

# Guide R pour Motus

*Tara L. Crewe, Zoe Crysler, et Philip Taylor*

# Contents

<b>Utilisation du langage R appliqu� aux donn�es du r�seau de radiotlm�trie automatis�</b>	
<b>Motus</b>	<b>4</b>
<b>1 Introduction</b>	<b>5</b>
1.1 Ce que le pr�sent guide ne montre pas . . . . .	7
1.2 Connaissances pr�alables . . . . .	7
1.3 Ensemble de donn�es utilis�es comme exemples . . . . .	8
1.4 Remerciements . . . . .	8
<b>2 Chargement des logiciels R</b>	<b>10</b>
2.1 Traitement interne des donn�es . . . . .	11
<b>3 Acc�s aux donn�es de d�tection</b>	<b>13</b>
3.1 Types de bases de donn�es . . . . .	13
3.2 Chargement des logiciels R pertinents . . . . .	13
3.3 R�glage de l'environnement syst�me . . . . .	14
3.4 Importation de donn�es de d�tection . . . . .	14
3.5 Structure des donn�es . . . . .	17
3.6 V�rification de la version de la base de donn�es {checkVersion} . . . . .	19
3.7 Conversion d'une table SQLITE en une trame de donn�es plate . . . . .	19
3.8 Exportation d'une trame de donn�es «plate» sous la forme d'un fichier CSV ou RDS . . . . .	21
3.9 Convention de noms d'objets R . . . . .	22
<b>4 D�ploiements d'�metteurs et de r�cepteurs</b>	<b>23</b>
4.1 Chargement des logiciels R pertinents et r�glage de l'environnement de travail . . . . .	24
4.2 Chargement du fichier .motus . . . . .	24
4.3 D�ploiements d'�metteurs . . . . .	24
4.4 V�rification des m�tadonn�es sur les r�cepteurs . . . . .	30
<b>5 Nettoyage des donn�es</b>	<b>42</b>
5.1 Chargement des logiciels requis . . . . .	43
5.2 Chargement des donn�es de d�tection . . . . .	43
5.3 V�rifications pr�liminaires des donn�es . . . . .	44
5.4 Recherche et examen de d�etections ambigu�es . . . . .	50
5.5 V�rification de la validit� des s�quences d'au moins 2 d�etections . . . . .	63
5.6 Filtrage des donn�es . . . . .	63
<b>6 Exploration des donn�es avec le logiciel R de Motus</b>	<b>65</b>
6.1 Chargement des logiciels requis . . . . .	65
6.2 Chargement des donn�es . . . . .	65
6.3 Sommaires de donn�es . . . . .	66
6.4 Graphiques de donn�es . . . . .	70
6.5 Cartes de donn�es . . . . .	76

<b>A Appendix - alltags structure</b>	<b>83</b>
<b>B Annexe B - Dépannage</b>	<b>85</b>
B.1 Fermeture du logiciel motus . . . . .	85
B.2 Reprise du téléchargement de données . . . . .	85
B.3 Messages d'erreur courants et solutions . . . . .	85
<b>C Annexe C - Le logiciel R de Motus</b>	<b>88</b>
C.1 checkVersion . . . . .	88
C.2 sunRiseSet . . . . .	89
C.3 plotAllTagsCoord . . . . .	91
C.4 plotAllTagsSite . . . . .	92
C.5 plotDailySiteSum . . . . .	93
C.6 plotRouteMap . . . . .	94
C.7 plotSite . . . . .	95
C.8 plotSiteSig . . . . .	96
C.9 plotTagSig . . . . .	97
C.10 simSiteDet . . . . .	98
C.11 siteSum . . . . .	101
C.12 siteSumDaily . . . . .	102
C.13 siteTrans . . . . .	103
C.14 tagSum . . . . .	104
C.15 tagSumSite . . . . .	105
C.16 timeToSunriset . . . . .	106
<b>D Annexe D - Le logiciel motusClient - Fonctions de filtrage de données</b>	<b>109</b>
D.1 listRunsFilters . . . . .	109
D.2 Dépendances . . . . .	109
D.3 Exemple . . . . .	110
D.4 createRunsFilter . . . . .	110
D.5 getRunsFilters . . . . .	111
D.6 writeRunsFilter . . . . .	111

# Utilisation du langage R appliquée aux données du réseau de radiotélémétrie automatisée Motus



Le présent guide Web a pour but de montrer aux utilisateurs de Motus (<https://motus.org>) comment se servir du langage de programmation statistique R (<https://www.r-project.org/>) pour importer les données de détection de signaux d'émetteurs dans le cadre d'un projet – ou à partir d'un récepteur – particulier; nettoyer les données et supprimer les faux positifs; explorer les données de détection par différents moyens de visualisation et sous forme de sommaires; transformer les données (par exemple en déterminant l'intervalle de temps depuis le lever/coucher du soleil ou la déclinaison magnétique); et exécuter diverses procédures d'analyse. Nous espérons que le contenu sera utile. Si vous avez des suggestions d'exemples supplémentaires à formuler, n'hésitez pas à nous le faire savoir en envoyant un courriel à [motus@birdscanada.org](mailto:motus@birdscanada.org).

Version 1.0  
Janvier 2018

# **Chapter 1**

## **Introduction**



Le Système de surveillance faunique Motus ('Motus'; Taylor et al. 2017; <https://motus.org/?lang=fr>) est un réseau international de recherche collaborative composé de stations de radiotélémétrie automatisée qui assure le suivi des déplacements et du comportement d'animaux volants portant des émetteurs radio encodés numériquement. Motus a été élaboré à l'Université Acadia en 2012-2013. En 2014, une importante expansion de l'infrastructure a été rendue possible grâce à une subvention accordée par la Fondation canadienne pour l'innovation aux universités Western, de Guelph et Acadia. Depuis, le réseau Motus s'est étendu grâce à la collaboration entre des chercheurs indépendants et divers organismes (voir <https://motus.org/about/?lang=fr>). Il est maintenant géré en tant que programme d'Études d'Oiseaux Canada (<https://www.birdscanada.org/?lang=FR>) en partenariat avec l'Université Acadia.

Motus se distingue des autres réseaux de télémétrie automatisée en ce que l'ensemble des chercheurs dans une région donnée (les Amériques ou l'Europe par exemple) utilisent une fréquence radio commune. Ainsi, les animaux portant un émetteur peuvent être détectés par n'importe quelle station réceptrice du réseau, ce qui élargit grandement la portée spatiale des sujets de recherche possibles. De plus, les utilisateurs de Motus partagent aussi une infrastructure de données et un portail Web. L'entreposage et l'archivage centralisés de toutes les données recueillies dans l'ensemble du réseau permettent aux utilisateurs d'accéder à toutes les données sur leurs émetteurs dont les signaux sont captés par n'importe quelle station réceptrice, et les responsables des récepteurs ont accès à l'ensemble des données captées par ceux-ci.

En outre, avec l'existence d'une infrastructure de données partagée, les utilisateurs peuvent bénéficier des fonctions R mises au point spécifiquement pour les données Motus par n'importe quel utilisateur du système. Le logiciel R de Motus décrit dans ce guide est en constante évolution. Le présent guide a pour but d'aider les utilisateurs à apprendre les différentes fonctionnalités du logiciel et de les inciter à contribuer à leur développement. Il montre également comment d'autres logiciels R, comme ggplot, peuvent être utilisés pour explorer, visualiser, transformer et analyser les données Motus.

Le contenu de ce guide continuera d'évoluer et d'augmenter en fonction des besoins du réseau en matière d'analyse. Les personnes intéressées à ajouter des codes au logiciel R ou à enrichir ce guide peuvent faire parvenir leurs propositions à <http://www.birdscanada.org/research/motus/?lang=FR>.

Taylor, P. D., T. L. Crewe, S. A. Mackenzie, D. Lepage, Y. Aubry, Z. Crysler, G. Finney, C. M. Francis, C. G. Guglielmo, D. J. Hamilton, R. L. Holberton, P. H. Loring, G. W. Mitchell, D. R. Noriis, J. Paquet, R. A. Ronconi, J. Smetzer, P. A. Smith, L. J. Welch et B. K. Woodworth. 2017. The Motus Wildlife Tracking System: a collaborative .research network to enhance the understanding of wildlife movement. Avian Conservation and Ecology 12(1):8. <https://doi.org/10.5751/ACE-00953-120108>.

## 1.1 Ce que le présent guide ne montre pas

Le présent guide ne montre pas comment enregistrer des émetteurs radio auprès de Motus, gérer les émetteurs et les déploiements des stations réceptrices ou téléverser des données de détection brutes en vue de leur traitement. On peut accéder aux instructions pour ces opérations en cliquant sur l'option de menu Ressources au site Web de Motus: <https://motus.org/resources/?lang=fr>. N'oubliez pas d'enregistrer vos émetteurs **avant de les déployer** et de saisir les métadonnées sur les émetteurs et les stations en ligne sans tarder. Veuillez également consulter la Politique de collaboration de Motus et le Barème des frais d'enregistrement d'émetteurs à <https://motus.org/policy/?lang=fr>.

## 1.2 Connaissances préalables

Nous tenons pour acquis que vous possédez une connaissance de base du langage et du logiciel R. Quel que soit votre degré de connaissance à cet égard, nous vous recommandons fortement de vous familiariser avec l'ouvrage de Garrett Grolemund et d'Hadley Wickham intitulé «R for Data Science» (<http://r4ds.had.co.nz/>). Cet ouvrage montre comment importer, visualiser et résumer des données en langage R en utilisant la collection tidyverse de logiciels R (<https://www.tidyverse.org/>). De plus, il fournit un cadre d'une valeur

inestimable qui vous permettra d'organiser votre flux de travail afin de produire du code propre et reproductible (<http://r4ds.had.co.nz/workflow-projects.html>). Nous suivons l'exemple de ces auteurs en utilisant le cadre tidyverse tout au long du présent guide dans la mesure du possible.

## 1.3 Ensemble de données utilisées comme exemples

Tout au long du présent guide, nous utilisons un sous-ensemble de données recueillies dans le cadre du Programme de suivi des oiseaux de rivage de la baie James dans le but de montrer comment accéder aux données Motus, les gérer et les analyser en langage R. Nous vous recommandons de passer en revue le code présenté à titre d'exemple dans chaque chapitre avec ce sous-ensemble de données **avant** d'utiliser vos propres données, car vous aurez sans aucun doute besoin de modifier le code que nous fournissons pour traiter celles-ci le plus efficacement possible (chaque situation est différente).

Le Programme de suivi des oiseaux de rivage de la baie James a pour objet de surveiller et d'étudier les oiseaux de rivage qui font halte sur la côte de la baie James. Il est mené en collaboration par le ministère des Richesses naturelles et des Forêts de l'Ontario, Études d'Oiseaux Canada, l'Université Trent et le Service canadien de la faune (Environnement et Changement climatique Canada), conjointement avec une initiative de conservation de plus grande portée mobilisant les Premières Nations de la baie James et Nature Canada. Le Musée royal de l'Ontario était un partenaire associé au programme jusqu'en 2016. Le programme vise trois buts: 1) améliorer la capacité d'estimer les indices d'abondance et les tendances démographiques des espèces d'oiseaux de rivage qui font halte sur la côte ouest de la baie James; 2) étudier les profils des déplacements des oiseaux et leurs causes; et 3) déterminer l'importance relative des haltes migratoires et des habitats qu'elles renferment. Une fois réunis, les renseignements collectés contribueront à établir des mesures de conservation du Bécasseau maubèche et d'autres espèces d'oiseaux de rivage par des moyens de protection des habitats, comme la désignation au sein du Réseau de réserves pour les oiseaux de rivage dans l'hémisphère occidental (RRORHO). De plus amples renseignements peuvent être obtenus sur le site Web du Programme de suivi des oiseaux de rivage de la baie James à l'adresse <https://www.jamesbayshorebirdproject.com/> (en anglais) et sur sa page Facebook à <https://www.facebook.com/jamesbayshorebirdproject/> ou en communiquant avec le chef du programme:

Christian Friis Wildlife Biologist Canadian Wildlife Service, Environment and Climate Change Canada / Government of Canada [christian.friis@canada.ca](mailto:christian.friis@canada.ca) / Tel: 416.739.4908

Biogiste de la faune  
Service Canadien de la faune, Environnement et Changement Climatique Canada / Gouvernement du Canada  
[christian.friis@canada.ca](mailto:christian.friis@canada.ca) / Tél.: 416.739.4908

## 1.4 Remerciements

Une partie du présent guide est adaptée du contenu de l'entrepôt de données github de John Brzustowski se rapportant au logiciel R de Motus, à l'adresse <https://github.com/jbrzusto/motus>.

Motus était à l'origine le réseau SensorGnome, conçu par Philip Taylor et John Brzustowski à l'Université Acadia. La première expansion du réseau a été financée par le biais d'une subvention de la Fondation canadienne pour l'innovation accordée aux universités Western (Christopher Guglielmo, Ph. D.), de Guelph (Ryan Norris, Ph. D.) et Acadia (Philip Taylor, Ph. D.). L'élaboration de l'interface Web de Motus et du logiciel R et la production du guide qui l'accompagne ont été rendues possibles grâce à une subvention accordée à Études d'Oiseaux Canada par CANARIE (<https://www.canarie.ca/fr/>). Motus poursuit sa croissance en tant que programme d'Études d'Oiseaux Canada grâce à la collaboration d'un grand nombre de chercheurs indépendants, d'organismes et de particuliers. Une liste non exhaustive des partenaires et collaborateurs de Motus est présentée à <https://motus.org/data/partners.jsp>. Si votre organisme devrait figurer sur cette liste mais ne s'y trouve pas, veuillez envoyer un courriel à [motus@birdscanada.org](mailto:motus@birdscanada.org).

Beaucoup d'intervenants ont travaillé de concert pour fédérer la technologie Motus, l'interface Web et le logiciel R. L'équipe centrale du programme Motus est formée des personnes suivantes: John Brzustowski, Tara Crewe, Zoe Crysler, Jeremy Hussell, Catherine Jardine, Denis Lepage, Stuart Mackenzie, Paul Morrill et Philip Taylor.

## Chapter 2

# Chargement des logiciels R

Deux logiciels R ont été élaborés pour les utilisateurs de Motus:

1. motus: ce logiciel intègre des fonctions permettant de produire des sorties graphiques sommaires et de transformer (ajouter les heures du lever et du coucher du soleil) et d'analyser les données Motus.
2. motusClient: ce logiciel intègre des fonctions pour le téléchargement et la mise à jour des données de détection et des métadonnées sur le déploiement des émetteurs et des récepteurs à partir du serveur de Motus.

Les **utilisateurs** de Motus peuvent installer les versions stables les plus récentes des logiciels R à l'aide du code suivant. Comme c'est le cas pour tous les logiciels R, vous n'avez besoin d'installer les logiciels qu'une seule fois. Après l'installation, vous devez charger chaque logiciel (en utilisant la fonction library() ou require()) chaque fois que vous ouvrez une nouvelle session R.

Veuillez noter que l'utilisation de certaines fonctionnalités du logiciel devtools peut nécessiter des versions à jour de R et de RStudio. Pour éviter les erreurs, veuillez-vous assurer d'utiliser les versions les plus récentes de R et de RStudio et mettre à jour vos logiciels R en utilisant update.packages() dans la console R.

Pour mettre à jour vos logiciels existants:

```
update.packages()
```

Commencez par installer les logiciels requis s'ils ne sont pas déjà installés. Notez que le logiciel motusClient, qu'il faut utiliser pour accéder aux données de détection dans le serveur de Motus, fait partie du logiciel motus (c.-à-d. que vous ne devriez avoir qu'à exécuter le code pour installer le logiciel motus, et le logiciel motusClient sera chargé automatiquement). Le code permettant d'installer le logiciel motusClient de manière indépendante est fourni ci-dessous, mais vous ne devriez pas avoir besoin de l'exécuter.

```
install.packages("devtools")
library(devtools)

# Installation du logiciel motus pour le
# téléchargement, la manipulation, la visualisation
# et l'analyse des données
install_github("MotusWTS/motus")

# Installation du logiciel motusClient pour le
# téléchargement des données
install_github("MotusWTS/motusClient")

library(motus)
```

Si vous devez mettre à jour un logiciel motus ou motusClient existant, vous devez spécifier ‘force = TRUE’:

```
# Réinstallation forcée du logiciel motus quand des
# mises à jour sont requises
install_github("MotusWTS/motus", force = TRUE)

# Réinstallation forcée du logiciel motusClient
# quand des mises à jour sont requises
install_github("MotusWTS/motusClient", force = TRUE)

library(motus)
```

Tout au long du guide, nous utilisons tidyverse, une collection de logiciels R consacrés à la science des données - y compris tidyr, dplyr, ggplot2 et lubridate - pour la gestion et la manipulation de dates. Vous trouverez de plus amples renseignements sur tidyverse à <https://www.tidyverse.org/> ou en parcourant (ou, mieux encore, en lisant attentivement) l’ouvrage de Garrett Grolemund et d’Hadley Wickham intitulé «R for Data Science» (<http://r4ds.had.co.nz/>). Pour la cartographie, nous utilisons aussi les logiciels rworldmap et ggmap, qui peuvent être installés à partir de CRAN, de la manière suivante:

```
library(maps)

install.packages("tidyverse")
library(tidyverse)

install.packages("tidyrr")
library(tidyrr)

install.packages("rworldmap")
library(rworldmap)

install.packages("ggmap")
library(ggmap)
```

Nous installons également le logiciel plyr, mais nous ne le chargeons pas. Nous l’utilisons directement pour nous servir de la pratique fonction round\_any, mais s’il est chargé, il peut survenir des problèmes touchant les fonctions dplyr:

```
install.packages("plyr")
```

## 2.1 Traitement interne des données

Lorsqu’un animal portant un émetteur se déplace dans la zone de détection d’une station Motus, les signaux (ou salves d’impulsions) émis sont captés par l’antenne ou les antennes de la station et enregistrés par un récepteur. Ces données de détection brutes sont téléchargées depuis le récepteur puis téléchargées dans la base de données Motus instantanément via une connexion Internet ou sont téléchargées depuis le récepteur puis téléchargées dans la base de données Motus manuellement. En arrière-plan, les données de détection brutes sont lues et traitées au moyen de différentes fonctions pour produire le fichier de données de détection des émetteurs auquel les utilisateurs du logiciel R peuvent accéder (voir le chapitre 3). La plupart des utilisateurs n’auront pas besoin de recourir aux fonctions de traitement interne des données, mais une liste complète des fonctions intégrées dans le logiciel R du serveur de Motus est fournie sur GitHub (<https://github.com/jbrzusto/motusServer>). Pour voir le code sous-jacent à chaque fonction, on peut aller sur GitHub. On peut aussi taper ce qui suit dans la console R après avoir chargé le logiciel R; il suffit de remplacer «function.name» par le nom de la fonction R d’intérêt:

```
function.name()
```

Dans le chapitre suivant, nous examinerons et chargerons des données.

# Chapter 3

## Accès aux données de détection

Avant de télécharger vos données de détection, rendez-vous à la page Web relative aux problèmes de données pour vérifier qu'il n'y a pas de problème en suspens en ce qui touche les métadonnées.

### 3.1 Types de bases de données

Il existe deux types de bases de données de détection:

1. **base de données de récepteur:** comprend toutes les données de détection des signaux de n'importe quel émetteur enregistré captés par un récepteur particulier. Une base de données de récepteur porte un nom comme SG-1234BBBK5678.motus, ce nom correspondant au numéro de série du récepteur.
2. **base de données de projet:** comprend toutes les données de détection des signaux de vos émetteurs enregistrés captés dans l'ensemble du réseau Motus. Une base de données de projet porte un nom comme project-123.motus, le numéro correspondant à l'identifiant du projet Motus en question.

Ces deux types de bases de données correspondent au modèle de base du partage de données:

1. Vous obtenez toutes les données de détection des signaux de tous les émetteurs captés par *vos* récepteurs (c.-à-d. une base de données de récepteur pour chacun des récepteurs que vous avez déployés).
2. Vous obtenez toutes les données de détection des signaux de *vos* émetteurs captés par *n'importe quel* récepteur (c.-à-d. une base de données de projet pour chacun de vos projets Motus).

### 3.2 Chargement des logiciels R pertinents

Avant de commencer à travailler avec des données, il faut charger les logiciels requis pour les opérations expliquées dans le présent chapitre. Si vous n'avez pas encore installé ces logiciels (à partir de github et de CRAN), retournez au chapitre 2 pour le faire.

```
# logiciel «motus» requis depuis github  
require(motus)
```

### 3.3 Réglage de l'environnement système

Réglez le fuseau horaire de l'environnement système au temps universel coordonné (UTC), pour vous assurer de toujours travailler à cette échelle de temps. C'est une étape très importante qui devrait toujours faire partie de chaque séance de travail. Si vous ne faites pas ce réglage, deux problèmes peuvent se produire. Premièrement, les indications des heures dans la base de données de Motus sont en UTC, et si vous ne réglez pas l'environnement système à l'échelle de temps UTC, ces indications peuvent être modifiées par inadvertance pendant l'importation des données. Deuxièmement, si les signaux des émetteurs ont été captés dans différents fuseaux horaires, les indications des heures peuvent aussi être modifiées par inadvertance.

```
Sys.setenv(TZ = "GMT")
```

### 3.4 Importation de données de détection

Pour importer des données de détection de signaux d'émetteurs dans le cadre de votre projet ou à partir de votre récepteur, vous devez avoir un identifiant numérique de projet ou un numéro de série de récepteur.

Pour que le réseau Motus fonctionne correctement, il est important que les collaborateurs versent en temps opportun dans le serveur de Motus les données de détection provenant de leur(s) récepteur(s) et qu'ils assurent l'exactitude des métadonnées sur leurs émetteurs et récepteurs, et les tiennent à jour. Nous vous encourageons, après avoir téléchargé vos données depuis le serveur de Motus, à mettre à jour vos données de détection et vos métadonnées chaque fois que vous exécutez une analyse, car des collaborateurs peuvent ajouter des données de détection et des métadonnées n'importe quand et, si ces nouvelles données ne sont pas incluses, cela peut faire en sorte que vos données de détection soient incomplètes.

#### 3.4.1 Téléchargement des données d'un projet ou d'un récepteur pour la première fois

Lorsque vous téléchargez des données depuis le serveur de Motus pour la première fois, vous devez spécifier ‘new = TRUE’ et ‘update = TRUE’. À moins que le répertoire dans lequel vous voulez enregistrer vos données soit indiqué spécifiquement dans l'appel de fonction, les données seront téléchargées dans le répertoire de travail courant.

#### 3.4.2 Authentification de l'utilisateur

Notez que la première fois que vous appellerez une fonction en vous servant du logiciel R de Motus, vous devrez entrer votre nom d'utilisateur et votre mot de passe motus.org dans la console R pour certifier votre droit d'accès aux données d'un projet. Cela se produira seulement une fois par session de travail avec le logiciel R. Si vous n'avez pas de nom d'utilisateur ni de mot de passe Motus, vous pouvez vous inscrire à <https://motus.org/data/user/new?lang=fr>. La permission d'accéder aux données d'un projet vous sera accordée par le personnel de Motus ou le chercheur principal engagé dans le projet en question.

Tout au long du présent guide, nous utiliserons des exemples de données (voyez la section 1.3) attribuées au projet 176 (Programme de suivi des oiseaux de rivage de la baie James). Pour accéder à ces données, vous devrez ouvrir une session en entrant votre nom d'utilisateur et votre mot de passe («motus.sample») dans la console R lorsque la fonction tagme() vous y invitera (voyez la section 3.4.4). Cela ressemblera à ce qui suit:

```
> tagme(176, update = TRUE, dir = "./data/")
Please enter a value for login name at motus.org
==>
motus.sample
Please enter a value for password at motus.org
==>
motus.sample
Checking for new data in project 176
src: sqlite 3.19.3 [F:\Motus\Motus_RPackage\MotusRBoo
tbls: admInfo, allambigs, alltags, antDeps, batches, b
, recvs,
runs, runsFilters, species, tagAmbig, tagDeps, tags
> |
```

### 3.4.3 Fermeture de session

Une fois que vous aurez ouvert une session dans un compte utilisateur, vous ne pourrez pas accéder aux données d'un autre compte. Si vous avez besoin de fermer une session dans le compte courant pour accéder à d'autres données, vous pouvez exécuter le code ci-dessous.

```
motusLogout()
```

### 3.4.4 Téléchargement de données de détection

Nous pouvons commencer. Notez qu'aucun récepteur n'est enregistré dans le cadre du projet 176, de sorte que le deuxième appel de fonction ne permettra pas d'obtenir des données. Toutefois, vous pouvez remplacer le numéro de série du récepteur par un véritable numéro de série enregistré dans le cadre de votre projet ou d'un de vos projets si vous avez ouvert une session dans votre propre compte (c.-à-d. non pas avec le mot de passe «motus.sample»; voyez la section 3.4.3).

\*\*\*\*Toutes les données téléchargées seront enregistrées dans votre répertoire de travail, sauf indication contraire dans l'appel de fonction tagme tel qu'indiqué ci-après.\*\*

Veuillez noter que si vous téléchargez pour la première fois ('new = TRUE' dans l'appel de fonction tagme) un grand ensemble de données à partir du serveur de Motus, cela peut prendre beaucoup de temps, parfois quelques heures. Une fois que vous aurez effectué un premier téléchargement, le chargement d'un fichier .motus dans R avec la commande «`tagme(proj.num, update = TRUE)`» se fera presque instantanément. L'indication du déroulement du téléchargement devrait s'afficher à la console; si vous ne la voyez pas, essayez de faire défiler la page vers le bas pendant que la fonction tagme est en cours d'exécution.

