    label=sprintf(
      "%.1f kb in problem %s",
      (problemEnd-problemStart)/1e3, problem.name)),
    showSelected=c("bases.per.problem", "problem.name"),
    data=data.frame(modelSelection.labels, variable="peaks"))+
  ylab("")+
  facet_grid(variable ~ ., scales="free"))
```

```
Warning: Removed 717 rows containing missing values (geom_segment).
```



The figure above shows `errors` (top panel) and `peaks` (bottom panel) as a function of `log(penalty)`. Again this static version is overplotted; interactivity will be used so that this figure is readable (only shows the subset of data corresponding to the selected values of`bases.per.problem` and `problem.name`).

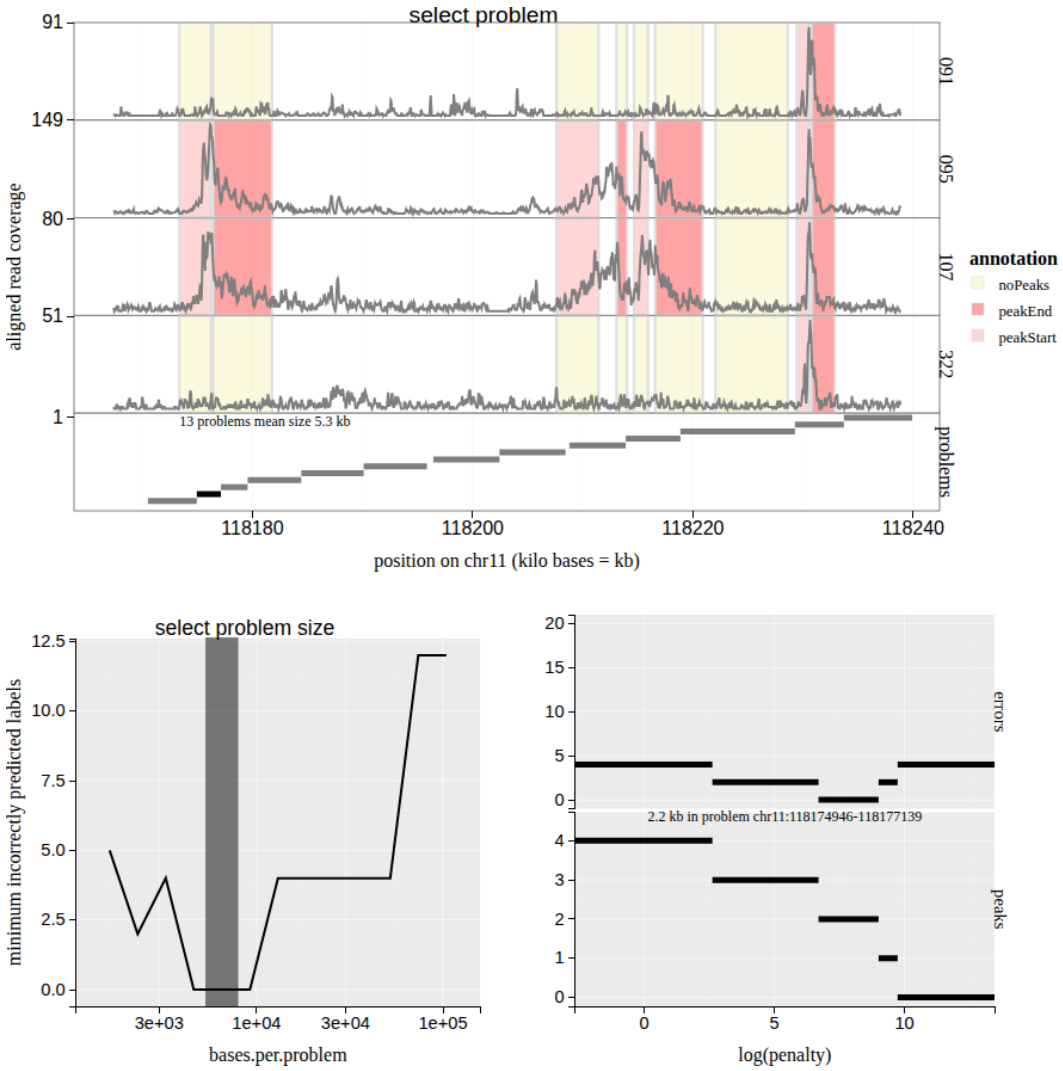
14.3 Interactive data visualization (incomplete)

In this section, we combine the ggplots from the previous section into a linked interactive data visualization. The code below uses `theme_animint()` to attach some display options to the previous coverage plot, and adds the `first` option to specify what data subsets should be displayed first.

```
timing.incomplete.construct <- system.time({
  coverage.counts <- table(PSJ$coverage$sample.id)
  facet.rows <- length(coverage.counts)+1
  viz.incomplete <- animint(
    first=list(
      bases.per.problem=6516,
      problem.name="chr11:118174946-118177139"),
    coverage=gg.cov.prob+theme_animint(
      last_in_row=TRUE, colspan=2,
      width=800, height=facet.rows*100),
    resError=gg.res.error,
    modelSelection=gg.model.selection)
})
```

The timing output above shows that the initial definition is fast. Rendering this preliminary (incomplete) data viz in the code below is also fast.

```
before.incomplete <- Sys.time()
viz.incomplete
```




```
cat(elapsed.incomplete <- Sys.time()-before.incomplete, "seconds\n")
```

```
7.577811 seconds
```

We see a data visualization above with three plots.

- On top, four ChIP-seq data profiles are shown, along with a problems panel which divides the X axis into problems in which the segmentation algorithm runs.
- Clicking the bottom left plot selects problem size, which updates the problems displayed on top.
- The bottom right plot shows the number of peaks and errors as a function of penalty (larger for fewer peaks).

14.4 Add interactivity using for loops

In this section, we add layers to the previous ggplots using a for loop, which is sub-optimal, but we show it for comparison with the better approach (named `clickSelects` and `showSelected`) which is presented in the next section. The visualization in the previous section is incomplete. We would like to add

- rectangles in the bottom left plot which would allow us to select the number of peaks predicted in the given problem.
- segments and rectangles in the top plot which would show the predicted peaks and label errors.

One (inefficient) way of adding those would be via a for loop, which is coded below. For every problem there is a selector (called `problem.dot`) for the number of peaks in that problem. So in this for loop we add a few layers with `clickSelects="problem.dot"` or `showSelected="problem.dot"` to the `coverage` and `modelSelection` plots.

```
viz.first <- viz.incomplete
viz.first$first <- c(viz.incomplete$first, PSJ$first)
viz.first$modelSelection <- viz.first$modelSelection+
  ggtitle("select number of samples with peak in problem")
print(timing.for.construct <- system.time({
  viz.for <- viz.first
  viz.for$title <- "PSJ with for loops"
  for(problem.dot in names(PSJ$modelSelection.by.problem)){
    if(problem.dot %in% names(PSJ$peaks.by.problem)){
      peaks <- PSJ$peaks.by.problem[[problem.dot]]
      peaks[[problem.dot]] <- peaks$peaks
      prob.peaks.names <- c(
        "bases.per.problem", "problem.i", "problem.name",
        "chromStart", "chromEnd", problem.dot)
      prob.peaks <- unique(data.frame(peaks)[, prob.peaks.names])
      prob.peaks$sample.id <- "problems"
      viz.for$coverage <- viz.for$coverage +
        geom_segment(aes(
          chromStart/1e3, 0,
          xend=chromEnd/1e3, yend=0),
          clickSelects="problem.name",
          showSelected=c(problem.dot, "bases.per.problem"),
          data=peaks, size=7, color="deepskyblue")
    }
    modelSelection.dt <- PSJ$modelSelection.by.problem[[problem.dot]]
    modelSelection.dt[[problem.dot]] <- modelSelection.dt$peaks
    viz.for$modelSelection <- viz.for$modelSelection+
      geom_tallrect(aes(
        xmin=min.log.lambda,
        xmax=max.log.lambda),
        clickSelects=problem.dot,
        showSelected=c("problem.name", "bases.per.problem"),
```

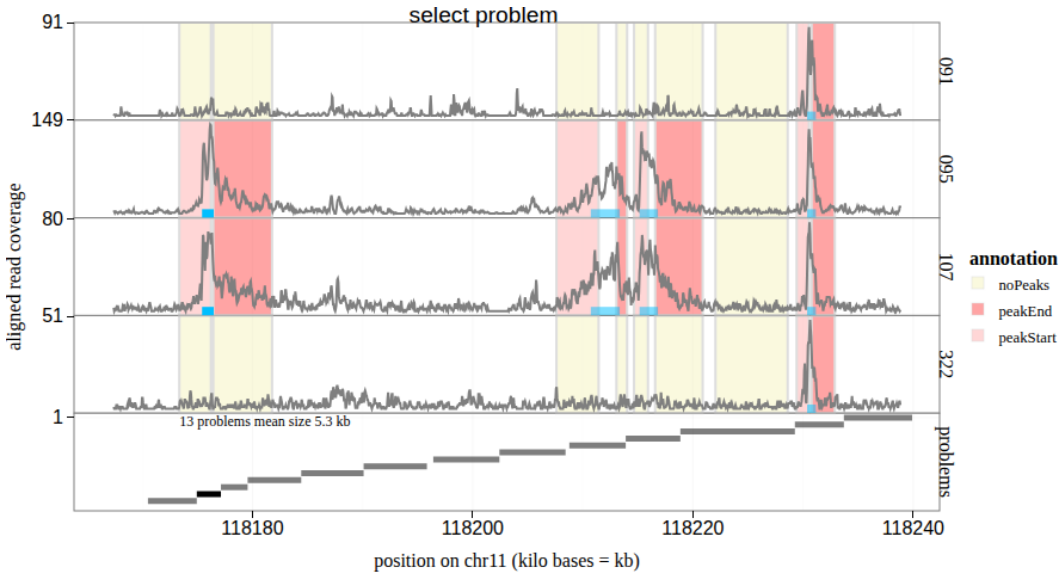
```
        data=modelSelection.dt, alpha=0.5)
  }
}))
```

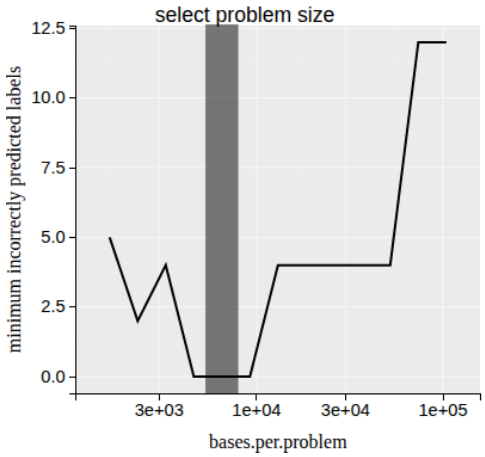
```
   user  system elapsed
  0.992   0.001   0.993
```

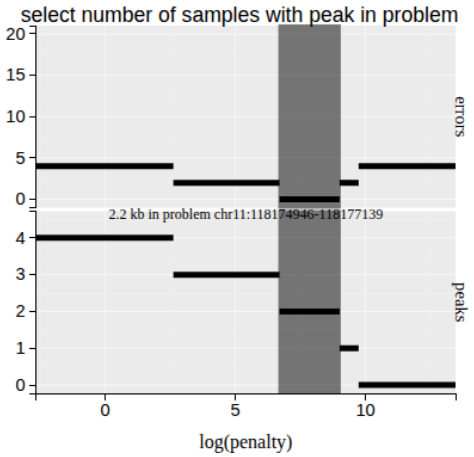
Note the timing of the code above, which just evaluates the R code that defines this data viz. Next, we compile the data visualization.

```
before.for <- Sys.time()
viz.for
```

```
Warning in checkSingleShowSelectedValue(meta$selectors): showSelected variables
with only 1 level: chr11.118184422.118184700peaks,
chr11.118192951.118193582peaks, chr11.118203893.118204314peaks
```







```
  cat(elapsed.for <- Sys.time()-before.for, "seconds\n")
```

```
1.204538 seconds
```

Note that the compilation takes several seconds, since there are so many geoms (click Show download status table to see all of them). Compared to the data visualization from the previous section, this one has

- blue segments that appear in the top coverage data plot, to indicate predicted peaks.
- selection rectangles that can be clicked in the bottom right plot, to change the number of samples with a peak in the selected problem.

In the next section we will create the same data viz, but more efficiently.

14.5 Add interactivity using named `clickSelects` and `showSelected`

In this section we use named `clickSelects` and `showSelected` to create a more efficient version of the previous data visualization. In general, any data visualization defined using for loops in R code can be made more efficient by instead using this method. First, we define some common data.

```
  (sample.peaks <- pdot(PSJ$peaks.by.problem))
```

```
                      problem.dot bases.per.problem problem.i
   1: chr11.118172963.118174591peaks              1629         1
   2: chr11.118172963.118174591peaks              1629         1
  ---
1998: chr11.118194818.118266077peaks            104267         2
1999: chr11.118194818.118266077peaks            104267         2
                  problem.name problemStart problemEnd peaks  sample.id
   1: chr11:118172963-118174591    118172963  118174591     1 McGill0091
   2: chr11:118172963-118174591    118172963  118174591     2 McGill0091
  ---
1998: chr11:118194818-118266077    118194818  118266077     4 McGill0091
1999: chr11:118194818-118266077    118194818  118266077     4 McGill0322
      chromStart  chromEnd     mean
   1:  118173535 118173819 3.802817
   2:  118173535 118173819 3.802817
  ---
1998:  118209753 118218118 2.190317
1999:  118209753 118218118 2.810879
```

The output above shows a table with one row per peak that can be displayed, for different samples, problems, and interactive choices of `bases.per.problem` and `peaks` parameters. Note the `problem.dot` column which defines the name of the selector that will store the currently selected number of peaks for that problem.

In the code below, the main idea is that for every problem, there is a selector defined by the `problem.dot` column, for the number of peaks in that problem. We use

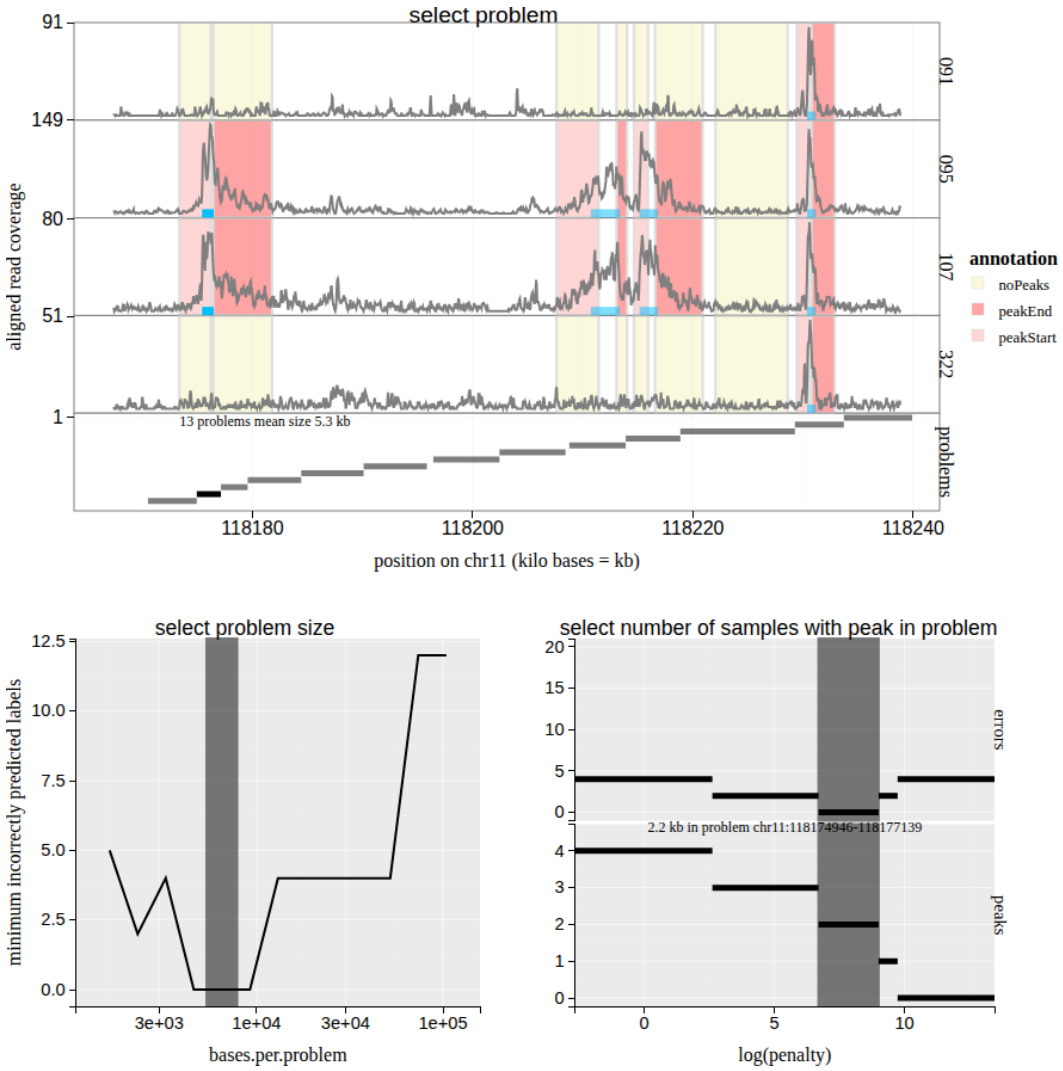
`showSelected=c(problem.dot="peaks")` and `clickSelects=c(problem.dot="peaks")` to indicate that the selector name is found in the `problem.dot` column, and the selection value is found in the `peaks`column. The`animint2dir()` compiler creates a selection variable for every unique value of `problem.dot` (and it uses corresponding values in `peaks` to set/update the selected value/geoms).

```
print(timing.named.construct <- system.time({
  viz.named <- viz.first
  viz.named$title <- "PSJ named clickSelects and showSelected"
  viz.named$coverage <- viz.named$coverage+
    geom_segment(aes(
      chromStart/1e3, 0,
      xend=chromEnd/1e3, yend=0),
      clickSelects="problem.name",
      showSelected=c(problem.dot="peaks", "bases.per.problem"),
      data=sample.peaks, size=7, color="deepskyblue")
  viz.named$modelSelection <- viz.named$modelSelection+
    geom_tallrect(aes(
      xmin=min.log.lambda,
      xmax=max.log.lambda),
      clickSelects=c(problem.dot="peaks"),
      showSelected=c("problem.name", "bases.per.problem"),
      data=all.modelSelection, alpha=0.5)
}))
```

```
   user  system elapsed
  0.002   0.000   0.003
```

It is clear that it takes much less time to evaluate the R code above which uses the named `clickSelects` and `showSelected`. We compile it below.

```
before.named <- Sys.time()
viz.named
```




```
cat(elapsed.named <- Sys.time()-before.named, "seconds\n")
```

```
7.302212 seconds
```

The animint produced above should appear to be the same as the other data viz from the previous section. The timings above show that named `clickSelects` and `showSelected` are much faster than for loops, in both the definition and compilation steps.

14.6 Disk usage comparison

In this section we compute the disk usage of both methods.

```
  viz.dirs.vec <- c("ch14incomplete", "ch14for", "ch14named")
  viz.dirs.text <- paste(viz.dirs.vec, collapse=" ")
  (cmd <- paste("du -ks", viz.dirs.text))
```

```
[1] "du -ks ch14incomplete ch14for ch14named"
```

```
  (kb.dt <- fread(cmd=cmd, col.names=c("kilobytes", "path")))
```

```
   kilobytes           path
1:       980 ch14incomplete
2:      3116        ch14for
3:      1400      ch14named
```

The table above shows that the data viz defined using for loops takes about twice as much disk space as the data viz that used named `clickSelects` and `showSelected`.

14.7 Chapter summary and exercises

The table below summarizes the disk usage and timings presented in this chapter.

```
  data.table(
    kb.dt,
    construct.seconds=c(
      timing.incomplete.construct[["elapsed"]],
      timing.for.construct[["elapsed"]],
      timing.named.construct[["elapsed"]]),
    compile.seconds=as.numeric(c(
      elapsed.incomplete,
      elapsed.for,
      elapsed.named)))
```

```
   kilobytes           path construct.seconds compile.seconds
1:       980 ch14incomplete             0.001        7.577811
2:      3116        ch14for             0.993       72.272287
3:      1400      ch14named             0.003        7.302212
```

It is clear from the table above that named `clickSelects` and `showSelected` are more efficient in both respects, and should be used instead of for loops.

Exercises:

- Use named `clickSelects` and `showSelected` to create a visualization which demonstrates over- and under-fitting, as in this visualization of linear model and nearest neighbors.
- Use named `clickSelects` and `showSelected` to create a visualization of some data from your domain of expertise.

Next, Chapter 15 explains how to visualize root-finding algorithms.

15

Newton's root-finding method

Roots of a function `f(x)` are values `x` such that `f(x)=0`. Some functions `f`have an explicit expression for their roots. For example:

- the linear function `f(x)=b*x+c=0` has a single root `x=-c/b`, if b is not zero.
- the quadratic function `f(x)=a*x^2+b*x+c=0` has two roots `x=(-b±sqrt(b^2-4*a*c))/(2*a)`, if the discriminant is positive `b^2-4*a*c>0`.
- the sin function `f(x)=sin(a*x)=0` has an infinite number of roots: `pi*z/a` for all integers `z`.

However, there are some functions which have no explicit expression for their roots. For example, the roots of the Poisson loss `f(x)=a*x+b*log(x)+c` have no explicit expression in terms of common mathematical functions. (actually it has a solution in terms of the Lambert W function but that function is not commonly available) This goal of this chapter is to create an interactive data visualization that explains Newton's method for finding the roots of such functions.

Chapter outline:

- We begin by implementing the Newton method for the Poisson loss, to find the root which is larger than the minimum. We create several static and one interactive data visualization.
- We then suggest an exercise for finding the root which is smaller than the minimum.

15.1 Larger root in mean space

We begin by defining coefficients of a Poisson Loss function with two roots.

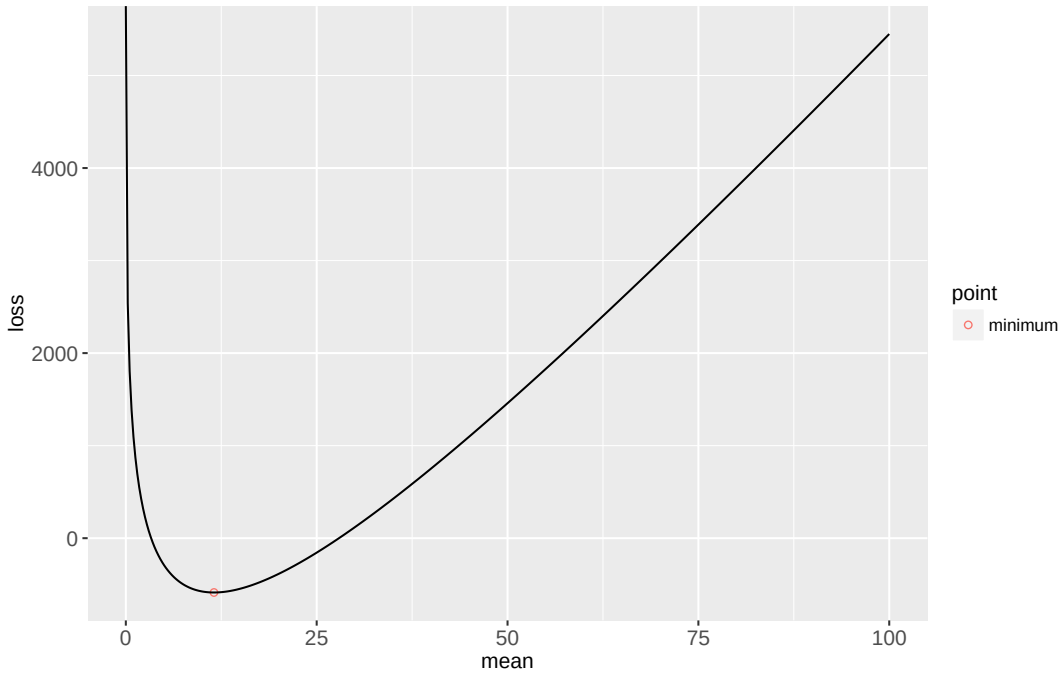
```
Linear <- 95
Log <- -1097
Constant <- 1000
loss.fun <- function(Mean){
  Linear*Mean + Log*log(Mean) + Constant
}
(mean.at.optimum <- -Log/Linear)
```

```
[1] 11.54737
```

```
(loss.at.optimum <- loss.fun(mean.at.optimum))
```

```
[1] -586.764
```

```
library(data.table)
loss.dt <- data.table(mean=seq(0, 100, l=400))
loss.dt[, loss := loss.fun(mean)]
opt.dt <- data.table(
  mean=mean.at.optimum,
  loss=loss.at.optimum,
  point="minimum")
library(animint2)
gg.loss <- ggplot()+
  geom_point(aes(mean, loss, color=point), data=opt.dt)+
  geom_line(aes(mean, loss), data=loss.dt)
print(gg.loss)
```



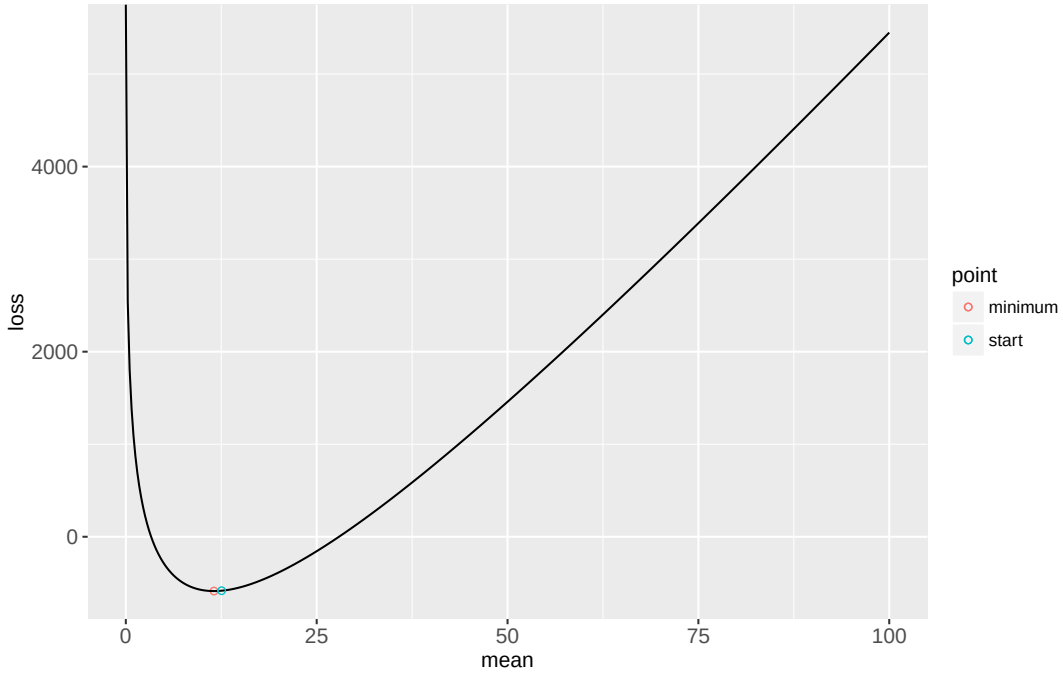
Our goal is to find the two roots of this function. Newton's root finding method starts from an arbitrary candidate root, and then repeatedly uses linear approximations to find more accurate candidate roots. To compute the linear approximation, we need the derivative:

```
loss.deriv <- function(Mean){
  Linear + Log/Mean
}
```

We begin the root finding at a point larger than the minimum,

```
possible.root <- mean.at.optimum+1
gg.loss+
  geom_point(aes(
```

```
      mean, loss, color=point),
      data=data.table(
        point="start",
        mean=possible.root,
        loss=loss.fun(possible.root)))
```



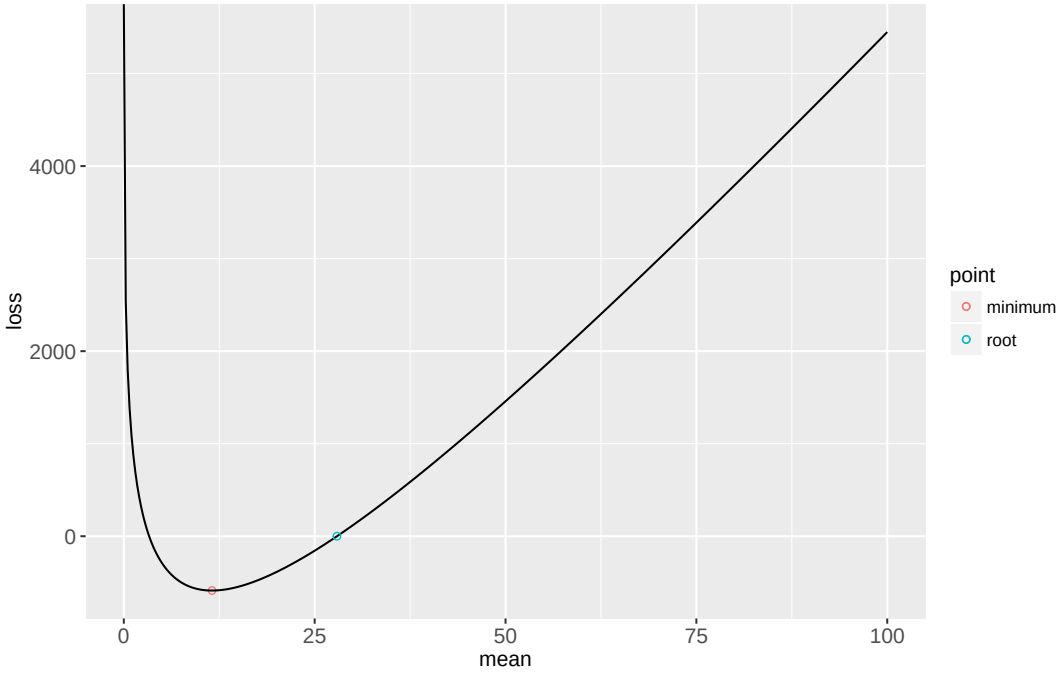
We then use the following implementation of Newton's method to find a root,

```
iteration <- 1
solution.list <- list()
thresh.dt <- data.table(thresh=1e-6)
while(thresh.dt$thresh < abs({
  fun.value <- loss.fun(possible.root)
})){
  cat(sprintf("mean=%e loss=%e\n", possible.root, fun.value))
  deriv.value <- loss.deriv(possible.root)
  new.root <- possible.root - fun.value/deriv.value
  solution.list[[iteration]] <- data.table(
    iteration, possible.root, fun.value, deriv.value, new.root)
  iteration <- iteration+1
  possible.root <- new.root
}
```

```
mean=1.254737e+01 loss=-5.828735e+02
mean=8.953188e+01 loss=4.574958e+03
mean=3.424363e+01 loss=3.768946e+02
mean=2.825783e+01 loss=1.901051e+01
mean=2.791944e+01 loss=7.929111e-02
```

```
mean=2.791802e+01 loss=1.425562e-06
```

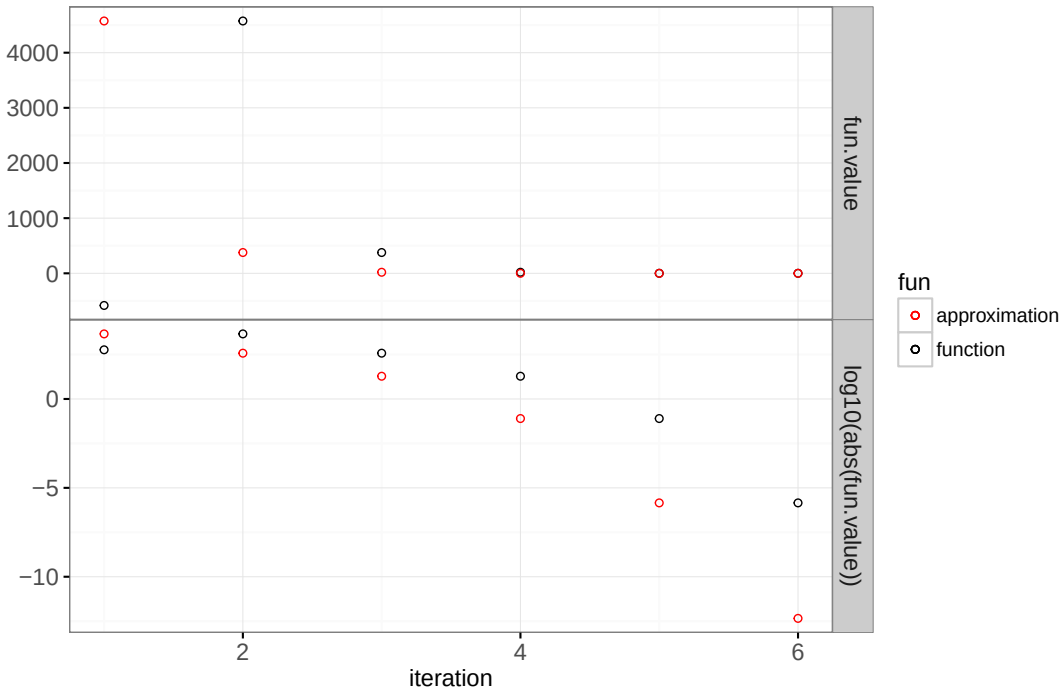
```
root.dt <- data.table(point="root", possible.root, fun.value)
gg.loss+
  geom_point(aes(possible.root, fun.value, color=point),
             data=root.dt)
```



The plot above shows the root that was found. The stopping criterion was an absolute cost value less than `1e-6` so we know that this root is at least that accurate. The following plot shows the accuracy of the root as a function of the number of iterations.

```
solution <- do.call(rbind, solution.list)
solution$new.value <- c(solution$fun.value[-1], fun.value)
gg.it <- ggplot()+
  geom_point(aes(
    iteration, fun.value, color=fun),
    data=data.table(solution, y="fun.value", fun="function"))+
  geom_point(aes(
    iteration, log10(abs(fun.value)), color=fun),
    data=data.table(
      solution, y="log10(abs(fun.value))", fun="function"))+
  scale_color_manual(values=c("function"="black", approximation="red"))+
  geom_point(aes(
    iteration, new.value, color=fun),
    data=data.table(solution, y="fun.value", fun="approximation"))+
  geom_point(aes(
    iteration, log10(abs(new.value)), color=fun),
    data=data.table(
```

```
      solution, y="log10(abs(fun.value))", fun="approximation"))+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(y ~ ., scales="free")+
  ylab("")
print(gg.it)
```

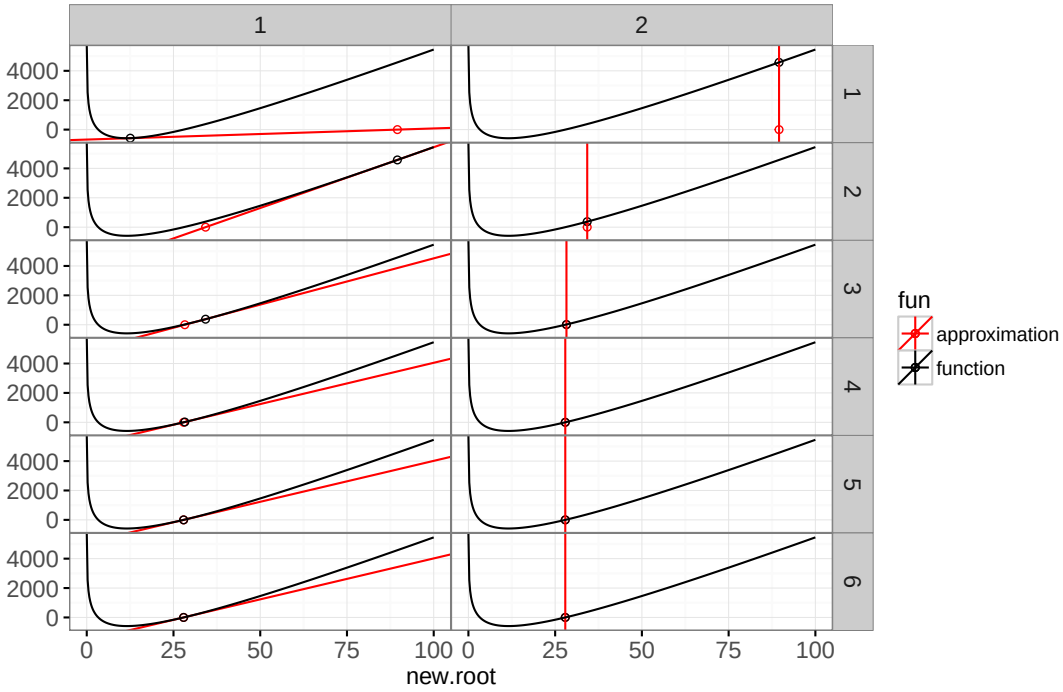


The plot above shows a horizontal line for the stopping criterion threshold, on the log scale. It is clear that the red dot in the last iteration is much below that threshold.

The plot below shows each step of the algorithm. The left panels show the linear approximation at the candidate root, along with the root of the linear approximation. The right panels show the root of the linear approximation, along with the corresponding function value (the new candidate root).

```
## y - fun.value = deriv.value * (x - possible.root)
## y = deriv.value*x + fun.value-possible.root*deriv.value
ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(iteration ~ step)+
  scale_color_manual(values=c("function"="black", approximation="red"))+
  geom_abline(aes(
    slope=deriv.value, intercept=fun.value-possible.root*deriv.value,
    color=fun),
    data=data.table(solution, fun="approximation", step=1))+
  geom_point(aes(
```

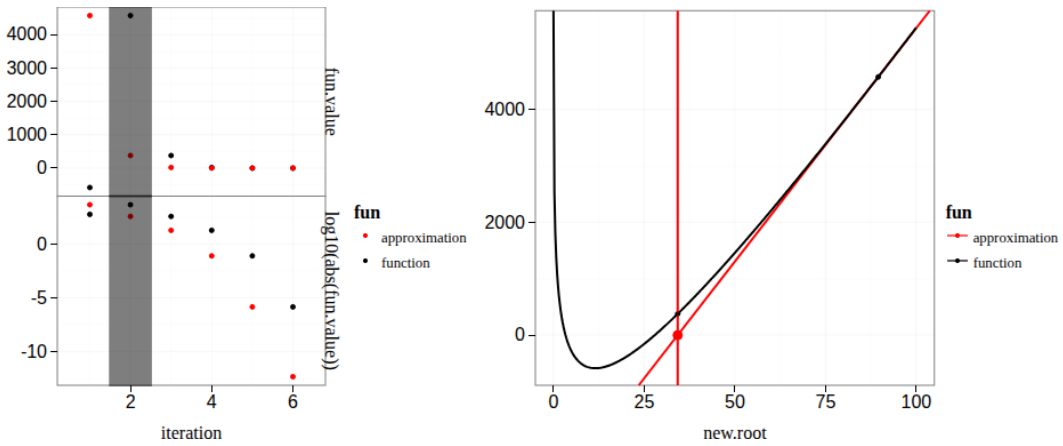
```
    new.root, 0, color=fun),
    data=data.table(solution, fun="approximation"))+
  geom_point(aes(
    new.root, new.value, color=fun),
    data=data.table(solution, fun="function", step=2))+
  geom_vline(aes(
    xintercept=new.root, color=fun),
    data=data.table(solution, fun="approximation", step=2))+
  geom_point(aes(
    possible.root, fun.value, color=fun),
    data=data.table(solution, fun="function", step=1))+
  geom_line(aes(
    mean, loss, color=fun),
    data=data.table(loss.dt, fun="function"))+
  ylab("")
```



It is clear that the algorithm quickly converges to the root. The following is an animated interactive version of the same data viz.

```
animint(
  time=list(variable="iteration", ms=2000),
  iterations=gg.it+
    theme_animint(width=300, colspan=1)+
    geom_tallrect(aes(
      xmin=iteration-0.5,
      xmax=iteration+0.5),
      clickSelects="iteration",
```

```
      alpha=0.5,
      data=solution),
  loss=ggplot()+
    theme_bw()+
    scale_color_manual(values=c(
      "function"="black",
      approximation="red"))+
    geom_abline(aes(
      slope=deriv.value, intercept=fun.value-possible.root*deriv.value,
      color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="approximation"))+
    geom_point(aes(
      new.root, 0, color=fun),
      showSelected="iteration",
      size=4,
      data=data.table(solution, fun="approximation"))+
    geom_point(aes(
      new.root, new.value, color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="function"))+
    geom_vline(aes(
      xintercept=new.root, color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="approximation"))+
    geom_point(aes(
      possible.root, fun.value, color=fun),
      showSelected="iteration",
      data=data.table(solution, fun="function"))+
    geom_line(aes(
      mean, loss, color=fun),
      data=data.table(loss.dt, fun="function"))+
    ylab(""))
```



15.2 Comparison with Lambert W solution

The code below uses the Lambert W function to compute a root, and compares its solution to the one we computed using Newton's method.

```
  inside <- Linear*exp(-Constant/Log)/Log
  root.vec <- Log/Linear*c(
    LambertW::W(inside, 0),
    LambertW::W(inside, -1))
```

```
Registered S3 methods overwritten by 'ggplot2':
  method                    from
  drawDetails.zeroGrob      animint2
  grobHeight.absoluteGrob   animint2
  grobHeight.zeroGrob       animint2
  grobWidth.absoluteGrob    animint2
  grobWidth.zeroGrob        animint2
  grobX.absoluteGrob        animint2
  grobY.absoluteGrob        animint2
  heightDetails.titleGrob   animint2
  heightDetails.zeroGrob    animint2
  makeContext.dotstackGrob  animint2
  print.ggplot2_bins        animint2
  print.rel                 animint2
  widthDetails.titleGrob    animint2
  widthDetails.zeroGrob     animint2
```

```
  loss.fun(c(
    Newton=root.dt$possible.root,
    Lambert=root.vec[2]))
```

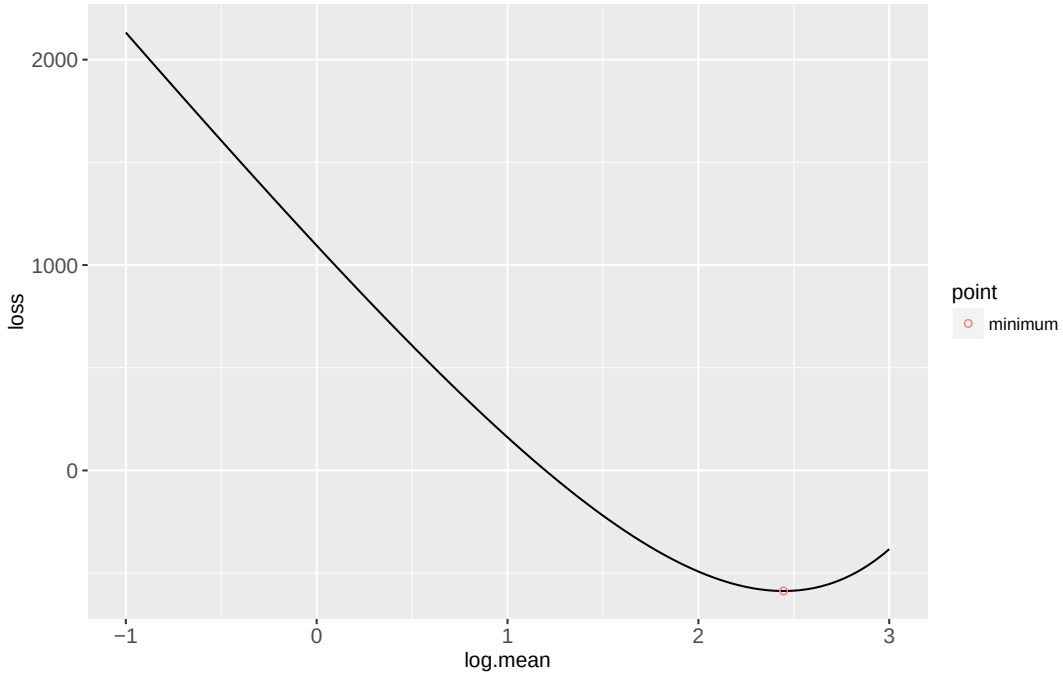
```
       Newton       Lambert
-4.547474e-13  0.000000e+00
```

For these data, the Lambert W function yields a root which is slightly more accurate than our implementation of Newton's method.

15.3 Root-finding in the log space

The previous section showed an algorithm for finding the root which is larger than the minimum. In this section we explore an algorithm for finding the other root (smaller than the minimum). Note that the Poisson loss is highly non-linear as mean goes to zero, so the linear approximation in the Newton root finding will not work very well. Instead, we equivalently perform root finding in the log space:

```
log.loss.fun <- function(log.mean){
  Linear*exp(log.mean) + Log*log.mean + Constant
}
log.loss.dt <- data.table(
  log.mean=seq(-1, 3, l=400)
)[
, loss := log.loss.fun(log.mean)]
ggplot()+
  geom_line(aes(
    log.mean, loss),
    data=log.loss.dt)+
  geom_point(aes(
    log(mean), loss, color=point),
    data=opt.dt)
```



Exercise: derive and implement the Newton method for this function, in order to find the root that is smaller than the minimum. Create an animint similar to the previous section.

15.4 Chapter summary and exercises

In this chapter we explored several visualizations of the Newton method for finding roots of smooth functions.

Exercises:

- Add a title to each plot.

- Add a size legend to the first plot (black points larger than red), so that we can see when red and black have the same value.
- Add a `geom_hline` to emphasize the loss=0 value in the second plot.
- Add a `geom_hline` to emphasize the stopping threshold in the first plot.
- Turn off one of the two legends, to save space.
- How to specify smooth transitions between iterations?
- Instead of using iteration as the animation/time variable, create a new one in order to show two distinct states/steps for each iteration, i.e. the `step` variable in the facetted plot above.
- What happens to the rate of convergence when you try to find the larger root in the log space, or the smaller root in the original space? Theoretically it should not converge as fast, since the functions are more nonlinear for those roots. Make a data visualization that allows you to select the starting value, and shows how many iterations it takes to converge to within the threshold.
- Create another plot that allows you to select the threshold. Plot the number of iterations as a function of threshold.
- Derive the loss function for Binomial regression, and visualize the corresponding Newton root finding method.
- Refactor `gg.it` code to use only one `geom_point`, instead of the four geoms in the current code. Hint: use `rbind()` to create a single table with all of the data.

Next, Chapter 16 explains how to visualize change-point detection models.

16

Supervised change-point detection

In this chapter we will explore several data visualizations of supervised changepoint detection models.

Chapter outline:

- We begin by making several static visualizations of the `intreg` data set.
- We then create an interactive visualization in which one plot can be click to select the number of changepoints/segments, and the other plot shows the corresponding model.
- We end by showing a static visualization of the max margin linear regression model, and suggesting exercises about creating an interactive version.

16.1 Static figures

We begin by loading the `intreg` data set.