Dans l'éventualité où votre connexion au serveur de Motus ferait défaut avant la fin du téléchargement (à cause d'une mauvaise connexion Internet par exemple), utilisez «`tagme(proj.num, update = TRUE)`» pour poursuivre le téléchargement à partir du point d'interruption, en veillant à indiquer le répertoire de réception des données si vous n'utilisez pas le répertoire de travail.

```
getwd() # Indiquez le répertoire de travail; utilisez la fonction setwd() pour le modifier.
proj.num <- 176 # Entrez 176 ou le numéro de votre projet.
```

```
sql.motus <- tagme(projRecv = proj.num, new = TRUE,
update = TRUE) # Pour la base de données du projet
sql.motus <- tagme(projRecv = "SG-123BBBBK1234", update = TRUE,
new = TRUE) # Pour la base de données du récepteur
```

Si vous ne voulez pas utiliser le répertoire de travail, spécifiez un répertoire à créer et ouvrez une base de données de détection locale en utilisant «dir =»:

```
sql.motus <- tagme(projRecv = proj.num, new = TRUE,
                     update = TRUE, dir = "./data/")
```

La fonction tagme() versera une copie de votre base de données de détection dans le répertoire de travail ou le répertoire que vous avez spécifié. Cette copie prendra la forme d'un fichier SQLite portant le suffixe .motus.

### 3.4.5 Ouverture et mise à jour d'une base de données de détection locale

Pour ouvrir et mettre à jour une base de données de détection qui existe déjà (qui a déjà été téléchargée):

```
sql.motus <- tagme(projRecv = proj.num, new = FALSE,
                     update = TRUE, dir = "./data/") # Utilisez «dir =» pour spécifier un répertoire.
```

Si vous travaillez hors ligne et que vous voulez charger une base de données déjà téléchargée sans vous connecter au serveur, utilisez:

```
sql.motus <- tagme(projRecv = proj.num, update = FALSE,
                     dir = "./data/")
```

### 3.4.6 Vérification de l'existence de nouvelles données

Pour vérifier si de nouvelles données sont disponibles sans télécharger ces données, vous pouvez utiliser la fonction tellme(). Vous verrez s'afficher une liste:

- **numHits**: nombre de nouvelles détections;
- **numBytes**: quantité approximative de données non compressées à transférer, en mégaoctets;
- **numRuns**: nombre de séquences de nouvelles détections, c'est-à-dire de séries de détections continues par une antenne particulière de signaux provenant d'un émetteur;
- **numBatches**: nombre de lots de nouvelles données;
- **numGPS** nombre d'enregistrements GPS de nouvelles données.

La fonction tellme () repose sur l'hypothèse qu'il existe déjà une copie locale de la base de données:

```
tellme(projRecv = proj.num) # Si db est dans le répertoire de travail
tellme(projRecv = proj.num, dir = "./data/") # Pour spécifier un autre répertoire
```

Pour vérifier combien de données sont disponibles pour un projet sans que vous ayez une copie locale de la base de données, utilisez le paramètre «new»:

```
tellme(projRecv = proj.num, new = TRUE)
```

### 3.4.7 Imposition d'une mise à jour/réimportation de métadonnées sur des émetteurs et des récepteurs

Les métadonnées sur les émetteurs et les récepteurs sont automatiquement fusionnées avec les données de détection d'émetteurs lorsque ces dernières sont téléchargées. Toutefois, si vous voulez imposer la réimportation des métadonnées lorsque vous mettez à jour une base de données, vous pouvez utiliser la commande suivante:

```
sql.motus <- tagme(projRecv = proj.num, forceMeta = TRUE)
```

### 3.4.8 Importation de l'ensemble des métadonnées sur les émetteurs et les récepteurs

Lorsque vous utilisez la fonction tagme() pour télécharger ou mettre à jour votre fichier .motus, vous obtenez les métadonnées sur:

1. tous les émetteurs enregistrés dans le cadre de votre projet dont des signaux ont été détectés;
2. les émetteurs enregistrés dans le cadre d'autres projets qui sont associés à des détections de signaux ambigus (voyez le chapitre 5) dans vos données;
3. les récepteurs qui ont détecté les signaux de vos émetteurs ainsi que les signaux ambigus.

Dans de nombreux cas, vous voudrez accéder à toutes les métadonnées sur l'ensemble des émetteurs et des récepteurs partout dans le réseau (par exemple pour déterminer combien de vos émetteurs déployés n'ont pas été repérés ou pour localiser les stations avec et sans détections). Vous pouvez utiliser la fonction metadata() pour ajouter la totalité des métadonnées Motus à votre fichier .motus enregistré. Cette fonction n'a besoin d'être exécutée qu'une seule fois, mais nous vous suggérons de réimporter les métadonnées occasionnellement pour vous assurer que vous avez l'information la plus récente et à jour.

L'exécution de la fonction metadata () de la manière suivante aura pour effet d'ajouter les métadonnées appropriées provenant de l'ensemble du réseau (tous les émetteurs et les récepteurs) aux tables «recvDeps» et «tagDeps» dans votre fichier .motus:

```
# Accéder aux métadonnées sur tous les émetteurs et
# les récepteurs dans le cadre de tous les projets
# dans le réseau.
metadata(sql.motus)
```

Vous pouvez aussi charger les métadonnées reliées à un ou des projets spécifiques, de la manière suivante:

```
# Accéder aux métadonnées sur les émetteurs et les
# récepteurs associés au projet 176.
metadata(sql.motus, projectIDs = 176)
```

```
# Accéder aux métadonnées sur les émetteurs et les
# récepteurs associés aux projets 176 et 1.
metadata(sql.motus, projectIDs = c(176, 1))
```

## 3.5 Structure des données

Chaque base de données de détection est stockée sous la forme d'un fichier SQLite («dplyr::src\_sqlite») portant le suffixe .motus. Nous avons choisi le format SQLite pour les raisons suivantes:

1. Il est souple; il autorise de nombreux formats de données.
2. Il est accessible à partir de nombreuses plates-formes logicielles (pas seulement à partir de R).
3. Il permet l'**ajonction**: la base de données peut être créée et mise à jour sur disque sans qu'il soit nécessaire de lire et de sauvegarder de nouveau tout le contenu de la base. Cela permet de gagner du temps et d'économiser de la mémoire lorsqu'on fait une recherche pour déterminer si de nouvelles données de détection sont disponibles pour un projet ou un récepteur donné.

Le fichier .motus contient une série de tables interdépendantes dans lesquelles les données sont stockées sous forme condensée pour économiser de la mémoire. Voici quelles sont ces tables:

1. antDeps: métadonnées sur les antennes, p. ex., hauteur, angle, type d'antenne;
2. batchRuns: métadonnées sur les runID et les batchID associés;
3. batches: données de détection pour un récepteur et un numéro de redémarrage de récepteur particuliers;

4. filters: métadonnées reliées à des filtres créés par l'utilisateur qui sont associés au récepteur en question;
5. gps: métadonnées reliées à la position GPS du récepteur;
6. hits: données de détection au niveau de chaque détection de signaux;
7. meta: métadonnées reliées au projet et au type de données (émetteurs par rapport aux récepteurs) qui sont inclus dans le fichier .motus;
8. projAmbig: métadonnées reliées aux projets pour lesquels il existe des détections ambiguës;
9. projBatch: métadonnées pour le nombre de détections contenues dans chaque lot;
10. projs: métadonnées reliées aux projets, p. ex., nom du projet, chercheur principal;
11. recvDeps: métadonnées reliées aux déploiements de récepteurs, p. ex., date du déploiement, lieu, caractéristiques du récepteur;
12. recvs: métadonnées reliées au numéro de série du récepteur et à l'identifiant d'appareil (deviceID) Motus associé;
13. runs: données de détection associées à une séquence de (détections continues d'un émetteur unique par un récepteur particulier);
14. runsFilters: liste de runIDs associés à des filtres créés par l'utilisateur et à des probabilités attribuées;
15. species: métadonnées reliées aux espèces, p. ex., identifiant unique, nom scientifique, nom commun;
16. tagAmbig: métadonnées reliées aux émetteurs ambigus, p. ex., ambigID et motusTagID associés;
17. tagDeps: métadonnées reliées à des déploiements d'émetteurs, p. ex., date du déploiement, lieu et espèce;
18. tags: métadonnées reliées à des émetteurs, p. ex., identifiant unique, caractéristiques de l'émetteur(p. ex., cadence d'émission).

Vous pouvez visualiser la liste des tables, et les variables contenues dans ces tables, en utilisant les codes suivants:

```
# Indiquer le lieu et le nom du fichier du projet.
file.name <- dbConnect(SQLite(), "./data/project-176.motus")

# Obtenir une liste des tables dans le fichier
# .motus spécifié ci-dessus.
dbListTables(file.name)

# Obtenir une liste des variables contenues dans la
# table «species» dans le fichier .motus.
dbListFields(file.name, "species")
```

En plus de ces tables, des tables «virtuelles» ou «vues» ont été créées à la suite de requêtes qui entraînent la fusion des données des différentes tables en une «vue» pratique contenant tous les champs dont vous pouvez avoir besoin. Les vues suivantes sont actuellement incluses dans chaque fichier .motus:

1. allambigs: liste en format long tous les motusTagID (jusqu'à 6) associés à chaque ambigID négatif;
2. alltags: fournit l'ensemble des données de détection pour tous les émetteurs et tous les émetteurs ambigus (en double) associés à votre projet. Les détections ambiguës sont répétées pour chaque motusTagID représenté par chaque ambigID.

Étant donné que le fichier est un fichier dplyr::src\_sqlite, toutes les fonctions dplyr peuvent être utilisées pour filtrer la base de données .motus et en présenter un sommaire sans avoir d'abord à enregistrer les données sous la forme d'un fichier *plat* (une trame de données bidimensionnelle typique). Le format SQL est très avantageux en présence d'un gros fichier – les requêtes effectuées dans ce format seront beaucoup plus rapides que celles effectuées sur une trame de données plate.

On peut accéder à chaque table et chaque vue dans le fichier .motus en utilisant la fonction `tbl()`:

```
# Obtenir la table des métadonnées sur les
# déploiements d'émetteurs pour le projet courant.
```

```
tbl.tagDeps <- tbl(sql.motus, "tagDeps")
```

La structure sous-jacente à ces tables est une liste de longueur 2:

```
str(tbl.tagDeps)
```

La première partie de la liste, «src», fournit de l'information sur la `SQLiteConnection`, entre autres sur l'emplacement de la base de données. La deuxième partie est une liste qui comprend la table sous-jacente. Dès lors, l'objet R «tagDeps» est une table *virtuelle* qui contient la structure de la base de données et l'information nécessaire pour la connexion avec les données sous-jacentes dans le fichier `.motus`. Tel qu'indiqué plus haut, cette méthode de stockage des données présente l'avantage d'économiser de la mémoire lorsqu'on accède à de très grosses bases de données, et les fonctions intégrées dans le logiciel `dplyr` peuvent être utilisées pour manipuler et résumer les tables avant de rassembler les résultats dans une trame de données «plate» typique.

Si vous voulez utiliser des fonctions familières pour accéder à des composantes de la trame de données sous-jacente, utilisez la fonction «`collect`». Par exemple, pour voir les noms des variables dans la table `tagDeps`:

```
tbl.tagDeps %>% collect() %>% names() # Présenter la liste des noms des variables dans la table.
```

```
## [1] "deployID"      "tagID"        "projectID"     "status"
## [5] "tsStart"       "tsEnd"         "deferSec"      "speciesID"
## [9] "bandNumber"    "markerNumber"  "markerType"    "latitude"
## [13] "longitude"     "elevation"     "comments"     "id"
## [17] "bi"            "tsStartCode"   "tsEndCode"    "fullID"
```

La table *virtuelle* «`alltags`» contient les données de détection ainsi que toutes les variables dont la plupart des utilisateurs pourront avoir besoin à partir des tables `.motus` sous-jacentes. Cette table est accessible elle aussi à l'aide de la fonction `dplyr` `tbl()`:

```
tbl.alltags <- tbl(sql.motus, "alltags") # Table virtuelle
```

La table suivante présente la liste des variables disponibles dans la vue «`alltags`». Vous trouverez une description complète de chaque champ dans l'annexe A.

## 3.6 Vérification de la version de la base de données {checkVersion}

Lorsque vous appelez la fonction `tagme` pour charger la base de données `SQLite`, un processus permet de vérifier que la version de votre base de données correspond à la version la plus récente du logiciel `motus` et de stocker la version dans une nouvelle table appelée `admInfo`. Au fil du temps, des changements sont apportés qui nécessitent l'ajout de nouvelles tables ou vues ou de nouveaux champs dans la base de données. Si vous n'avez pas la version correcte de la base de données, il se pourrait que certains des exemples contenus dans le présent guide ne fonctionnent pas. Cette fonction permet de vérifier que votre base de données a été actualisée à la version courante du logiciel `motus`. Si vous n'avez pas la version courante, reportez-vous au chapitre 2, qui fournit des instructions sur la mise à jour des logiciels `motus` et `motusClient`. Si le système affiche un avertissement, reportez-vous à l'annexe B.

```
checkVersion(sql.motus)
```

## 3.7 Conversion d'une table SQLITE en une trame de données plate

Pour convertir la vue «`alltags`» ou une autre table dans le fichier `.motus` en un format «plat» typique (c.-à-d. avec tous les enregistrements pour chaque champ entrés), utilisez les fonctions `collect()` et `as.data.frame()`.

La sortie pourra ensuite subir d'autres manipulations ou être utilisée pour générer un fichier RDS de vos données à des fins d'archivage ou d'exportation.

Nous suggérons le flux de travail suivant. Préparez un script qui effectue le téléchargement/la mise à jour de vos données, élimine les variables nécessaires et effectue un premier nettoyage s'il y a lieu, puis enregistrez les données résultantes sous la forme d'un fichier RDS. Nous suggérons d'utiliser le format RDS plutôt que le format CSV, car il conserve la structure sous-jacente des données (p. ex. les heures POSIX demeurent des heures POSIX). Toutefois, si vous voulez exporter vos données dans un autre programme, il pourrait être préférable d'utiliser le format CSV.

Créer un fichier plat avec l'ensemble des champs peut exiger beaucoup de mémoire et peut ralentir le logiciel R considérablement lorsqu'il faut traiter de grands ensembles de données. Pour certaines combinaisons d'ensembles de données et d'ordinateurs, il peut être impossible d'utiliser directement des trames de données dans R. Si cela se produit, c'est l'étape de votre flux de travail où vous devriez examiner attentivement l'information dont vous avez besoin dans votre ensemble de données (par exemple comment les données sont agrégées) et la simplifier. Il est toujours possible de revenir au script dont il est question ici pour créer un nouveau fichier RDS avec des variables différentes ou avec des données agrégées à une échelle différente.

Nous développons cette idée dans les sections qui suivent.

Créez une trame de données ...

```
df.alltags <- tbl.alltags %>% collect() %>% as.data.frame() # Pour tous les champs dans la trame de données
```

... et examinez brièvement le fichier résultant.

```
names(df.alltags) # Noms des champs
str(df.alltags) # Examinez la structure de vos champs de données.
head(df.alltags) # Examinez les 6 premières lignes de la trame de données.
summary(df.alltags) # Sommaire de chaque colonne dans la trame de données
```

Notez que le champ de l'estampille temporelle (ts) est numérique; il indique le nombre de secondes écoulé depuis le 1er janvier 1970. Nous recommandons que, lorsque vous transformez vos tables en trames de données plates, vous formatez l'estampille temporelle en utilisant le logiciel lubridate en fonction de cette estampille, par exemple:

```
df.alltags <- tbl.alltags %>%
  collect() %>%
  as.data.frame() %>% # Pour tous les champs dans la trame de données
  mutate(ts = as_datetime(ts, tz = "UTC", origin = "1970-01-01"))

# tz = "UTC" n'est pas nécessaire ici si vous avez réglé votre environnement système à l'échelle de temps
# ... mais c'est un rappel utile!
```

Si vous voulez charger seulement une partie de votre table virtuelle complète (p. ex., certains champs, certains émetteurs ou tous les émetteurs se rapportant à un projet ou à une espèce spécifique), vous pouvez utiliser des fonctions dplyr pour filtrer les données avant de les réunir en une trame de données. Voici des exemples:

1. Pour choisir certaines variables:

```
# Pour choisir un sous-ensemble de variables, dans
# le présent cas une liste unique d'identifiants
# d'émetteurs (tag ID) Motus à chaque récepteur et
# à chaque antenne.
df.alltagsSub <- select(tbl.alltags, recv, port, motusTagID) %>%
  distinct() %>% collect() %>% as.data.frame()
```

2. Pour choisir certains identifiants d'émetteurs (tag ID):

### 3.8. EXPORTATION D'UNE TRAME DE DONNÉES «PLATE» SOUS LA FORME D'UN FICHIER CSV OU RDS21

```
# Filtrez pour inclure seulement les identifiants
# d'émetteurs Motus 16011 et 23316.
df.alltagsSub <- filter(tbl.alltags, motusTagID %in%
  c(16011, 23316)) %>% collect() %>% as.data.frame() %>%
  mutate(ts = as_datetime(ts, tz = "UTC", origin = "1970-01-01"))
```

3. Pour choisir une espèce en particulier:

```
# Filtrez pour inclure seulement le Bécasseau
# maubèche (en utilisant la variable speciesID).
df.4670 <- filter(tbl.alltags, speciesID == 4670) %>%
  collect() %>% as.data.frame() %>% mutate(ts = as_datetime(ts,
  tz = "UTC", origin = "1970-01-01"))

# Filtrez pour inclure seulement le Bécasseau
# maubèche (en utilisant le nom commun anglais).
df.redKnot <- filter(tbl.alltags, speciesEN == "Red Knot") %>%
  collect() %>% as.data.frame() %>% mutate(ts = as_datetime(ts,
  tz = "UTC", origin = "1970-01-01"))
```

En utilisant la fonction dplyr(), vous pouvez aussi résumer votre table virtuelle avant de la convertir en un fichier plat. Par exemple, pour déterminer le nombre de détections différentes pour chaque émetteur à chaque récepteur:

```
df.detectSum <- tbl.alltags %>% group_by(motusTagID,
  recv) %>% tally() %>% collect() %>% as.data.frame()
```

Dans les chapitres suivants, nous vous montrerons d'autres façons de travailler avec vos données et de les résumer.

## 3.8 Exportation d'une trame de données «plate» sous la forme d'un fichier CSV ou RDS

Nous avons mentionné qu'un flux de travail approprié consiste à créer un script qui permet de traiter tous les problèmes de données et qui crée ensuite une trame de données (ou un espace de travail) qui peut être réutilisée. Si vous faites cela, vous pouvez rapidement amorcer une session d'analyse ou de visualisation à partir d'un point de départ connu (et cohérent). Nous utilisons un fichier RDS, qui conserve toutes les structures de données R associées, par exemple les estampilles temporelles.

```
# Enregistrez un fichier RDS.

saveRDS(df.alltags, "./data/df.alltags.RDS")

# ou enregistrez comme fichier CSV, qui ne conserve
# pas les estampilles temporelles, mais qui peut
# être lu facilement par d'autres programmes.

write_csv(df.alltags, "./data/df.alltags.CSV")
```

### 3.9 Convention de noms d'objets R

Dans le présent chapitre et tout au long du guide, nous nommons les objets R en fonction de leur structure et de la source des données que ces objets contiennent. Ainsi, les objets SQLite portent le préfixe «sql.», les objets de tables virtuelles, le préfixe «tbl.» et les objets de trames de données, le préfixe «df.». Le reste du nom de l'objet inclut le nom de la table .motus d'où proviennent les données. Dans le reste du guide, nous utiliserons et référencerons les formats de nommage ci-dessous; veuillez vous assurer de bien connaître ces formats avant de passer au prochain chapitre. Le code ci-dessous tient pour acquis que vous avez déjà téléchargé les données du projet 176 et que vous n'avez pas besoin de les mettre à jour; dans le cas contraire, reportez-vous à la section 3.4.4, qui montre comment effectuer le téléchargement initial:

```
# Objet R SQLite, lié au fichier .motus:
sql.motus <- tagme(176, update = TRUE, dir = "./data")

# Objet de la table virtuelle alltags dans le fichier .motus du projet 176:
tbl.alltags <- tbl(sql.motus, "alltags")
df.alltags <-tbl.alltags %>%
  collect() %>%
  as.data.frame() %>% # Objet de trame de données («plate») de la table alltags
  mutate(ts = as_datetime(ts, tz = "UTC", origin = "1970-01-01"))
```

Dans le prochain chapitre, il sera question de métadonnées manquantes.

## Chapter 4

# Déploiements d'émetteurs et de récepteurs

Avant de travailler avec vos données de détection, il convient d'abord de résumer et de visualiser les métadonnées relatives aux déploiements des émetteurs et des récepteurs enregistrés dans le cadre de votre projet. La production de sommaires, de graphiques et de cartes des données sur vos déploiements peut permettre de trouver des erreurs possibles dans les métadonnées sur les émetteurs et les récepteurs. De telles erreurs peuvent faire en sorte que les données de détection recueillies dans le cadre de votre projet soient incomplètes, tout comme les projets d'autres chercheurs dont des émetteurs ont été détectés par vos récepteurs.

Le présent chapitre est un complément de la page sur les problèmes de données du site Web de Motus, qui présente pour chaque projet une liste des problèmes touchant les métadonnées (valeurs manquantes ou aberrantes), problèmes qu'il faut accepter ou ne pas prendre en compte. Veuillez traiter toutes les erreurs associées à votre projet qui sont indiquées dans la page des problèmes de données **avant** d'importer vos données par l'intermédiaire de R. Les indications fournies ici ne permettent pas de faire une vérification complète des métadonnées sur les déploiements de vos émetteurs et récepteurs, mais elles aideront à déceler les erreurs que les interrogations automatiques sur la page des problèmes de données n'ont pas permis de trouver.

Nous utilisons l'ensemble de données du Programme de suivi des oiseaux de rivage de la baie James (projet 176) à titre d'exemple tout au long du présent chapitre (voyez la section 1.3). Pendant que vous parcourez le code pour examiner vos propres déploiements, **si vous trouvez des erreurs ou des omissions dans vos métadonnées, veuillez les corriger; pour ce faire, rendez-vous à la page <https://motus.org/>.** Cliquez sur l'option «Gérer les données» et dans le menu, choisissez «Gérer vos émetteurs» pour corriger les métadonnées sur les déploiements d'émetteurs ou «Gérer vos récepteurs» pour corriger les métadonnées sur les récepteurs. Il est important de corriger en ligne les erreurs dans les métadonnées; ainsi, les erreurs sont corrigées à la source et les données corrigées sont archivées dans le serveur de Motus, de sorte que tous les utilisateurs ont accès aux métadonnées correctes sur les émetteurs et les récepteurs. Les métadonnées corrigées en ligne seront automatiquement corrigées dans vos fichiers de données de détection. Si vous avez déjà téléchargé vos données de détection, vous pouvez mettre à jour le fichier existant afin d'inclure les nouvelles métadonnées et données de détection (voyez les sections 3.4.7 et 3.4.5).

## 4.1 Chargement des logiciels R pertinents et réglage de l'environnement de travail

Avant de commencer à travailler avec des données, il faut charger les logiciels requis pour les opérations expliquées dans le présent chapitre. Si vous n'avez pas encore installé ces logiciels (à partir de github et de CRAN), retournez au chapitre 2 pour le faire.

```
library(tidyverse)
library(tidyr)
library(motus)

# Réglez le fuseau horaire de l'environnement
# système au temps universel coordonné (UTC), pour
# vous assurer de toujours travailler à cette
# échelle de temps.
Sys.setenv(TZ = "GMT")
```

## 4.2 Chargement du fichier .motus

Dans le présent chapitre, il est tenu pour acquis que vous avez déjà téléchargé le fichier .motus. Si ce n'est pas le cas, retournez au chapitre 3, qui présente la marche à suivre pour ce faire. Pour mettre à jour le fichier existant et le charger dans R, utilisez la fonction tagme(); pour ce faire, il se peut que vous ayez à ouvrir une session tel qu'indiqué dans le chapitre précédent en entrant votre nom d'utilisateur **et** le mot de passe «motus.sample».

```
proj.num <- 176

sql.motus <- tagme(proj.num, update = TRUE, dir = "./data")
```

## 4.3 Déploiements d'émetteurs

Dans votre fichier .motus, lorsque vous utilisez la fonction tagme(), vous obtenez seulement les métadonnées sur tous les émetteurs dont les signaux ont été détectés dans le cadre de votre projet ainsi que les métadonnées sur les émetteurs ambigus associés dont les signaux ont été détectés dans le cadre d'autres projets et les métadonnées sur les récepteurs des stations où des signaux de vos émetteurs ont été détectés. Ici, nous:

1. téléchargerons les métadonnées complètes sur les émetteurs pour votre projet seulement;
2. déterminerons combien d'émetteurs sont enregistrés dans le cadre de votre projet;
3. déterminerons combien de ces émetteurs enregistrés ont été déployés;
4. localiserons les émetteurs déployés;
5. vérifierons si les métadonnées sur les déploiements d'émetteurs sont complètes et exactes.

Voyons maintenant chacune de ces opérations dans l'ordre.