```
library(animint2)
data(intreg)
library(data.table)
lapply(intreg, function(df)data.table(df)[1:2])
```

```
$model
         line      min.L    max.L min.feature max.feature
1: regression -1.4001088 2.139100   -2.476391   -1.584656
2:      limit -0.1493744 3.389835   -2.476391   -1.584656

$annotations
   signal first.base last.base   annotation  logratio
1:   11.2   55103411 161558770 0breakpoints 0.9847716
2:    4.2  140080934 201712984  1breakpoint 0.9847716

$intervals
   signal   feature       min.L    max.L
1:    4.2 -2.152421 -1.36503599 1.136433
2:   11.2 -1.797948  0.04183316      Inf

$selection
   signal     min.L     max.L segments cost
1:    4.2      -Inf -3.566634       20    2
2:    4.2 -3.566634 -3.301154       19    2
```

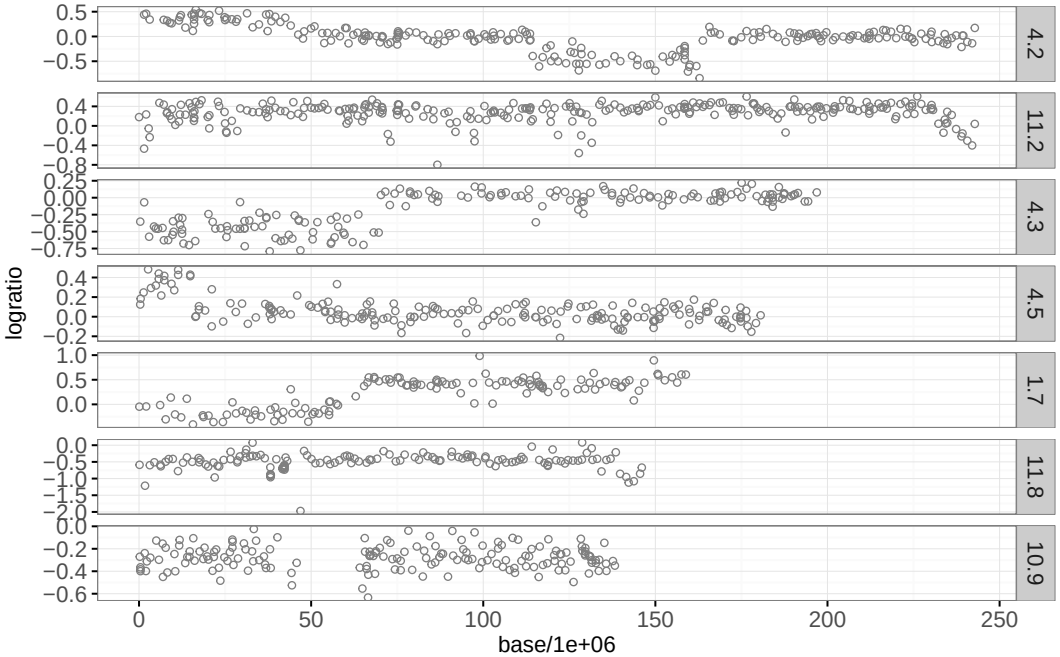
```
$segments
   signal segments first.base last.base        mean
1:    4.2        1    1472476 242801018 -0.02092153
2:    4.2        2    1472476  45164626  0.35123108

$breaks
   signal      base segments
1:    4.2  45164626        2
2:    4.2 114042112        3

$signals
   signal    base  logratio
1:    4.2 1472476 0.4404207
2:    4.2 2063049 0.4594316
```

As shown above, it is a named list of 7 related data.frames. We begin our exploration of these data by plotting the signals in separate facets.

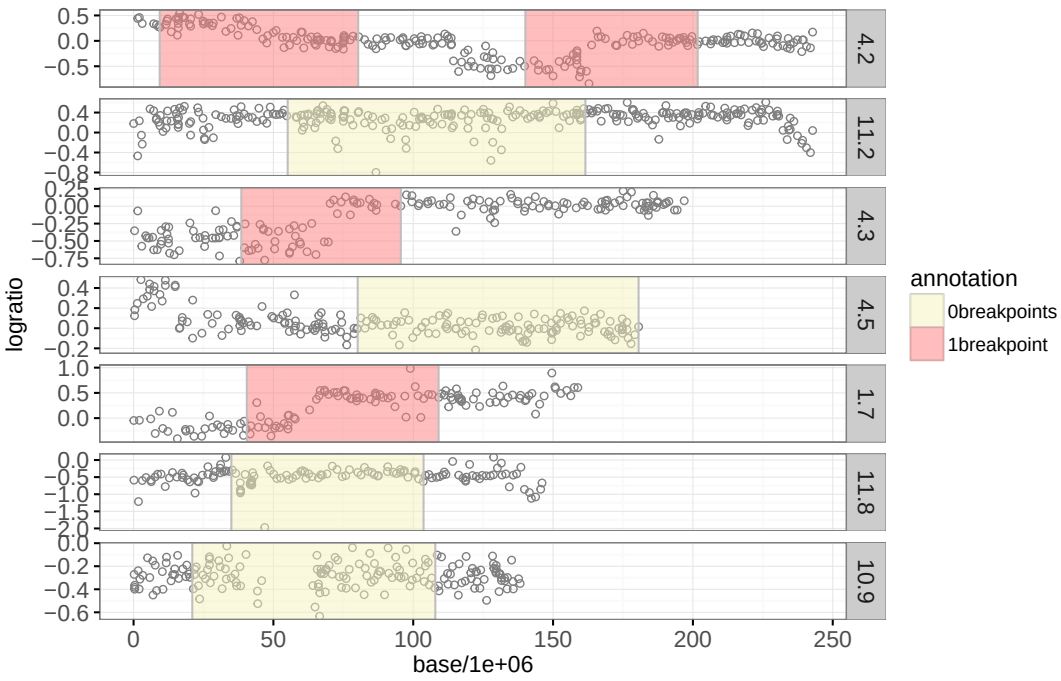
```
data.color <- "grey50"
gg.signals <- ggplot()+
  theme_bw()+
  facet_grid(signal ~ ., scales="free")+
  geom_point(aes(
    base/1e6, logratio,
    showSelected="signal"),
    color=data.color,
    data=intreg$signals)
gg.signals
```



Each data point plotted above shows an approximate measurement of DNA copy number (logratio), as a function of base position on a chromosome. Such data come from high-throughput assays which are important for diagnosing certain types of cancer such as neuroblastoma.

An important part of the diagnosis is detecting "breakpoints" or abrupt changes, within a given chromosome (panel). It is clear from the plot above that there are several breakpoint in these data. In particular signal 4.2 appears to have three breakpoints, signal 4.3 appears to have one, etc. In fact these data come from medical doctors at the Institute Curie (Paris, France) who have visually annotated regions with and without breakpoints. These data are available as `intreg$annotations` and are plotted below.

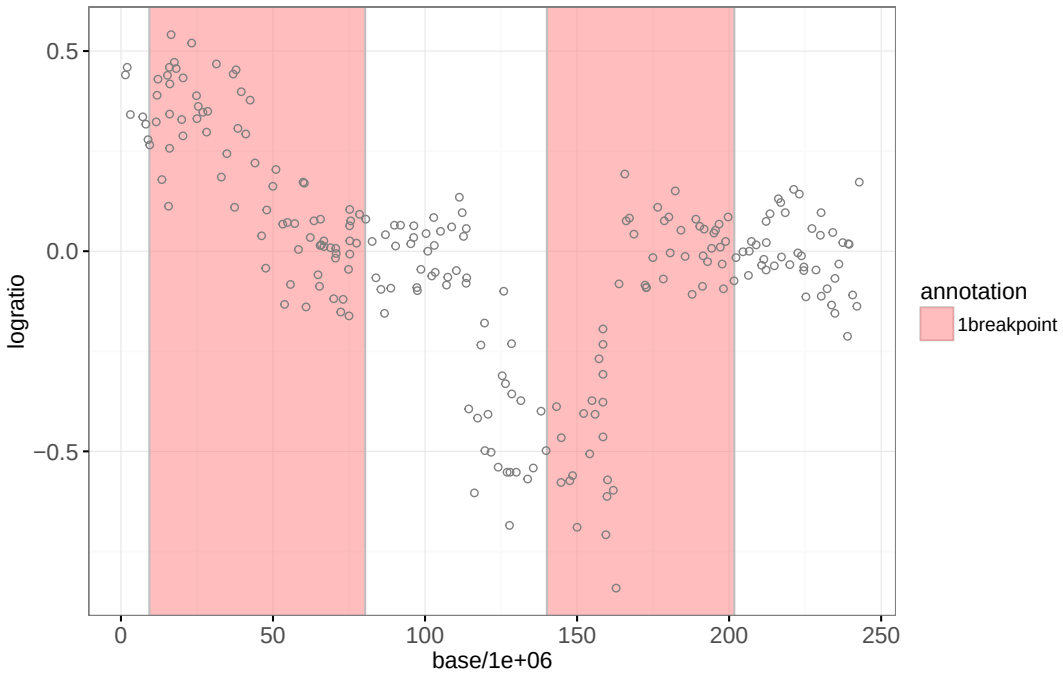
```
breakpoint.colors <- c(
  "1breakpoint"="#ff7d7d",
  "0breakpoints"='#f6f4bf')
gg.ann <- gg.signals+
  scale_fill_manual(values=breakpoint.colors)+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    fill=annotation),
    color="grey",
    alpha=0.5,
    data=intreg$annotations)
gg.ann
```



The plot above shows yellow regions where the doctors have determined that there are no significant breakpoints, and red regions where there is one breakpoint. The goal in analyzing these data is to learn from the limited labeled data (colored regions) and provide consistent breakpoint predictions throughout (even in un-labeled regions).

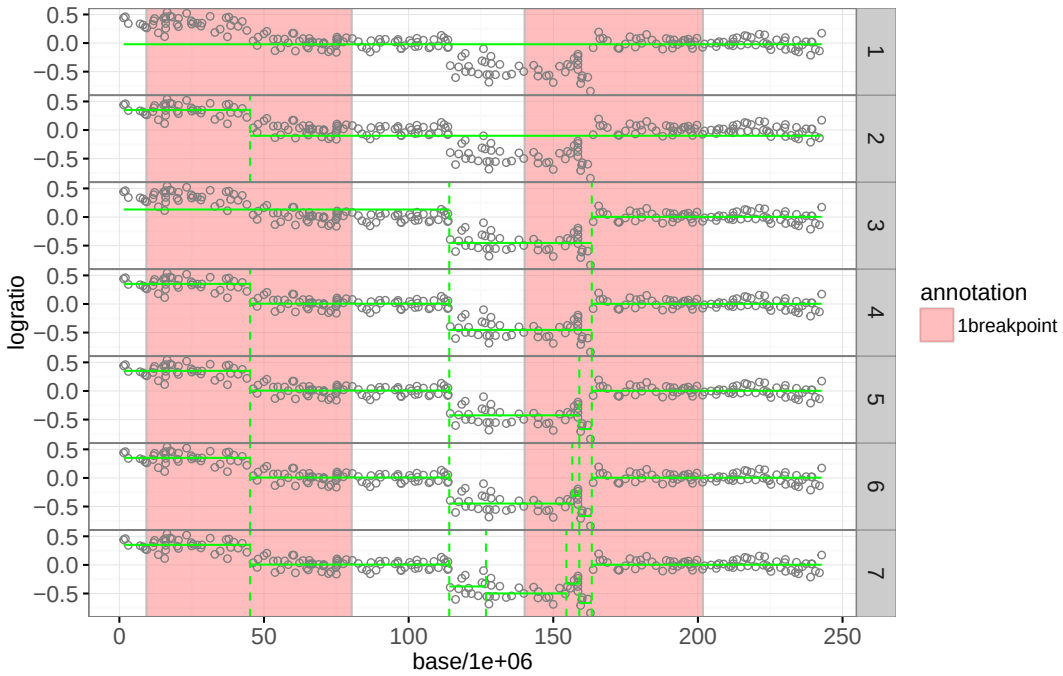
In order to detect these breakpoints we have fit some maximum likelihood segmentation models, using the efficient algorithm implemented in `jointseg::Fpsn`. The segment means are available in `intreg$segments` and the predicted breakpoints are available in `intreg$breaks`. For each signal there is a sequence of models from 1 to 20 segments. First let's zoom in on one signal:

```
sig.name <- "4.2"
show.segs <- 7
sig.labels <- subset(intreg$annotations, signal==sig.name)
gg.one <- ggplot()+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    fill=annotation),
    color="grey",
    alpha=0.5,
    data=sig.labels)+
  geom_point(aes(
    base/1e6, logratio),
    color=data.color,
    data=subset(intreg$signals, signal==sig.name))+
  scale_fill_manual(values=breakpoint.colors)
gg.one
```



We plot some of these models for one of the signals below:

```
sig.segs <- data.table(
  intreg$segments)[signal == sig.name & segments <= show.segs]
sig.breaks <- data.table(
  intreg$breaks)[signal == sig.name & segments <= show.segs]
model.color <- "green"
gg.models <- gg.one+
  facet_grid(segments ~ .)+
  geom_segment(aes(
    first.base/1e6, mean,
    xend=last.base/1e6, yend=mean),
    color=model.color,
    data=sig.segs)+
  geom_vline(aes(
    xintercept=base/1e6),
    color=model.color,
    linetype="dashed",
    data=sig.breaks)
gg.models
```



The plot above shows the maximum likelihood segmentation models in green (from one to six segments). Below we use the `penaltyLearning::labelError` function to compute the label error, which quantifies which models agree with which labels.

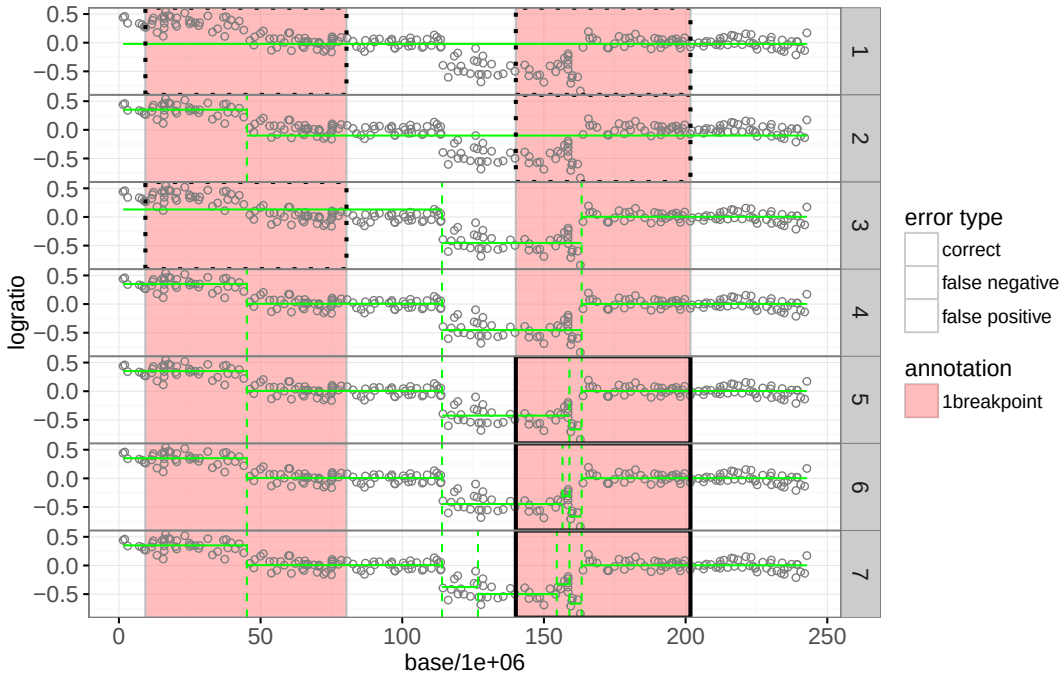
```
sig.models <- data.table(segments=1:show.segs, signal=sig.name)
sig.errors <- penaltyLearning::labelError(
  sig.models, sig.labels, sig.breaks,
  change.var="base",
```

```
    label.vars=c("first.base", "last.base"),
    model.vars="segments",
    problem.vars="signal")
```

```
Registered S3 methods overwritten by 'ggplot2':
  method                   from
  drawDetails.zeroGrob     animint2
  grobHeight.absoluteGrob  animint2
  grobHeight.zeroGrob      animint2
  grobWidth.absoluteGrob   animint2
  grobWidth.zeroGrob       animint2
  grobX.absoluteGrob       animint2
  grobY.absoluteGrob       animint2
  heightDetails.titleGrob  animint2
  heightDetails.zeroGrob   animint2
  makeContext.dotstackGrob animint2
  print.ggplot2_bins       animint2
  print.rel                animint2
  widthDetails.titleGrob   animint2
  widthDetails.zeroGrob    animint2
```

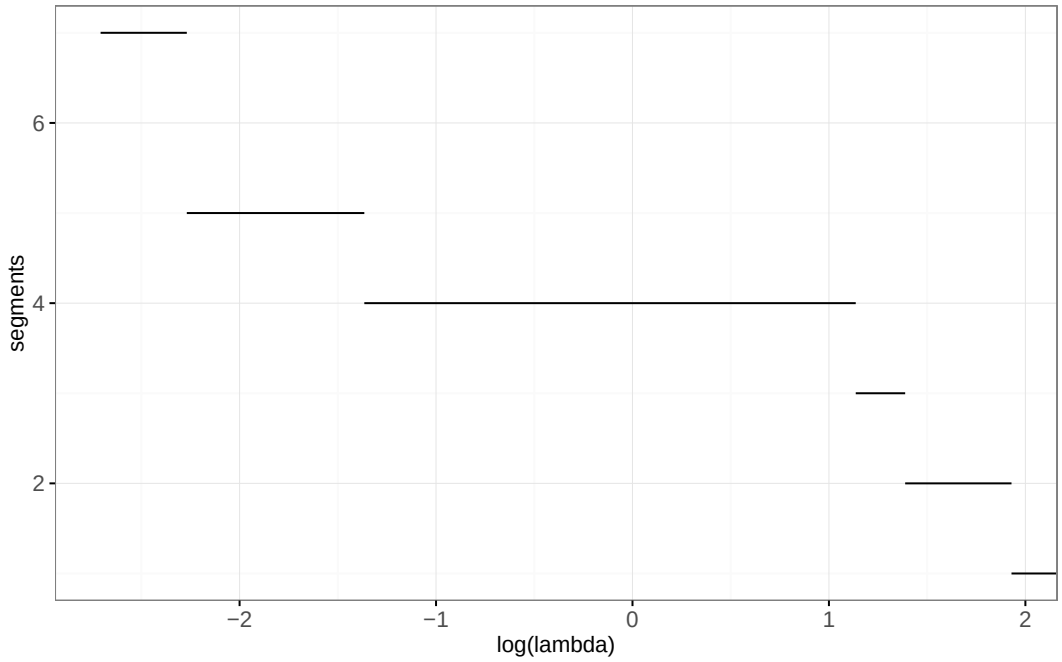
The `sig.errors$label.errors` data.table contains one row for every (model,label) combination. The`status` column can be used to show the label error: `false negative` for too few changes, `false positive` for too many changes, or `correct` for the right number of changes.

```
gg.models+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    linetype=status),
    data=sig.errors$label.errors,
    color="black",
    size=1,
    fill=NA)+
  scale_linetype_manual(
    "error type",
    values=c(
      correct=0,
      "false negative"=3,
      "false positive"=1))
```



Looking at the label error plot above, it is clear that the model with four segments should be selected, because it achieves zero label errors. There are a number of criteria that can be used to select which one of these models is best. One way to do that is by selecting the model with s segments is $S^*(\lambda) = L_s + \lambda s$, where L_s is the total loss of the model with s segments, and λ is a non-negative penalty. In the plot below we show the model selection function $S^*(\lambda)$ for this data set:

```
sig.selection <- data.table(
  intreg$selection)[signal == sig.name & segments <= show.segs]
gg.selection <- ggplot()+
  theme_bw()+
  geom_segment(aes(
    min.L, segments,
    xend=max.L, yend=segments),
    data=sig.selection)+
  xlab("log(lambda)")
gg.selection
```

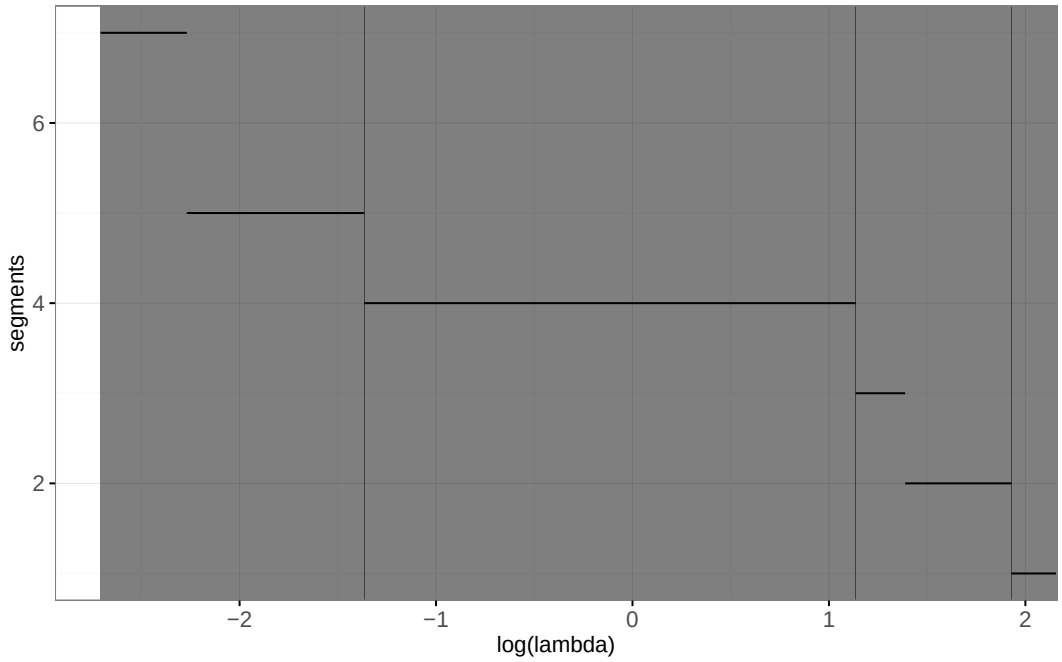


It is clear from the plot above that the model selection function is decreasing. In the next section we make an interactive version of these two plots where we can actually click on the model selection plot in order to select the model.

16.2 Interactive figures for one signal

We will create an interactive figure for one signal by adding a `geom_tallrect()` with `clickSelects=segments` to the plot above:

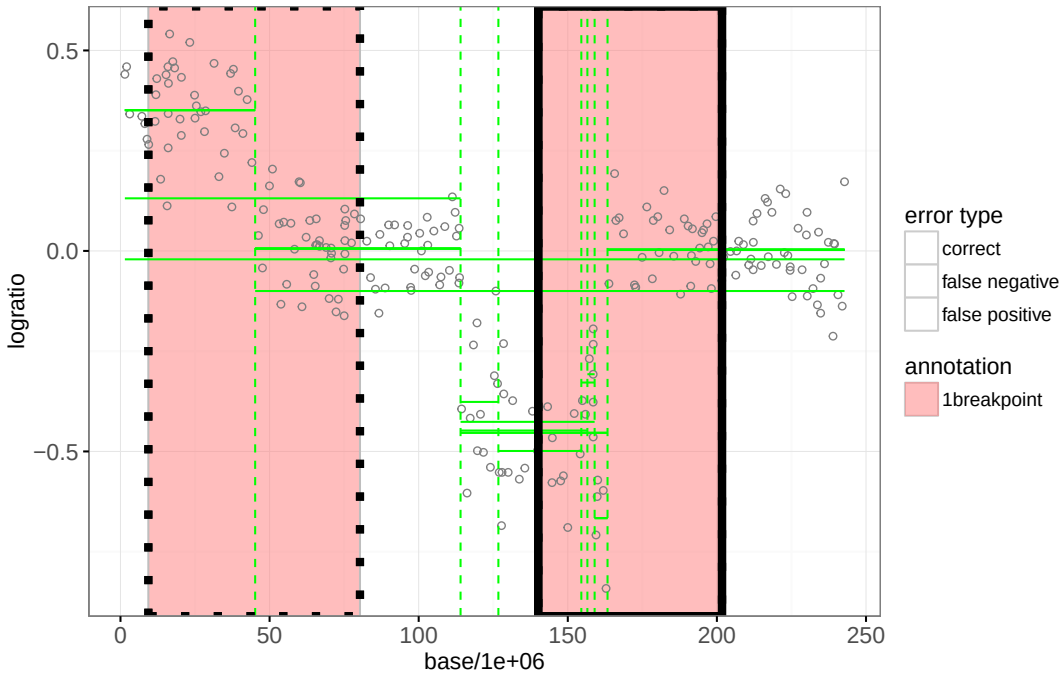
```
interactive.selection <- gg.selection+
  geom_tallrect(aes(
    xmin=min.L, xmax=max.L),
    clickSelects="segments",
    data=sig.selection,
    color=NA,
    fill="black",
    alpha=0.5)
interactive.selection
```



We will combine that with the non-facetted version of the data/models plot below, in which we have added `showSelected=segments` to the model geoms:

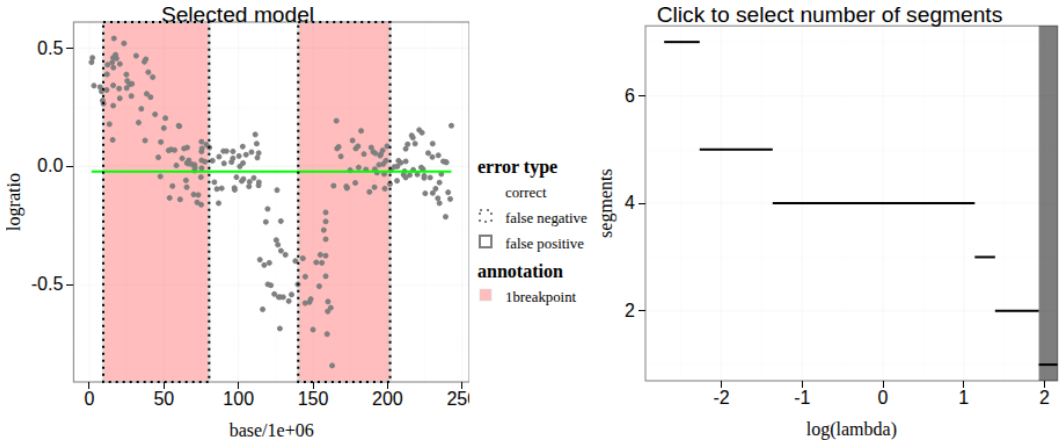
```
interactive.models <- gg.one+
  geom_segment(aes(
    first.base/1e6, mean,
    xend=last.base/1e6, yend=mean),
    showSelected="segments",
    color=model.color,
    data=sig.segs)+
  geom_vline(aes(
    xintercept=base/1e6),
    showSelected="segments",
    color=model.color,
    linetype="dashed",
    data=sig.breaks)+
  geom_tallrect(aes(
    xmin=first.base/1e6, xmax=last.base/1e6,
    linetype=status),
    showSelected="segments",
    data=sig.errors$label.errors,
    size=2,
    color="black",
    fill=NA)+
  scale_linetype_manual(
    "error type",
    values=c(
      correct=0,
      "false negative"=3,
```

```
      "false positive"=1))
interactive.models
```



Of course the plot above is not very informative because it is not interactive. Below we combine the two interactive ggplots in a single linked animint:

```
animint(
  models=interactive.models+
    ggtitle("Selected model"),
  selection=interactive.selection+
    ggtitle("Click to select number of segments"))
```



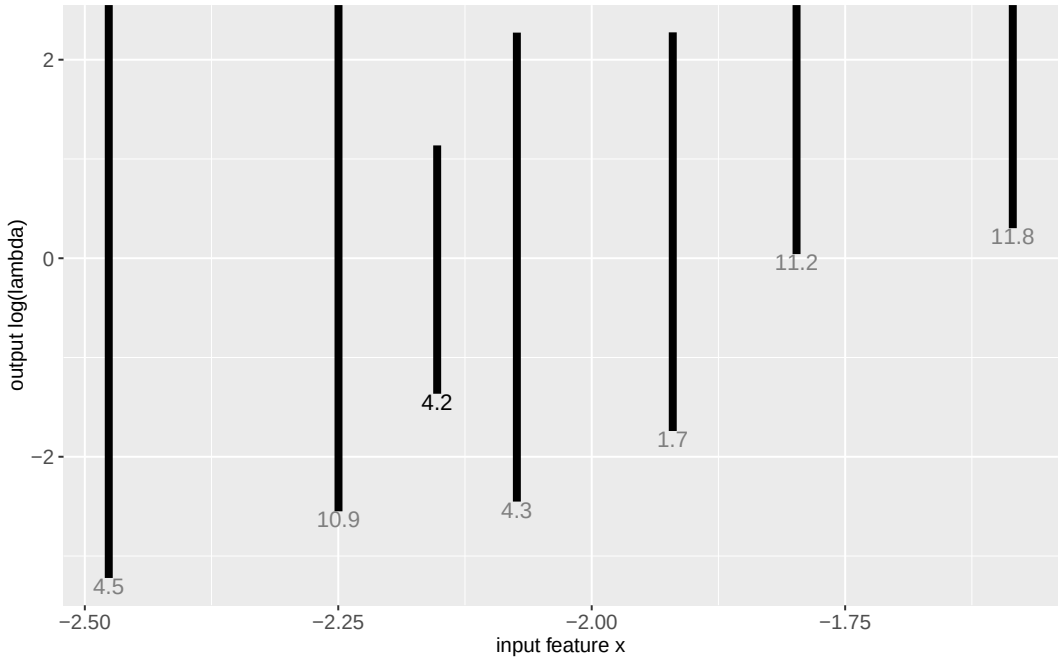
Note that in the data viz above the model with 6 segments is not selectable for any value of lambda, so there is no way to click on the plot to select that model. However it is possible

to select the model using the segments selection menu (click "Show selection menus" at the bottom of the data viz).

16.3 Static max margin regression plot

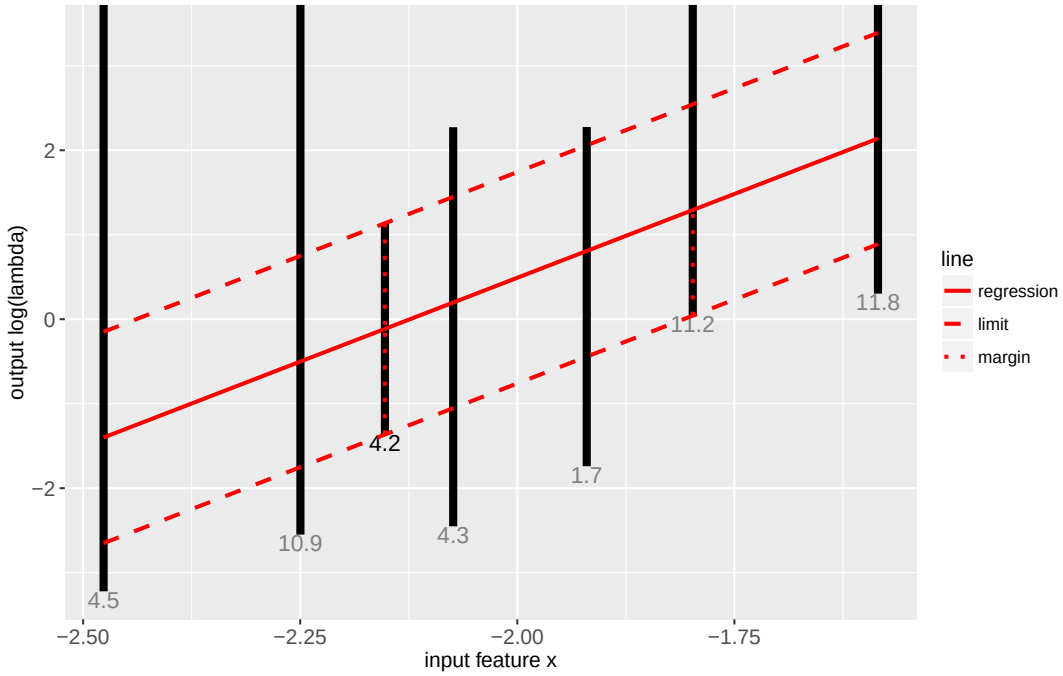
Another part of this data set is `intreg$intervals` which has one row for every signal. The columns`min.L` and `max.L` indicate the min/max values of the target interval, which is the largest range of log(penalty) values with minimum label errors. Below we plot this interval as a function of a feature of the data (log number of data points):

```
gg.intervals <- ggplot()+
  geom_segment(aes(
    feature, min.L,
    xend=feature, yend=max.L),
    size=2,
    data=intreg$intervals)+
  geom_text(aes(
    feature, min.L, label=signal,
    color=ifelse(signal==sig.name, "black", "grey50")),
    vjust=1,
    data=intreg$intervals)+
  scale_color_identity()+
  ylab("output log(lambda)")+
  xlab("input feature x")
gg.intervals
```



The target intervals in the plot above denote the region of log(lambda) space that will select a model with minimum label errors. There is one interval for each signal; we made an animint in the previous section for the signal indicated in black text. Machine learning algorithms can be used to find a penalty function that intersects each of the intervals, and maximizes the margin (the distance between the regression function and the nearest interval limit). Data for the linear max margin regression function are in `intreg$model` which is shown in the plot below:

```
gg.mm <- gg.intervals+
  geom_segment(aes(
    min.feature, min.L,
    xend=max.feature, yend=max.L,
    linetype=line),
    color="red",
    size=1,
    data=intreg$model)+
  scale_linetype_manual(
    values=c(
      regression="solid",
      margin="dotted",
      limit="dashed"))
gg.mm
```



The plot above shows the linear max margin regression function f(x) as the solid red line. It is clear that it intersects each of the black target intervals, and maximizes the margin (red vertical dotted lines). For more information on the subject of supervised changepoint detection, please see my useR 2017 tutorial.

Now that you know how to visualize each of the seven parts of the `intreg` data set, the rest

of the chapter is devoted to exercises.

16.4 Chapter summary and exercises

Exercises:

- Add a `geom_text()` which shows the currently selected signal name at the top of the plot, in `interactive.models` in the first animint above.
- Make an animint with two plots that shows the data set that corresponds to each interval on the max margin regression plot. One plot should show an interactive version of the max margin regression plot where you can click on an interval to select a signal. The other plot should show the data set for the currently selected signal.
- In the animint you created in the previous exercise, add a third plot with the model selection function for the currently selected signal.
- Re-design the previous animint so that instead of using a third plot, add a facet to the max margin regression regression plot such that the log(lambda) axes are aligned. Add another facet that shows the number of incorrect labels (`intreg$selection$cost`) for each log(lambda) value.
- Add geoms for selecting the number of segments. Clicking the model selection plot should select the number of segments, which should update the displayed model and label errors on the plot of the data for the currently selected signal. Furthermore add a visual indication of the selected model to the max margin regression plot. The result should look something like this.
- Make another data viz by starting with the facetted `gg.signals` plot in the beginning of this chapter. Add a plot that can be used to select the number of segments for each signal. For each signal in the facetted plot of the data, show the currently selected model for that signal (there should be a separate selection variable for each signal – you can use named `clickSelects` and `showSelected` as explained in Chapter 14). The result should look something like this.

Next, Chapter 17 explains how to visualize the K-means clustering algorithm.

17

K-means clustering

In this chapter we will explore several data visualizations of K-means clustering, which is an unsupervised learning algorithm.

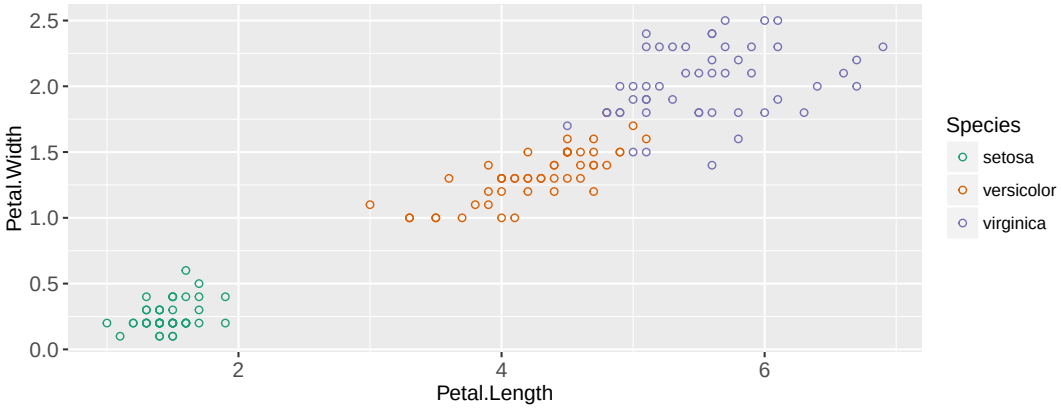
Chapter outline:

- We begin by visualizing two features of the iris data.
- We choose three random data points to use as cluster centers.
- We show how all distances between data points and cluster centers can be computed and visualized.
- We end by showing a visualization of how the k-means model parameters change with each iteration.

17.1 Visualize iris data with labels

We begin with a typical visualization of the iris data, including a color legend to indicate the Species.

```
library(animint2)
color.code <- c(
  setosa="#1B9E77",
  versicolor="#D95F02",
  virginica="#7570B3",
  "1"="#E7298A",
  "2"="#66A61E",
  "3"="#E6AB02",
  "4"="#A6761D")
ggplot()+
  scale_color_manual(values=color.code)+
  geom_point(aes(
    Petal.Length, Petal.Width, color=Species),
    data=iris)+
  coord_equal()
```



We will illustrate the K-means clustering algorithm using these two dimensions,

```
data.mat <- as.matrix(iris[,c("Petal.Width","Petal.Length")])
head(data.mat)
```

```
     Petal.Width Petal.Length
[1,]         0.2          1.4
[2,]         0.2          1.4
[3,]         0.2          1.3
[4,]         0.2          1.5
[5,]         0.2          1.4
[6,]         0.4          1.7
```

```
str(data.mat)
```

```
num [1:150, 1:2] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
- attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:2] "Petal.Width" "Petal.Length"
```

To run K-means, the number of clusters hyper-parameter (K) must be fixed in advance. Then K random data points are selected as the initial cluster centers,

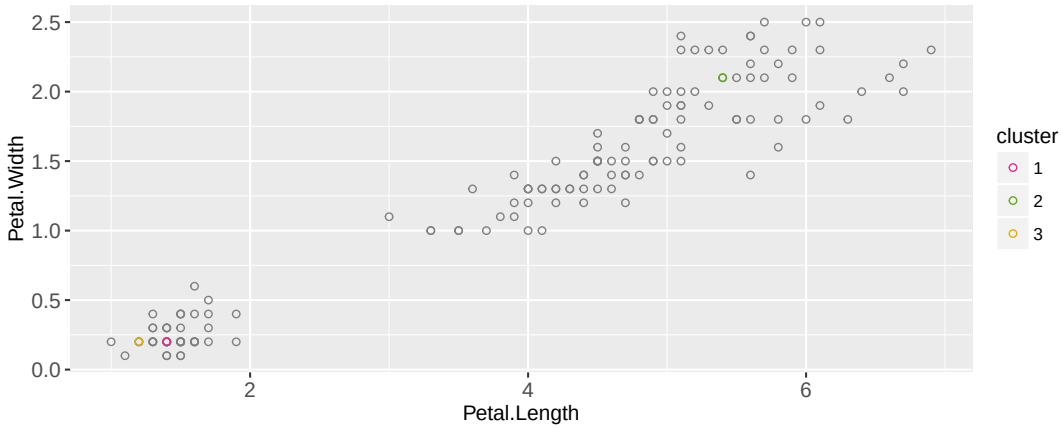
```
K <- 3
library(data.table)
data.dt <- data.table(data.mat)
set.seed(3)
centers.dt <- data.dt[sample(1:.N, K)]
(centers.mat <- as.matrix(centers.dt))
```

```
     Petal.Width Petal.Length
[1,]         0.2          1.4
[2,]         2.1          5.4
[3,]         0.2          1.2
```

```
centers.dt[, cluster := factor(1:K)]
centers.dt
```

```
   Petal.Width Petal.Length cluster
1:         0.2          1.4       1
2:         2.1          5.4       2
3:         0.2          1.2       3
```

```
gg.centers <- ggplot()+
  scale_color_manual(values=color.code)+
  geom_point(aes(
    Petal.Length, Petal.Width),
    color="grey50",
    data=data.dt)+
  geom_point(aes(
    Petal.Length, Petal.Width, color=cluster),
    data=centers.dt)+
  coord_equal()
gg.centers
```



Above we displayed the two data sets (cluster centers and data) using two instances of `geom_point()`. Below we compute the distance between each data point and each cluster center,

```
pairs.dt <- data.table(expand.grid(
  centers.i=1:nrow(centers.mat),
  data.i=1:nrow(data.mat)))
```

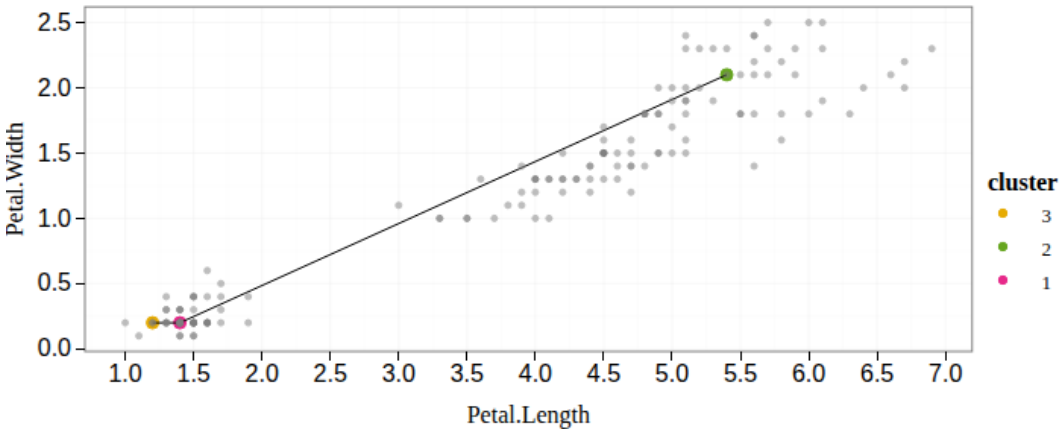
These can be visualized via a `geom_point()`,

```
seg.dt <- pairs.dt[, data.table(
  data.i,
  data=data.mat[data.i,],
  center=centers.mat[centers.i,])]
gg.centers+
  geom_segment(aes(
    data.Petal.Length, data.Petal.Width,
    xend=center.Petal.Length, yend=center.Petal.Width),
```

```
      size=1,
      data=seg.dt)
```

There are 450 segments overplotted above, so interactivity would be useful to emphasize the segments connected to a particular data point. To do that we create a`data.i` selection variable,

```
animint(
  ggplot()+
    theme_bw()+
    theme_animint(height=300, width=640)+
    scale_color_manual(values=color.code)+
    scale_x_continuous(breaks=seq(1,7,by=0.5))+
    scale_y_continuous(breaks=seq(0, 2.5, by=0.5))+
    geom_point(aes(
      Petal.Length, Petal.Width, color=cluster),
      size=4,
      data=centers.dt)+
    geom_segment(aes(
      data.Petal.Length, data.Petal.Width,
      xend=center.Petal.Length, yend=center.Petal.Width),
      size=1,
      showSelected="data.i",
      data=seg.dt)+
    geom_point(aes(
      Petal.Length, Petal.Width),
      clickSelects="data.i",
      size=2,
      color="grey50",
      data=data.table(data.mat, data.i=1:nrow(data.mat))))
```



In the data viz above you can click on a data point to show the distances from that data point to each cluster center.

Exercises for this section:

- edit the x/y scales so that the same ticks are shown.
- change the color of each segment so that it matches the corresponding cluster.
- add a tooltip that shows the distance value.
- make the segment width depend on its optimality (segment connected to closest cluster center should be emphasized with greater width).

17.2 Visualizing iterations of algorithm

Next we compute the closest cluster center for each data point,

```
  pairs.dt[, error := rowSums(
  (data.mat[data.i,]-centers.mat[centers.i,])^2)]
  (closest.dt <- pairs.dt[, .SD[which.min(error)], by=data.i])
```

```
     data.i centers.i error
  1:      1         1  0.00
  2:      2         1  0.00
 ---
149:    149         2  0.04
150:    150         2  0.18
```

```
  (closest.data <- closest.dt[, .(
    data.dt[data.i],
    cluster=factor(centers.i)
  )])
```

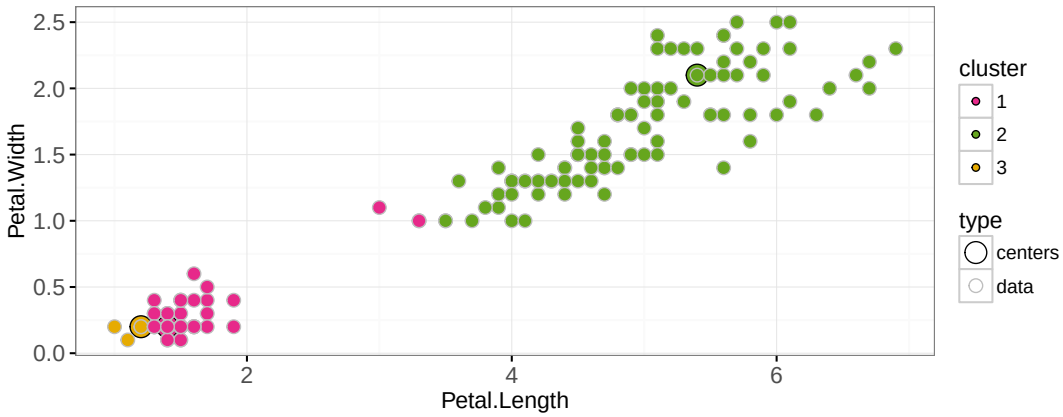
```
     Petal.Width Petal.Length cluster
  1:         0.2          1.4       1
  2:         0.2          1.4       1
 ---
```

```
149:          2.3          5.4       2
150:          1.8          5.1       2
```

```
(both.dt <- rbind(
  data.table(type="centers", centers.dt),
  data.table(type="data", closest.data)))
```

```
       type Petal.Width Petal.Length cluster
  1: centers         0.2          1.4       1
  2: centers         2.1          5.4       2
 ---
152:    data         2.3          5.4       2
153:    data         1.8          5.1       2
```

```
ggplot()+
  scale_fill_manual(values=color.code)+
  scale_color_manual(values=c(centers="black", data="grey"))+
  scale_size_manual(values=c(centers=5, data=3))+
  geom_point(aes(
    Petal.Length, Petal.Width, fill=cluster, size=type, color=type),
    data=both.dt)+
  coord_equal()+
  theme_bw()
```



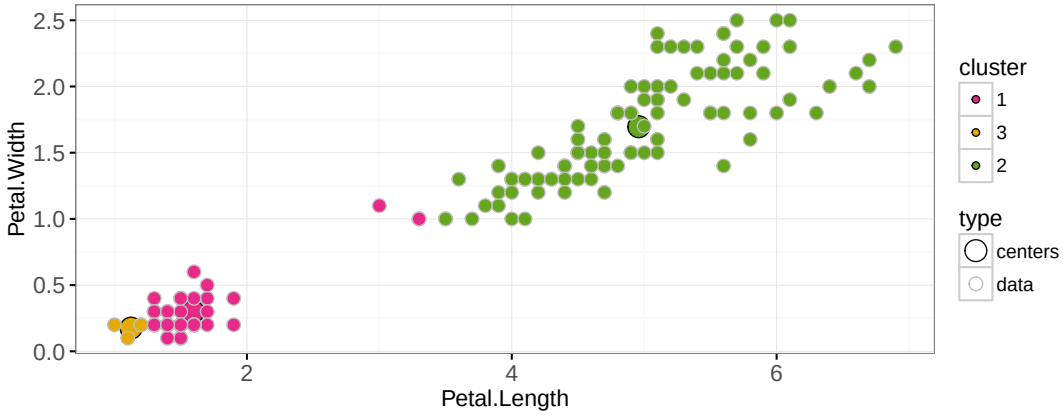
Then we update the cluster centers,

```
new.centers <- closest.dt[, data.table(
  t(colMeans(data.dt[data.i]))
), by=.(cluster=centers.i)]
(new.both <- rbind(
  data.table(type="centers", new.centers),
  data.table(type="data", closest.data)))
```

```
       type cluster Petal.Width Petal.Length
1: centers       1       0.300     1.595918
2: centers       3       0.175     1.125000
```

```
 ---
152:    data       2       2.300     5.400000
153:    data       2       1.800     5.100000
```

```
ggplot()+
  scale_fill_manual(values=color.code)+
  scale_color_manual(values=c(centers="black", data="grey"))+
  scale_size_manual(values=c(centers=5, data=3))+
  geom_point(aes(
    Petal.Length, Petal.Width, fill=cluster, size=type, color=type),
    data=new.both)+
  coord_equal()+
  theme_bw()
```



So the visualizations above show the steps of k-means: (1) updating cluster assignment based on closest center, then (2) updating center based on data assigned to that cluster. To visualize several iterations of the above two steps, we can use a for loop,

```
set.seed(3)
centers.dt <- data.dt[sample(1:.N, K)]
(centers.mat <- as.matrix(centers.dt))
```