### 4.3.1 Téléchargement des métadonnées complètes sur les émetteurs pour votre projet

Si les métadonnées sont incomplètes ou s'il manque des enregistrements d'émetteurs, il se peut qu'il manque des données de détection. Il faut donc vérifier si tous les émetteurs enregistrés pour votre projet sont pris en compte et pas seulement ceux dont des signaux ont été détectés. Pour ce faire, nous utiliserons la fonction `metadata()` pour le projet 176, dont il est question plus en détail dans la section 3.4.8.

```
metadata(sql.motus, projectIDs = proj.num)
```

### 4.3.2 Détermination du nombre d'émetteurs enregistrés

Maintenant que nous avons l'ensemble des métadonnées sur les émetteurs pour notre projet, nous pouvons vérifier le nombre d'émetteurs enregistrés en chargeant la table «tags» dans le fichier .motus. Cette table contient les métadonnées sur chaque émetteur enregistré, dont son identifiant unique et de l'information sur le fabricant, le modèle, les fréquences nominale et décalée, la cadence d'émission et la durée de l'impulsion. La table «tags» ne fournit pas d'information sur le déploiement des émetteurs. Nous choisissons les métadonnées propres au Programme de suivi des oiseaux de rivage de la baie James (projet 176) et ne tenons pas compte des métadonnées associées à de possibles émetteurs en double utilisés pour d'autres projets:

```
tbl.tags <- tbl(sql.motus, "tags")
df.tags <-tbl.tags %>% filter(projectID == proj.num) %>%
  collect() %>% as.data.frame()
```

Le nombre de lignes dans la base de données «df.tags» équivaut au nombre d'émetteurs dans l'ensemble de données du Programme de suivi des oiseaux de rivage de la baie James qui sont enregistrés (projet 176; 18 émetteurs):

```
nrow(df.tags) # Nombre d'émetteurs enregistrés dans la base de données
```

```
## [1] 18
```

Vous pouvez voir les identifiants des émetteurs (motusTagID):

```
unique(df.tags$tagID)
```

```
## [1] 16011 16035 16036 16037 16038 16039 16044 16047 16048 16052 17357
## [12] 19129 22867 22897 22902 22905 23316 23319
```

Si des émetteurs enregistrés sont manquants, suivez les instructions fournies à <https://motus.org/tag-registration/?lang=fr>.

### 4.3.3 Détermination du nombre d'émetteurs enregistrés qui ont été déployés

Pour vérifier quels émetteurs enregistrés ont été déployés, il faut utiliser la table de métadonnées «tagDeps» qui se trouve dans le fichier .motus. Cette table indique la date, l'heure et le lieu du déploiement ainsi que le nom d'espèce de l'animal portant l'émetteur. La base de données se rapporte au projet 176, et nous utilisons la fonction `anti_join` afin de déterminer pour quels émetteurs enregistrés il existe (ou n'existe pas) d'information sur le déploiement.

```
tbl.tagDeps <-tbl(sql.motus, "tagDeps")
df.tagDeps <-tbl.tagDeps %>%
  filter(projectID == proj.num) %>%
  collect() %>%
  as.data.frame() %>% # dans le format df, les dates peuvent être formatées avec la logique
  mutate(tsStart = as_datetime(tsStart, tz = "UTC", origin = "1970-01-01"),
```

```

tsEnd = as_datetime(tsEnd, tz = "UTC", origin = "1970-01-01"))

anti_join(df.tags, df.tagDeps, by = "tagID")

## [1] tagID      projectID    mfgID      type       codeSet
## [6] manufacturer model      lifeSpan    nomFreq    offsetFreq
## [11] bi          pulseLen
## <0 rows> (or 0-length row.names)

```

Dans les données du programme de la baie James, il n'y a pas de métadonnées sur les déploiements manquantes pour les émetteurs enregistrés, ce qui porte à croire que tous les émetteurs ont été déployés. Si vous avez des émetteurs non déployés dans vos propres fichiers, vérifiez si c'est bien le cas dans vos dossiers; sans les métadonnées sur les déploiements, les données de détection reliées aux émetteurs enregistrés mais non déployés seront manquantes dans votre base de données de détection.

#### 4.3.4 Détermination du nombre de déploiements par émetteur

Un émetteur peut être déployé plus d'une fois. Par exemple, un émetteur déjà déployé peut avoir été récupéré puis déployé de nouveau sur un autre animal. Quand un émetteur est déployé plus d'une fois, les données de détection doivent être considérées indépendamment pour chaque déploiement.

Tout au long du présent guide, nous utilisons «motusTagID» comme identifiant unique pour un déploiement. Toutefois, lorsqu'un émetteur est déployé plus d'une fois, l'identifiant «motusTagID» demeure inchangé d'un déploiement à l'autre. Il faut utiliser «tagDeployID» ou une combinaison de «motusTagID» et de «tagDeployID» pour distinguer les détections propres à chaque déploiement.

Vérifions s'il y a plus d'un déploiement pour des émetteurs dans les données du programme de la baie James (en réalité, il n'y a pas de cas de plus d'un déploiement) puis voyons comment créer une variable combinée tagID/deployID pour l'utiliser à la place de l'identifiant «motusTagID» dans l'éventualité où il y aurait plus d'un déploiement d'un émetteur dans vos propres données:

```

df.alltags %>%
  select(motusTagID, tagDeployID) %>%
  filter(!is.na(tagDeployID)) %>% # Enlever NA tagDeployID.
  distinct() %>%
  group_by(motusTagID) %>%
  mutate(n = n()) %>%
  filter(n > 1)

```

```

## # A tibble: 0 x 3
## # Groups:   motusTagID [0]
## # ... with 3 variables: motusTagID <int>, tagDeployID <int>, n <int>

```

Si vous avez plus d'un déploiement pour un émetteur, nous vous recommandons de créer une variable combinée «motusTagDepID», que vous utiliserez à la place de «motusTagID» pour définir un déploiement unique d'un émetteur. Tout au long du présent guide, vous remplacerez donc «motusTagID» par «motusTagDepID»:

```

df.alltags <- df.alltags %>% mutate(motusTagDepID = paste(motusTagID,
  tagDeployID, sep = "."))

```

```

# et faire de même pour les métadonnées sur les
# émetteurs

```

```

df.tagDeps <- df.tagDeps %>% mutate(motusTagDepID = paste(tagID,
  deployID, sep = "."))

```

### 4.3.5 Localisation des émetteurs déployés

La production d'une carte indiquant les lieux où vos émetteurs ont été déployés peut permettre de détecter d'éventuelles erreurs évidentes dans les indications des degrés de latitude ou de longitude que les interrogations effectuées en ligne au centre de messages sur les métadonnées n'auraient pas permis de saisir.

#### a. Chargement des fichiers de cartes de base

Chargez les fichiers de cartes de base à partir du logiciel rworldmap:

```
na.lakes <- map_data(map = "lakes")
na.lakes <- mutate(na.lakes, long = long - 360)

# Incluez l'ensemble des Amériques pour commencer.
na.map <- map_data(map = "world2")
na.map <- filter(na.map, region %in% c("Canada", "USA"))

na.map <- mutate(na.map, long = long - 360)

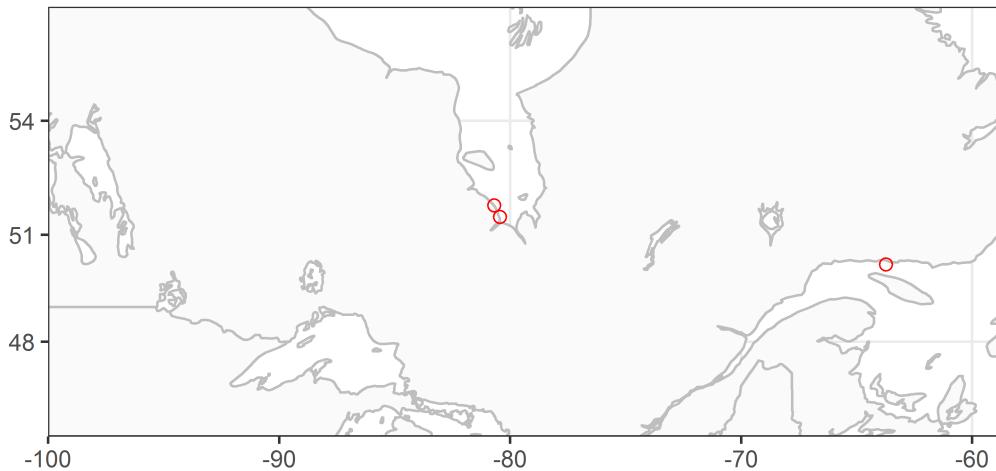
# Selon la position, vous voudrez peut-être
# localiser des émetteurs dans d'autres pays des
# Amériques: «Mexico», «lakes», «Belize», «Costa
# Rica», «Panama», «Guatemala», «Honduras»,
# «Nicaragua», «El Salvador», «Colombia»,
# «Venezuela», «Ecuador», «Peru», «Brazil»,
# «Guyana», «Suriname», «Bolivia», «French Guiana»,
# «Jamaica», «Cuba», «Haiti», «Dominican Republic»,
# «The Bahamas», «Turks and Caicos Islands»,
# «Puerto Rico», «British Virgin Islands»,
# «Montserrat», «Dominica», «Saint Lucia»,
# «Barbados», «Grenada», «Trinidad and Tobago»,
# «Chile», «Argentina», «Uruguay»
```

#### b. Pointage des positions des émetteurs déployés

Pointez les positions des émetteurs déployés à partir des données du programme de la baie James (projet 176):

```
# Fixez des limites à la carte en fonction des
# positions des détections, en vous assurant que la
# carte englobe ces positions.
xmin <- -100 #min(df.tagDeps$longitude, na.rm = TRUE) - 5
xmax <- max(df.tagDeps$longitude, na.rm = TRUE) + 5
ymin <- min(df.tagDeps$latitude, na.rm = TRUE) - 5
ymax <- max(df.tagDeps$latitude, na.rm = TRUE) + 5

# Pointez en utilisant ggplot.
ggplot(na.lakes, aes(long, lat)) + geom_polygon(data = na.map,
  aes(long, lat, group = group), colour = "grey",
  fill = "grey98") + geom_polygon(aes(group = group),
  colour = "grey", fill = "white") + coord_map(projection = "mercator",
  xlim = c(xmin, xmax), ylim = c(ymin, ymax)) + xlab("") +
  ylab("") + theme_bw() + geom_point(data = filter(df.tagDeps,
  projectID == 176), aes(longitude, latitude), cex = 2,
  pch = 1, colour = "red")
```



S'il y a une erreur dans la position d'un émetteur déployé, veuillez la corriger en ligne à <https://motus.org/data/>.

#### 4.3.6 Vérification de la complétude et de l'exactitude des métadonnées sur les déploiements d'émetteurs

Les métadonnées sur les émetteurs requises sont les suivantes : date/heure du début du déploiement, date/heure de la fin du déploiement (s'il y a lieu), latitude, longitude et espèce. L'absence d'information - en particulier les dates et les heures et les latitude et longitude - peut influer sur la durée de vie estimée de l'émetteur en cause; par conséquent, il faudra se demander si tagFinder «cherchera» l'émetteur au(x) moment(s) approprié(s). Cela peut aussi accroître le risque d'ambiguités causées par l'existence d'émetteurs en double dans le réseau.

##### a. Vérifier les gammes de valeurs des métadonnées

Tout d'abord, il faut utiliser le sommaire (df.tagDeps) pour avoir une idée de la gamme des valeurs de chaque variable et pour vérifier s'il manque des valeurs (NA) ou s'il y a des valeurs aberrantes pour une ou des variables. Nous présentons ci-dessous le résumé d'un sous-ensemble des variables dans la base de données df.tagDeps. Il faut répondre à plusieurs questions: est-ce que la plage des valeurs des dates du début et de la fin du déploiement est raisonnable pour vos déploiements? y a-t-il des erreurs évidentes dans les dates et les heures des déploiements? est-ce que la gamme des latitudes et longitudes est raisonnable? est-ce que les valeurs identifiant les espèces sont correctes?

```
df.tagDeps %>% select(tagID, projectID, tsStart, tsEnd,
  speciesID, latitude, longitude) %>% summary()
```

```
##      tagID      projectID      tsStart
##  Min.   :16011   Min.   :176   Min.   :2015-08-02 11:40:00
##  1st Qu.:16038   1st Qu.:176   1st Qu.:2015-08-13 15:25:00
##  Median :16050   Median :176   Median :2015-09-10 17:50:30
##  Mean   :18616   Mean   :176   Mean   :2016-01-24 12:49:36
##  3rd Qu.:22890   3rd Qu.:176   3rd Qu.:2016-09-25 15:34:15
##  Max.   :23319   Max.   :176   Max.   :2016-10-15 16:00:00
##      tsEnd                  speciesID      latitude
##  Min.   :2015-12-17 11:40:00   Min.   :4180   Min.   :50.19
##  1st Qu.:2015-12-28 15:25:00   1st Qu.:4670   1st Qu.:50.52
```

```

## Median :2016-03-10 17:50:30 Median :4690 Median :51.48
## Mean   :2016-07-28 18:09:36 Mean   :4674 Mean   :51.18
## 3rd Qu.:2017-06-06 09:53:45 3rd Qu.:4690 3rd Qu.:51.48
## Max.   :2017-06-26 16:00:00 Max.   :4820 Max.   :51.80
## longitude
## Min.   :-80.69
## 1st Qu.:-80.45
## Median :-80.45
## Mean   :-75.85
## 3rd Qu.:-67.92
## Max.   :-63.75

```

Il ne manque pas de dates de début (tsStart), et les dates de début des déploiements vont de 2015 à 2016, ce qui est raisonnable pour ce projet.

Les identifiants d'espèces sont numériques et n'ont pas de signification particulière si l'on ne peut pas leur attribuer chacun un nom d'espèce, ce que nous ferons ci-après. Dans le cas présent, il n'y a pas de valeurs manquantes.

#### b. Vérifier que les identifiants d'espèces concordent avec vos données

La table ‘species’ dans le fichier .motus associe chaque identifiant numérique d'espèce à un nom commun anglais, un nom commun français et un nom scientifique. Nous chargeons cette table et l'intégrons comme sous-ensemble de la suite d'identifiants numériques dans les métadonnées sur les émetteurs:

```

# Générez la liste des identifiants d'espèces dans
# les métadonnées du projet 176.
sp.list <- unique(df.tagDeps$speciesID)

# Métadonnées sur les espèces
tbl.species <- tbl(sql.motus, "species")
tbl.species %>% filter(id %in% sp.list) %>% collect() %>%
  as.data.frame()

```

	id	english	french
## 1	4180	Semipalmated Plover	Pluvier semipalmé
## 2	4670	Red Knot	Bécasseau maubèche
## 3	4680	Sanderling	Bécasseau sanderling
## 4	4690	Semipalmated Sandpiper	Bécasseau semipalmé
## 5	4760	White-rumped Sandpiper	Bécasseau à croupion blanc
## 6	4780	Pectoral Sandpiper	Bécasseau à poitrine cendrée
## 7	4820	Dunlin	Bécasseau variable
		scientific group	sort
## 1	Charadrius semipalmatus	BIRDS	NA
## 2	Calidris canutus	BIRDS	NA
## 3	Calidris alba	BIRDS	NA
## 4	Calidris pusilla	BIRDS	NA
## 5	Calidris fuscicollis	BIRDS	NA
## 6	Calidris melanotos	BIRDS	NA
## 7	Calidris alpina	BIRDS	NA

Cette opération génère la liste de toutes les espèces incluses dans les métadonnées sur les déploiements d'émetteurs pour le projet. Si des noms d'espèces sont inappropriés, cela s'explique probablement par une erreur de saisie de données au moment de l'attribution d'un déploiement à une espèce. Vous pouvez chercher les enregistrements dans vos métadonnées sur les émetteurs en utilisant le code suivant; ensuite, vous utilisez la variable deployID associée à l'entrée ou aux entrées pour trouver et mettre à jour en ligne l'enregistrement du déploiement dans vos métadonnées sur le projet:

```
filter(df.tagDeps, speciesID == 4780)

##   deployID tagID projectID status          tsStart          tsEnd
## 1    10517  22867      176    <NA> 2016-09-06 15:35:00 2017-05-18 15:35:00
##   deferSec speciesID bandNumber markerNumber markerType latitude longitude
## 1       NA      4780        <NA>     2641-20877 metal band 51.79861 -80.69139
##   elevation
## 1       NA
##
## 1 Sex:F, Age:HY, Bill28, Tarsus:26.2, Wing Chord:123, Wing Flat:129, Mass:57.7, Flag: (FEW)7P6, Blood
##   id bi tsStartCode tsEndCode           fullID
## 1 NA NA      1L        3L SampleData#272.1:5.3@166.38(M.22867)
##   motusTagDepID
## 1    22867.10517
```

Rappelez-vous que les corrections des métadonnées doivent être apportées en ligne.

## 4.4 Vérification des métadonnées sur les récepteurs

Il existe deux sources de métadonnées sur les récepteurs dans les données de détection Motus: les récepteurs enregistrés dans le cadre de votre projet et les récepteurs enregistrés dans le cadre de projets d'autres chercheurs. Vous pouvez accéder aux métadonnées sur tous les récepteurs du réseau, car les données négatives (par exemple, mon émetteur *n'a pas été* détecté à la station x même si cette station était active) sont souvent aussi importantes que les données positives. Cela vous permet aussi de connaître les positions où vos émetteurs ont été détectés par rapport à la répartition de l'ensemble des récepteurs partout dans le réseau Motus.

Les erreurs ou les omissions touchant les métadonnées sur les récepteurs que vous détectez dans votre fichier .motus peuvent être corrigées seulement pour les récepteurs enregistrés dans le cadre de votre propre projet.

Les utilisateurs de Motus sont encouragés à entrer des métadonnées sur les récepteurs complètes et exactes pour le bénéfice de l'ensemble des utilisateurs du réseau. Si vous prévoyez d'avoir besoin d'information spécifique sur le déploiement de récepteurs ou d'antennes à des stations déployées par d'autres chercheurs, vous pouvez utiliser le forum de Motus (<https://motus.org/discussion/>) pour demander que ces derniers enregistrent l'information sur le déploiement dont vous avez besoin. Vous devez indiquer avec précision l'information qui vous intéresse et indiquer également quand et où dans le réseau vos émetteurs seront déployés et éventuellement détectés. Ici, nous:

1. téléchargerons les métadonnées complètes sur les récepteurs dans l'ensemble du réseau;
2. déterminerons combien de récepteurs sont enregistrés dans le cadre de votre projet;
3. déterminerons à quel(s) moment(s) les récepteurs enregistrés dans le cadre de votre projet ont été déployés;
4. localiserons les récepteurs déployés dans tout le réseau et dans le cadre de votre projet;
5. vérifierons si les métadonnées sur les déploiements de récepteurs sont complètes et exactes.

### 4.4.1 Téléchargement des métadonnées sur tous les récepteurs dans le réseau

Plus loin dans le présent chapitre, nous produirons la carte indiquant la position de tous les récepteurs dans le réseau. Nous chargerons donc maintenant les métadonnées relatives à tous les projets plutôt que seulement celles propres au projet 176 comme nous l'avons fait précédemment. La fonction `metadata()` est décrite plus en détail dans la section 3.4.8.

```
metadata(sql.motus)
```

#### 4.4.2 Détermination du nombre de récepteurs déployés dans le cadre de votre projet

Pour savoir quels récepteurs déployés sont enregistrés dans le cadre de votre projet (s'il y en a), importez les données sur les déploiements et faites-en un sous-ensemble de données et un sommaire:

```
tbl.recvDeps <- tbl(sql.motus, "recvDeps")
df.projRecvs <- tbl.recvDeps %>% filter(projectID ==
  proj.num) %>% collect() %>% as.data.frame() %>%
  mutate(tsStart = as_datetime(tsStart, tz = "UTC",
    origin = "1970-01-01"), tsEnd = as_datetime(tsEnd,
    tz = "UTC", origin = "1970-01-01"))

summary(df.projRecvs)

##      deployID      serno      receiverType      deviceID
##  Min.   :1134  Length:29      Length:29      Min.   : 74.0
##  1st Qu.:2289  Class :character  Class :character  1st Qu.:138.5
##  Median :3101  Mode  :character  Mode  :character  Median :280.0
##  Mean   :3007
##  3rd Qu.:4048
##  Max.   :4222
##                               NA's   :2
##
##      macAddress      status      name
##  Length:29      Length:29      Length:29
##  Class :character  Class :character  Class :character
##  Mode  :character  Mode  :character  Mode  :character
##
##      siteName      fixtureType      latitude      longitude
##  Length:29      Length:29      Min.   :51.15  Min.   :-80.80
##  Class :character  Class :character  1st Qu.:51.29  1st Qu.:-80.57
##  Mode  :character  Mode  :character  Median :51.48  Median :-80.45
##                               Mean   :51.51  Mean   :-80.35
##                               3rd Qu.:51.66  3rd Qu.:-80.12
##                               Max.   :51.88  Max.   :-79.69
##                               NA's   :4     NA's   :4
##
##      isMobile      tsStart
##  Min.   :0.0000  Min.   :2014-07-12 00:00:00
##  1st Qu.:0.0000  1st Qu.:2015-05-25 00:00:00
##  Median :0.0000  Median :2016-05-18 00:00:00
##  Mean   :0.1379  Mean   :2016-02-20 17:19:14
##  3rd Qu.:0.0000  3rd Qu.:2017-05-16 15:55:00
##  Max.   :1.0000  Max.   :2017-08-20 23:30:00
##
##      tsEnd      projectID      elevation
##  Min.   :2014-11-06 00:00:00  Min.   :176  Min.   :-7
##  1st Qu.:2015-09-09 12:00:00  1st Qu.:176  1st Qu.:-4
##  Median :2015-11-03 00:00:00  Median :176  Median :-4
##  Mean   :2016-02-22 05:38:15  Mean   :176  Mean   : 3
##  3rd Qu.:2016-11-29 12:00:00  3rd Qu.:176  3rd Qu.: 0
##  Max.   :2017-08-20 23:30:00  Max.   :176  Max.   :30
```

```
##  NA's     :6          NA's     :24
```

Il y a 29 récepteurs déployés d'enregistrés dans le cadre du projet 176. Les valeurs de latitude et de longitude sont manquantes dans quatre cas et les valeurs des dates de fin sont manquantes dans six cas, ce qui porte à croire que les récepteurs en cause sont encore déployés.

L'opération suivante permet de conserver seulement les variables qui nous intéressent (en enlevant celles dont nous n'avons pas besoin) et structure les enregistrements restants par identifiant de récepteur, par latitude et par date de début:

```
df.projRecvs %>% mutate(dateStart = date(tsStart)) %>%
  select(-serno, -fixtureType, -macAddress, -tsStart,
         -tsEnd, -elevation, -projectID, -status, -receiverType,
         -siteName) %>% arrange(deviceID, latitude,
                                    dateStart)
```

	deployID	deviceID		name	latitude	longitude	isMobile
## 1	3100	74		Washkaugou	51.1540	-79.8144	0
## 2	2291	75		North Bluff	51.4839	-80.4500	0
## 3	3102	75		North Bluff	51.4839	-80.4501	0
## 4	4051	75		North Bluff	51.4839	-80.4501	0
## 5	4221	75		North Bluff	51.4839	-80.4501	0
## 6	3103	78		Piskwamish	51.6579	-80.5678	0
## 7	4050	78		Piskwamish	51.6580	-80.5679	0
## 8	3101	199		Netitishi	51.2912	-80.1167	0
## 9	4052	199		Netitishi	51.2913	-80.1167	0
## 10	4222	261		North Bluff	51.4839	-80.4501	0
## 11	1140	261		North Bluff	51.4840	-80.4500	0
## 12	1134	280		Longridge	51.8230	-80.6911	0
## 13	2285	280		Longridge	51.8231	-80.6912	0
## 14	3097	280		Longridge	51.8244	-80.6909	0
## 15	4048	280	Halfway Point	51.8753	-80.7973	0	
## 16	1135	285	Netitishi	51.2913	-80.1167	0	
## 17	2289	285	Netitishi	51.2913	-80.1168	0	
## 18	2290	324	Washiskougau Creek	51.1542	-79.8145	0	
## 19	1136	324	Washiskougau Creek	51.1545	-79.8137	0	
## 20	2286	349	Piskwamish	51.6578	-80.5676	0	
## 21	1137	349	Piskwamish	51.6582	-80.5669	0	
## 22	3814	527	LR mobile	NA	NA	NA	1
## 23	3813	528	NP mobile	NA	NA	NA	1
## 24	4001	528	BurntPointAerial	NA	NA	NA	1
## 25	4002	528	JamesBayAerial	NA	NA	NA	1
## 26	3099	609	East Point	51.2301	-79.7124	0	
## 27	3098	647	Halfway Point	51.8752	-80.7973	0	
## 28	4053	NA	East Point	51.3818	-79.6857	0	
## 29	4049	NA	Longridge	51.8246	-80.6909	0	
			dateStart				
## 1			2016-05-18				
## 2			2015-05-25				
## 3			2016-05-18				
## 4			2017-05-17				
## 5			2017-08-20				
## 6			2016-05-18				
## 7			2017-05-17				
## 8			2016-05-18				
## 9			2017-05-17				

```
## 10 2017-08-20
## 11 2014-08-01
## 12 2014-07-16
## 13 2015-05-24
## 14 2016-05-17
## 15 2017-05-16
## 16 2014-07-12
## 17 2015-05-25
## 18 2015-05-25
## 19 2014-07-12
## 20 2015-05-24
## 21 2014-07-16
## 22 2015-07-07
## 23 2015-07-06
## 24 2016-07-19
## 25 2016-08-09
## 26 2016-05-18
## 27 2016-05-17
## 28 2017-05-18
## 29 2017-05-16
```

Le nombre de récepteurs déployés indiqué dans les métadonnées devrait correspondre au nombre réellement déployé sur le terrain.