```
     Petal.Width Petal.Length
[1,]         0.2          1.4
[2,]         2.1          5.4
[3,]         0.2          1.2
```

```
data.and.centers.list <- list()
iteration.error.list <- list()
for(iteration in 1:20){
  pairs.dt[, error := {
    rowSums((data.mat[data.i,]-centers.mat[centers.i,])^2)
  }]
  closest.dt <- pairs.dt[, .SD[which.min(error)], by=data.i]
  iteration.error.list[[iteration]] <- data.table(
    iteration, error=sum(closest.dt[["error"]]))
```

```
    iteration.both <- rbind(
      data.table(type="centers", centers.dt, cluster=1:K),
      closest.dt[, data.table(
        type="data", data.dt[data.i], cluster=factor(centers.i))])
    data.and.centers.list[[iteration]] <- data.table(
      iteration, iteration.both)
    new.centers <- closest.dt[, data.table(
      t(colMeans(data.dt[data.i]))
    ), keyby=.(cluster=centers.i)]
    centers.dt <- new.centers[, names(centers.dt), with=FALSE]
    centers.mat <- as.matrix(centers.dt)
  }
  (data.and.centers <- do.call(rbind, data.and.centers.list))
```

```
      iteration    type Petal.Width Petal.Length cluster
   1:         1 centers         0.2          1.4       1
   2:         1 centers         2.1          5.4       2
  ---
3059:        20    data         2.3          5.4       2
3060:        20    data         1.8          5.1       2
```

```
  (iteration.error <- do.call(rbind, iteration.error.list))
```

```
    iteration     error
 1:         1 123.63000
 2:         2  85.82705
---
19:        19  31.37136
20:        20  31.37136
```

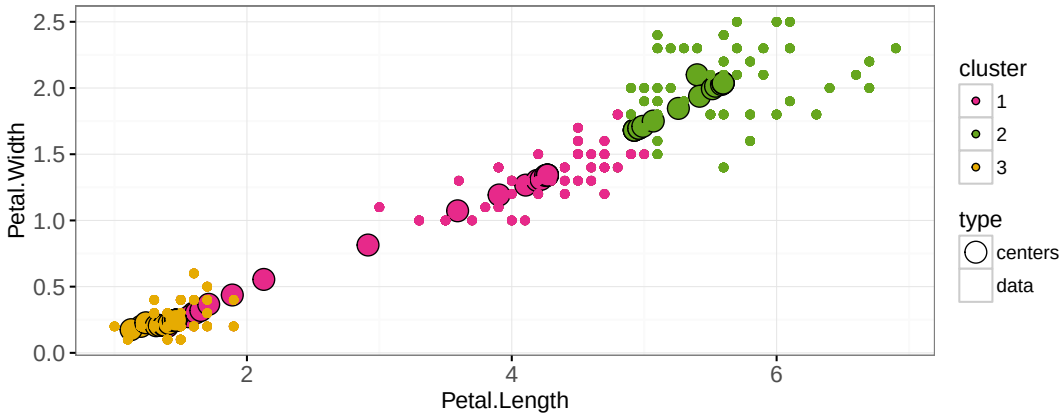
First we create an overview plot with an error curve that will be used to select the model size,

```
gg.err <- ggplot()+
  theme_bw()+
  geom_point(aes(
    iteration, error),
    data=iteration.error)+
  make_tallrect(iteration.error, "iteration", alpha=0.3)
```

We also make a plot which will show the current iteration,

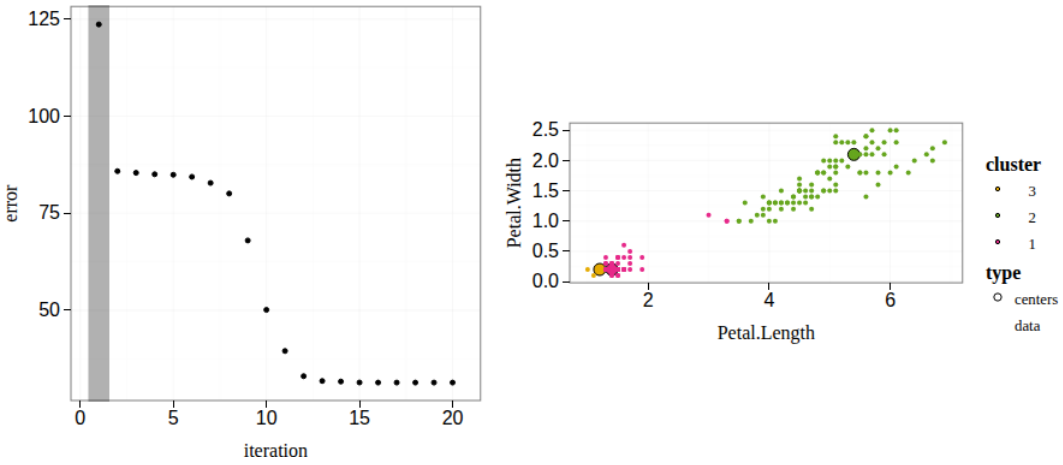
```
gg.iteration <- ggplot()+
  scale_fill_manual(values=color.code)+
  scale_color_manual(values=c(centers="black", data=NA))+
  scale_size_manual(values=c(centers=5, data=2))+
  geom_point(aes(
    Petal.Length, Petal.Width, fill=cluster, size=type, color=type),
    showSelected="iteration",
    data=data.and.centers)+
```

```
  coord_equal()+
  theme_bw()
gg.iteration
```



Combining the two plots results in an interactive data viz,

```
animint(gg.err, gg.iteration)
```



17.3 Chapter summary and exercises

Exercises:

- Make centers always show up in front (on top) of the data.
- Add smooth transitions.
- Add animation on iteration variable.
- Current code has fixed max number of iterations, so it is possible for the last few iterations to make no progress. For example in the viz above, iteration=16 was the last one that resulted in a decrease in error (iterations 17-20 resulted in no decrease). Modify

the code so that it stops iterating if there is no decrease in error.
- Current viz has only one animation frame (`showSelected` subset) per iteration (the mean shown is before it is updated). Add another animation frame that shows the mean after the update.
- Add interactive segments that show the distance from each data point to each cluster center (as in first animint on this page).
- Add the features described in the exercises in the previous section on this page.
- Compute results for several different random seeds, then display error rates for each seed on the error overview plot, and allow the user to select any of those results.
- Compute results for several different numbers of clusters (K). Compute the Adjusted Rand Index using `pdfCluster::adj.rand.index(species, cluster)` for each different K and seed. Add an overview plot that shows the ARI value of each model, and allows selecting the number of clusters.
- Make a similar visualization using another data set such as `data("penguins", package="palmerpenguins")`.

Next, Chapter 18 explains how to visualize the gradient descent learning algorithm for neural network learning.

18

Neural networks

In this chapter we will explore several data visualizations of the gradient descent learning algorithm for Neural networks.

Chapter outline:

- We begin by simulating and visualizing some 2d data for binary classification.
- We then show how a classification function in 2d can be visualized by computing predictions on a grid, and then using `geom_tile()` or `geom_path()` with contour lines.
- We compute linear model predictions, and gradient descent updates, using a simple automatic differentiation (auto-grad) system.
- We end by implementing gradient descent for a neural network, and using an interactive data visualization to show how the predictions get more accurate with iterations of the learning algorithm.

18.1 Visualize simulated data

In this section, we simulate a simple data set with a non-linear pattern for binary classification.

```
sim.col <- 2
sim.row <- 100
set.seed(1)
features.hidden <- matrix(runif(sim.row*sim.col), sim.row, sim.col)
head(features.hidden)
```

```
         [,1]      [,2]
[1,] 0.2655087 0.6547239
[2,] 0.3721239 0.3531973
[3,] 0.5728534 0.2702601
[4,] 0.9082078 0.9926841
[5,] 0.2016819 0.6334933
[6,] 0.8983897 0.2132081
```

In the simulation, the data table above has the "hidden" features which are used to create the labels, but are not available for learning. The latent/true function used for classification is the following,

```
bayes <- function(DT)DT[, (V1>0.2 & V2<0.8)]
library(data.table)
```

```
hidden.dt <- data.table(features.hidden)
label.vec <- ifelse(bayes(hidden.dt), 1, -1)
table(label.vec)
```

```
label.vec
-1  1
31 69
```

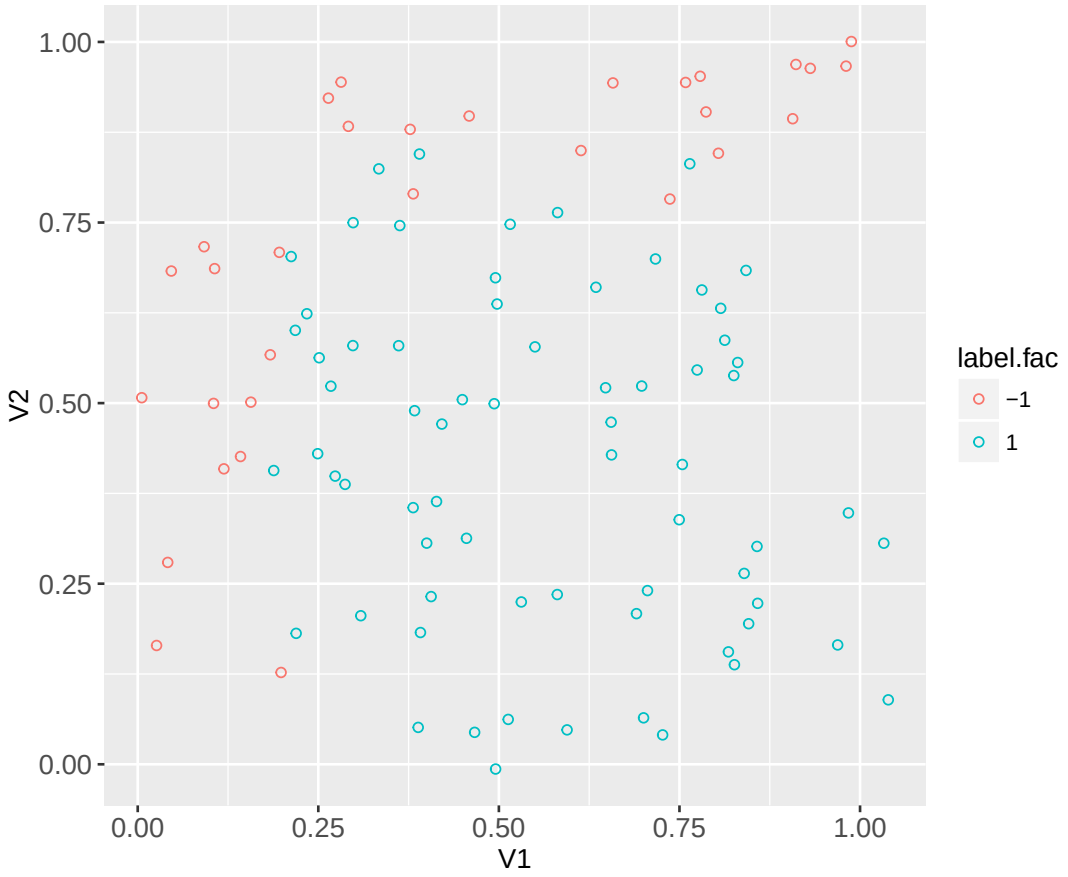
The binary labels above are created from the hidden features, but for learning we only have access to the noisy features below,

```
set.seed(1)
features.noisy <- features.hidden+rnorm(sim.row*sim.col, sd=0.05)
head(features.noisy)
```

```
         [,1]      [,2]
[1,] 0.2341860 0.6237056
[2,] 0.3813061 0.3553031
[3,] 0.5310719 0.2247141
[4,] 0.9879718 1.0005855
[5,] 0.2181573 0.6007640
[6,] 0.8573663 0.3015725
```

To plot the data and visualize the pattern, we use the code below,

```
library(animint2)
label.fac <- factor(label.vec)
sim.dt <- data.table(features.noisy, label.fac)
ggplot()+
  geom_point(aes(
    V1, V2, color=label.fac),
    data=sim.dt)+
  coord_equal()
```



The plot above shows each row in the data set as a point, with the two features on the two axes, and the two labels in two different colors. The lower right part of the feature space tends to have positive labels, and the left and top areas have negative labels. This is the pattern that the neural network will learn. To properly train a neural network, we need to split the data into two sets:

- subtrain: used to compute gradients, which are used to update weight parameters, and predicted values. With enough iterations/epochs of the gradient descent learning algorithm, and a powerful enough neural network model (large enough number of hidden units/layers), it should be possible to get perfect prediction on the subtrain set.
- validation: used to avoid overfitting. By computing the prediction error on the validation set, and choosing the number of gradient descent iterations/epochs which minimizes the validation error, we can ensure the learned model has good generalization properties (provides good predictions on not only the subtrain set, but also new data points like in the validation set).

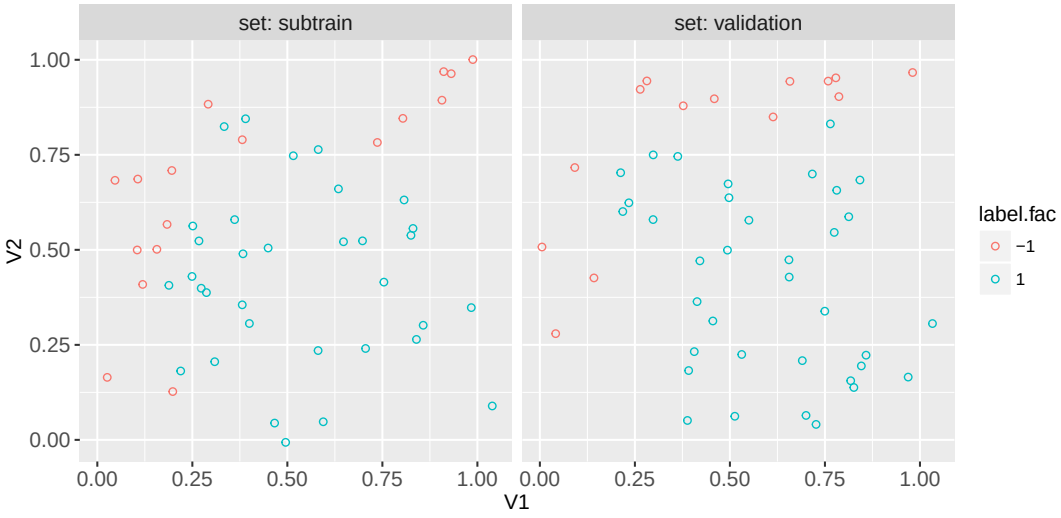
```
is.set.list <- list(
  validation=rep(c(TRUE,FALSE), l=nrow(features.noisy)))
is.set.list$subtrain <- !is.set.list$validation
set.vec <- ifelse(is.set.list$validation, "validation", "subtrain")
table(set.vec)
```

```
set.vec
```

```
subtrain validation
      50         50
```

The code above is used to randomly assign half of the data into each of the subtrain and validation sets. Below, we plot the two sets in separate facets,

```
sim.dt[, set := set.vec]
ggplot()+
  facet_grid(. ~ set, labeller=label_both)+
  geom_point(aes(
    V1, V2, color=label.fac),
    data=sim.dt)+
  coord_equal()
```



18.2 Visualize Bayes optimal classification function

To visualize the optimal/Bayes decision boundary, we need to evaluate the function on a 2d grid of points that spans the feature space. To create such a grid, we first create a list which contains the two 1d grids for each feature,

```
(grid.list <- lapply(sim.dt[, .(V1, V2)], function(V){
  seq(min(V), max(V), l=30)
}))
```

```
$V1
 [1] 0.005600558 0.041238397 0.076876237 0.112514077 0.148151916 0.183789756
 [7] 0.219427595 0.255065435 0.290703274 0.326341114 0.361978954 0.397616793
[13] 0.433254633 0.468892472 0.504530312 0.540168151 0.575805991 0.611443831
[19] 0.647081670 0.682719510 0.718357349 0.753995189 0.789633028 0.825270868
[25] 0.860908708 0.896546547 0.932184387 0.967822226 1.003460066 1.039097905
```

```
$V2
 [1] -0.006562821  0.028166431  0.062895684  0.097624936  0.132354189
 [6]  0.167083441  0.201812694  0.236541946  0.271271199  0.306000451
[11]  0.340729703  0.375458956  0.410188208  0.444917461  0.479646713
[16]  0.514375966  0.549105218  0.583834471  0.618563723  0.653292975
[21]  0.688022228  0.722751480  0.757480733  0.792209985  0.826939238
[26]  0.861668490  0.896397742  0.931126995  0.965856247  1.000585500
```

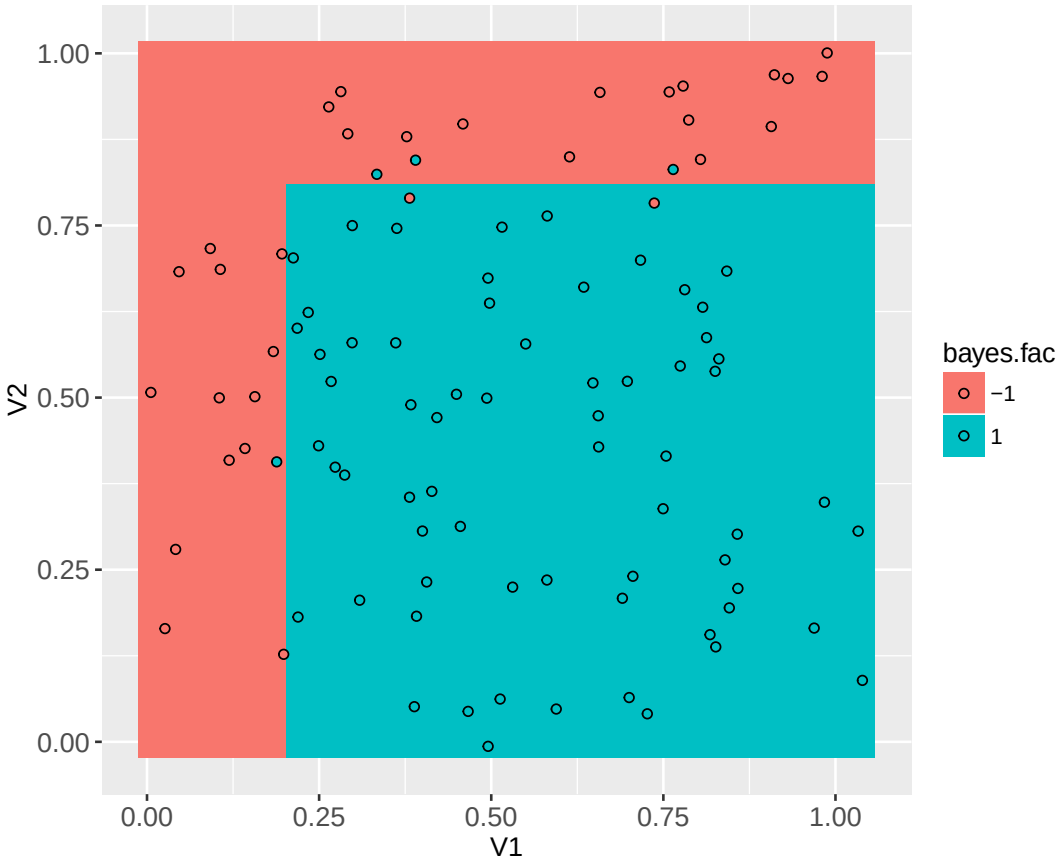
Then we use `CJ` (cross-join) to create a data table representing the 2d grid, for which we evaluate the best/Bayes classification function,

```
(grid.dt <- do.call(
  CJ, grid.list
)[
, bayes.num := ifelse(bayes(.SD), 1, -1)
][
, bayes.fac := factor(bayes.num)
][])
```

```
             V1           V2 bayes.num bayes.fac
  1: 0.005600558 -0.006562821        -1        -1
  2: 0.005600558  0.028166431        -1        -1
 ---
899: 1.039097905  0.965856247        -1        -1
900: 1.039097905  1.000585500        -1        -1
```

The best classifier is visualized below in the feature space,

```
ggplot()+
  geom_tile(aes(
    V1, V2, fill=bayes.fac),
    color=NA,
    data=grid.dt)+
  geom_point(aes(
    V1, V2, fill=label.fac),
    color="black",
    data=sim.dt)+
  coord_equal()
```



The plot above shows that even using the best possible function, there are still some prediction errors (points on a background of different color). Another way to visualize that best classification function is via the decision boundary, which can be computed using the code below,

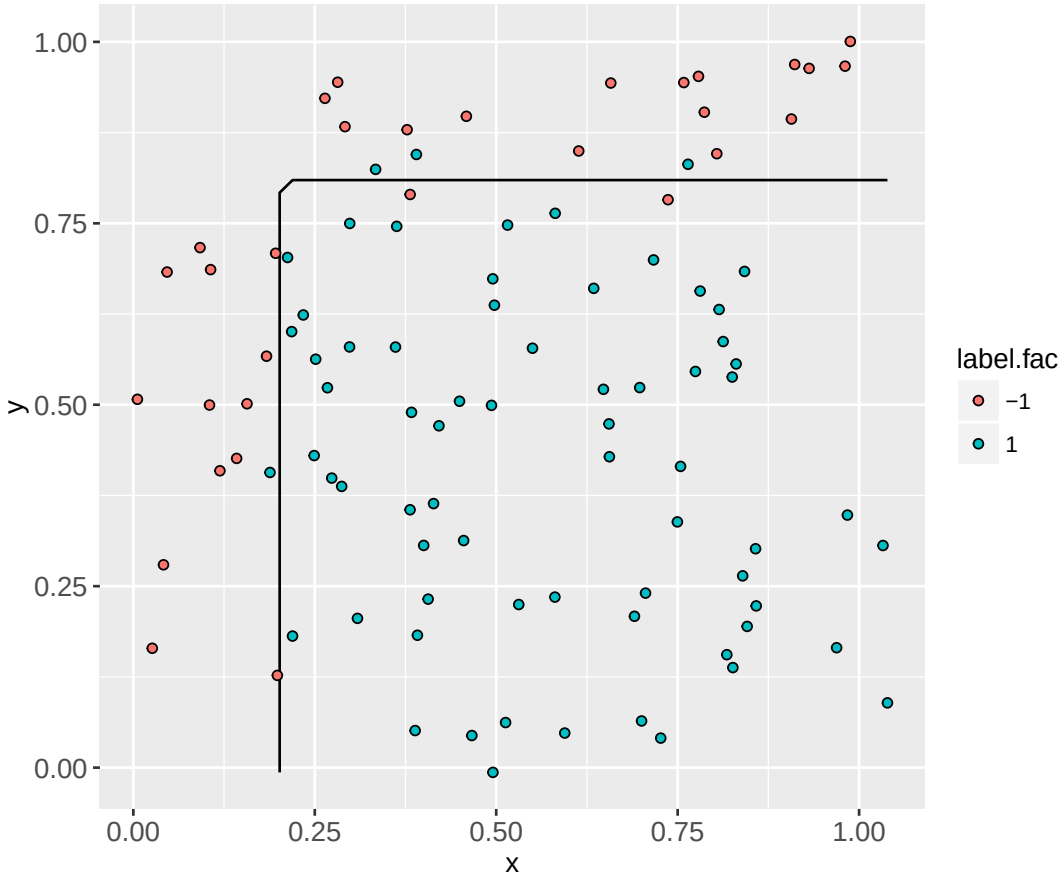
```
get_boundary <- function(score){
  contour.list <- contourLines(
    grid.list$V1, grid.list$V2,
    matrix(
      score,
      length(grid.list$V1),
      length(grid.list$V2),
      byrow=TRUE),
    levels=0)
  if(length(contour.list)){
    data.table(contour.i=seq_along(contour.list))[, {
      with(contour.list[[contour.i]], data.table(level, x, y))
    }, by=contour.i]
  }
}
(bayes.contour.dt <- get_boundary(grid.dt$bayes.num))
```

```
  contour.i level         x           y
```

```
 1:          1     0 0.2016087 -0.006562821
 2:          1     0 0.2016087  0.028166431
---
47:          1     0 1.0034601  0.809574611
48:          1     0 1.0390979  0.809574611
```

The best decision boundary is visualized in the feature space below,