Si l'on examine les valeurs dans la colonne ‘isMobile’ qui correspondent aux quatre récepteurs déployés pour lesquels les valeurs de latitude et longitude sont manquantes, il est évident qu'il s'agit de récepteurs mobiles qui n'ont pas une position fixe (c.-à-d. qu'ils ont une valeur de 1). Étant donné que ces récepteurs sont mobiles, on ne s'attend pas à voir des coordonnées géographiques; l'indication demeure donc NA dans ces cas. Lorsque des coordonnées sont indiquées dans la table pour des récepteurs mobiles, celles-ci représentent le point de départ du déploiement.

#### 4.4.3 Détermination du moment du déploiement des récepteurs dans le cadre de votre projet

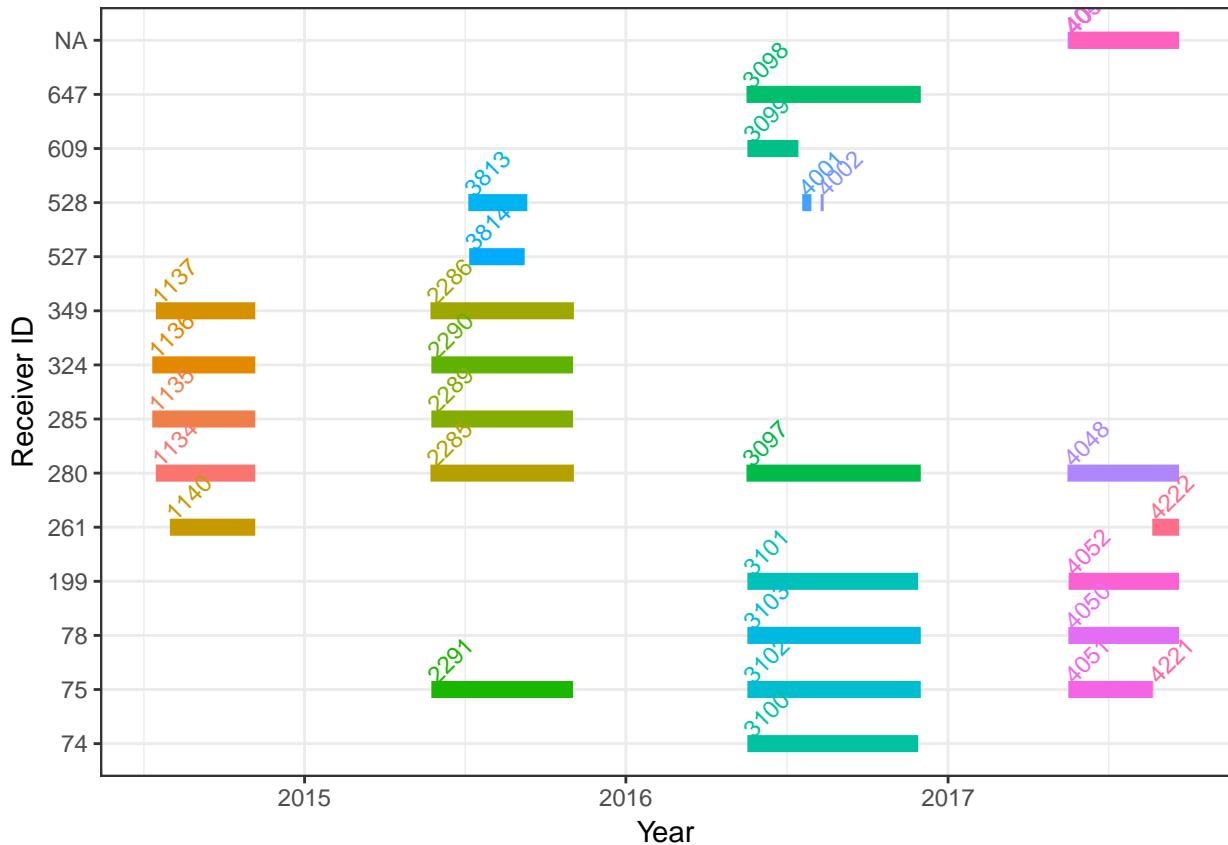
L'information sur le moment des déploiements peut être affichée sous forme de graphique. Les lignes horizontales dans le graphique ci-dessous correspondent à l'intervalle de temps pendant lequel a été déployé chaque récepteur (deviceID) enregistré dans le cadre du Programme de suivi des oiseaux de rivage de la baie James (projet 176). Dans le cas des deux récepteurs pour lesquels la date de fin est manquante, le code attribue une date de fin arbitraire correspondant à la date de fin la plus éloignée parmi celles des autres récepteurs plus un mois. Sans cette opération, l'information sur les déploiements sans dates de fin ne serait pas affichée. Les périodes de déploiement d'un récepteur donné ne devraient pas se chevaucher dans le temps:

```
# Mettez les données dans un format long pour
# simplifier le pointage(ou utilisez geom_segment).
```

```
df.projRecvs.long <- select(df.projRecvs, deviceID,
  deployID, tsStart, tsEnd) %>% tidyrr::gather(when,
  ts, c(tsStart, tsEnd)) %>% mutate(ts = if_else(is.na(ts),
  max(ts, na.rm = TRUE) + duration(1, "month"), ts)) # fake end date

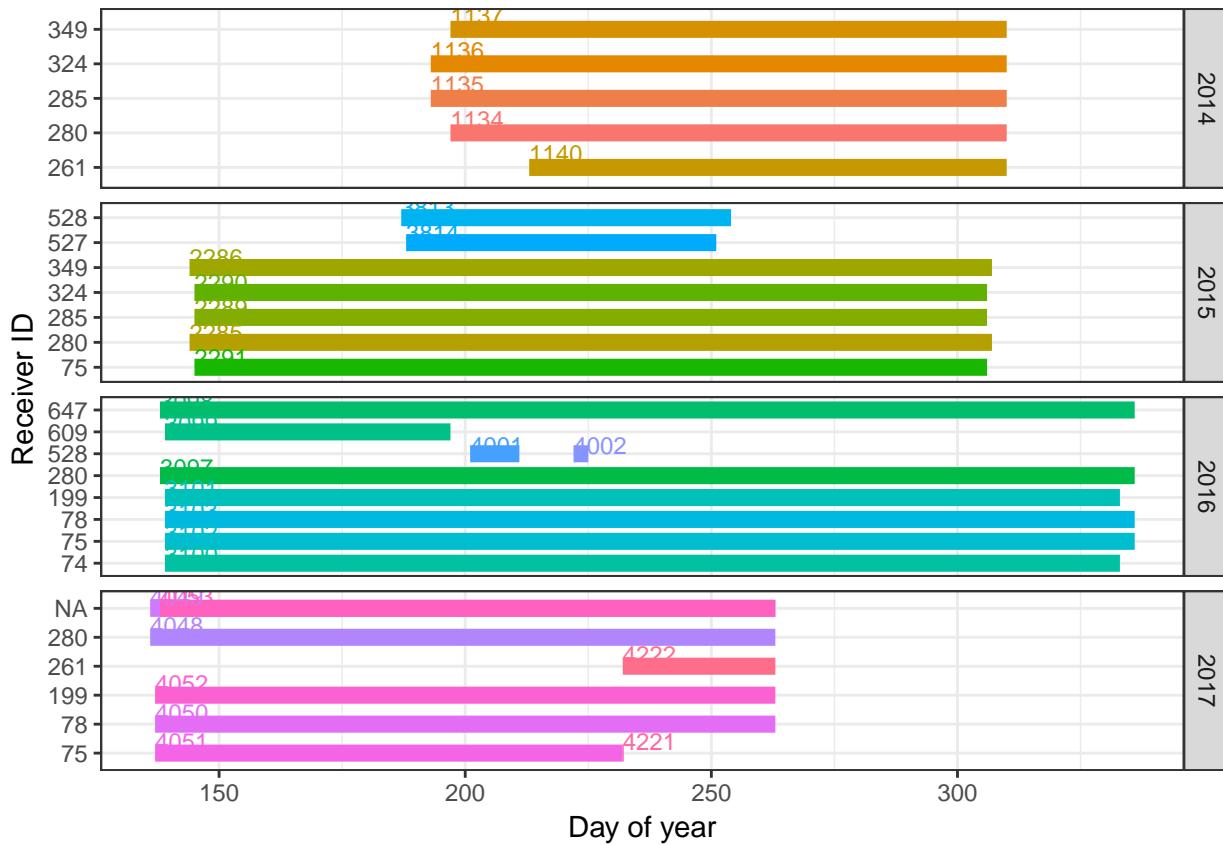
ggplot(df.projRecvs.long, aes(y = as.factor(deviceID),
  x = ts, colour = as.factor(deployID))) + geom_line(lwd = 3) +
  # Centrez plutôt sur la droite.
```

```
geom_text(data = filter(df.projRecvs.long, when == "tsStart"), aes(label = deployID), hjust = "left",
nudge_y = 0.2, size = 3, angle = 45) + theme_bw() +
ylab("Receiver ID") + xlab("Year") + theme(legend.position = "none")
```



Si vous voulez obtenir de l'information plus détaillée pour une année donnée (ou pour toutes les années), vous pouvez générer un sous-ensemble de données et pointer celles-ci de nouveau sur un graphique ou utiliser le jour de l'année sur l'axe des x et ensuite faire la synthèse par année (facet\_wrap).

```
ggplot(df.projRecvs.long, aes(y = as.factor(deviceID),
x = yday(ts), colour = as.factor(deployID))) +
geom_line(lwd = 3) +
# Centrez les étiquettes sur la gauche.
geom_text(data = filter(df.projRecvs.long, when == "tsStart"), aes(label = deployID), hjust = "left",
nudge_y = 0.4, size = 3) + theme_bw() + ylab("Receiver ID") +
xlab("Day of year") + theme(legend.position = "none") +
facet_grid(year(ts) ~ ., scales = "free")
```



#### 4.4.4 Localisation des récepteurs déployés

Les cartes assurent une meilleure représentation spatiale que les simples graphiques. Les opérations suivantes permettent de pointer la position des récepteurs Motus sur une carte de l'Amérique du Nord. Les récepteurs déployés dans le cadre du projet 176 (Programme de suivi des oiseaux de rivage de la baie James) sont indiqués en rouge.

##### a. Charger les métadonnées sur tous les récepteurs

```
df.recvDeps <- tbl.recvDeps %>% collect() %>% as.data.frame() %>%
  mutate(tsStart = as_datetime(tsStart, tz = "UTC",
    origin = "1970-01-01"), tsEnd = as_datetime(tsEnd,
    tz = "UTC", origin = "1970-01-01"))
```

##### b. Charger les fichiers des cartes de base

```
na.lakes <- map_data(map = "lakes")
na.lakes <- mutate(na.lakes, long = long - 360)

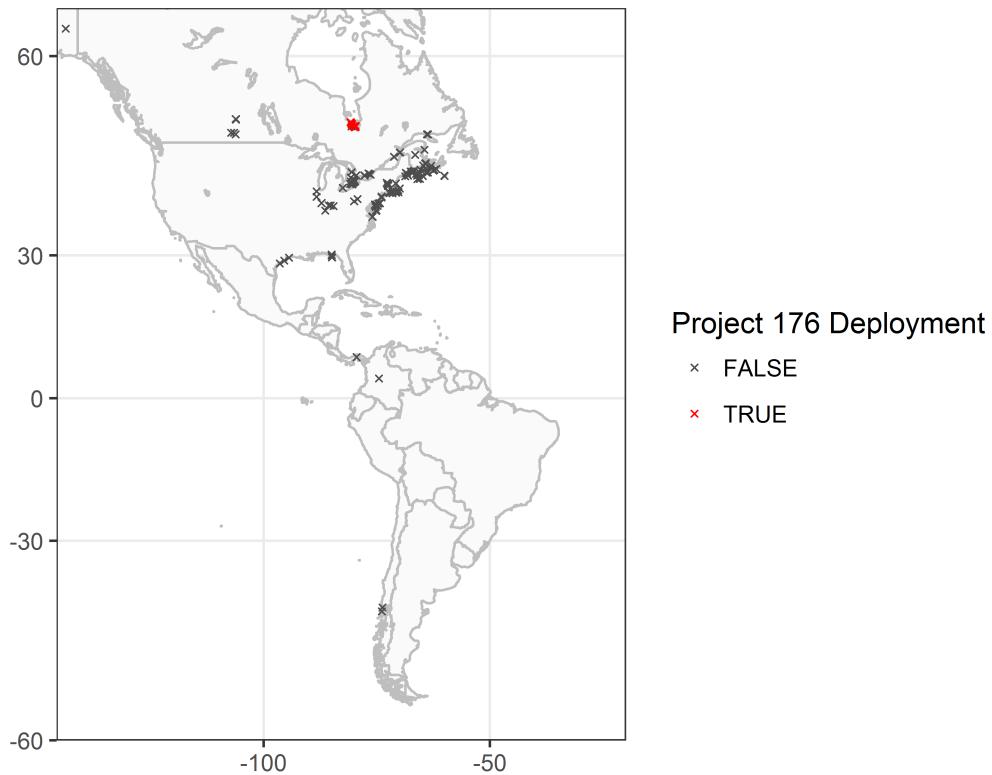
# Include all of the Americas to begin
na.map <- map_data(map = "world2")
na.map <- filter(na.map, region %in% c("Canada", "USA",
  "Mexico", "lakes", "Belize", "Costa Rica", "Panama",
  "Guatemala", "Honduras", "Nicaragua", "El Salvador",
  "Colombia", "Venezuela", "Ecuador", "Peru", "Brazil",
  "Guyana", "Suriname", "Bolivia", "French Guiana",
```

```
"Jamaica", "Cuba", "Haiti", "Dominican Republic",
"The Bahamas", "Turks and Caicos Islands", "Puerto Rico",
"British Virgin Islands", "Montserrat", "Dominica",
"Saint Lucia", "Barbados", "Grenada", "Trinidad and Tobago",
"Chile", "Argentina", "Uruguay", "Paraguay")) %>%
  mutate(long = long - 360)
```

- c. Pointer sur la carte la position des récepteurs dans les Amériques Carte montrant la position des récepteurs déployés dans l'ensemble du réseau (les «x» gris foncé) et des récepteurs déployés dans le cadre du Programme de suivi des oiseaux de rivage de la baie James (projet 176; les «x» rouges).

```
# Fixez des limites à la carte en fonction des
# positions des detections, en vous assurant que la
# carte englobe ces positions.
xmin <- min(df.recvDeps$longitude, na.rm = TRUE) -
  2
xmax <- -20 # Limitez aux Amériques (sauf quelques points en Europe).
ymin <- -60 #min(df.recvDeps$longitude, na.rm = TRUE) - 2
ymax <- max(df.recvDeps$latitude, na.rm = TRUE) + 2

# map
ggplot(na.lakes, aes(long, lat)) + geom_polygon(data = na.map,
  aes(long, lat, group = group), colour = "grey",
  fill = "grey98") + geom_polygon(aes(group = group),
  colour = "grey", fill = "white") + coord_map(projection = "mercator",
  xlim = c(xmin, xmax), ylim = c(ymin, ymax)) + xlab("") +
  ylab("") + theme_bw() + geom_point(data = df.recvDeps,
  aes(longitude, latitude, colour = as.logical(projectID ==
  176)), cex = 0.8, pch = 4) + scale_colour_manual(values = c("grey30",
  "red"), name = "Project 176 Deployment")
```



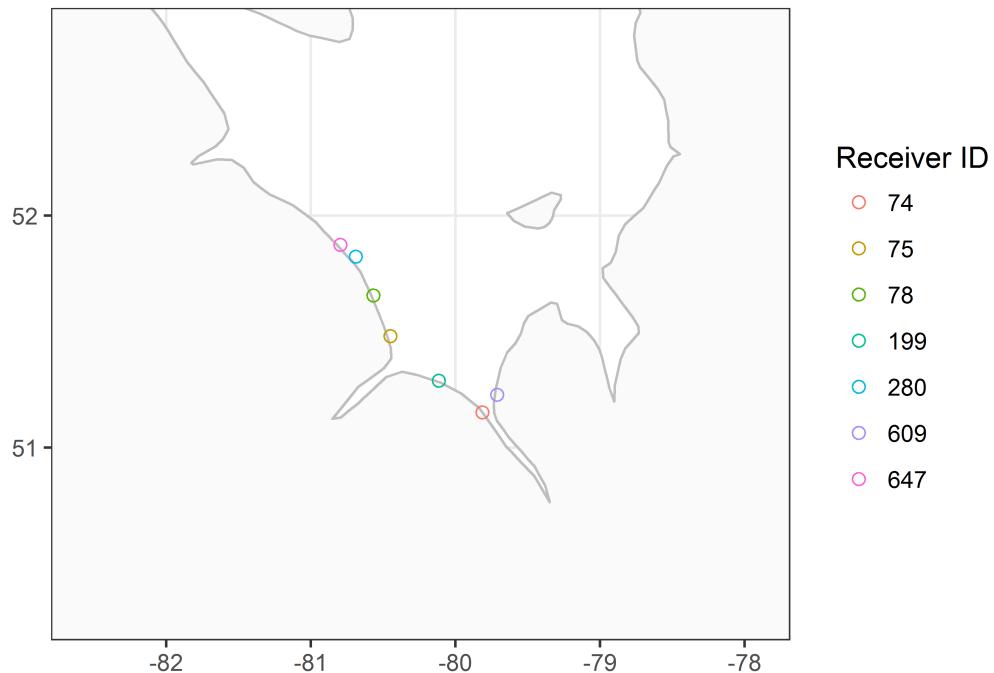
#### d. Pointer sur la carte la position seulement des récepteurs déployés dans le cadre d'un projet

Carte montrant la position des récepteurs déployés dans le cadre d'un projet. Les limites de la carte sur les axes des x (longitude) et des y (latitude) sont fixées à l'aide de la trame de données «df.projRecvs» qui a été créée précédemment. Seuls les récepteurs qui étaient actifs en 2016 figurent sur cette carte.

```
# Fixez des limites à la carte en fonction des positions des détections, en vous assurant que la carte
xmin <- min(df.projRecvs$longitude, na.rm = TRUE) - 2
xmax <- max(df.projRecvs$longitude, na.rm = TRUE) + 2
ymin <- min(df.projRecvs$latitude, na.rm = TRUE) - 1
ymax <- max(df.projRecvs$latitude, na.rm = TRUE) + 1

# map
ggplot(na.lakes, aes(long, lat)) +
  geom_polygon(data = na.map,
               aes(long, lat, group=group), colour = "grey", fill="grey98") +
  geom_polygon(aes(group = group), colour = "grey", fill = "white") +
  coord_map(projection="mercator", xlim = c(xmin, xmax), ylim = c(ymin, ymax)) +
#  coord_map(projection="mercator", xlim = c(xmin, xmax), ylim = c(50.154, 52.875)) +
  xlab("") + ylab("") +
  theme_bw() +
  geom_point(data = filter(df.projRecvs,
                           year(tsStart) == 2016,
                           !is.na(latitude)), # remove mobile receivers
```

```
aes(longitude, latitude, colour = as.factor(deviceID)), cex = 2, pch = 1) +
  scale_colour_discrete(name = "Receiver ID")
```



#### 4.4.5 Vérification de la complétude et de l'exactitude des métadonnées sur les déploiements des récepteurs

Les utilisateurs de Motus se préoccupent principalement de savoir si les métadonnées sur les déploiements des récepteurs qui ont détecté des signaux de leurs émetteurs sont complètes, parce que cela peut avoir un effet direct sur l'interprétation des données de ces détections. Par exemple, si les valeurs de latitude ou de longitude sont manquantes, on ne saura pas quelle était la position de l'émetteur dont les signaux ont été détectés. De même, si l'information sur le type et/ou l'orientation de l'antenne est manquante, cela peut empêcher de bien estimer l'orientation du déplacement de l'émetteur ou son cap au moment du départ de l'animal portant l'émetteur.

Dans de nombreux cas cependant, les métadonnées sur les déploiements des récepteurs *sans* information sur la détection des signaux des émetteurs peuvent quand même être utiles, par exemple pour estimer la probabilité de détection d'un animal qui passe dans la zone de détection d'une station réceptrice.

Dans la présente section, nous nous intéressons aux métadonnées sur les récepteurs enregistrés dans le cadre d'un projet particulier. Selon vos intérêts, les sommaires de métadonnées peuvent être appliqués à un plus grand groupe de récepteurs, par exemple à tous les récepteurs qui ont détecté des signaux d'émetteurs ou à tous les récepteurs dans une région déterminée (même s'ils n'ont pas détecté de signaux d'émetteurs).

##### a. Chargement des données sur les récepteurs et les antennes

```
# Métadonnées sur TOUTES les antennes Motus
# déployées; pour simplifier, conservez seulement
# les variables d'intérêt.
tbl.antDeps <- tbl(sql.motus, "antDeps")
```

```

df.antDeps <- tbl.antDeps %>% select(deployID, port,
                                         antennaType, bearing, heightMeters) %>% collect() %>%
                                         as.data.frame()

# Métadonnées sur les récepteurs déployés;
# conservez les variables d'intérêt.
df.recvDeps <- df.recvDeps %>% select(deployID, receiverType,
                                         deviceID, name, latitude, longitude, isMobile,
                                         tsStart, tsEnd, projectID, elevation)

df.stationDeps <- left_join(df.recvDeps, df.antDeps,
                             by = "deployID")

```

Faites de ces métadonnées un sous-ensemble des données sur les récepteurs enregistrés dans le cadre d'un projet:

```

df.stationDeps <- filter(df.stationDeps, projectID ==
                           proj.num)

```

### b. Examen de la gamme des valeurs des métadonnées

Utilisez la fonction `summary()` pour avoir une idée générale de la distribution des variables dans les données.

```
summary(df.stationDeps)
```

```

##      deployID    receiverType        deviceID       name
##  Min.   :1134    Length:91          Min.   : 74.0  Length:91
##  1st Qu.:2289    Class  :character  1st Qu.: 78.0  Class  :character
##  Median :3100    Mode   :character  Median :280.0  Mode   :character
##  Mean   :2996           Mean   :258.7
##  3rd Qu.:4049           3rd Qu.:324.0
##  Max.   :4222           Max.   :647.0
##                  NA's   :6
##      latitude     longitude       isMobile
##  Min.   :51.15    Min.   :-80.80    Min.   :0.00000
##  1st Qu.:51.29    1st Qu.:-80.57   1st Qu.:0.00000
##  Median :51.48    Median :-80.45   Median :0.00000
##  Mean   :51.52    Mean   :-80.36   Mean   :0.04396
##  3rd Qu.:51.66    3rd Qu.:-80.12   3rd Qu.:0.00000
##  Max.   :51.88    Max.   :-79.69   Max.   :1.00000
##  NA's   :4         NA's   :4
##      tsStart            tsEnd            projectID
##  Min.   :2014-07-12 00:00:00  Min.   :2014-11-06 00:00:00  Min.   :176
##  1st Qu.:2015-05-25 00:00:00  1st Qu.:2015-11-02 00:00:00  1st Qu.:176
##  Median :2016-05-18 00:00:00  Median :2016-07-15 00:00:00  Median :176
##  Mean   :2016-03-28 08:40:07  Mean   :2016-04-04 01:07:25  Mean   :176
##  3rd Qu.:2017-05-16 19:42:00  3rd Qu.:2016-12-01 00:00:00  3rd Qu.:176
##  Max.   :2017-08-20 23:30:00  Max.   :2017-08-20 23:30:00  Max.   :176
##                  NA's   :21
##      elevation      port      antennaType      bearing
##  Min.   :-7.000    Min.   :1.000  Length:91          Min.   : 0.0
##  1st Qu.:-4.000    1st Qu.:1.000  Class  :character  1st Qu.: 50.0
##  Median :-4.000    Median :2.000  Mode   :character  Median :145.0
##  Mean   : 4.235    Mean   :2.231           Mean   :152.6
##  3rd Qu.: 0.000    3rd Qu.:3.000           3rd Qu.:290.0
##  Max.   :30.000    Max.   :4.000           Max.   :357.5

```

```

##  NA's    :74
##  heightMeters
##  Min.   :4.800
##  1st Qu.:5.600
##  Median :5.800
##  Mean   :5.832
##  3rd Qu.:6.000
##  Max.   :6.200
##  NA's    :4

```

Les valeurs de latitude et de longitude sont manquantes dans quatre cas associés au déploiement de récepteurs mobiles, comme nous l'avons vu précédemment.

Les valeurs d'élévation sont manquantes dans 74 enregistrements sur 91, mais il s'agit d'un champ de données dont le contenu n'est pas obligatoire. On peut estimer l'élévation à partir d'autres sources ou directement dans R (par exemple, voyez <https://stackoverflow.com/questions/8973695/conversion-for-latitude-longitude-to-altitude-in-r>).

Sur 91 enregistrements, les valeurs de l'angle de relèvement d'antenne sont manquantes dans 18 enregistrements et les valeurs de la hauteur d'antenne dans 4 enregistrements. Formez un sous-ensemble de données avec les enregistrements dans lesquels les valeurs de l'angle de relèvement d'antenne sont manquantes pour voir si cela peut être corrigé:

```

filter(df.stationDeps, is.na(bearing)) %>% select(-elevation,
  -deviceID, -tsEnd)

```

	deployID	receiverType	name	latitude	longitude	isMobile
## 1	3097	SENSORGNAME	Longridge	51.8244	-80.6909	0
## 2	3098	SENSORGNAME	Halfway Point	51.8752	-80.7973	0
## 3	3099	SENSORGNAME	East Point	51.2301	-79.7124	0
## 4	3100	SENSORGNAME	Washkaugou	51.1540	-79.8144	0
## 5	3101	SENSORGNAME	Netitishi	51.2912	-80.1167	0
## 6	3102	SENSORGNAME	North Bluff	51.4839	-80.4501	0
## 7	3103	SENSORGNAME	Piskwamish	51.6579	-80.5678	0
## 8	3813	LOTEKSRX800	NP mobile	NA	NA	1
## 9	3814	LOTEKSRX800	LR mobile	NA	NA	1
## 10	4001	LOTEKSRX800	BurntPointAerial	NA	NA	1
## 11	4002	LOTEKSRX800	JamesBayAerial	NA	NA	1
## 12	4048	SENSORGNAME	Halfway Point	51.8753	-80.7973	0
## 13	4049	<NA>	Longridge	51.8246	-80.6909	0
## 14	4050	SENSORGNAME	Piskwamish	51.6580	-80.5679	0
## 15	4051	SENSORGNAME	North Bluff	51.4839	-80.4501	0
## 16	4052	SENSORGNAME	Netitishi	51.2913	-80.1167	0
## 17	4221	SENSORGNAME	North Bluff	51.4839	-80.4501	0
## 18	4222	SENSORGNAME	North Bluff	51.4839	-80.4501	0
	tsStart	projectID	port	antennaType	bearing	heightMeters
## 1	2016-05-17 00:00:00	176	4	omni-whip	NA	6.0
## 2	2016-05-17 00:00:00	176	4	omni-whip	NA	6.0
## 3	2016-05-18 00:00:00	176	4	omni-whip	NA	6.0
## 4	2016-05-18 00:00:00	176	4	omni-whip	NA	6.0
## 5	2016-05-18 00:00:00	176	4	omni-whip	NA	6.0
## 6	2016-05-18 00:00:00	176	4	omni-whip	NA	6.0
## 7	2016-05-18 00:00:00	176	4	omni-whip	NA	6.0
## 8	2015-07-06 00:00:00	176	1	yagi-3	NA	NA
## 9	2015-07-07 00:00:00	176	1	yagi-3	NA	NA
## 10	2016-07-19 08:00:00	176	1	yagi-3	NA	NA

## 11 2016-08-09 07:15:00	176	1	yagi-3	NA	NA
## 12 2017-05-16 15:55:00	176	3	omni-whip	NA	6.2
## 13 2017-05-16 19:42:00	176	4	omni-whip	NA	6.2
## 14 2017-05-17 15:19:00	176	4	omni-whip	NA	6.2
## 15 2017-05-17 15:00:00	176	4	omni-whip	NA	6.2
## 16 2017-05-17 22:47:00	176	4	omni-whip	NA	6.2
## 17 2017-08-20 23:30:00	176	4	omni-whip	NA	6.2
## 18 2017-08-20 23:30:00	176	4	omni-whip	NA	6.2

Il ressort que les stations réceptrices pour lesquelles les valeurs de l'angle de relèvement d'antenne sont manquantes sont limitées à celles dont les antennes sont omnidirectionnelles ou aux récepteurs mobiles, si bien que l'absence de valeurs est compréhensible. On constate aussi que les quatre enregistrements dans lesquels les valeurs de hauteur d'antenne sont manquantes sont également associés aux quatre récepteurs mobiles. Par conséquent, l'absence de valeurs de hauteur d'antenne est elle aussi compréhensible, et il n'est pas nécessaire d'apporter des corrections.

Rappelez-vous que les corrections relatives aux métadonnées manquantes doivent être apportées en ligne. Les métadonnées corrigées en ligne seront automatiquement corrigées dans vos fichiers de données de détection. Si vous avez déjà téléchargé vos données de détection, vous pouvez mettre à jour le fichier existant afin d'inclure les nouvelles métadonnées et données de détection (voyez les sections 3.4.7 et 3.4.5).

Dans le prochain chapitre, nous examinerons nos données à la recherche de faux positifs et nous enlèverons les données de détection des signaux d'émetteurs ambigus.

## Chapter 5

# Nettoyage des données

Il peut arriver que des données de détection d'émetteurs dans votre base soient incorrectes; les «erreurs» peuvent avoir trois causes.

Premièrement, les récepteurs peuvent détecter du bruit radioélectrique aléatoire (des parasites) qu'ils interprètent comme étant de véritables signaux provenant d'émetteurs. Le phénomène produit ce qu'on appelle des faux positifs.

Deuxièmement, même si nous faisons tout pour l'éviter, il arrive parfois que des émetteurs en double émettent des signaux en même temps. Lorsque sont déployés en même temps deux émetteurs ayant le même identifiant, la même cadence d'émission et la même fréquence d'émission nominale, il se peut que les signaux captés proviennent de l'un ou de l'autre des émetteurs. Si cela se produit, il faut s'appuyer sur l'information contextuelle pour les départager (si possible). Les signaux en question sont appelés signaux d'émetteurs ambigus.

Troisièmement, il peut arriver que deux émetteurs diffusent chacun un signal en même temps et, ce faisant, produisent par hasard un signal semblant provenir d'un troisième émetteur, qui est en fait inexistant. Cette situation survient la plupart du temps à des sites de repos d'oiseaux ou dans des colonies de nidification, où de nombreux émetteurs fonctionnent simultanément. Dans ces cas, on qualifie ces émetteurs de faux. Il n'est pas expressément question des faux émetteurs dans le présent chapitre; nous cherchons une façon de les détecter de manière globale et d'éliminer les données qui s'y rapportent. Il en est question ici parce que vous pourriez rencontrer des situations où des données de détection semblent très plausibles sans pourtant avoir de sens sur le plan biologique. Communiquez avec nous si vous croyez qu'il existe des données de détection de signaux de faux émetteurs dans votre base.

*Le présent chapitre a pour but* de vous fournir les outils dont vous avez besoin pour trouver les détections erronées dans vos données et les éliminer. Nous vous donnons ci-après des exemples de flux de travail permettant de traiter les faux positifs et les signaux d'émetteurs ambigus:

- 1) Utiliser un filtre préliminaire pour supprimer toutes les détections faisant partie d'une séquence d'au moins 2. Une séquence est un groupe de détections consécutives des signaux d'un émetteur par une antenne et un récepteur donnés. En général, le risque est élevé qu'une séquence d'au moins 2 détections (c.-à-d. 2 salves d'impulsions) représente un faux positif. Nous recommandons généralement d'éliminer toutes les détections faisant partie d'une séquence d'au moins 2, sauf si celles-ci ont eu lieu à l'une de quelques stations «tranquilles» où il y a peu de bruit radioélectrique. Toutefois, comme il est probable que cette opération entraîne la perte de détections non erronées, nous recommandons également qu'après avoir effectué une analyse complète de vos données, vous réexaminez les détections en cause une à une pour déterminer (habituellement selon le contexte) si elles peuvent être considérées comme acceptables.
- 2) Déterminer combien de vos détections d'émetteurs peuvent être des détections de signaux ambigus.

- 3) Fournir un flux de travail pour examiner les détections individuellement et déterminer si elles font partie d'une séquence d'au moins 2 et si elles correspondent à des erreurs.
- 4) Éliminer les erreurs dans vos données.

## 5.1 Chargement des logiciels requis

Suivez les instructions dans le chapitre 2 pour installer les logiciels suivants avant de charger les données, si cela n'est pas déjà fait.

```
Sys.setenv(tz = "GMT")

# library(devtools)
library(motus)
library(tidyverse)
library(lubridate)
# library(rworldmap) # pour la production de cartes
```

## 5.2 Chargement des données de détection

Il est indiqué au chapitre 3 que pour accéder à la base de données du projet 176 (données du Programme de suivi des oiseaux de rivage de la baie James), il faut entrer «motus.sample» dans la console R comme nom d'utilisateur et mot de passe à l'invite de la fonction tagme() du processus d'authentification de l'utilisateur. Nous tenons pour acquis que vous avez déjà effectué le téléchargement initial de ces données.

En accédant à la table alltags, nous supprimons certaines variables non nécessaires pour réduire la taille globale de l'ensemble de données et rendre son utilisation plus facile. **C'est particulièrement important dans le cas des projets de grande envergure et complexes;** la section 3.7 présente la marche à suivre pour visualiser les variables dans une table et pour filtrer les données et en faire des sous-ensembles avant de les réunir dans une trame de données. Nous créons ensuite les variables de latitude et de longitude des récepteurs («recvLat», «recvLon», «recvAlt») en nous basant sur les coordonnées enregistrées par leur GPS («gpsLat», «gpsLon», «gpdAlt»). Lorsque ces coordonnées ne sont pas disponibles, il faut utiliser celles qui sont contenues dans les métadonnées sur les récepteurs déployés («recvDeployLat», «recvDeployLon», «recvDeployAlt»). Nous utilisons les instructions «collect()» et «as.data.frame()» pour transformer la trame de données en un fichier «plat» puis nous transformons toutes les variables temporelles exprimées en secondes depuis le 1er janvier 1970 dans le format de type datetime (POSIXct). Enfin, nous créons des noms de récepteurs («receiver names») à partir des variables de latitude et de longitude propres aux récepteurs dans la base de données pour lesquels les valeurs de ces variables ne sont pas inscrites.

```
proj.num <- 176

# Charger les données de détection, choisir les variables, créer des variables de latitude et transform
# la trame de données en un fichier plat. De plus, nous intervenons pour les sites dont les données sur
# sont manquantes ou dont les récepteurs n'ont pas de nom. Quand plus d'utilisateurs auront examiné (et
# il devrait commencer à y avoir moins de données manquantes.
sql.motus <- tagme(proj.num, update = TRUE, dir = "./data/")
tbl.alltags <- tbl(sql.motus, "alltags")

df.alltags <-tbl.alltags %>%
  mutate(recvLat = if_else((is.na(gpsLat)|gpsLat == 0),
                           recvDeployLat, gpsLat),
        recvLon = if_else((is.na(gpsLon)|gpsLon == 0),
                           recvDeployLon, gpsLon),
```

```

    recvAlt = if_else(is.na(gpsAlt), recvDeployAlt, gpsAlt)) %>%
  select(-noise, -slop, -burstSlop, -done, -bootnum, -mfgID,
         -codeSet, -mfg, -nomFreq, -markerNumber, -markerType,
         -tagDeployComments, -fullID, -deviceID, -recvDeployLat,
         -recvDeployLon, -recvDeployAlt, -speciesGroup, -gpsLat,
         -gpsLon, -recvAlt, -recvSiteName) %>%
  collect() %>%
  as.data.frame() %>%
  mutate(ts = as_datetime(ts), # Travailler avec les dates APRÈS avoir transformé la trame de données en
        tagDeployStart = as_datetime(tagDeployStart),
        tagDeployEnd = as_datetime(tagDeployEnd),
        recvLat = plyr::round_any(recvLat, 0.05),
        recvLon = plyr::round_any(recvLon, 0.05),
        recvDeployName = if_else(is.na(recvDeployName),
                                paste(recvLat, recvLon, sep=":"),  

                                recvDeployName))

# Notez que dans l'instruction «select», vous pouvez simplement choisir les variables dont vous avez besoin
# par exemple select(runID, ts, sig, freqsd, motusTagID, ambigID, runLen, tagProjID,
#                    tagDeployStart, tagDeployEnd, etc.)

```

## 5.3 Vérifications préliminaires des données

Avant de filtrer les données, il faut produire quelques sommaires et graphiques de données.

### 5.3.1 Sommaires de données de détection d'émetteurs

Premièrement, déterminez quels émetteurs ont été détectés et combien sont caractérisés par une séquence d'au moins 2 détections. Il y a plusieurs raisons pour lesquelles il se peut que des émetteurs déployés ne soient pas détectés:

- 1) L'émetteur n'a pas été activé correctement au moment du déploiement. Pour éviter cela, il faut toujours s'assurer que l'émetteur est activé en utilisant un récepteur portatif pour vérifier l'émission avant de fixer l'émetteur sur l'animal et de remettre celui-ci en liberté.
- 2) Il se peut qu'un animal portant un émetteur activé correctement ne soit pas passé dans la zone de détection d'une station réceptrice. La conception des études comprenant l'installation des récepteurs à des endroits stratégiques en fonction des objectifs du projet peut augmenter la probabilité de détection des émetteurs.
- 3) S'il y a des métadonnées sur les émetteurs déployés qui sont manquantes ou incorrectes dans la base de données Motus, il se peut que l'algorithme de traitement des données ne «voie» pas les émetteurs en question au moment où ils ont été déployés, ou à quelque moment que ce soit. Il faut s'assurer que les métadonnées sur les émetteurs sont entrées correctement.

Avant de poursuivre, vérifiez si vous avez des émetteurs qui ont été déployés plus d'une fois, tel que décrit dans la section 4.3.4. Si c'est le cas, vous devrez utiliser «tagDeployID» ou une combinaison de «motusTagID» et de «tagDeployID» pour distinguer les détections propres à chaque déploiement (l'un ou l'autre des identifiants d'émetteurs peut faire l'affaire, mais en les combinant, vous saurez quel identifiant est associé à chaque déploiement).

Dans le projet 176, tous les émetteurs ont été déployés seulement une fois chacun. Nous utiliserons donc «motusTagID» comme identifiant unique pour un émetteur déployé dans tout le code R tout au long du

présent guide.

L'opération suivante montre que 18 émetteurs déployés dans le cadre du projet 176 ont été détectés et que beaucoup sont caractérisés par une séquence d'au moins 2 détections (TRUE):

```
df.alltags %>%
  filter(tagProjID == proj.num) %>% # sous-ensemble devant inclure seulement les émetteurs enregistrés
  mutate(r1.gt.2 = runLen == 2) %>%
  group_by(motusTagID, r1.gt.2) %>%
  tally() %>%
  spread(key = r1.gt.2, value=n)

## # A tibble: 18 x 3
## # Groups:   motusTagID [18]
##       motusTagID `FALSE` `TRUE`
## * <int>     <int>    <int>
## 1       16011      125      2
## 2       16035      454      2
## 3       16036      106     12
## 4       16037     1307     46
## 5       16038       84     78
## 6       16039     1098     28
## 7       16044      289     16
## 8       16047      773     66
## 9       16048       84     14
## 10      16052      133     26
## 11      17357      277     12
## 12      19129      568    720
## 13      22867      5545    222
## 14      22897     34308    488
## 15      22902      2815    108
## 16      22905     25684    326
## 17      23316      5518    216
## 18      23319     22471    288
```

Dans certains cas, il peut s'agir de détections valides, mais nous jugeons qu'il est plus simple de les retirer de l'analyse, pour y revenir éventuellement. Nous filtrons donc en fonction d'une séquence de détections (runLen) supérieure à 2 pour la plupart des opérations subséquentes. Enregistrons les données filtrées dans un bloc pour l'ajouter à nos autres filtres plus tard.

```
df.alltags.sub <- filter(df.alltags, runLen > 2)

df.block.0 <- filter(df.alltags, runLen == 2) %>% select(motusTagID,
  runID) %>% distinct()
```

La meilleure façon d'obtenir une première représentation des données consiste à les pointer sous forme de graphiques. Nous vous montrerons plus tard comment pointer les données de détection sur une carte, mais nous préférons adopter d'abord une approche plus simple, soit le pointage selon la latitude et la longitude. Toutefois, il faut d'abord simplifier les données. Autrement, nous risquerions de tenter de répartir des milliers ou des millions de points, ce qui peut prendre beaucoup de temps. Nous simplifierons les données en créant une petite fonction qui nous servira de nouveau à de futures étapes.

Notez qu'il faut enlever environ 150 détections qu'il est impossible de localiser parce qu'il n'y a aucune référence géographique associée aux métadonnées sur les récepteurs qui ont capté les signaux. Faites une simple vérification pour déterminer si les récepteurs en cause vous appartiennent; dans l'affirmative, **vous devez corriger les métadonnées en ligne!**

```
filter(df.alltags.sub, is.na(recvLat)) %>% select(recvLat,
  recvLon, recvDeployName, recvDeployID, recv, recvProjID,
  recvProjName) %>% distinct()
```

```
##   recvLat recvLon recvDeployName recvDeployID      recv recvProjID
## 1      NA      NA        NP mobile      3813    Lotek-280      176
## 2      NA      NA       NA:NA      NA SG-1415BBBK0382      NA
## 3      NA      NA       NA:NA      NA SG-2814BBBK0547      NA
##   recvProjName
## 1   SampleData
## 2      <NA>
## 3      <NA>
```

### Simplification des données en vue du pointage

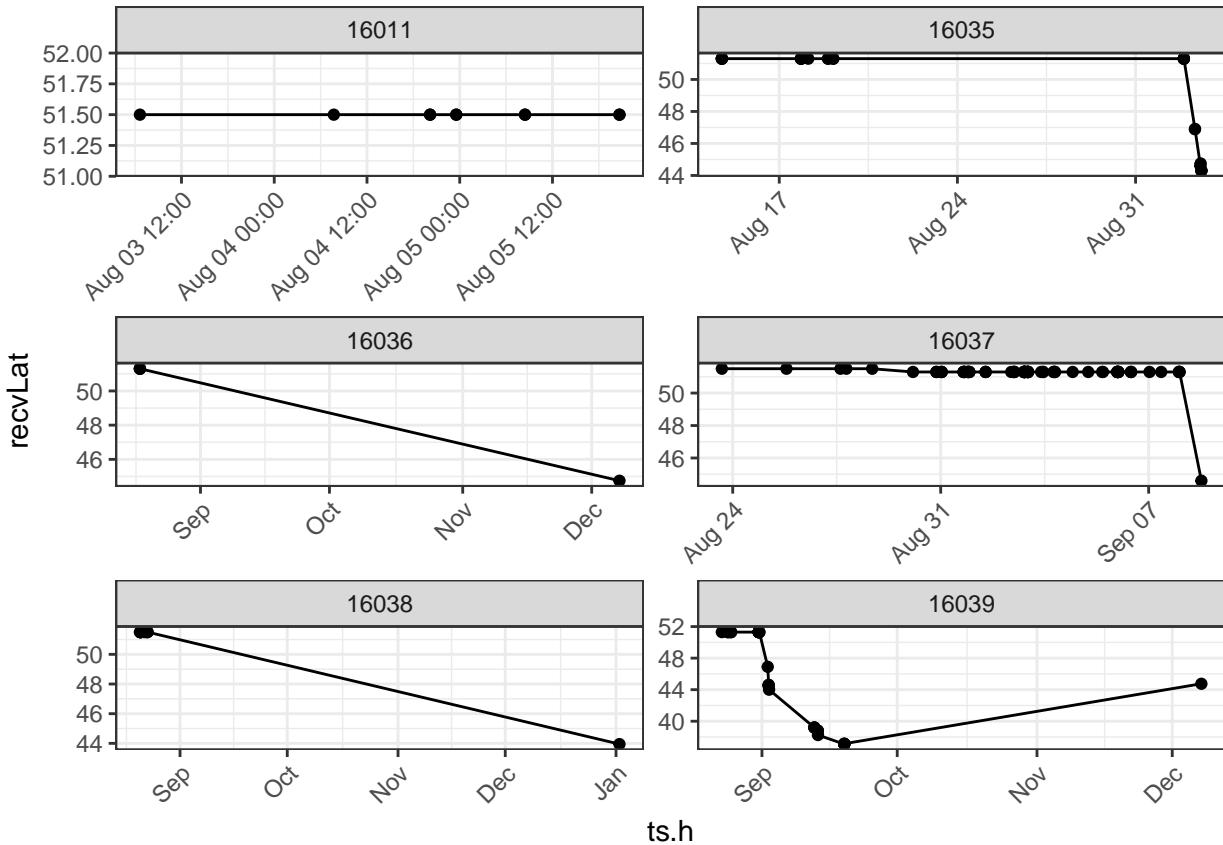
```
# Simplifiez les données en produisant un sommaire en fonction du runID.
# Si vous voulez produire un sommaire à une échelle plus fine ou plus grossière, vous pouvez aussi créer
# L'option de recharge la plus simple est une variable d'estampille temporelle arrondie; par exemple en
# l'appel de fonction mutate(ts.h = plyr::round_any(ts, 3600)).
# Une autre option consiste à utiliser juste la date (p. ex. date = as_date(ts)).

#
fun.getpath <- function(df)
{
  df %>%
    filter(tagProjID == proj.num, # Conservez seulement les émetteurs enregistrés dans le cadre du projet
          !is.na(recvLat) | !(recvLat == 0)) %>% # Rejette les données sans indication de longitude/latitude
    group_by(motusTagID, runID, recvDeployName, ambigID,
             tagDeployLon, tagDeployLat, recvLat, recvLon) %>%
    # Production d'un sommaire par runID pour obtenir la séquence de détections maximale et la référence
    summarize(max.runLen = max(runLen), ts.h = mean(ts)) %>%
    arrange(motusTagID, ts.h)
} # Fin de l'appel de fonction

df.alltags.path <- fun.getpath(df.alltags.sub)
```

Nous commencerons par localiser un sous-ensemble d'émetteurs en fonction de la latitude ou de la longitude afin d'avoir un aperçu des problèmes possibles. Ici, pour simplifier l'exemple, nous pointons la position de seulement six émetteurs. Pour l'instant, nous évitons d'examiner les émetteurs ambigus.

```
p <- ggplot(data = filter(df.alltags.path, motusTagID %in%
  c(16011, 16035, 16036, 16037, 16038, 16039)), aes(ts.h,
  recvLat))
p + geom_point() + geom_path() + theme_bw() + facet_wrap(~motusTagID,
  scales = "free", ncol = 2) + theme(axis.text.x = element_text(angle = 45,
  vjust = 1, hjust = 1))
```



Nous voyons tout de suite qu'il pourrait y avoir un problème, car la position indiquée de certains émetteurs est à environ 44 degrés de latitude en hiver, ce qui est possible mais non probable dans le cas des oiseaux visés par le Programme de suivi des oiseaux de rivage de la baie James (projet 176). Examinons ces émetteurs de plus près en vérifiant les séquences de détections dans la trame de données qui sont associées aux détections effectuées en décembre et en janvier.

```
filter(df.alltags.sub, month(ts) %in% c(12, 1), motusTagID %in%
      c(16036, 16038, 16039)) %>% group_by(recvDeployName,
      month(ts), runLen) %>% summarize(n = length(ts),
      n.tags = length(unique(motusTagID)))
```

```
## # A tibble: 2 x 5
## # Groups:   recvDeployName, month(ts) [?]
##   recvDeployName `month(ts)` runLen     n n.tags
##   <chr>           <dbl>    <int> <int> <int>
## 1 Sable West Light 2       1.00      3     3     1
## 2 Swallowtail        12.0       3     6     2
```

Ces détections correspondent à des positions dans les Provinces maritimes du Canada (l'île de Sable, en Nouvelle-Écosse, et l'île Grand Manan, au Nouveau-Brunswick) et sont caractérisées par des séquences de 3 détections. Elles indiquent la présence probable de faux positifs. Commençons un compte des séquences précises en cause afin de les réunir pour effectuer un filtrage plus tard.

Si cela vous intéresse, vous pouvez exécuter de nouveau le code ci-dessus, mais avec la trame de données complète (df.alltags) contenant des séquences d'au moins 2 détections. Vous constaterez qu'il existe à ces positions d'autres détections de faux positifs qui sont déjà éliminées par filtrage en fonction d'une séquence d'au moins 2 détections (runLen > 2). Ces détections supplémentaires montrent une nouvelle fois qu'il y avait du bruit radioélectrique à ces positions pendant ces mois précis (décembre et janvier), ce qui a causé

certaines détections de faux positifs.