```
ggplot()+
  geom_path(aes(
    x, y, group=contour.i),
    data=bayes.contour.dt)+
  geom_point(aes(
    V1, V2, fill=label.fac),
    color="black",
    data=sim.dt)+
  coord_equal()
```



18.3 Forward and back propagation in linear model

To implement the gradient descent algorithm for learning neural network model parameters, we will use a simple auto-grad system. Auto-grad is the idea that the neural network model structure should be defined once, and that structure should be used to derive both the forward (prediction) and backward (gradient) propagation computations. Below we use a simple auto-grad system where each node in the computation graph is represented by an R environment (a mutable data structure, necessary so that the gradients are back-propagated to all the model parameters). The function below is a constructor for the most basic building block of the auto-grad system, a node in the computation graph:

```
new_node <- function(value, gradient=NULL, ...){
  node <- new.env()
  node$value <- value
  node$parent.list <- list(...)
  node$backward <- function(){
    grad.list <- gradient(node)
    for(parent.name in names(grad.list)){
      parent.node <- node$parent.list[[parent.name]]
      parent.node$grad <- grad.list[[parent.name]]
      parent.node$backward()
    }
  }
  node
}
```

The code in the function above starts by creating a new environment, then populates it with three objects:

- `value` is a matrix computed by forward propagation at this node in the computation graph.
- `parent.list` is a list of parent nodes, each of which is used to compute `value`.
- `backward` is a function which should be called by the user on the final/loss node in the computation graph. It calls `gradient`, which should compute the gradient of the loss with respect to the parent nodes, which are stored in the `grad` attribute in each corresponding parent node, before recursively calling `backward` on each parent node.

The simplest kind of node is an initial node, defined by the code below,

```
initial_node <- function(mat){
  new_node(mat, gradient=function(...)list())
}
```

The code above says that an initial node simply stores the input matrix `mat` as the value, and has a `gradient` method that does nothing (because initial nodes in the computation graph have no parents for which gradients could be computed). The code below defines `mm`, a node in the computation graph which represents a matrix multiplication,

```
mm <- function(feature.node, weight.node)new_node(
  cbind(1, feature.node$value) %*% weight.node$value,
  features=feature.node,
  weights=weight.node,
  gradient=function(node)list(
    features=node$grad %*% t(weight.node$value),
    weights=t(cbind(1, feature.node$value)) %*% node$grad))
```

The `mm` definition above assumes that there is a weight node with the same number of rows as the number of columns (plus one for intercept) in the feature node. The forward/value and gradient computations use matrix multiplication. For instance, we can use`mm` as follows to define a simple linear model,

```
feature.node <- initial_node(features.noisy[is.set.list$subtrain,])
weight.node <- initial_node(rep(0, ncol(features.noisy)+1))
linear.pred.node <- mm(feature.node, weight.node)
str(linear.pred.node$value)
```

```
 num [1:50, 1] 0 0 0 0 0 0 0 0 0 0 ...
```

It can be seen in the code above that the `mm` function returns a node representing predicted values, one for each row in the feature matrix. To use the gradient features we need a loss function, which in the case of binary classification is the logistic (cross-entropy) loss,

```
log_loss <- function(pred.node, label.node)new_node(
  mean(log(1+exp(-label.node$value*pred.node$value))),
  pred=pred.node,
  label=label.node,
  gradient=function(...)list(
    pred=-label.node$value/(
      1+exp(label.node$value*pred.node$value)
    )/length(label.node$value)))
```

The code above defines the logistic loss and gradient, assuming the label is either -1 or 1, and the prediction is a real number (not necessarily between 0 and 1, maybe negative). The code below creates nodes for the labels and loss,

```
label.node <- initial_node(label.vec[is.set.list$subtrain])
loss.node <- log_loss(linear.pred.node, label.node)
loss.node$value
```

```
[1] 0.6931472
```

Now that we have computed the loss, we can compute the gradient of the loss with respect to the weights, which is used to perform the updates during learning. Remember that we should now call`backward` (on the subtrain loss), which should eventually store the gradient as `weight.node$grad`. Below we first verify that it has not yet been computed, then we compute it:

```
weight.node$grad
```

```
NULL
```

```
  loss.node$backward()
  weight.node$grad
```

```
            [,1]
[1,] -0.16000000
[2,] -0.10511619
[3,] -0.02447615
```

Note that since `loss.node` contains recursive back-references to its parent nodes (including predictions and weights), the `backward` call above is able to conveniently compute and store `weight.node$grad`, the gradient of the loss with respect to the weight parameters. The gradient is the direction of steepest ascent, meaning the direction weights could be modified to maximize the loss. Because we want to minimize the loss, the learning algorithm performs updates in the negative gradient direction, of steepest descent.

```
  (descent.direction <- -weight.node$grad)
```

```
           [,1]
[1,] 0.16000000
[2,] 0.10511619
[3,] 0.02447615
```

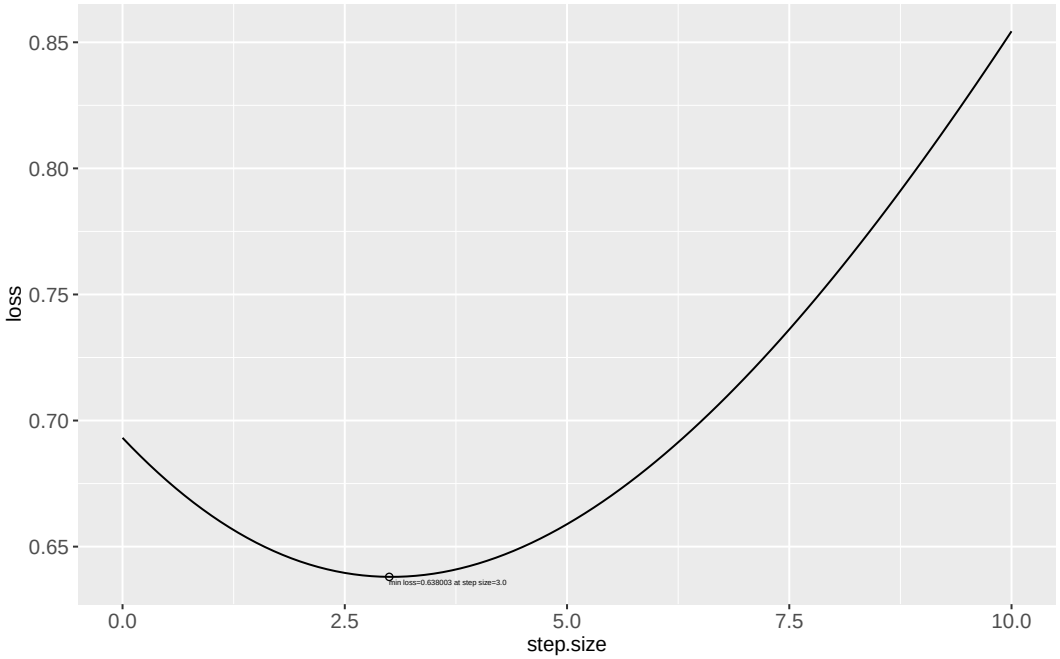
In gradient descent for this linear model, we update the weight vector in this direction. Each update to the weight vector is referred to as an iteration or step. A small step in this direction is guaranteed to decrease the loss, but too small of a step will not make much progress toward minimizing the loss. It is unknown how far in this direction is best, so we typically need to search over a grid of step sizes (aka learning rates). Or we can perform a line search, which means making the following plot of loss as a function of step size, then choosing the step size with minimal loss.

```
  (line.search.dt <- data.table(step.size=seq(0, 10, l=101))[, .(
    loss=log_loss(mm(
      feature.node,
      initial_node(weight.node$value+step.size*descent.direction)
    ), label.node)$value
  ), by=step.size])
```

```
     step.size      loss
  1:       0.0 0.6931472
  2:       0.1 0.6894865
 ---
100:       9.9 0.8490474
101:      10.0 0.8543739
```

```
  line.search.min <- line.search.dt[which.min(loss)]
  ggplot()+
    geom_line(aes(
      step.size, loss),
```

```
    data=line.search.dt)+
  geom_point(aes(
    step.size, loss),
    data=line.search.min)+
  geom_text(aes(
    step.size, loss, label=sprintf(
      "min loss=%f at step size=%.1f",
      loss, step.size)),
    data=line.search.min,
    size=4,
    hjust=0,
    vjust=1.5)
```



The plot above shows that the min loss occurs at a step size of about 3, which means that the line search would choose that step size for this gradient descent parameter update/iteration, before re-computing the gradient in the next iteration.

18.4 Neural network learning

Defining a linear model in the previous section was relatively simple, because there is only one weight matrix parameter (actually, a weight vector, with the same number of elements as the number of columns/features, plus one for an intercept). In contrast, a neural network has more than one weight matrix parameter to learn. We initialize these weights as nodes in the code below,

```
new_weight_node_list <- function(units.per.layer, intercept=TRUE){
  weight.node.list <- list()
  for(layer.i in seq(1, length(units.per.layer)-1)){
    input.units <- units.per.layer[[layer.i]]+intercept
    output.units <- units.per.layer[[layer.i+1]]
    weight.mat <- matrix(
      rnorm(input.units*output.units), input.units, output.units)
    weight.node.list[[layer.i]] <- initial_node(weight.mat)
  }
  weight.node.list
}
(units.per.layer <- c(ncol(features.noisy), 40, 1))
```

```
[1]  2 40  1
```

```
(weight.node.list <- new_weight_node_list(units.per.layer))
```

```
[[1]]
<environment: 0x55bc315dca70>

[[2]]
<environment: 0x55bc315dbae8>
```

```
lapply(weight.node.list, function(node)dim(node$value))
```

```
[[1]]
[1]  3 40

[[2]]
[1] 41  1
```

The output above shows that there is a single layer with 40 hidden units in the neural network, meaning there are two weight matrices to learn. Each of these weight matrices is used to predict the units in a given layer, from the units in the previous layer. In order to learn a prediction function which is a non-linear function of the features, each layer except for the last must have a non-linear activation function, applied element-wise to the units after matrix multiplication. For example, a common and efficient non-linear activation function is the ReLU (Rectified Linear Units), which is implemented below,

```
relu <- function(before.node)new_node(
  ifelse(before.node$value < 0, 0, before.node$value),
  before=before.node,
  gradient=function(node)list(
    before=ifelse(before.node$value < 0, 0, node$grad)))
hidden.before.act <- mm(feature.node, weight.node.list[[1]])
str(hidden.before.act$value)
```

```
 num [1:50, 1:40] 1.62 3.67 2.34 2.42 1.57 ...
```

```
  hidden.before.act$value[1:5,1:5]
```

```
         [,1]        [,2]      [,3]      [,4]       [,5]
[1,] 1.617099 -0.31485515 0.8687167 0.5969050 -0.5717334
[2,] 3.665478 -0.08953216 1.1884750 0.7686507 -0.7655569
[3,] 2.335856 -1.53696561 1.1271410 0.4960481 -1.2771119
[4,] 2.418533 -0.61750628 1.0377415 0.6157081 -0.8390015
[5,] 1.571396  1.26836784 0.6830969 0.7897418  0.2105804
```

```
  hidden.after.act <- relu(hidden.before.act)
  hidden.after.act$value[1:5,1:5]
```

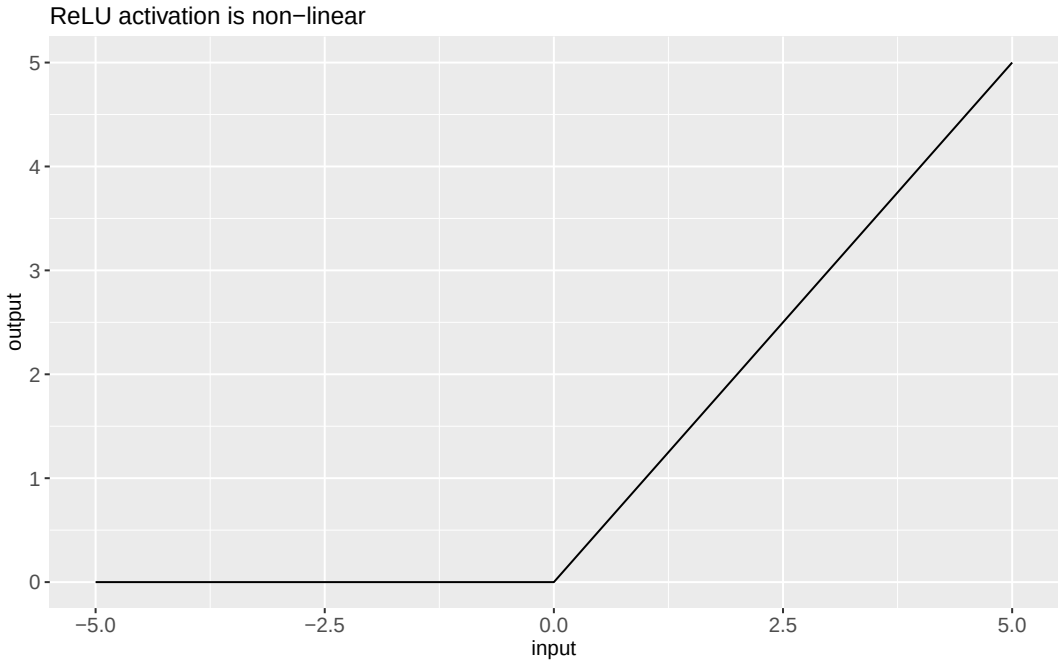
```
         [,1]     [,2]      [,3]      [,4]      [,5]
[1,] 1.617099 0.000000 0.8687167 0.5969050 0.0000000
[2,] 3.665478 0.000000 1.1884750 0.7686507 0.0000000
[3,] 2.335856 0.000000 1.1271410 0.4960481 0.0000000
[4,] 2.418533 0.000000 1.0377415 0.6157081 0.0000000
[5,] 1.571396 1.268368 0.6830969 0.7897418 0.2105804
```

Note in the output above how the ReLU activation sets negative values to zero, and keeps positive values the same:

```
  (relu.dt <- data.table(
    input=seq(-5, 5, l=101)
  )[, output := relu(initial_node(input))$value][])
```

```
     input output
  1:  -5.0    0.0
  2:  -4.9    0.0
 ---
100:   4.9    4.9
101:   5.0    5.0
```

```
  ggplot()+
    ggtitle("ReLU activation is non-linear")+
    geom_line(aes(
      input, output),
      data=relu.dt)
```



Finally, the last node that we need to implement our neural network is a node for predictions, computed via the for loop over weight nodes in the function below,

```
pred_node <- function(set.features){
  feature.node <- initial_node(set.features)
  for(layer.i in seq_along(weight.node.list)){
    weight.node <- weight.node.list[[layer.i]]
    before.node <- mm(feature.node, weight.node)
    feature.node <- if(layer.i < length(weight.node.list)){
      relu(before.node)
    }else{
      before.node
    }
  }
  feature.node
}
nn.pred.node <- pred_node(features.noisy[is.set.list$subtrain,])
str(nn.pred.node$value)
```

```
num [1:50, 1] 3.85 6.42 2.14 3.54 8.18 ...
```

The `pred_node` function is also useful for computing predictions on the grid of features, which will be useful later for visualizing the learned function,

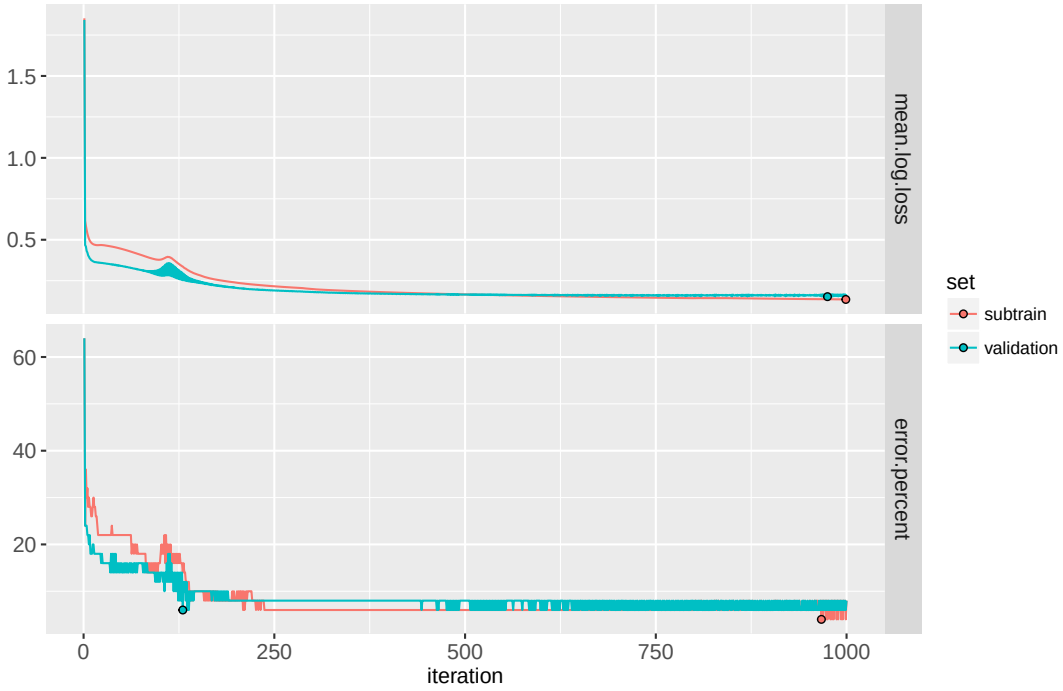
```
grid.mat <- grid.dt[, cbind(V1,V2)]
nn.grid.node <- pred_node(grid.mat)
str(nn.grid.node$value)
```

```
num [1:900, 1] 1.74 2.09 2.43 2.78 3.12 ...
```

The code below combines all of the pieces above into a gradient descent learning algorithm. The hyper-parameters are the constant step size, and the maximum number of iterations.

```
step.size <- 0.5
max.iterations <- 1000
units.per.layer <- c(ncol(features.noisy), 40, 1)
loss.dt.list <- list()
err.dt.list <- list()
pred.dt.list <- list()
set.seed(10)
weight.node.list <- new_weight_node_list(units.per.layer)
for(iteration in 1:max.iterations){
  loss.node.list <- list()
  for(set in names(is.set.list)){
    is.set <- is.set.list[[set]]
    set.label.node <- initial_node(label.vec[is.set])
    set.features <- features.noisy[is.set,]
    set.pred.node <- pred_node(set.features)
    set.loss.node <- log_loss(set.pred.node, set.label.node)
    loss.node.list[[set]] <- set.loss.node
    set.pred.num <- ifelse(set.pred.node$value<0, -1, 1)
    is.error <- set.pred.num != set.label.node$value
    err.dt.list[[paste(iteration, set)]] <- data.table(
      iteration, set,
      set.features,
      label=set.label.node$value,
      pred.num=as.numeric(set.pred.num))
    loss.dt.list[[paste(iteration, set)]] <- data.table(
      iteration, set,
      mean.log.loss=set.loss.node$value,
      error.percent=100*mean(is.error))
  }
  grid.node <- pred_node(grid.mat)
  pred.dt.list[[paste(iteration)]] <- data.table(
    iteration,
    grid.dt,
    pred=as.numeric(grid.node$value))
  loss.node.list$subtrain$backward()#<-back-prop.
  for(layer.i in seq_along(weight.node.list)){
    weight.node <- weight.node.list[[layer.i]]
    weight.node$value <- #learning/param updates:
      weight.node$value-step.size*weight.node$grad
  }
}
loss.dt <- rbindlist(loss.dt.list)
err.dt <- rbindlist(err.dt.list)
pred.dt <- rbindlist(pred.dt.list)
loss.tall <- melt(loss.dt, measure=c("mean.log.loss", "error.percent"))
loss.tall[, log10.iteration := log10(iteration)]
min.dt <- loss.tall[
```

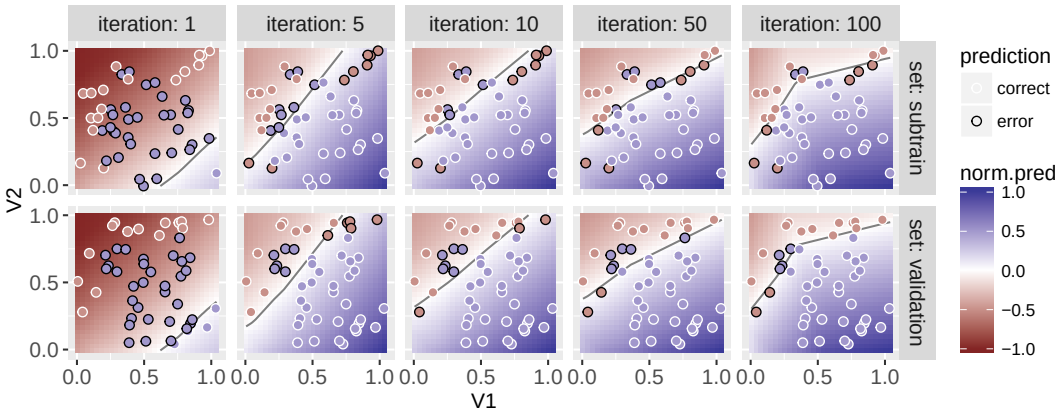
```
, .SD[which.min(value)], by=.(set, variable)]
ggplot()+
  facet_grid(variable ~ ., scales="free")+
  scale_y_continuous("")+
  geom_line(aes(
    iteration, value, color=set),
    data=loss.tall)+
  geom_point(aes(
    iteration, value, fill=set),
    color="black",
    data=min.dt)
```



In the code above we saved loss and predictions for all of the iterations of gradient descent, but in the code below we visualize only some of them, due to limited space:

```
some <- function(DT)DT[iteration%in%c(1,5,10,50,100)]
err.dt[, prediction := ifelse(label==pred.num, "correct", "error")]
iteration.contours <- pred.dt[
, get_boundary(pred), by=.(iteration)]
some.loss <- some(loss.dt)
pred.dt[, norm.pred := pred/max(abs(pred)), by=.(iteration)]
some.pred <- some(pred.dt)
some.err <- some(err.dt)
some.contours <- some.pred[
, get_boundary(pred), by=.(iteration)]
ggplot()+
  facet_grid(set ~ iteration, labeller="label_both")+
```

```
  geom_tile(aes(
    V1, V2, fill=norm.pred),
    color=NA,
    data=some.pred)+
  geom_path(aes(
    x, y, group=contour.i),
    color="grey50",
    data=some.contours)+
  scale_fill_gradient2()+
  geom_point(aes(
    V1, V2, color=prediction, fill=label/2),
    size=2,
    data=some.err)+
  scale_color_manual(
    values=c(correct="white", error="black"))+
  coord_equal()+
  scale_y_continuous(breaks=seq(0,1,by=0.5))+
  scale_x_continuous(breaks=seq(0,1,by=0.5))
```

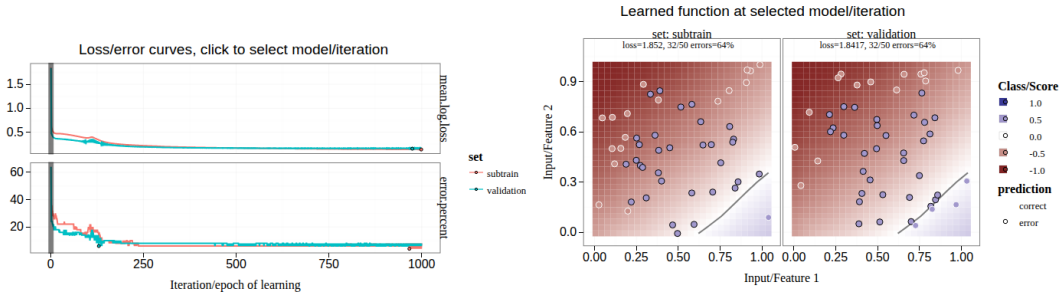


From the plot above we can see that as the number of iterations increases, the predictions get more accurate. Finally, we conclude with an interactive plot where you can click the loss plot to select an iteration of gradient descent, for which the corresponding decision boundary is shown on the predictions plot.