Vous pourriez aussi être intéressés d'une manière plus générale à déterminer quelles données sont caractérisées seulement par de courtes séquences de détections. Par exemple, le code suivant indique la séquence de détections maximale à tous les sites par mois (pour les séquences d'au moins 2 détections [ $\text{runLen} > 2$ ]).

```
df.alltags.sub %>% mutate(month = month(ts)) %>% group_by(recvDeployName,
  month) %>% summarize(max.rl = max(runLen)) %>%
  spread(key = month, value = max.rl)
```

```
## # A tibble: 49 x 9
## # Groups:   recvDeployName [49]
##   recvDeployName      `1`   `3`   `4`   `5`   `8`   `9`   `10`   `12`
##   * <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Assateague State Park     NA     NA    NA NA     NA  6.00 NA     NA
## 2 BennettMeadow            NA     NA    NA NA     NA NA 11.0  NA
## 3 Binbrook_Conservation_~   NA     NA    NA  3.00     NA NA    NA NA
## 4 BISE                     NA     NA    NA NA     NA NA    6.00 NA
## 5 Bombay Hook               NA     NA    NA NA     NA 53.0  NA     NA
## 6 Brier2                   NA     NA    NA NA     NA 29.0  NA     NA
## 7 BSC HQ                   NA     NA    NA 21.0     NA NA    NA NA
## 8 BULL                      NA     NA    NA NA     NA 38.0  5.00  NA
## 9 Comeau (Marshalltown)    NA     NA    NA NA     NA  4.00 NA     NA
## 10 CONY                     NA     NA    NA NA     NA    NA    7.00 NA
## # ... with 39 more rows
```

Vous pouvez aussi produire une liste des sites où la séquence de détections maximale ne dépasse jamais 4 (par exemple), ce qui peut parfois (mais pas toujours!) indiquer qu'il s'agit simplement de faux positifs.

```
df.alltags.sub %>% mutate(month = month(ts)) %>% group_by(recvDeployName,
  month) %>% summarize(max.rl = max(runLen)) %>%
  filter(max.rl < 5) %>% spread(key = month, value = max.rl)
```

```
## # A tibble: 12 x 8
## # Groups:   recvDeployName [12]
##   recvDeployName      `1`   `3`   `4`   `5`   `9`   `10`   `12`
##   * <chr>          <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 Binbrook_Conservation_Area NA     NA    3.00 NA     NA     NA
## 2 Comeau (Marshalltown)    NA     NA    NA  4.00  NA     NA
## 3 Koffler                  NA     NA    3.00 NA     NA     NA
## 4 LLICALDAD                NA     4.00 NA    NA     NA     NA
## 5 MountToby                 NA     NA    NA  NA     NA    3.00 NA
## 6 NP mobile                 NA     NA    NA  3.00  NA     NA
## 7 Old Cut                   NA     NA    4.00 NA     NA     NA
## 8 OSCT                      NA     NA    NA  NA     NA    3.00 NA
## 9 Quempillen (Chile)       NA     3.00 NA    NA     NA     NA
## 10 Sable West Light 2      3.00  NA    NA  NA     NA     NA
## 11 Swallowtail              NA     NA    NA  NA     NA    3.00
## 12 TRUS                     NA     NA    NA  NA     NA    4.00 NA
```

Il est impossible de passer en revue ici tous les problèmes possibles. Nous vous encourageons fortement à explorer vos données minutieusement avant de prendre des décisions éclairées quant à savoir quelles détections sont improbables ou indéterminées. Dans le reste du présent chapitre, nous vous montrerons comment recueillir l'information sur les séquences de détections et les appliquer à vos données avant l'analyse.

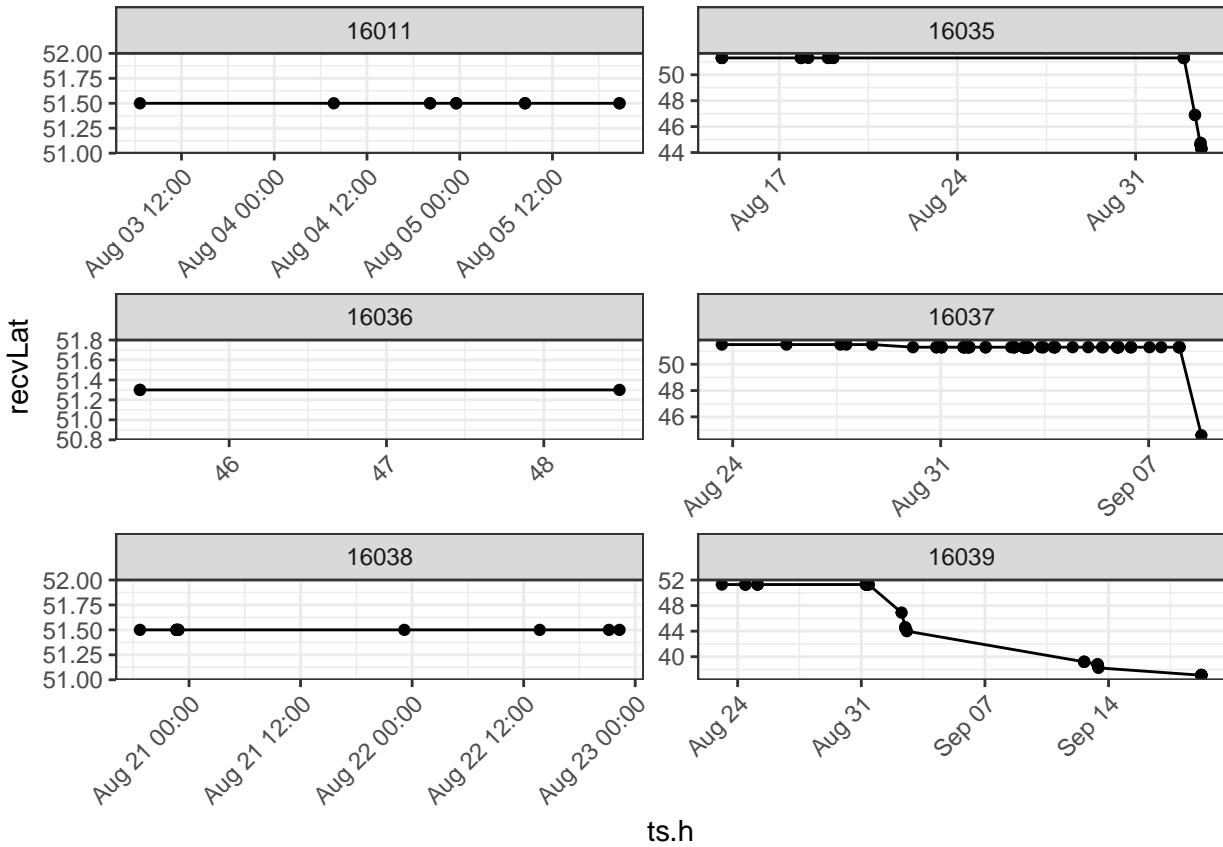
Pour commencer, créons une trame de données contenant les motusTagID et les runID correspondant aux faux positifs indiqués précédemment.

Ensuite, produisons de nouveau le graphique avec les données nouvellement filtrées.

```
# Créez le filtre.
df.block.1 <- filter(df.alltags.sub, month(ts) %in%
  c(12, 1), motusTagID %in% c(16036, 16038, 16039)) %>%
  select(motusTagID, runID) %>% distinct()

# Utilisez la fonction que nous avons créée
# précédemment pour produire une nouvelle trame de
# données «path» en vue de créer le graphique.
df.alltags.path <- fun.getpath(filter(df.alltags.sub,
  motusTagID %in% c(16011, 16035, 16036, 16037, 16038,
  16039), !(runID %in% df.block.1$runID)))

p <- ggplot(data = df.alltags.path, aes(ts.h, recvLat))
p + geom_point() + geom_path() + theme_bw() + facet_wrap(~motusTagID,
  scales = "free", ncol = 2) + theme(axis.text.x = element_text(angle = 45,
  vjust = 1, hjust = 1))
```



Nous pouvons voir que la majorité des détections qui restent semblent maintenant avoir plus de sens. Les émetteurs 16035, 16037 et 16039 ont été détectés pendant la migration dans ce qui semble être une progression latitudinale raisonnable dans le temps, et les trois autres émetteurs n'ont pas été détectés très loin du lieu de leur déploiement.

Nous vous encourageons à explorer le reste des émetteurs dans ce groupe pour déterminer s'il y a d'autres faux positifs.

## 5.4 Recherche et examen de détections ambiguës

Avant d'aller plus loin, nous devons vérifier s'il y a des détections ambiguës d'émetteurs. S'il y en a, nous devons les examiner et créer d'autres filtres pour les éliminer de notre base de données.

### Est-ce que vous avez des émetteurs associés à des détections ambiguës?

La fonction «clarify()» du logiciel R motusClient permet de produire un sommaire des ambiguïtés dans les données de détection. Chaque ambigID se rapporte à une sélection de détections qui pourraient correspondre à au moins un motusTagID (jusqu'à concurrence de 6), lesquels sont indiqués dans les champs id1 à id6:

```
clarify(sql.motus)
```

```
##   ambigID numHits    id1                      fullID1   id2
## 1      -56     5734 22867 SampleData#272.1:5.3@166.38(M.22867) 23316
## 2     -106      279 17021           Selva#172:6.1@166.38(M.17021) 17357
## 3     -114       86 22897 SampleData#303.1:5.3@166.38(M.22897) 24298
## 4     -134     22749 22905 SampleData#301:5.3@166.38(M.22905) 23319
## 5     -171      2074 22778 RBrownAMW0#308:5.3@166.38(M.22778) 22902
## 6     -337        4 10811      Niles#152:6.1@166.38(M.10811) 16011
##                               fullID2   id3
## 1 SampleData#272:5.3@166.38(M.23316) NA
## 2 SampleData#172:6.1@166.38(M.17357) NA
## 3 NEONICS#303:5.3@166.38(M.24298) NA
## 4 SampleData#301.1:5.3@166.38(M.23319) NA
## 5 SampleData#308.1:5.3@166.38(M.22902) 24303
## 6 SampleData#152:6.1@166.38(M.16011) NA
##                               fullID3 id4 fullID4 id5 fullID5 id6 fullID6
## 1                  <NA> NA  <NA> NA  <NA> NA  <NA>
## 2                  <NA> NA  <NA> NA  <NA> NA  <NA>
## 3                  <NA> NA  <NA> NA  <NA> NA  <NA>
## 4                  <NA> NA  <NA> NA  <NA> NA  <NA>
## 5 NEONICS#308:5.3@166.38(M.24303) NA  <NA> NA  <NA> NA  <NA>
## 6                  <NA> NA  <NA> NA  <NA> NA  <NA>
##   motusTagID tsStart tsEnd
## 1          NA      NA     NA
## 2          NA      NA     NA
## 3          NA      NA     NA
## 4          NA      NA     NA
## 5          NA      NA     NA
## 6          NA      NA     NA
```

Nous pouvons voir qu'il y a dans cet ensemble de données six émetteurs associés à des détections ambiguës. Les détections associées à cinq des six «ambigID» pourraient correspondre à un de deux émetteurs et celles associées à un «ambigID» (-171), à un de trois émetteurs. Les champs «fullID» indiquent les noms des projets associés aux émetteurs en double (p. ex., «SampleData», «Selva» et «Niles»), ainsi que les caractéristiques des émetteurs (identifiant du fabricant, cadence d'émission et fréquence d'émission).

Obtenons un vecteur et produisons des graphiques pour voir où il pourrait y avoir des problèmes.

```
df.ambigTags <- select(df.alltags.sub, ambigID, motusTagID) %>%
  filter(!is.na(ambigID)) %>% distinct()
```

En utilisant notre fonction «getpath», nous créerons des trajectoires et nous pointerons les détections sur des graphiques. Nous ajouterons de l'information dans les graphiques pour indiquer où (dans le temps) les émetteurs sont effectivement ambiguës. Nous pourrons alors inspecter l'ensemble (ou des parties) de chaque graphique pour déterminer si nous pouvons attribuer sans ambiguïté selon le contexte la détection d'un

émetteur ambigu à un déploiement particulier.

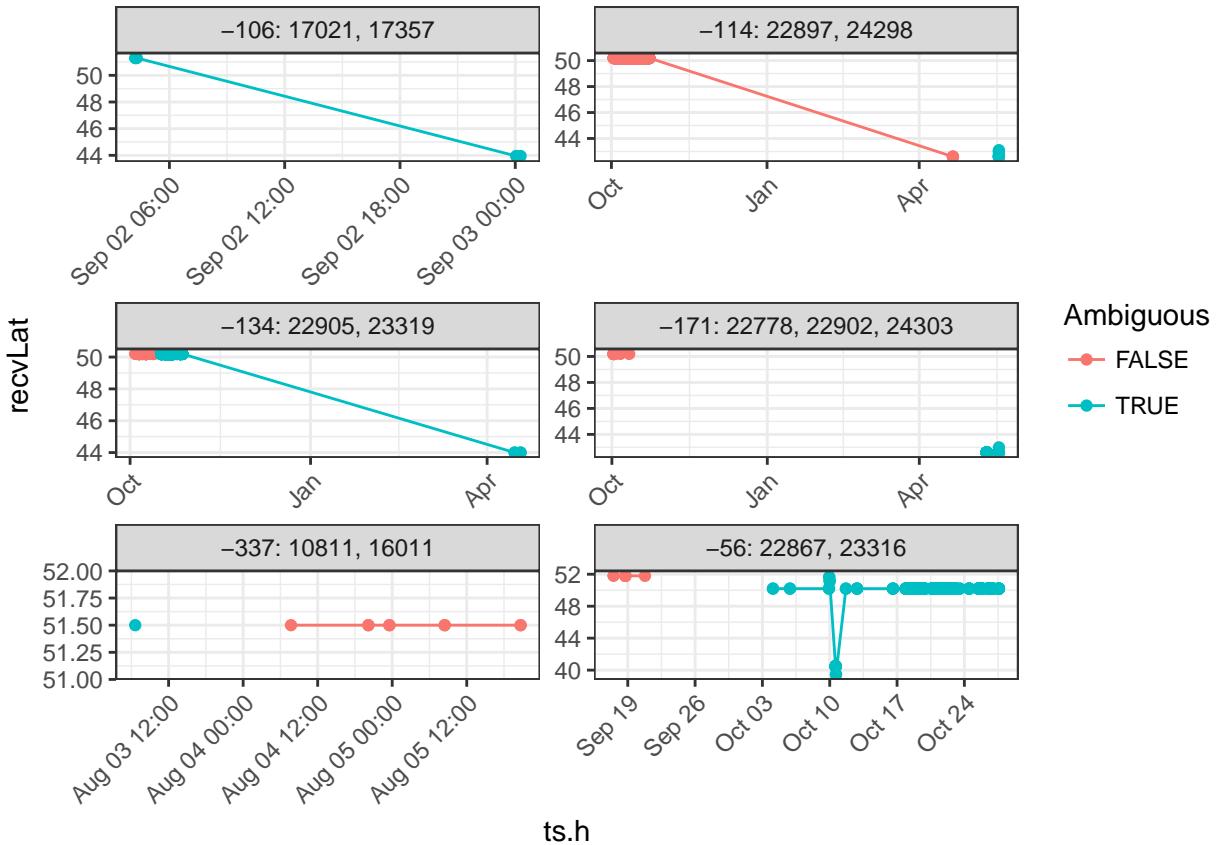
```
df.alltags.path <- fun.getpath(filter(df.alltags.sub,
  motusTagID %in% df.ambigTags$motusTagID, tagProjID ==
    proj.num)) %>% # Créez une variable booléenne pour les détections
# ambiguës:
mutate(Ambiguous = !(is.na(ambigID)))

# Pour pointer tous les émetteurs ambigus se
# rapportant à un même projet ensemble sur le même
# graphique, nous devons créer une nouvelle
# variable «ambig tag» que nous appelons «newID»

ambigTags.2 <- filter(df.alltags.sub) %>% select(ambigID,
  motusTagID) %>% filter(!is.na(ambigID)) %>% distinct() %>%
  group_by(ambigID) %>% summarize(newID = paste(unique(ambigID),
  toString(motusTagID), sep = ":")) %>% left_join(df.ambigTags,
  by = "ambigID")

# et la fusionner avec «df.alltags.path».
df.alltags.path <- left_join(df.alltags.path, ambigTags.2,
  by = "motusTagID") %>% arrange(ts.h)

p <- ggplot(data = df.alltags.path, aes(ts.h, recvLat,
  group = Ambiguous, colour = Ambiguous))
p + geom_point() + geom_path() + theme_bw() + facet_wrap(~newID,
  scales = "free", ncol = 2) + theme(axis.text.x = element_text(angle = 45,
  vjust = 1, hjust = 1))
```



Occupons-nous d'abord des cas faciles.

#### ambigID -337: motusTagID 10811 et 16011

```
filter(df.alltags.sub, ambigID == -337) %>% group_by(motusTagID,
  tagDeployStart, tagDeployEnd, tagDeployLat, tagDeployLon) %>%
  tally()
```

```
## # A tibble: 2 x 6
## # Groups:   motusTagID, tagDeployStart, tagDeployEnd, tagDeployLat [?]
##   motusTagID tagDeployStart      tagDeployEnd      tagDeployLat
##       <int>     <dttm>          <dttm>          <dbl>
## 1     10811 2014-10-28 07:00:00 2015-08-03 07:00:00    39.1
## 2     16011 2015-08-02 11:39:59 2015-12-17 11:39:59    51.5
## # ... with 2 more variables: tagDeployLon <dbl>, n <int>
```

Le graphique montre que l'émetteur ambigu -337 est ambigu seulement au début du déploiement.

En examinant le sommaire des données sur les déploiements des émetteurs, nous constatons qu'il y avait seulement 4 détections, à la latitude exacte du déploiement de l'émetteur 16011 et juste avant les détections non ambiguës du motusTagID 16011. Dès lors, le problème ici est seulement que la queue de la trajectoire de déploiement de l'émetteur 10811 chevauche légèrement la trajectoire de déploiement de l'émetteur 16011. Nous pouvons affirmer en toute confiance que ces détections sont reliées au motusTagID 16011 et supprimer les détections ambiguës attribuées à l'autre émetteur.

Nous créerons une autre trame de données pour assurer le suivi de ces séquences de détections.

```
# Nous voulons trouver les détections associées au
# motusTagID que nous souhaitons ultimement
```

```
# SUPPRIMER de la trame de données.
df.block.2 <- filter(df.alltags.sub, ambigID == -337,
  motusTagID == 10811) %>% select(motusTagID, runID) %>%
  distinct()
```

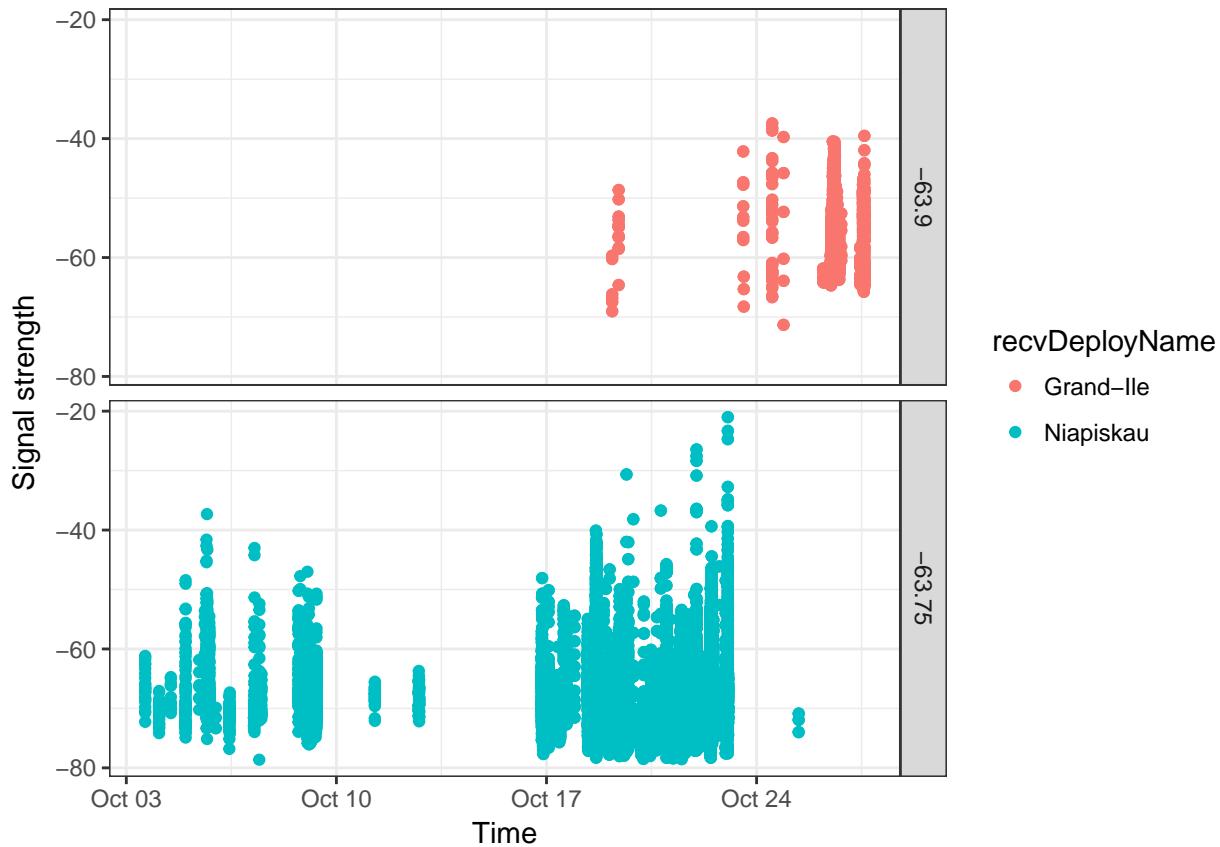
**ambigID -134: motusTagIDs 22905 et 23319**

```
filter(df.alltags.sub, ambigID == -134) %>% group_by(motusTagID,
  tagDeployStart, tagDeployEnd, tagDeployLat, tagDeployLon,
  month(ts)) %>% tally()
```

```
## # A tibble: 4 x 7
## # Groups:   motusTagID, tagDeployStart, tagDeployEnd, tagDeployLat,
## #   tagDeployLon [?]
##   motusTagID tagDeployStart     tagDeployEnd      tagDeployLat
##   <int> <dttm>           <dttm>           <dbl>
## 1     22905 2016-10-01 16:00:00 2017-06-12 16:00:00    50.2
## 2     22905 2016-10-01 16:00:00 2017-06-12 16:00:00    50.2
## 3     23319 2016-10-15 16:00:00 2017-06-26 16:00:00    50.2
## 4     23319 2016-10-15 16:00:00 2017-06-26 16:00:00    50.2
## # ... with 3 more variables: tagDeployLon <dbl>, `month(ts)` <dbl>,
## #   n <int>
```

Ici, la situation est semblable à la précédente, mais un peu plus complexe. Deux émetteurs identiques ont été déployés peu de temps l'un après l'autre au même endroit. Examinons un graphique simple.

```
filter(df.alltags.sub, motusTagID %in% c(22905, 23319),
  month(ts) == 10) %>% ggplot(aes(ts, sig, group = recvDeployName,
  colour = recvDeployName)) + geom_point() + theme_bw() +
  xlab("Time") + ylab("Signal strength") + facet_grid(recvLon ~
  .)
```



Il semble que les détections à deux sites proches l'un de l'autre se chevauchent. De l'information supplémentaire fournie par les chercheurs sur le terrain pourrait nous permettre de dénouer l'intrigue, mais à la seule vue des données la situation n'est pas claire.

Nous examinons également les détections non ambiguës de l'émetteur -134 effectuées à la mi-avril. Comme c'est très tôt pour qu'un Bécasseau maubèche passe au-dessus du sud de l'Ontario, il y a lieu de s'interroger sur ces détections.

L'utilisation du filtre suivant révèle l'existence de deux séquences distinctes de 3 détections séparées par un intervalle de 3 jours. Si nous inspectons le reste de ce lot (c.-à-d. si nous examinons aussi les séquences d'au moins 2 détections dans la trame de données originale) ...

```
filter(df.alltags, batchID == 79646) %>% select(runLen,
  recvDeployName) %>% group_by(runLen, recvDeployName) %>%
  tally()
```

```
## # A tibble: 2 x 3
## # Groups:   runLen [?]
##   runLen recvDeployName     n
##   <int>   <chr>        <int>
## 1      2 Koffler        110
## 2      3 Koffler         12
```

... nous constatons qu'il y a beaucoup de faux positifs à cette tour à peu près au même moment (dans le même lot), de sorte que les séquences de 3 détections sont vraisemblablement de faux positifs. Nous supprimerons donc de la base de données toutes les détections de cet émetteur ambigu.

```
# Nous voulons trouver les détections associées au
# motusTagID que nous souhaitons ultimement
```

```
# SUPPRIMER de la trame de données.

df.block.3 <- filter(df.alltags.sub, ambigID == -134) %>%
  select(motusTagID, runID) %>% distinct()
```

ambigID -171: motusTagIDs 22778, 22902 et 22403

Les détections ambiguës reliées à cet émetteur, effectuées dans la région des Grands Lacs, pourraient aussi être reliées au motusTagID 22778 enregistré dans le cadre du projet RBrownAMWO ou au motusTagID 24303 enregistré dans le cadre du projet Neonics. Examinons ces détections de plus près.

Premièrement, trouvez la date et le lieu du déploiement de chaque émetteur.

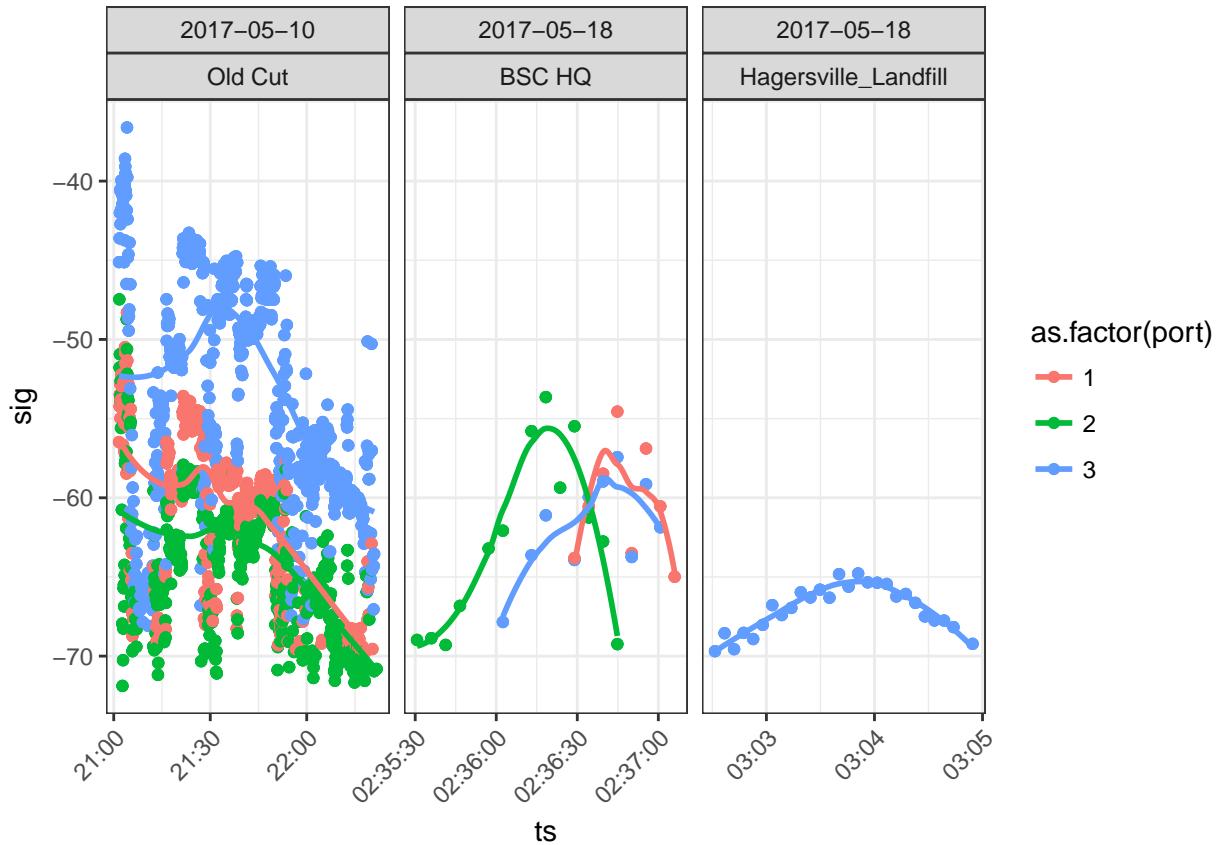
```
filter(df.alltags, ambigID == -171) %>% filter(!is.na(tagDeployStart)) %>%
  select(motusTagID, tagProjID, start = tagDeployStart,
         end = tagDeployEnd, lat = tagDeployLat, lon = tagDeployLon,
         species = speciesEN) %>% distinct() %>% arrange(start)
```

	motusTagID	tagProjID	start	end	lat
## 1	22902	176	2016-10-01 16:00:00	2017-06-12 16:00:00	50.19278
## 2	22778	82	2016-10-21 00:00:00	2018-09-09 00:00:00	45.13535
## 3	24303	146	2017-05-10 22:30:59	2017-06-30 22:30:59	42.60600
##	lon	species			
## 1	-63.74528	Red Knot			
## 2	-67.29323	American Woodcock			
## 3	-80.46900	White-crowned Sparrow			

Et pointez sur des graphiques les détections ambiguës.

```
df.ambig.171 <- filter(df.alltags.sub, ambigID == -171)

p <- ggplot(data = df.ambig.171, aes(ts, sig, colour = as.factor(port)))
p + geom_point() + geom_smooth(method = "loess", se = FALSE) +
  theme_bw() + facet_wrap(as_date(ts) ~ recvDeployName,
                         scales = "free_x") + theme(axis.text.x = element_text(angle = 45,
                         vjust = 1, hjust = 1))
```



Nous voyons qu'il y a un grand nombre de détections ambiguës le 10 mai 2017 à Old Cut (Long Point, lac Érié, Ontario), ce qui s'expliquerait par la présence d'un oiseau qui «flâne» dans les environs. Il s'agit presque certainement de détections des signaux de l'émetteur 24303, qui a été déployé à Old Cut le 10 mai 2017. Les détections ultérieures ont été effectuées le 18 mai près d'Old Cut (siège d'Études d'Oiseaux Canada, Port Rowan, Ontario) puis au nord d'Old Cut (Hagersville, Ontario). Ces détections correspondent au départ en migration d'un oiseau. Notez en particulier les courbes dans les graphiques du centre et de droite qui correspondent à une hausse puis une baisse de la puissance du signal, ce qui indiquerait qu'un oiseau passe en volant dans le faisceau d'une antenne.

Comme ces détections sont reliées à un autre projet, nous supprimons simplement toutes les détections de cet émetteur ambigu de notre base de données.

```
# Nous voulons trouver les détections associées au
# motusTagID que nous souhaitons ultimement
# SUPPRIMER de la trame de données.
```

```
df.block.4 <- filter(df.alltags.sub, ambigID == -171) %>%
  select(motusTagID, runID) %>% distinct()
```

#### ambigID -114: motusTagIDs 22897 et 24298

Examinons maintenant les ambiguïtés se rapportant à l'émetteur -114.

```
filter(df.alltags, ambigID == -114) %>% filter(!is.na(tagDeployStart)) %>%
  select(motusTagID, tagProjID, start = tagDeployStart,
         end = tagDeployEnd, lat = tagDeployLat, lon = tagDeployLon,
         species = speciesEN) %>% distinct() %>% arrange(start)
```

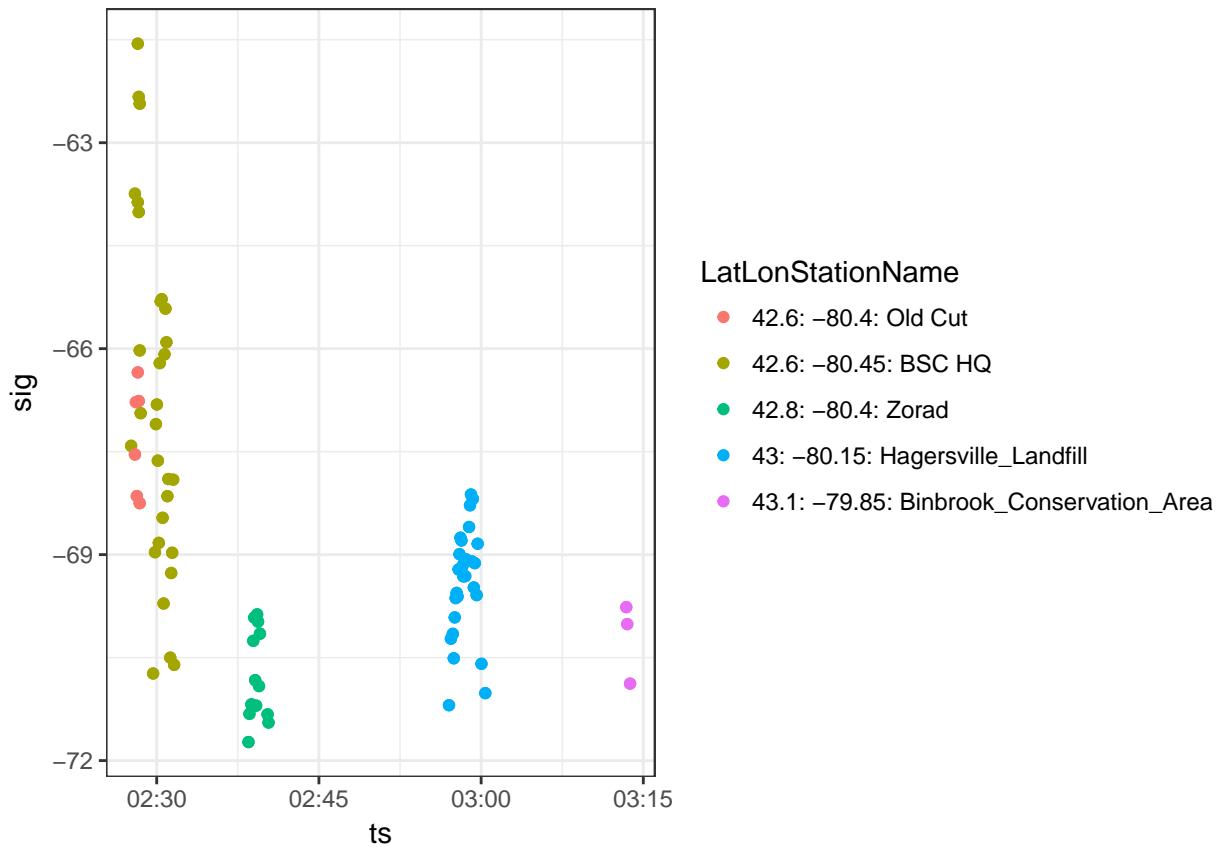
```
##   motusTagID tagProjID           start          end      lat
## 1     22897       176 2016-10-01 16:00:00 2017-06-12 16:00:00 50.19278
## 2     24298       146 2017-05-10 03:00:00 2017-06-30 03:00:00 42.60690
##   lon      species
## 1 -63.74528    Red Knot
## 2 -80.46900 White-crowned Sparrow
```

Ici encore, nous formons un sous-ensemble de données que nous pointons sur un graphique. Comme un premier graphique semble indiquer que toutes les détections se rapportent à un vol migratoire, nous produisons un graphique quelque peu différent du précédent qui illustre mieux cette situation.

```
df.ambig.114 <- filter(df.alltags.sub, ambigID == -114) %>%
  mutate(LatLonStationName = paste(recvLat, recvLon,
  recvDeployName, sep = ": "))

p <- ggplot(data = df.ambig.114, aes(ts, sig, colour = LatLonStationName))

p + geom_point() + theme_bw()
```



Notez que les détections correspondent à un départ en migration à partir du secteur de Long Point (station d'Old Cut, lac Érié, Ontario) environ une semaine après le déploiement de l'émetteur ambigu 24298 au même endroit. Dans ce cas aussi, il semble que les détections ambiguës peuvent être supprimées de la base de données car elles sont reliées à un autre projet.

```
df.block.5 <- filter(df.alltags.sub, ambigID == -114) %>%
  select(motusTagID, runID) %>% distinct()
```

ambigID -106: motusTagIDs 17021 et 17357

Ces deux émetteurs posent un problème intéressant. Il y a seulement une courte période de chevauchement, entre la mi-août et la mi-septembre 2015. Un des émetteurs a été fixé sur une Grive à joues grises en Colombie et l'autre sur un Bécasseau à croupion blanc, qui était associé au projet 176 (Programme de suivi des oiseaux de rivage de la baie James).

```
filter(df.alltags, ambigID == -106) %>% filter(!is.na(tagDeployStart)) %>%
  select(motusTagID, tagProjID, start = tagDeployStart,
         end = tagDeployEnd, lat = tagDeployLat, lon = tagDeployLon,
         species = speciesEN) %>% distinct() %>% arrange(start)
```

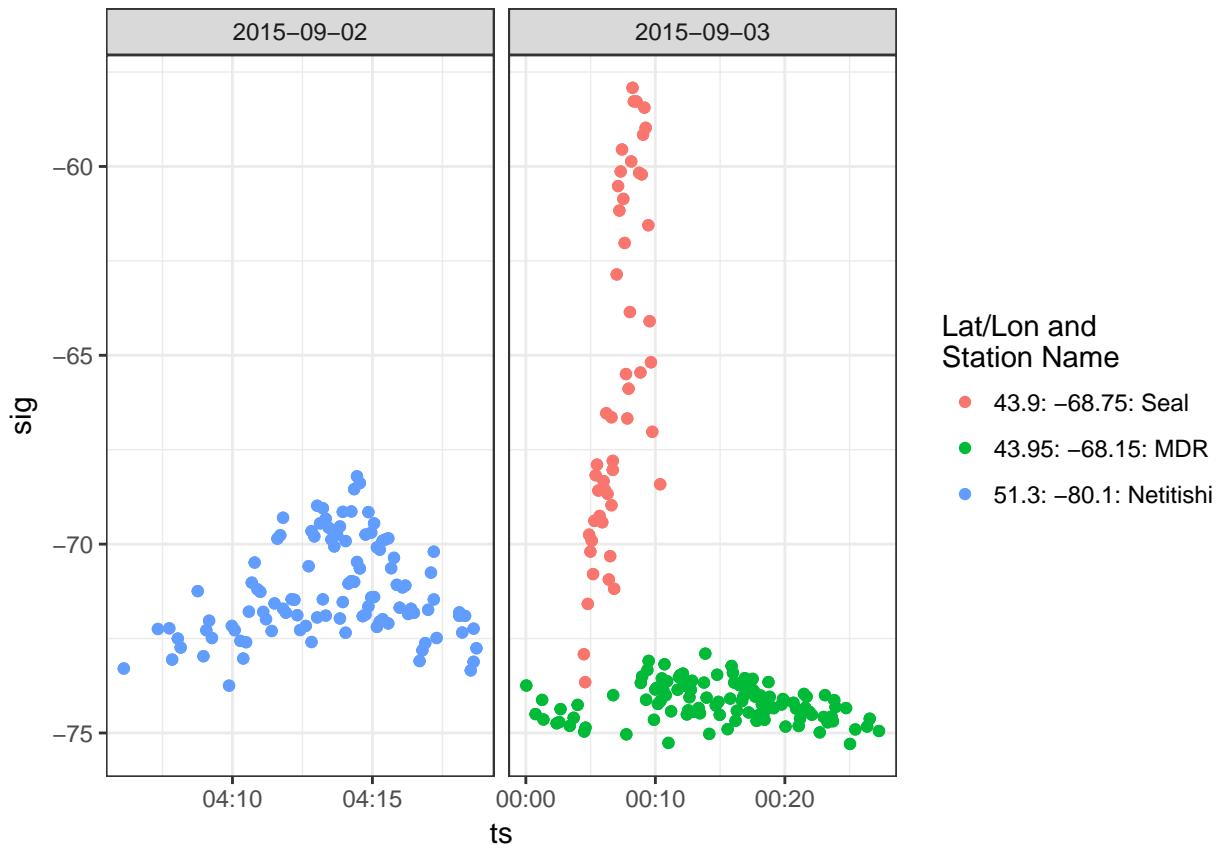
```
##   motusTagID tagProjID           start           end      lat
## 1       17021        57 2015-04-30 05:00:00 2015-09-14 05:00:00 11.12265
## 2       17357       176 2015-08-11 07:20:00 2015-12-26 07:20:00 51.48390
##   lon           species
## 1 -74.08735    Gray-cheeked Thrush
## 2 -80.45000 White-rumped Sandpiper
```

Nous pointons les détections ambiguës sur des graphiques pour examiner la période de chevauchement.

```
df.ambig.106 <- filter(df.alltags.sub, ambigID == -106)

p <- ggplot(data = df.ambig.106, aes(ts, sig, colour = paste(recvLat,
  recvLon, recvDeployName, sep = " : ")))

p + geom_point() + scale_colour_discrete(name = "Lat/Lon and\nStation Name") +
  theme_bw() + facet_wrap(~as_date(ts), scales = "free_x")
```



Les séquences de détections sont longues dans les deux ensembles, qui semblent valides (housse puis baisse

de la puissance du signal). Les deux ensembles de détections sont séparés d'environ une journée; il se peut donc qu'ils correspondent à deux oiseaux différents ou encore au vol de départ du Bécasseau à croupion blanc depuis sa halte migratoire. Utilisons la fonction siteTrans (dans le logiciel motus; voir la section C.13) pour examiner le vol entre Netitishi et MDR/Seal (dans le golfe du Maine).

```
df.ambig.106 %>% filter(motusTagID == 17021) %>% # Choisissez seulement l'identifiant d'un des deux éme
siteTrans(latCoord = "recvLat", lonCoord = "recvLon") %>%
ungroup() %>%
filter(rate < 60) %>% # Supprimez les détections simultanées des signaux provenant de Seal et de MDR.
mutate(total.time = as.numeric(round(seconds_to_period(tot_ts)))) %>%
select(start=recvDeployName.x, end=recvDeployName.y, date=ts.x, "rate(m/s)" = rate,
dist, total.time = total.time, bearing)
```

```
## # A tibble: 1 x 7
##   start     end     date           `rate(m/s)`    dist total.time bearing
##   <chr>     <chr>   <dttm>          <dbl>    <dbl>      <dbl>    <dbl>
## 1 Netiti~ MDR_4~ 2015-09-02 04:18:42       17.1  1.21e6      70879     128
```

Ces détections sont à plus de 1200 km l'une de l'autre, mais la vitesse du vol (17 m/s) correspond à celle d'un Bécasseau à croupion blanc. Étant donné que la durée de vie prévue de l'émetteur porté par la Grive à joues grises était proche de sa fin, nous pouvons raisonnablement affirmer que ces détections sont reliées au projet 176 et supprimer les détections ambiguës associées au motusTagID 17021.

```
df.block.6 <- filter(df.alltags.sub, ambigID == -106,
motusTagID == 17021) %>% select(motusTagID, runID) %>%
distinct()
```

### ambigID -56: motusTagIDs 22867 et 23316

Ces deux émetteurs ont aussi été déployés dans le cadre du même projet.

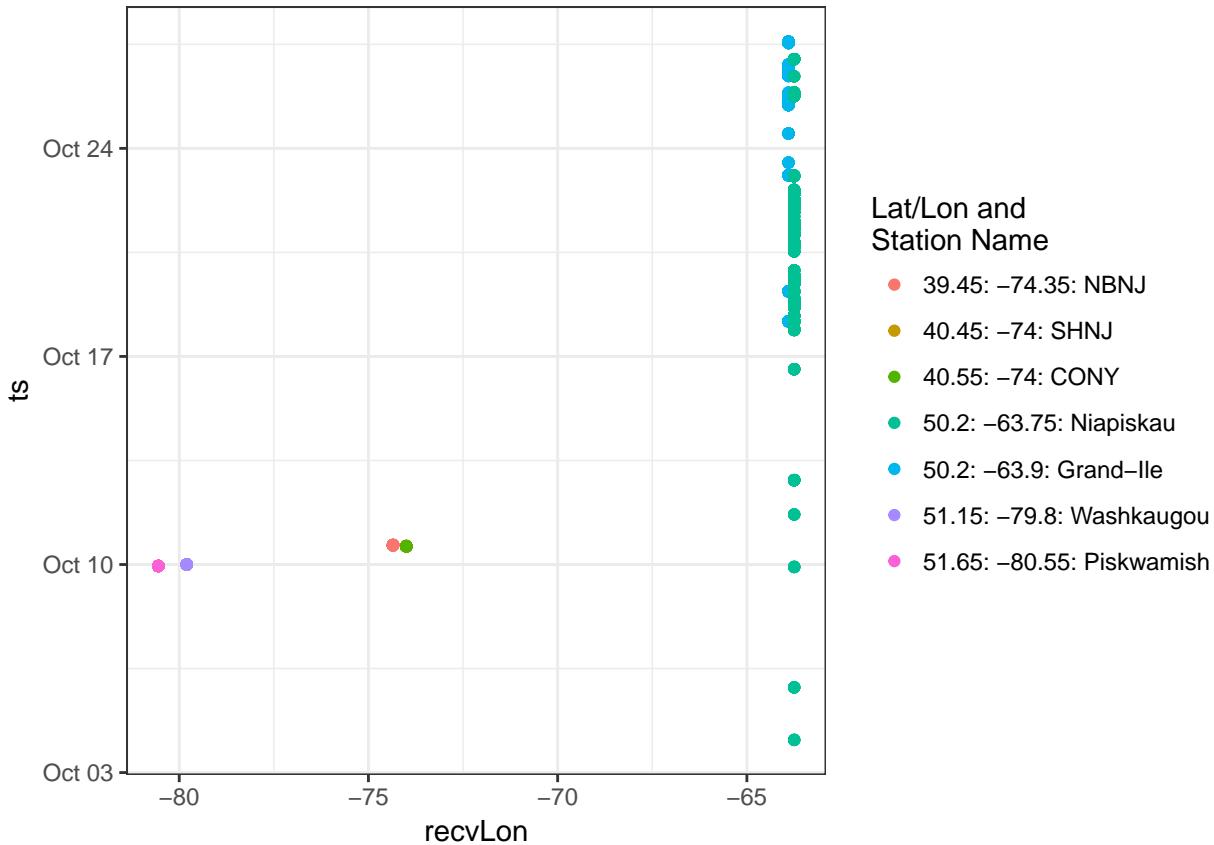
```
filter(df.alltags, ambigID == -56) %>% filter(!is.na(tagDeployStart)) %>%
select(motusTagID, tagProjID, start = tagDeployStart,
end = tagDeployEnd, lat = tagDeployLat, lon = tagDeployLon,
species = speciesEN) %>% distinct() %>% arrange(start)
```

```
##   motusTagID tagProjID           start           end        lat
## 1      22867      176 2016-09-06 15:35:00 2017-05-18 15:35:00 51.79861
## 2      23316      176 2016-10-02 16:00:00 2017-06-13 16:00:00 50.19278
##   lon      species
## 1 -80.69139 Pectoral Sandpiper
## 2 -63.74528      Red Knot
```

L'émetteur 23316 a été déployé dans le cadre du Programme de suivi des oiseaux de rivage de la baie James (projet 176) environ trois semaines après l'émetteur 22867, dont le lieu de déploiement se trouvait loin à l'ouest.

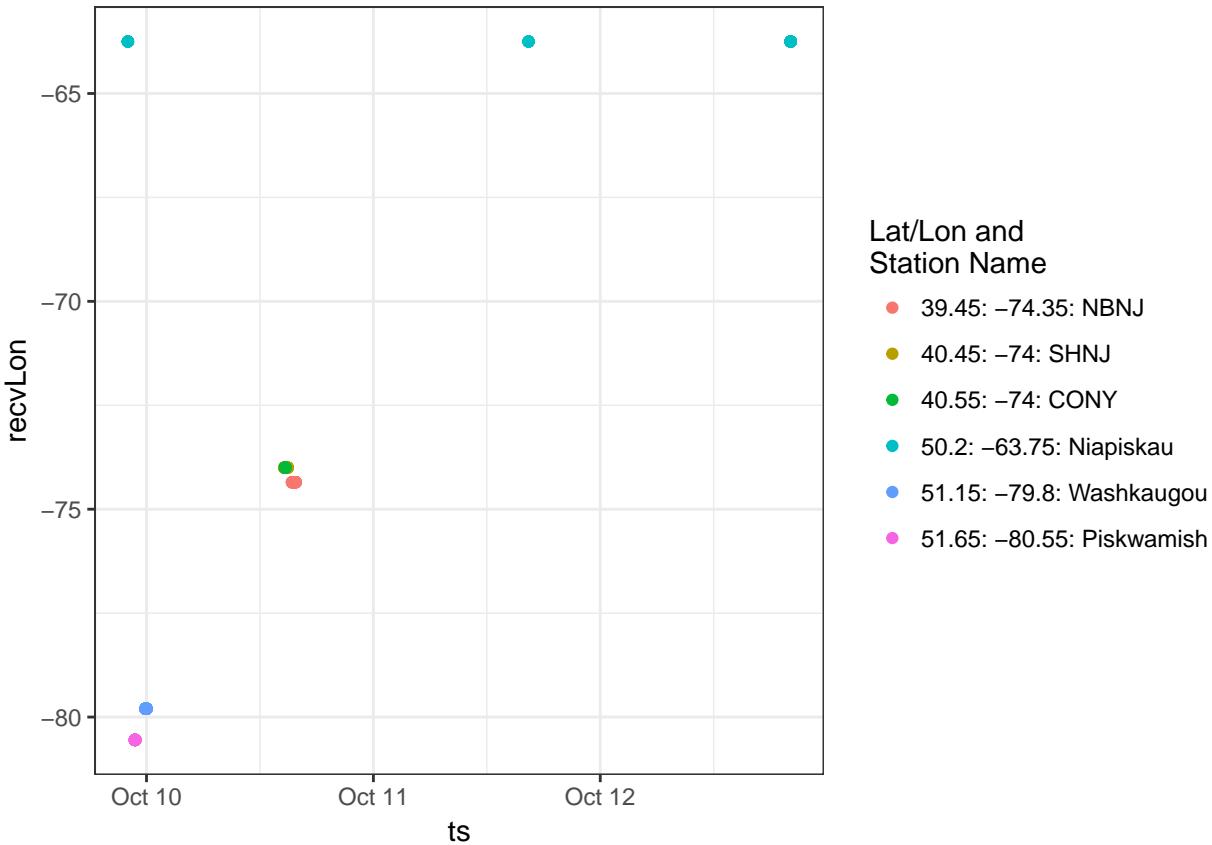
```
df.ambig.56 <- filter(df.alltags.sub, ambigID == -56) %>%
mutate(sig = ifelse(sig > 0, sig * -1, sig))

p <- ggplot(data = df.ambig.56, aes(recvLon, ts, colour = paste(recvLat,
recvLon, recvDeployName, sep = ": ")))
p + geom_point() + theme_bw() + scale_colour_discrete(name = "Lat/Lon and\nStation Name")
```



Nous voyons sur le graphique qu'un émetteur est détecté constamment près de la longitude -65, près du lieu de déploiement de l'émetteur 23316. Après la date du début du déploiement, cet émetteur était également présent à -65 pendant et après les détections effectuées loin à l'ouest. Selon toute probabilité, toutes les détections effectuées à -65 se rapportent à l'émetteur 23316. Toutefois, il est également clair que l'explication de l'ambiguïté se trouve dans la période du 9 au 11 octobre, de sorte que nous devons nous concentrer sur cette partie de l'ensemble de données.

```
ts.begin <- ymd_hms("2016-10-06 00:00:00")
ts.end <- ymd_hms("2016-10-12 23:00:00")
p <- ggplot(data = filter(df.ambig.56, ts > ts.begin,
                           ts < ts.end), aes(ts, recvLon, colour = paste(recvLat,
                           recvLon, recvDeployName, sep = ": ")))
p + geom_point() + theme_bw() + scale_colour_discrete(name = "Lat/Lon and\nStation Name")
```