```
n.subtrain <- sum(is.set.list$subtrain)
loss.dt[, n.set := ifelse(
  set=="subtrain", n.subtrain, sim.row-n.subtrain
)][, error.count := n.set*error.percent/100]
it.by <- 10
some <- function(DT)DT[iteration %in% as.integer(
  seq(1, max.iterations, by=it.by))]
animint(
  title="Neural network vs linear model",
  out.dir="neural-networks-sim",
  loss=ggplot()+
    ggtitle("Loss/error curves, click to select model/iteration")+
```

```
    theme_bw()+
    theme_animint(width=600, height=350)+
    theme(panel.margin=grid::unit(1, "lines"))+
    facet_grid(variable ~ ., scales="free")+
    scale_y_continuous("")+
    scale_x_continuous(
      "Iteration/epoch of learning")+
    geom_line(aes(
      iteration, value, color=set, group=set),
      data=loss.tall)+
    geom_point(aes(
      iteration, value, fill=set),
      color="black",
      data=min.dt)+
    geom_tallrect(aes(
      xmin=iteration-it.by/2,
      xmax=iteration+it.by/2),
      alpha=0.5,
      clickSelects="iteration",
      data=some(loss.tall[set=="subtrain"])),
  data=ggplot()+
    ggtitle("Learned function at selected model/iteration")+
    theme_bw()+
    theme_animint(width=600)+
    facet_grid(. ~ set, labeller="label_both")+
    geom_tile(aes(
      V1, V2, fill=norm.pred),
      color=NA,
      showSelected="iteration",
      data=some(pred.dt))+
    geom_text(aes(
      0.5, 1.1, label=paste0(
        "loss=", round(mean.log.loss, 4),
        ", ", error.count, "/", n.set,
        " errors=", error.percent, "%")),
      showSelected="iteration",
      data=loss.dt)+
    geom_path(aes(
      x, y, group=contour.i),
      showSelected="iteration",
      color="grey50",
      data=some(iteration.contours))+
    geom_point(aes(
      V1, V2, fill=label/2, color=prediction),
      showSelected=c("iteration", "set"),
      size=4,
      data=some(err.dt))+
    scale_fill_gradient2(
      "Class/Score")+
```

```
      scale_color_manual(
        values=c(correct="white", error="black"))+
      scale_x_continuous(
        "Input/Feature 1")+
      scale_y_continuous(
        "Input/Feature 2")+
      coord_equal())
```



18.5 Chapter summary and exercises

Exercises:

- Add animation over the number of iterations.
- Add smooth transitions when changing the selected number of iterations.
- Add a for loop over random seeds (or cross-validation folds) in the data splitting step, and create a visualization that shows how that affects the results.
- Add a for loop over random seeds at the weight matrix initialization step, and create a visualization that shows how that affects the results.
- Compute results for another neural network architecture (and/or linear model, by adding a for loop over different values of `units.per.layer`). Add another plot or facet which allows selecting the neural network architecture, and allows easy comparison of the min validation loss between models (for example, add facet columns to loss plot, and add horizontal lines to emphasize min loss).
- Modify the learning algorithm to use line search rather than constant step size, and then create a visualization which compares the two approaches in terms of min validation loss.

Next, Chapter 19 explains how to visualize P-values, including how to work around overplotting, using a heat map linked to a zoomed scatter plot. .

19

P-values

In this chapter we will explore several data visualizations related to P-values:

- we begin by explaining the concept of a P-value.
- we simulate some data for two-sample T-testing.
- we show how to create a volcano plot to summarize a set of P-values.
- we explain how to convert an overplotted scatterplot into a linked heat map and zoomed scatterplot.
- we end with another heat map of simulated parameter values, with a linked plot which shows the simulated data.

19.1 What is a P-value?

{#what-is-p-value}

A P-value is used to measure significance of a statistical test. P-values can be used in a wide range of tests, but a typical application is testing for difference between two conditions:

- in a medical experiment, does the treatment do better than the control? For example, with a weight loss drug, we would be interested to know if the treatment group weight is significantly less than the control group weight. Say we have 10 people assigned to the control group, and 15 people assigned to the treatment group. Then we could compute a P-value using an un-paired one-sided T-test to evaluate statistical significance (un-paired because each person in the treatment group does not have a corresponding member in the control group).
- in a machine learning experiment, is the neural network more accurate than the linear model? For example, consider a benchmark image classification data set, and evaluation using 5-fold cross-validation. We would have five measures of test accuracy for each of the two learning algorithms (neural network and linear model). Then we could compute a P-value using a paired one-sided T-test to evaluate statistical significance (paired because the fold 1 test accuracy for the linear model has a corresponding fold 1 test accuracy for the neural network, etc).

After computing the measurements (weight of each person or test accuracy of each machine learning algorithm), we can use them as input to `t.test()`, which will compute a P-value (smaller for a more significant difference). To understand the P-value, we must first adopt the *null hypothesis*: we assume there is no difference between the conditions. Then, the P-value of the test is defined as the probability that we observe a difference as large as the given measurements, or larger. Since under the null hypothesis, there is no difference, it is extremely unlikely to see large differences, so that is why small P-values are more significant.

19.2 Simulated data

We begin by simulating data for use with `t.test()`. Our simulation has four parameters:

- `true_offset` is the true difference between conditions,
- `sd` is the standard deviation of the simulated data,
- `sample` is a sample number (half of samples in one condition, half in the other),
- `trial` is the number of times that we repeat the experiment (for each offset and standard deviation).

The code below uses `CJ()` to define the values of each parameter:

```
library(data.table)
offset_by <- 0.1
sd_by <- 0.1
set.seed(1)
(sim_dt <- CJ(
  true_offset=round(
    seq(-3, 3, by=offset_by),
    ceiling(-log10(offset_by))),
  sd=seq(0.1, 1, by=sd_by),
  sample=seq(0, 9),
  trial=seq_len(100)
)[, let(
  condition = sample %% 2,
  pair = sample %/% 2
)][, let(
  value = rnorm(.N, true_offset*condition, sd)
)][])
```

```
        true_offset  sd sample trial condition pair       value
     1:          -3 0.1      0     1         0    0 -0.06264538
     2:          -3 0.1      0     2         0    0  0.01836433
    ---
609999:           3 1.0      9    99         1    4  2.54687307
610000:           3 1.0      9   100         1    4  2.70192000
```

The result above shows several hundred thousand rows, one for each simulated random normal `value` (with mean depending on `true_offset` and `condition`).

19.3 T-test and volcano plot

In this section we compute T-test results and visualize them using a volcano plot, which is a plot of negative log P-values versus estimated effect sizes. The test result visualization is called a volcano plot because the typical distribution of points resembles a volcano erupting from the origin upwards to the left and right. First, we add columns which we will use for visualization:

- `true_tile` is a text string for selecting a combination of one offset and one standard deviation.
- `Condition` is a string indicating the condition (either `zero` or `offset`).

```
  sim_dt[, let(
    true_tile=paste(true_offset, sd),
    Condition = ifelse(condition, "offset", "zero")
  )]
```

Next, we do a reshape to obtain a table with a column for each condition (`zero` and `offset`).

```
  (sim_wide <- dcast(
    sim_dt,
    true_tile + true_offset + sd + trial + pair ~ Condition))
```

```
       true_tile true_offset  sd trial pair      offset       zero
    1:  -0.1 0.1        -0.1 0.1     1    0 -0.08965774 -0.1451173
    2:  -0.1 0.1        -0.1 0.1     1    1 -0.10685583 -0.0862245
   ---
304999:       3 1         3.0 1.0   100    3  2.48334683 -0.7826612
305000:       3 1         3.0 1.0   100    4  2.70192000  0.3343277
```

The output above shows a table with half as many rows as the previous table. The code below computes a T-test for each repetition of the simulation:

```
  (sim_p <- sim_wide[, {
    t.result <- t.test(zero, offset, var.equal=TRUE)
    with(t.result, data.table(
      p.value,
      mean_zero=estimate[1],
      mean_offset=estimate[2]
    ))
  }, by=.(true_tile, true_offset, sd, trial)])
```

```
       true_tile true_offset  sd trial     p.value   mean_zero mean_offset
    1:  -0.1 0.1        -0.1 0.1     1 0.807971951 -0.15151092  -0.1383895
    2:  -0.1 0.1        -0.1 0.1     2 0.075025835  0.01060259  -0.1238523
   ---
60999:       3 1         3.0 1.0    99 0.001067247  0.14249170   2.4716192
61000:       3 1         3.0 1.0   100 0.005101731 -0.04792505   2.3569228
```

The output above has one row for each T-test, and columns for mean difference (`mean_zero`) and `p.value`. Since there are ten samples per trial, there are ten times fewer rows than the original simulated data table. Each trial involves a T-test of 5 control/zero samples, versus 5 treatment/offset samples. Next, we add columns for the volcano plot:

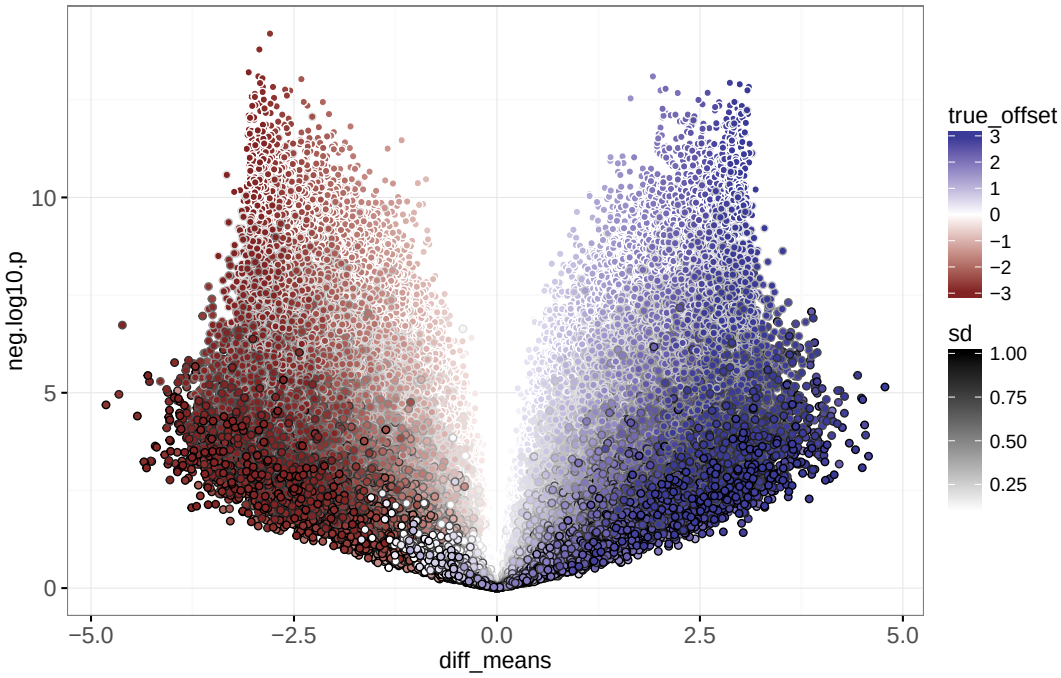
- `diff_means` is the difference between means of the two conditions, sometimes called the "effect size."
- `neg.log10.p` is the negative log-transformed P-value (larger for more significant).

```
sim_p[, let(
  diff_means = mean_offset - mean_zero,
  neg.log10.p = -log10(p.value)
)]
```

Next, we draw the volcano plot:

- X axis shows the difference between conditions (effect size).
- Y axis shows negative log P-value.

```
library(animint2)
(gg.volcano <- ggplot()+
  geom_point(aes(
    diff_means, neg.log10.p, fill=true_offset, color=sd),
    data=sim_p)+
  scale_fill_gradient2()+
  scale_color_gradient(low="white", high="black")+
  theme_bw())
```



The static graphic above shows one dot per T-test result. Dots close to the origin (0,0) represent tests which did not yield a significant difference, whereas dots with large Y values represent significant differences. It also includes `fill=true_offset` and `color=sd` so we can see how the simulation parameters affect the volcano plot:

- Larger `sd` values appear at the bottom (more variance makes it more difficult to detect a difference).
- Darker colors appear near the left/right edges (larger true offsets tend to result in larger computed differences in means).

Finally, the plot above is overplotted, meaning we are unable to see all of the details, because

there are too many data points plotted on top of one another.

19.4 Fix overplotting by heat map and zoom

In this section, we show how the previous volcano plot can be revised to show more detail, using a heat map linked to a zoomed scatterplot. To begin, we define the `round_rel()` function below, which is used to add `round_*` and `rel_*` columns which we will use to define heat map tiles.

- `round_*` columns are rounded to the nearest bin size (integer by default), and are used to define the heat map tile `x` and `y` positions.
- `rel_*` columns are used for `x` and `y` positions in the zoomed display, and are units relative to the corresponding `round_*` values.

```
round_rel <- function(DT, col_name, bin_size=1, offset=0){
  value <- DT[[col_name]]
  round_value <- round((value+offset)/bin_size)*bin_size
  DT[, paste0(c("round","rel"), "_", col_name) := list(
    round_value, value-round_value)]
}
round_rel(sim_p, "diff_means")
round_rel(sim_p, "neg.log10.p")
```

Next, we define `volcano_tile`, a text string combination of `round_*` variables, which will be used for selection.

```
sim_p[, volcano_tile := paste(round_diff_means, round_neg.log10.p)]
```

Next, we compute a table with one row per tile in the heat map we will display.

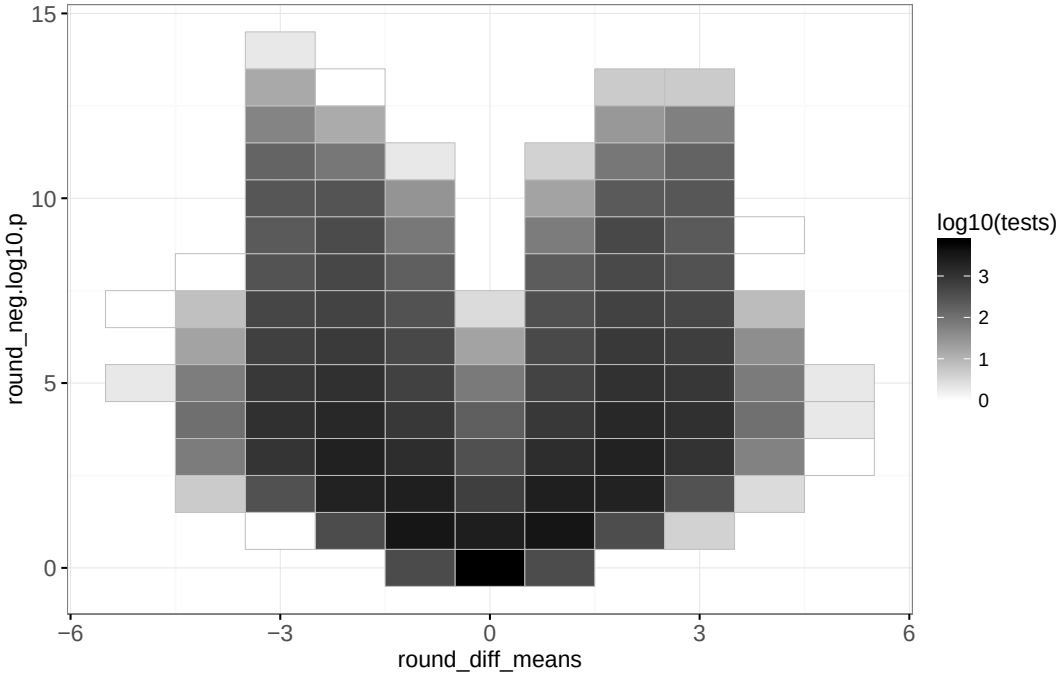
```
(volcano_tile_dt <- sim_p[, .(
  tests=.N
), by=.(volcano_tile, round_diff_means, round_neg.log10.p)])
```

```
    volcano_tile round_diff_means round_neg.log10.p tests
 1:          0 0                0                 0  6529
 2:          0 1                0                 1  2391
---
103:         5 5                5                 5     2
104:         4 9                4                 9     1
```

The output above shows one row per heat map tile, with the `tests` column indicating how many points appear in the corresponding area of the volcano plot. Below we create the volcano heat map.

```
(gg.volcano.tiles <- ggplot()+
  geom_tile(aes(
```

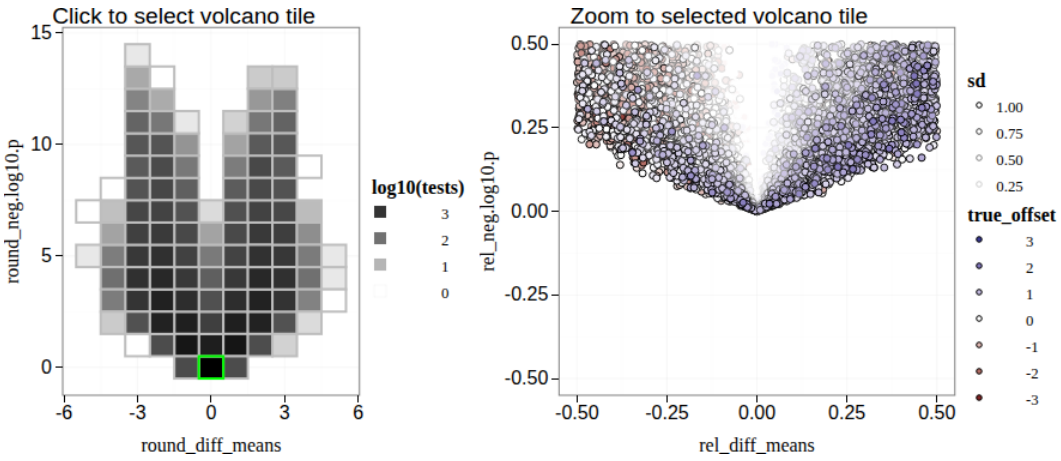
```
    round_diff_means, round_neg.log10.p, fill=log10(tests)),
    color="grey",
    data=volcano_tile_dt)+
  scale_fill_gradient(low="white",high="black")+
  theme_bw())
```



The output above is a heat map, with darker regions showing areas of the volcano plot which have more test results. The code below combines the volcano heat map with a zoomed scatterplot, using the `volcano_tile` selector to link them.

```
(viz.volcano <- animint(
  volcanoTiles=gg.volcano.tiles+
    ggtitle("Click to select volcano tile")+
    theme_animint(width=300, rowspan=1)+
    geom_tile(aes(
      round_diff_means, round_neg.log10.p),
      clickSelects="volcano_tile",
      color="green",
      fill="transparent",
      data=volcano_tile_dt),
  volcanoZoom=ggplot()+
    ggtitle("Zoom to selected volcano tile")+
    geom_point(aes(
      rel_diff_means, rel_neg.log10.p, fill=true_offset, color=sd),
      showSelected="volcano_tile",
      size=3,
      data=sim_p)+
```

```
      scale_fill_gradient2()+
      scale_color_gradient(low="white", high="black")+
      theme_bw()))
```



Above, after clicking the heat map on the left, the data shown in the right plot changes.

19.5 Visualizing grid of simulations

In this section, we create a new visualization to reveal details of each simulation parameter combination. First, in the code below, we compute the mean effect size (`diff_means`) and negative log P-value (`neg.log10.p`), over all 100 repetitions of each parameter combination.

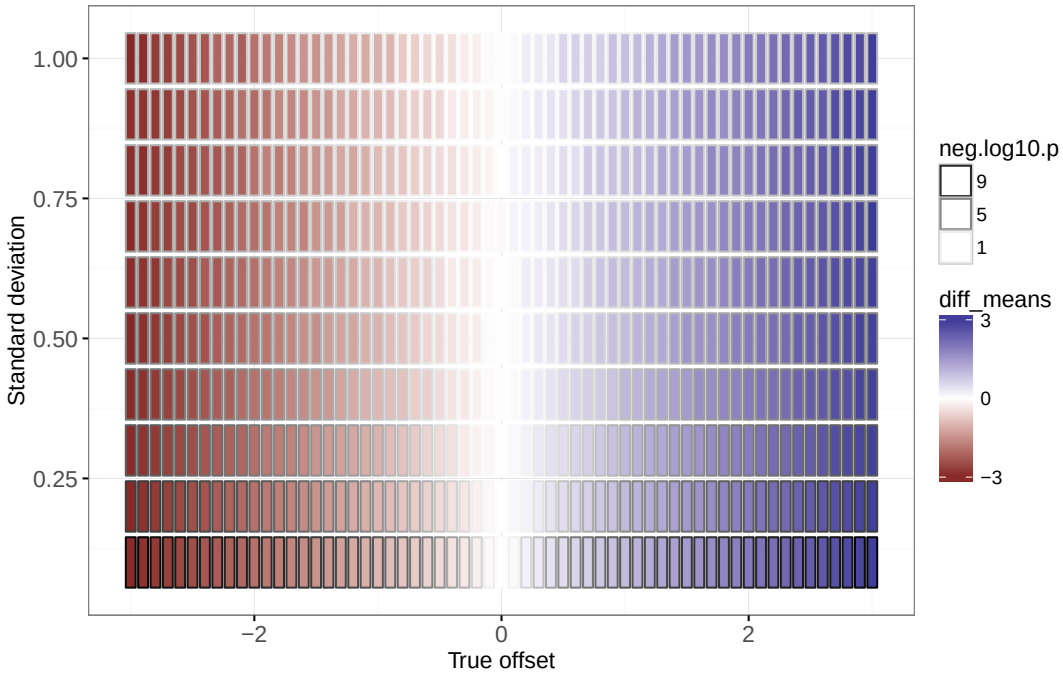
```
(sim_true_tiles <- dcast(
  sim_p,
  true_tile + true_offset + sd ~ .,
  mean,
  value.var=c("diff_means", "neg.log10.p")))
```

```
    true_tile true_offset  sd  diff_means neg.log10.p
 1:  -0.1 0.1        -0.1 0.1 -0.09920214   0.9730339
 2:  -0.1 0.2        -0.1 0.2 -0.09573305   0.6240886
---
609:     3 0.9          3.0 0.9  3.04724871   3.4218491
610:       3 1          3.0 1.0  3.01733378   2.8571784
```

The output above has one row per combination of simulation parameters (`true_offset` and `sd`). We use the code below to visualize this grid of parameter combinations.

```
width <- offset_by*0.4
height <- sd_by*0.45
(gg.true.tiles <- ggplot()+
  scale_x_continuous("True offset")+
```

```
  scale_y_continuous("Standard deviation")+
  scale_fill_gradient2(breaks=c(3,0,-3))+
  scale_color_gradient(
    guide=guide_legend(override.aes=list(fill='white')),
    low="white", high="black", breaks=c(9,5,1))+
  theme_bw()+
  geom_rect(aes(
    xmin=true_offset-width, xmax=true_offset+width,
    ymin=sd-height, ymax=sd+height,
    fill=diff_means, color=neg.log10.p),
    data=sim_true_tiles))
```



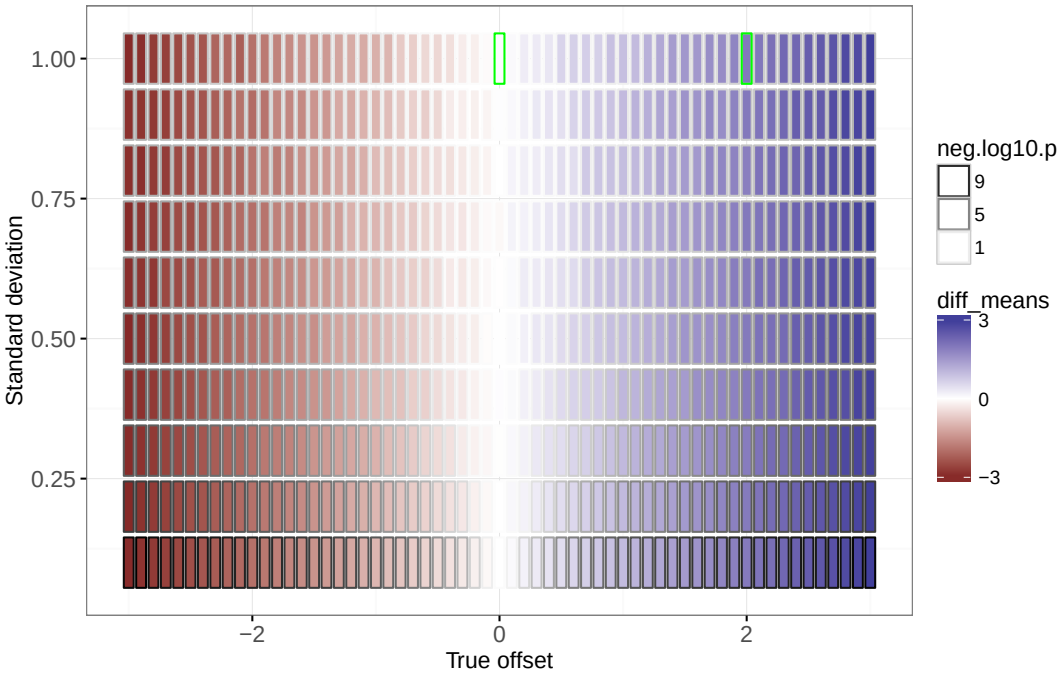
The figure above shows a rectangle for each simulation parameter combination. It is clear that:

- the estimated effect size (`fill=diff_means`) is consistent with the true offset (X axis).
- the more significant P-values (large `neg.log10.p`) are associated with small standard deviation parameters (because the signal to noise ratio is larger).

We will create a linked plot that shows details of the simulations for each parameter combination. First we prototype the details plots by examining a subset:

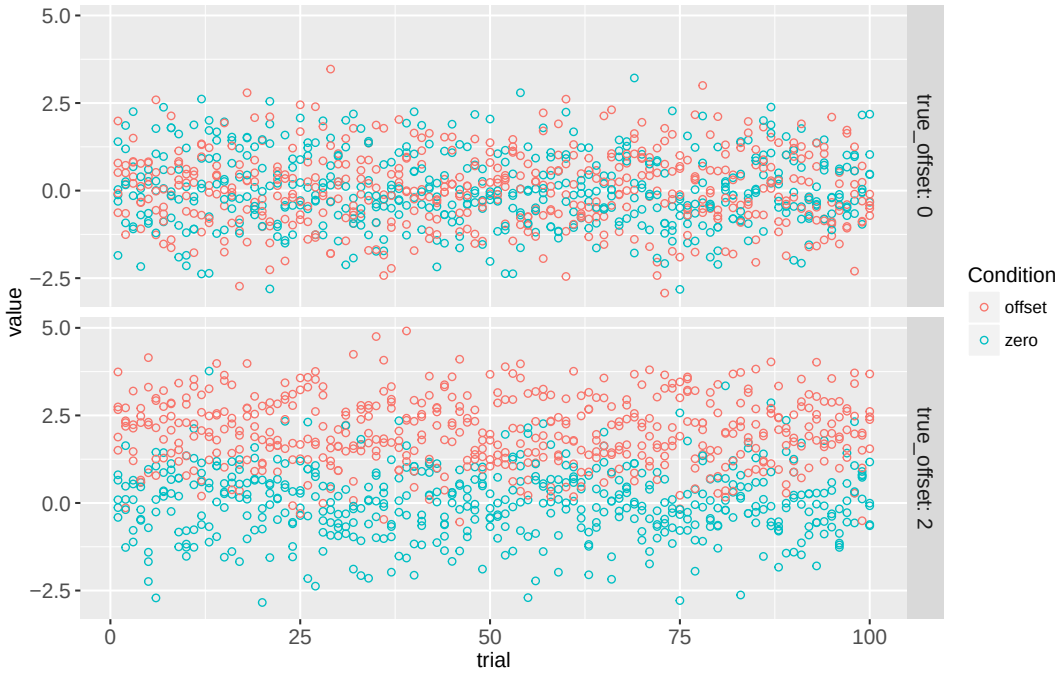
```
some <- function(DT)DT[true_offset %in% c(0,2) & sd == 1]
gg.true.tiles+
  geom_rect(aes(
    xmin=true_offset-width, xmax=true_offset+width,
    ymin=sd-height, ymax=sd+height),
    color="green",
    fill=NA,
```

```
      data=some(sim_true_tiles))
```



The figure above shows two rectangles emphasized with a green outline, for which we create a facetted non-interactive plot using the code below.

```
(gg.some.values <- ggplot()+
  facet_grid(true_offset ~ ., labeller=label_both)+
  geom_point(aes(
    trial, value, color=Condition),
    data=some(sim_dt)))
```



The figure above shows a panel for each of the two parameter combinations emphasized in the previous plot. The X axis represents `trial`, which ranges from 1 to 100, each one with different random values simulated using the same assumptions (`true_offset=0` or 2). The T-test is used to determine if there is any difference in means, between the two conditions.

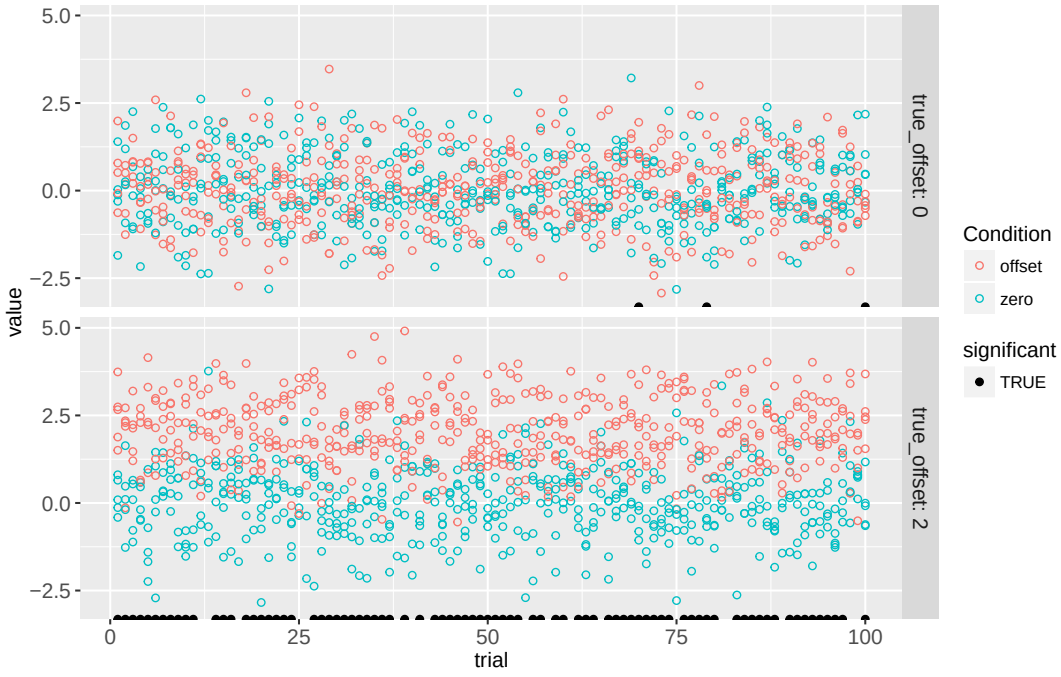
- When `true_offset=0`, there is no difference between the two conditions, so the T-test should have a small effect size, and a large P-value.
- When `true_offset=2`, there is a larger difference between the two conditions, so the T-test should have a larger effect size, and a small P-value.

The code below creates a new `significant` column which indicates if the test rejects the null hypothesis at the traditional threshold of 5%.

```
sim_p[, significant := p.value < 0.05]
```

The code below adds a new `geom_point()` to emphasize the trials which showed a significant difference at the 5% level.

```
only_significant <- sim_p[significant==TRUE]
gg.some.values+
  geom_point(aes(
    trial, -Inf, fill=significant),
    data=some(only_significant))+
  scale_fill_manual(values=c("TRUE"="black"))
```



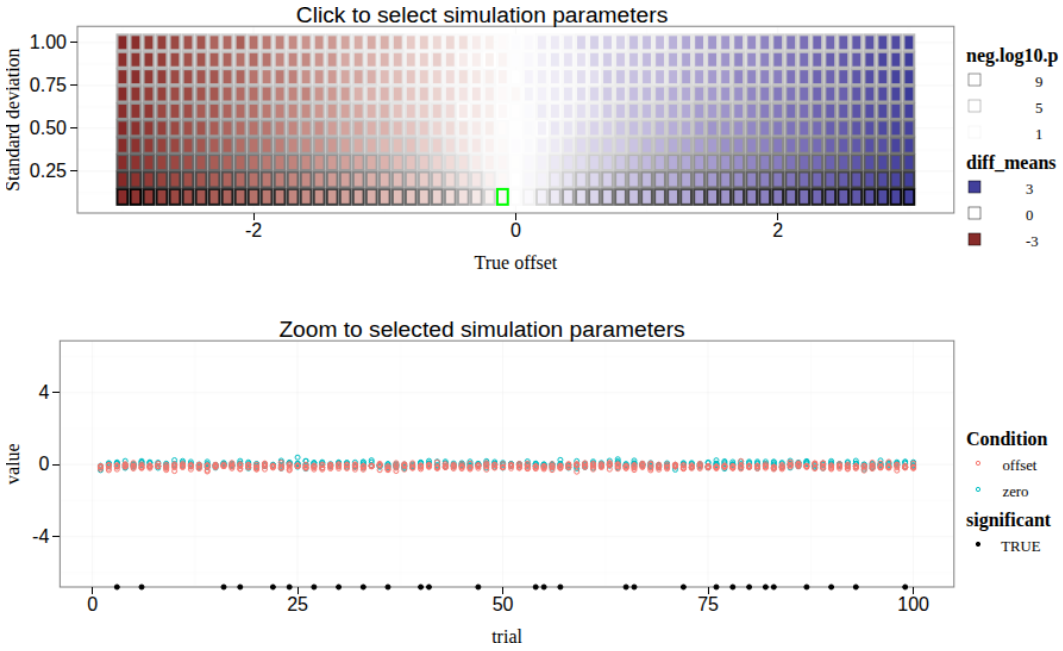
The figure above shows a black dot for each trial with a significant difference.

- For `true_offset=0`, we see that there are 3 trials with a significant difference, even though there is no difference in the true means in the simulation. This number of false positives is consistent with the 5% P-value threshold (type I error rate) that we used to define significance.
- For `true_offset=2`, we see that only 82 of the trials are significant, even though there is a true difference in means. The estimated power (true positive rate) is therefore 82% and the type II error rate (false negative rate) is therefore 18%.

Below we create a visualization which links the overview heat map to the details scatter plot, by replacing `facet_grid(true_offset ~ .)` in the code above, with `showSelected="true_tile"` in the code below.

```
(viz.parameters <- animint(
  tiles=gg.true.tiles+
    ggtitle("Click to select simulation parameters")+
    theme_animint(width=800, height=250, last_in_row=TRUE)+
    geom_rect(aes(
      xmin=true_offset-width, xmax=true_offset+width,
      ymin=sd-height, ymax=sd+height),
      fill="transparent",
      color="green",
      clickSelects="true_tile",
      data=sim_true_tiles),
  zoom=ggplot()+
    ggtitle("Zoom to selected simulation parameters")+
    theme_bw()+
    theme_animint(width=800, height=300)+
```

```
      geom_point(aes(
        trial, value, color=Condition),
        fill=NA,
        showSelected="true_tile",
        data=sim_dt)+
      geom_point(aes(
        trial, -Inf, fill=significant),
        showSelected="true_tile",
        data=only_significant)+
      scale_fill_manual(values=c("TRUE"="black"))))
```



In the visualization above, you can click on the top plot to select a simulation parameter combination. The 100 trials for the selected parameter combination are shown in the bottom plot.

19.6 Chapter summary and exercises

We have created several data visualizations using simulations to illustrate P-values. Since there were too many T-tests to show on the volcano plot, we instead used a heat map linked to a zoomed scatter plot. We also showed how to link a heat map of parameter combinations to a scatter plot which shows details of the corresponding values in the simulations.

Exercises:

- In `viz.volcano`, when clicking on one of the bottom `volcanoTiles`, we see only have of the space occupied in `volcanoZoom`. To fix this, and have all of the space occupied, go back to the definition of `round_neg.log10.p`, and use the `offset=0.5` argument in

`round_rel()`, which will shift the relative values by a certain offset.

- In `viz.volcano`, add `aes(tooltip)` to `volcanoTiles` to show how many points are in each heat map tile.
- Note how two `geom_tile()` were used in `viz.volcano`, and two `geom_rect()` were used in `viz.parameters`. The first geom uses color and fill to visualize the data, whereas the second geom uses `fill="transparent"` with `color="green"` for selection. Try a re-design with only one geom, which only uses `aes(fill)` and instead uses `color` as a geom parameter. What are the disadvantages of the approach using only one geom?
- In `viz.parameters`, add `geom_hline()` to emphasize the selected value of the true offset.
- In `viz.parameters$zoom`, add `geom_segment()` to represent the difference between means in each trial, using `aes(linetype=signficant)` to show which differences are significant at the traditional P-value threshold of `0.05`.
- In `viz.parameters$zoom`, add `geom_point()` to represent the mean in each trial and condition. Hint: you can either add two instances of `geom_point()`, or one `geom_point()` with a longer data table created via `melt(sim_p, measure.vars=measure(Condition, sep="mean_(.*)"))`.
- Add graphical elements to `gg.volcano` to emphasize the traditional P-value threshold of `0.05`: `geom_hline()` can show the threshold, and `geom_text()` can show how many tests fall above or below the threshold (use text like "1500 tests not significant").
- Add `aes(tooltip)` to `gg.true.tiles` to show values of `neg.log10.p` and `diff_means`.
- Add animation to `viz.parameters` so that we see a new parameter combination every second.
- Add `first` option to `viz.parameters` so that the first selection we see is the same as one of the parameter combinations shown in the static facetted plot.
- Combine `viz.volcano` with `viz.parameters` to create a new data visualization with four linked plots. Different kinds of interactions are possible: - In `volcanoZoom`, use `clickSelects="true_tile"` so that interactivity can be used to map volcano plot points back into the simulation parameter space. Add green dots with `showSelected="true_tile"` to `volcanoTiles` to show where the 100 trials for the selected parameter combination appear in volcano plot space.
- Create a new selection variable that is a unique combination of `true_tile` and `trial`, then use it for `clickSelects` in both `volcanoZoom` and `viz.parameters$zoom`, so that we can see a correspondence between a point in volcano space, and a trial in the parameter zoom details plot.

Next, the appendix explains some R programming idioms that are generally useful for interactive data visualization design.

A

Useful R programming idioms

This appendix describes several R programming idioms which are useful for creating animints.

A.1 Space saving facets

To emphasize the plotted data in facetted ggplots, eliminate the space between facets using the following idiom.

```
ggplot()+
  geom_point(aes(Petal.Width, Sepal.Width), iris)+
  theme_bw()+
  theme(panel.margin=grid::unit(0, "lines"))+
  facet_grid(. ~ Species)
```

There are three parts of this idiom:

- `panel.margin=0` eliminates space between panels.
- `theme_bw` activates a black and white theme (black panel borders and white panel backgrounds). This is necessary in order to see the boundaries between panels, since the ggplot default `theme_grey` uses grey panel backgrounds and no panel borders.
- `facet_*` creates a multi-panel ggplot.

Note that we use the grid unit `lines`, which equals the height of one line of text at the default size. This is the only grid unit which animint knows how to translate. It is not recommended to use other units such as `cm`.

A.2 List of data tables

The list of data tables idiom is very useful for creating interactive data visualizations of arbitrary complexity. The general form looks like

```
library(data.table)
outer.data.list <- list()
inner.data.list <- list()
for(outer in outer.vec){
  outer.dt <- computeOuter(outer)
```

```
  outer.data.list[[paste(outer)]] <- data.table(outer, outer.dt)
  for(inner in inner.vec){
    inner.dt <- computeInner(outer.dt, inner)
    inner.data.list[[paste(outer, inner)]] <-
      data.table(outer, inner, inner.dt)
  }
}
outer.data <- do.call(rbind, outer.data.list)
inner.data <- do.call(rbind, inner.data.list)
```

Some comments:

- The first part of the idiom involves initializing empty lists. Here there are two, `outer.data.list` and `inner.data.list`. However there can be as many as necessary.
- The second part of the idiom is a bunch of nested for loops that assign data tables to elements of those lists.
- Functions like `computeOuter` and `computeInner` can be used, or you can just do the computations directly inside the for loop.
- To ensure that your code will run as fast as possible, use matrix-vector or vector-scalar operations in the innermost for loop. If you only do scalar-scalar operations in your innermost for loop, then you can definitely improve the performance of your code by removing that for loop and re-writing the computation in terms of vector-scalar operations.
- The `paste` function is used to assign a `data.table` to a named list element. Although in principle one could use either `data.frame` or `data.table`, in practice `data.table` is often much faster during the last combination step.
- The last part of the idiom uses `do.call` with `rbind` to combine the data tables stored during the for loops.

A.3 addColumn then facet

This idiom is useful for creating multi-panel ggplots with aligned axes. First, define a function which takes as input a data table and one or more values which will be used to add factors to that data table.

```
addColumn <- function(df, time.period)data.frame(
  df, time.period=factor(time.period, c("1975", "1960-2010")))
animint(
  ggplot()+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    data=addColumn(WorldBank1975, "1975"))+
  geom_path(aes(
    x=life.expectancy, y=fertility.rate, color=region,
    group=country),
    data=addColumn(WorldBankBefore1975, "1975"))+
  geom_line(aes(
```

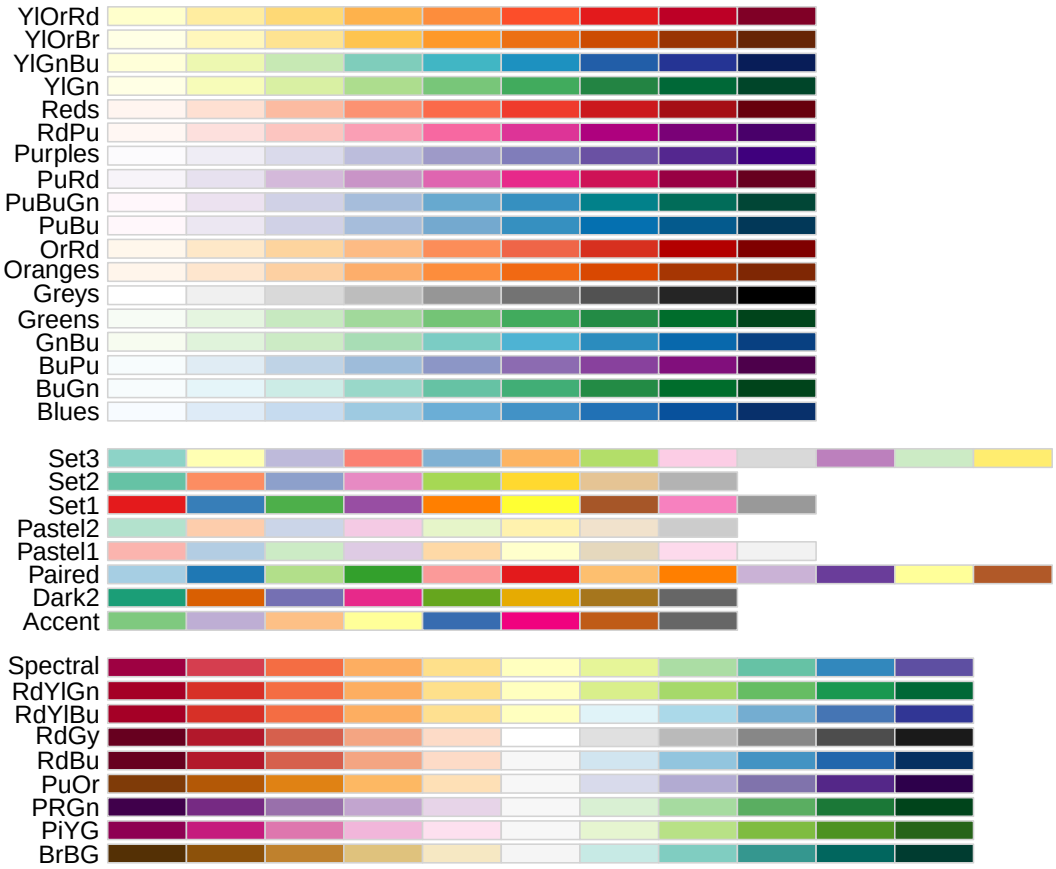
```
    x=year, y=fertility.rate, color=region, group=country),
    data=addColumn(WorldBank, "1960-2010"))+
  facet_grid(. ~ time.period, scales="free")+
  xlab(""))
```

Note that `scales="free"` and `xlab("")` are used since the x axes now have very different units (year and life expectancy).

A.4 Manual color legends

Color and fill legends in `ggplot2` can be manually specified via `scale_color_manual` and `scale_fill_manual`. Typically we will choose one of the ColorBrewer palettes:

```
RColorBrewer::display.brewer.all()
```



For example to get the R code for the Set1 palette, we can write

```
dput(RColorBrewer::brewer.pal(7, "Set1"))
```

```
c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
"#A65628")
```

We can then copy that R code from the terminal and paste it into our text editor

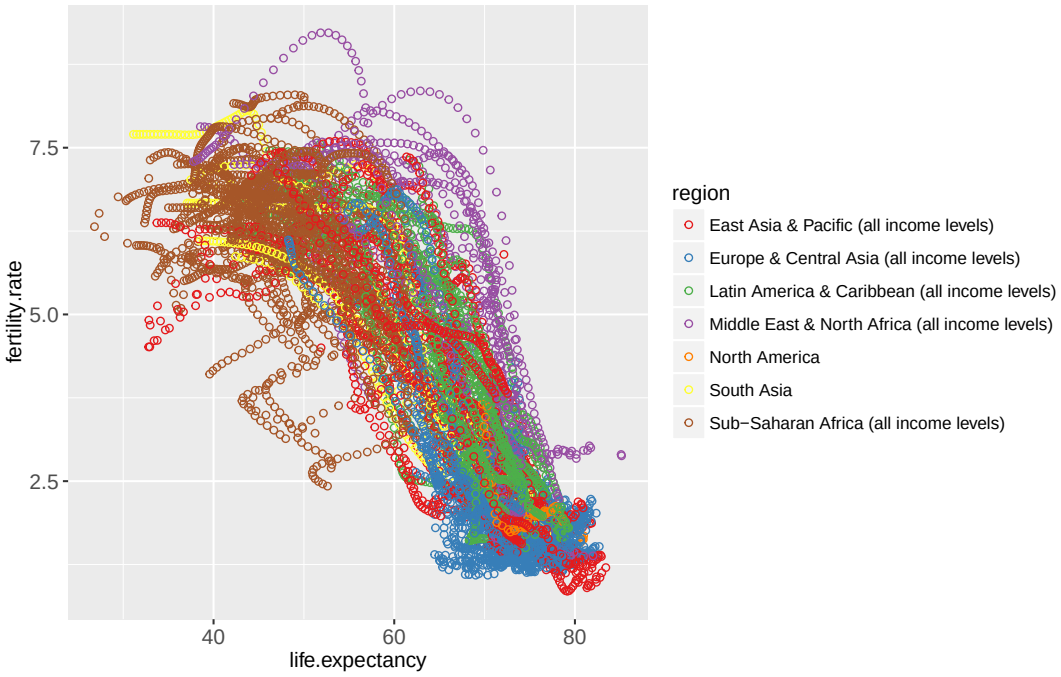
```
data(WorldBank, package="animint2")
region.colors <- c(
  "#E41A1C", "#377EB8", "#4DAF4A", "#984EA3", "#FF7F00", "#FFFF33",
  "#A65628")
names(region.colors) <- levels(WorldBank$region)
region.colors
```

```
      East Asia & Pacific (all income levels)
                                    "#E41A1C"
    Europe & Central Asia (all income levels)
                                    "#377EB8"
 Latin America & Caribbean (all income levels)
                                    "#4DAF4A"
Middle East & North Africa (all income levels)
                                    "#984EA3"
                                 North America
                                    "#FF7F00"
                                    South Asia
                                    "#FFFF33"
        Sub-Saharan Africa (all income levels)
                                    "#A65628"
```

Then we can use it with `scale_color_manual`

```
library(animint2)
ggplot()+
  scale_color_manual(values=region.colors)+
  geom_point(aes(
    x=life.expectancy, y=fertility.rate, color=region),
    data=WorldBank)
```

```
Warning: Removed 1490 rows containing missing values (geom_point).
```


7.5
5.0
2.5
fertility.rate
40
60
80
life.expectancy
region
East Asia & Pacific (all income levels)
Europe & Central Asia (all income levels)
Latin America & Caribbean (all income levels)
Middle East & North Africa (all income levels)
North America
South Asia
Sub−Saharan Africa (all income levels)