Nous pouvons constater que l'émetteur ambigu a été détecté constamment à Niapiskau et Grand Île avant et après la période durant laquelle il a aussi été détecté au nord et à l'ouest (à Washkaugou et à Piskwamish) et ensuite au sud (NBNJ, SHNJ et CONY). Nous pouvons examiner cette transition en filtrant la portion des détections non proches de Niapiskau et de nouveau en utilisant la fonction siteTrans du logiciel motus.

```
# L'autre émetteur est un double.
df.56.tmp <- filter(df.ambig.56, !(recvLat == 50.2), motusTagID == 22867)

siteTrans(df.56.tmp, latCoord = "recvLat", lonCoord = "recvLon") %>%
ungroup() %>%
filter(rate < 60) %>% # get rid of simultaneous detections
mutate(total.time = as.numeric(round(seconds_to_period(tot_ts)))) %>%
select(start=recvDeployName.x,
       end=recvDeployName.y,
       date=ts.x, "rate(m/s)" = rate,
       dist, total.time = total.time, bearing)

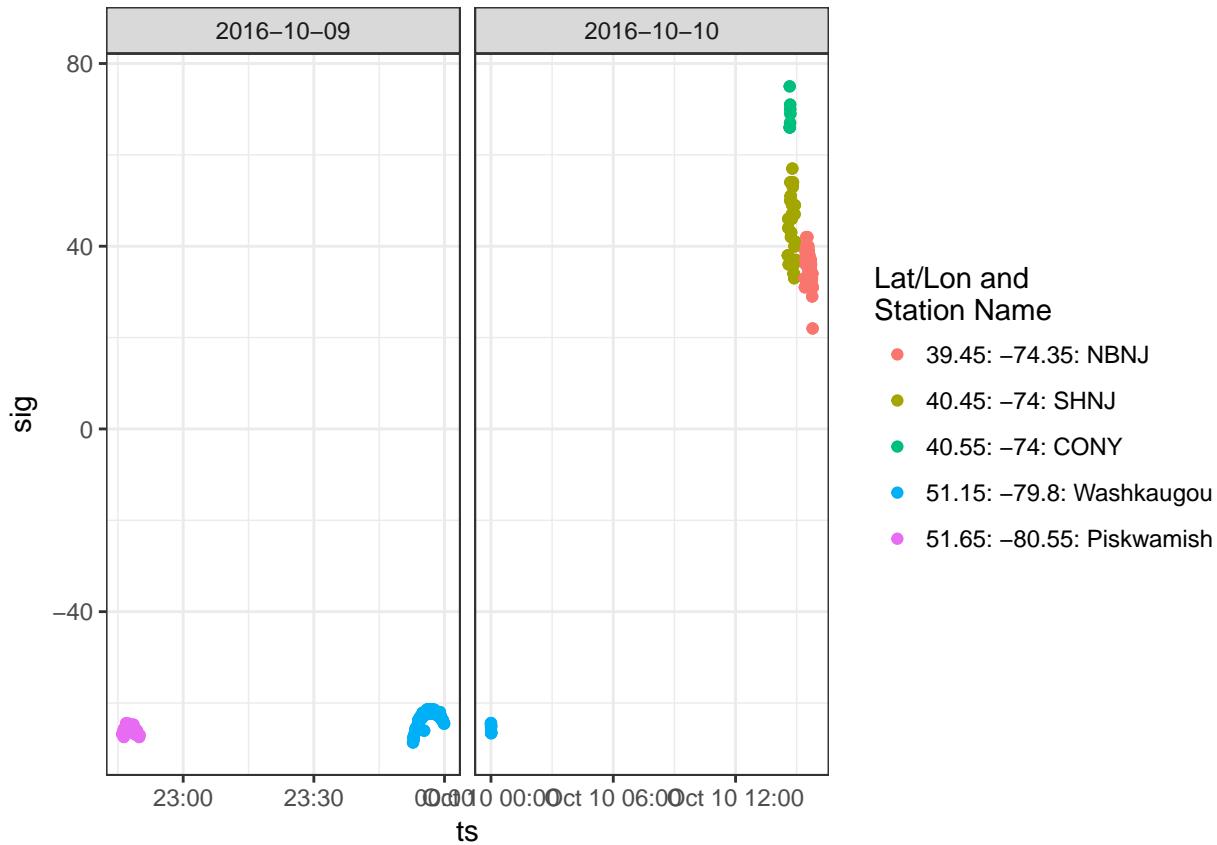
## # A tibble: 2 x 7
##   start   end     date      `rate(m/s)`    dist total.time bearing
##   <chr>   <chr>   <dttm>        <dbl>    <dbl>      <dbl>    <dbl>
## 1 Piskwa~ Washk~ 2016-10-09 22:49:59    20.3  7.63e4      3767     137
## 2 Washka~ SHNJ_~ 2016-10-10 00:00:42    24.3  1.27e6     52386     157
```

Le Bécasseau maubèche a effectué un vol de 14,5 heures entre Washkaugou et SHNJ à la vitesse de 24 m/s, ce qui est plausible. Les chercheurs engagés dans le projet peuvent avoir d'autres données pour soutenir ou réfuter l'hypothèse (par exemple, l'observation de l'individu encore présent à Niapiskau après que le vol a été enregistré), mais il semble probable que, tandis qu'un émetteur est demeuré à différents points aux

environs de la longitude -65, un autre émetteur porté par un individu qui a effectué les vols migratoires susmentionnés a été détecté. Nous pouvons produire un autre graphique plus détaillé de la puissance du signal pour examiner de plus près ces vols migratoires potentiels:

```
df.56.tmp <- filter(df.alltags.sub, ambigID == -56,
                     recvLon < -70)

p <- ggplot(data = df.56.tmp, aes(ts, sig, colour = paste(recvLat,
                recvLon, recvDeployName, sep = " : ")))
p + geom_point() + theme_bw() + scale_colour_discrete(name = "Lat/Lon and\nStation Name") +
    facet_wrap(~as_date(ts), scales = "free_x")
```



Le graphique présente les profils typiques de passages en vol correspondant à la hausse puis à la baisse de la puissance du signal. Cela, ajouté aux profils de détection généraux et aux connaissances sur l'espèce, nous mène à croire que les détections ambiguës peuvent être raisonnablement divisées entre deux individus, l'un détecté en continu aux environs de la longitude -65 (émetteur 23316) et l'autre en migration vers le sud-ouest pendant la même période (émetteur 22867).

Pour résoudre le problème, nous devons créer deux filtres, l'un qui exclut les détections ambiguës de l'émetteur 22867 et l'autre qui exclut certaines détections de l'émetteur 23316. Dans le cas présent, l'approche la plus facile consiste à filtrer en fonction du motusTagID et du recvDeployName.

```
# L'émetteur 23316 était seulement à «Grande Île»
# et à «Niapiskau» et l'émetteur 22867 n'a jamais
# été détecté à ces deux endroits. Nous excluons
# donc toutes les détections non effectuées à
# «Grande Île» et à «Niapiskau» pour motusTag 23316
# et nous faisons l'inverse pour l'émetteur 22867.
```

```
df.block.7 <- filter(df.alltags.sub, ambigID == -56,
  motusTagID == 23316, !(recvDeployName %in% c("Grand-Ile",
    "Niapiskau"))) %>% select(motusTagID, runID) %>%
  distinct()

df.block.8 <- filter(df.alltags.sub, ambigID == -56,
  motusTagID == 22867, recvDeployName %in% c("Grand-Ile",
    "Niapiskau")) %>% select(motusTagID, runID) %>%
  distinct()
```

## 5.5 Vérification de la validité des séquences d'au moins 2 détections

Au début du présent chapitre, nous avons supprimé toutes les détections faisant partie d'une séquence d'au moins 2, car on considère comme très probable qu'elles correspondent à de faux positifs. Maintenant que nous avons nettoyé les données et que nous avons un bon niveau de confiance quant aux détections qu'il reste, vous pourriez décider de revenir en arrière et d'examiner de plus près ces détections faisant partie d'une séquence d'au moins 2. Vous pourriez le faire, par exemple, en réexécutant les différents graphiques décrits dans ce chapitre (en commençant par les graphiques des coordonnées géographiques [latitude et longitude] en fonction du temps) pour voir si certaines de ces détections peuvent être considérées comme acceptables dans le contexte des positions correspondant aux détections acceptées. Il revient à l'utilisateur de déterminer quelles détections sont raisonnables en fonction de la biologie et du comportement de chaque animal portant un émetteur.

## 5.6 Filtrage des données

### 5.6.1 Filtrer et enregistrer le résultat dans le format RDS

Pour filtrer les données, nous pouvons simplement joindre de nouveau les trames de données df.block aux données originales en utilisant une fonction left\_join() puis les éliminer des données:

```
# Combinez les trames de données df.block en une
# seule et ajoutez une probabilité de 0 pour les
# enregistrements filtrés.

df.block.all <- bind_rows(df.block.0, df.block.1, df.block.2,
  df.block.3, df.block.4, df.block.5, df.block.6,
  df.block.7, df.block.8) %>% mutate(probability = 0)

df.alltags.sub <- left_join(df.alltags, df.block.all,
  by = c("runID", "motusTagID")) %>% # Attribuez une probabilité de 1 aux
# enregistrements qui ne seront pas filtrés.
mutate(probability = ifelse(is.na(probability), 1,
  probability)) %>% filter(probability > 0)
```

Maintenant, enregistrons la trame de données locale sous la forme d'un fichier RDS, que nous utiliserons dans le prochain chapitre. Comme il a été mentionné dans la section 3.8, le format RDS conserve la structure des données R, y compris les estampilles temporelles. Le format RDS présente également l'avantage que les sorties d'un flux de travail déterminé sont enregistrées sous la forme d'un fichier plat, auquel on peut accéder de nouveau au moyen d'une simple instruction readRDS.

```
saveRDS(df.alltags.sub, file = "./data/dfAlltagsSub.rds")
```

Et pour lire les données de nouveau:

```
df.alltags.sub <- readRDS("./data/dfAlltagsSub.rds")
```

### 5.6.2 Enregistrer un filtre personnalisé dans la base de données motus et l'appliquer aux données

Comme options de recharge à l'enregistrement des données sous la forme d'un fichier dans le format RDS, le logiciel R Motus intègre des fonctionnalités vous permettant d'enregistrer vos filtres directement dans votre fichier .motus. Une fois que vous aurez enregistré les filtres dans votre base de données, vous pouvez utiliser la fonction `left_join()` tel qu'indiqué précédemment sans avoir à recourir à des trames de données ou à un fichier RDS pour stocker vos données. Pour obtenir de plus amples renseignements sur les fonctions permettant de travailler avec des filtres Motus, reportez-vous à l'annexe D.

```
# Combinez les trames de données df.block en une
# seule et ajoutez une probabilité de 0 pour les
# enregistrements filtrés.
df.block.all <- bind_rows(df.block.0, df.block.1, df.block.2,
                           df.block.3, df.block.4, df.block.5, df.block.6,
                           df.block.7, df.block.8) %>% mutate(probability = 0)

# Créez un nouveau filtre appelé filtAmbigFalsePos
# et intégrer-y les données de la trame
# df.block.all.
tbl.filter = writeRunsFilter(sql.motus, "filtAmbigFalsePos",
                             df = df.block.all, delete = TRUE)

# Obtenez un objet table duquel les enregistrements
# filtrés à partir de tbl.filter.1 ont été retirés.
tbl.alltags.sub <- left_join(tbl.alltags, tbl.filter,
                               by = c("runID", "motusTagID")) %>% mutate(probability = ifelse(is.na(probability),
                               1, probability)) %>% filter(probability > 0)
```

# Chapter 6

## Exploration des données avec le logiciel R de Motus

Une fois que vous avez résolu les problèmes reliés aux faux positifs, s'il y en a, et que vous avez supprimé les faux positifs, vous pouvez commencer à analyser votre ensemble de données nettoyées. Dans ce chapitre, nous présentons des procédures simples avec lesquelles vous pouvez vous familiariser pour visualiser l'ensemble de données nettoyées du projet 176 utilisé ici comme exemple. Vous pouvez modifier ces scripts pour travailler avec vos propres données. Pour obtenir des instructions plus détaillées sur le logiciel R, nous vous recommandons fortement de vous familiariser avec l'ouvrage de Garrett Grolemund et d'Hadley Wickham intitulé «R for Data Science» (<http://r4ds.had.co.nz/>).

### 6.1 Chargement des logiciels requis

Suivez les instructions fournies dans le chapitre 2 pour installer les logiciels suivants, si cela n'est pas déjà fait, avant de charger les données.

```
library(motus)
library(tidyverse)
library(ggmap)

Sys.setenv(TZ = "GMT")
```

### 6.2 Chargement des données

Si vous avez suivi les instructions du chapitre précédent (chapitre 5) et que vous travaillez avec le fichier «df.alltags.sub» dont les données ont été nettoyées, vous pouvez omettre cette étape et passer à la section 6.3.

Autrement, si vous avez enregistré vos données sous la forme d'un fichier RDS, vous pouvez les charger comme suit:

```
df.alltags.sub <- readRDS("./data/dfAlltagsSub.rds") # Modifier le répertoire en sélectionnant le répe
```

Ou, si vous avez appliqué un filtre personnalisé à votre fichier .motus, vous pouvez charger les données .motus du projet 176 que vous avez téléchargées précédemment (voir le chapitre 3 et les nettoyer maintenant). Actuellement, le principal avantage du filtre personnalisé est que vous l'appliquez au fichier .motus, ce qui vous procure plus de souplesse pour appliquer les fonctions dplyr pour gérer et filtrer les données (p. ex., vous pouvez choisir des variables à inclure dans les données qui sont différentes de celles que nous avons incluses

dans le fichier RDS dans le chapitre 5. Cette approche vous permet également d'intégrer plus facilement les nouvelles données ajoutées à votre base de données avec la fonction tagme. Étant donné que nous choisissons les mêmes variables et que nous filtrons les mêmes enregistrements, les opérations suivantes vous donnent le même ensemble de données que le fait l'instruction readRDS ci-dessus:

```
# Chargez le fichier .motus.
proj.num = 176
sql.motus <- tagme(proj.num, update = TRUE, dir = "./data/")
tbl.alltags <- tbl(sql.motus, "alltags")

# Obtenez un objet table du filtre.
tbl.filter = getRunsFilters(sql.motus, "filtAmbigFalsePos")

# Filtrez et convertissez la table en une trame de données, avec quelques modifications.
df.alltags.sub <- left_join(tbl.alltags, tbl.filter, by = c("runID", "motusTagID")) %>%
  mutate(probability = ifelse(is.na(probability), 1, probability),
        recvLat = if_else((is.na(gpsLat) | gpsLat == 0),
                          recvDeployLat,
                          gpsLat),
        recvLon = if_else((is.na(gpsLon) | gpsLon == 0),
                          recvDeployLon,
                          gpsLon),
        recvAlt = if_else(is.na(gpsAlt),
                          recvDeployAlt,
                          gpsAlt)) %>%
  filter(probability > 0) %>%
  select(-noise, -slop, -burstSlop, -done, -bootnum, -codeSet,
         -mfg, -nomFreq, -markerNumber, -markerType, -tagDeployComments,
         -fullID, -deviceID, -recvDeployLat, -recvDeployLon, -recvDeployAlt,
         -speciesGroup, -gpsLat, -gpsLon, -recvAlt, -recvSiteName) %>%
  collect() %>%
  as.data.frame() %>%
  mutate(ts = as_datetime(ts), # Travaillez avec les dates APRÈS la transformation en un fichier plat.
        tagDeployStart = as_datetime(tagDeployStart),
        tagDeployEnd = as_datetime(tagDeployEnd))
```

**Si votre projet est de très grande envergure, il conviendrait peut-être d'en convertir seulement une portion à la trame de données pour éviter des problèmes de mémoire.** La section 3.7 présente en détail la marche à suivre pour filtrer la table avant de produire une trame de données.

Ici nous le faisons en ajoutant un filtre à la commande ci-dessus. Dans le cas présent, nous créons une trame de données seulement pour l'émetteur (motusTagID) 16047, mais vous pouvez décider quelle est la meilleure façon de produire un sous-ensemble de données en fonction de vos besoins (p. ex. par espèce ou par année):

```
# Créez un sous-ensemble pour un seul émetteur,
# pour maintenir une trame de données de petite
# taille.
df.alltags.16047 <- df.alltags.sub %>% filter(motusTagID ==
  16047)
```

### 6.3 Sommaires de données

Nous verrons ici des commandes de base, en commençant par la fonction summary(), qui permet de voir une sélection de variables dans une trame de données:

```

sql.motus %>%tbl("alltags") %>% select(ts, motusTagID,
  runLen, speciesEN, tagDeployLat, tagDeployLon,
  recvDeployLat, recvDeployLon) %>% collect() %>%
  summary()

##      ts          motusTagID       runLen       speciesEN
##  Min.   :1.438e+09   Min.   :10811   Min.   :  2.0  Length:108826
##  1st Qu.:1.476e+09   1st Qu.:22897   1st Qu.: 30.0  Class  :character
##  Median :1.477e+09   Median :22905   Median :122.0  Mode   :character
##  Mean   :1.476e+09   Mean   :22660   Mean   :355.9 
##  3rd Qu.:1.477e+09   3rd Qu.:23316   3rd Qu.:404.0 
##  Max.   :1.498e+09   Max.   :24303   Max.   :2474.0 
##
##    tagDeployLat     tagDeployLon     recvDeployLat     recvDeployLon
##  Min.   :11.12      Min.   :-80.69     Min.   :-42.50     Min.   :-143.68
##  1st Qu.:50.19      1st Qu.:-63.75     1st Qu.: 50.20     1st Qu.:-63.75
##  Median :50.19      Median :-63.75     Median : 50.20     Median :-63.75
##  Mean   :50.14      Mean   :-65.77     Mean   : 49.05     Mean   :-65.64
##  3rd Qu.:50.19      3rd Qu.:-63.75     3rd Qu.: 50.20     3rd Qu.:-63.75
##  Max.   :51.80      Max.   :-63.75     Max.   : 62.89     Max.   :-60.02
##  NA's   :2025      NA's   :2025      NA's   :173        NA's   :173

# Même sommaire pour les données sql filtrées
df.alltags.sub %>% select(ts, motusTagID, runLen, speciesEN,
  tagDeployLat, tagDeployLon, recvLat, recvLon) %>%
  summary()

```

```

##      ts          motusTagID       runLen
##  Min.   :2015-08-03 06:37:11   Min.   :16011   Min.   :  3.0
##  1st Qu.:2016-10-06 10:30:26   1st Qu.:22897   1st Qu.: 25.0
##  Median :2016-10-09 21:49:41   Median :22897   Median : 92.0
##  Mean   :2016-09-04 20:32:07   Mean   :22247   Mean   :230.7 
##  3rd Qu.:2016-10-19 10:41:54   3rd Qu.:22897   3rd Qu.:286.0 
##  Max.   :2017-04-20 22:33:19   Max.   :23316   Max.   :1371.0 
##
##    speciesEN        tagDeployLat     tagDeployLon     recvLat
##  Length:49076      Min.   :50.19      Min.   :-80.69     Min.   :-42.50
##  Class :character   1st Qu.:50.19     1st Qu.:-63.75     1st Qu.: 50.20
##  Mode  :character   Median :50.19     Median :-63.75     Median : 50.20
##                    Mean   :50.34     Mean   :-65.63     Mean   : 49.97
##                    3rd Qu.:50.19     3rd Qu.:-63.75     3rd Qu.: 50.20
##                    Max.   :51.80     Max.   :-63.75     Max.   : 51.82
##                                     NA's   :167
##
##    recvLon
##  Min.   :-80.69
##  1st Qu.:-63.75
##  Median :-63.75
##  Mean   :-65.32
##  3rd Qu.:-63.75
##  Max.   :-62.99
##  NA's   :167

```

Le logiciel dplyr vous permet de résumer facilement les données par groupes, de manipuler les variables ou de créer de nouvelles variables en fonction de vos données.

Nous pouvons manipuler des variables existantes ou en créer de nouvelles au moyen de la fonction `mutate` du logiciel `dplyr`. Ici, nous convertirons `ts` (estampille temporelle) à un format `POSIXct` puis nous créerons une nouvelle variable pour l'année et le jour de l'année (`doy`).

De plus, nous supprimerons l'ensemble de points où les degrés de latitude et de longitude sont manquants. Cela peut être utile dans certains contextes (par exemple si l'on connaît la position approximative du récepteur), mais cela peut aussi entraîner des avertissements ou des erreurs au moment du pointage sur un graphique ou une carte.

```
df.alltags.sub <- df.alltags.sub %>%
  mutate(ts = as_datetime(ts, tz = "UTC"), # Convertissez ts au format POSIXct.
         year = year(ts), # Extrayez l'année depuis ts
         doy = yday(ts)) %>% # Extrayez le numéro du jour de l'année depuis ts.
  filter(!is.na(recvLat))
head(df.alltags.sub)
```

```
##   hitID runID batchID          ts sig sigsd freq freqsd
## 1 45107  8886      53 2015-10-26 11:19:49  52     0    4     0
## 2 45108  8886      53 2015-10-26 11:20:28  54     0    4     0
## 3 45109  8886      53 2015-10-26 11:21:17  55     0    4     0
## 4 45110  8886      53 2015-10-26 11:21:55  52     0    4     0
## 5 45111  8886      53 2015-10-26 11:22:44  49     0    4     0
## 6 199885 23305     64 2015-10-26 11:12:04  33     0    4     0
##   motusTagID ambigID port runLen tagProjID mfgID tagType tagModel
## 1       16047      NA   3     5      176    378      ID NTQB-3-2
## 2       16047      NA   3     5      176    378      ID NTQB-3-2
## 3       16047      NA   3     5      176    378      ID NTQB-3-2
## 4       16047      NA   3     5      176    378      ID NTQB-3-2
## 5       16047      NA   3     5      176    378      ID NTQB-3-2
## 6       16047      NA   1    11      176    378      ID NTQB-3-2
##   tagLifespan tagBI pulseLen tagDeployID speciesID      tagDeployStart
## 1        NA 9.6971     2.5      1839     4670 2015-09-10 18:00:00
## 2        NA 9.6971     2.5      1839     4670 2015-09-10 18:00:00
## 3        NA 9.6971     2.5      1839     4670 2015-09-10 18:00:00
## 4        NA 9.6971     2.5      1839     4670 2015-09-10 18:00:00
## 5        NA 9.6971     2.5      1839     4670 2015-09-10 18:00:00
## 6        NA 9.6971     2.5      1839     4670 2015-09-10 18:00:00
##   tagDeployEnd tagDeployLat tagDeployLon tagDeployAlt recvDeployID
## 1 2016-03-10 18:00:00      51.4839    -80.45        NA     2510
## 2 2016-03-10 18:00:00      51.4839    -80.45        NA     2510
## 3 2016-03-10 18:00:00      51.4839    -80.45        NA     2510
## 4 2016-03-10 18:00:00      51.4839    -80.45        NA     2510
## 5 2016-03-10 18:00:00      51.4839    -80.45        NA     2510
## 6 2016-03-10 18:00:00      51.4839    -80.45        NA     2512
##   recv recvDeployName isRecvMobile recvProjID antType antBearing
## 1 Lotek-159      Shelburne        0      74  yagi-9      127
## 2 Lotek-159      Shelburne        0      74  yagi-9      127
## 3 Lotek-159      Shelburne        0      74  yagi-9      127
## 4 Lotek-159      Shelburne        0      74  yagi-9      127
## 5 Lotek-159      Shelburne        0      74  yagi-9      127
## 6 Lotek-164  BennettMeadow        0      74  yagi-9      243
##   antHeight speciesEN      speciesFR      speciesSci tagProjName
## 1        NA Red Knot Bécasseau maubèche Calidris canutus SampleData
## 2        NA Red Knot Bécasseau maubèche Calidris canutus SampleData
## 3        NA Red Knot Bécasseau maubèche Calidris canutus SampleData
```

```

## 4      NA Red Knot Bécasseau maubèche Calidris canutus SampleData
## 5      NA Red Knot Bécasseau maubèche Calidris canutus SampleData
## 6      NA Red Knot Bécasseau maubèche Calidris canutus SampleData
##   recvProjName gpsAlt filterID probability recvLat   recvLon year doy
## 1      <NA>     NA     NA           1 42.60699 -72.71657 2015 299
## 2      <NA>     NA     NA           1 42.60699 -72.71657 2015 299
## 3      <NA>     NA     NA           1 42.60699 -72.71657 2015 299
## 4      <NA>     NA     NA           1 42.60699 -72.71657 2015 299
## 5      <NA>     NA     NA           1 42.60699 -72.71657 2015 299
## 6      <NA>     NA     NA           1 42.68067 -72.47392 2015 299

```

Nous pouvons aussi résumer l'information par groupes, dans le cas présent motusTagID, et appliquer différentes fonctions à ces groupes, par exemple pour obtenir le nombre total de détections (n) de chaque émetteur, le nombre de récepteurs qui ont détecté chaque émetteur, les premières et dernières dates de détection et le nombre total de jours où il y a eu au moins une détection:

```

tagSummary <- df.alltags.sub %>% group_by(motusTagID) %>%
  summarize(nDet = n(), nRecv = length(unique(recvDeployName)),
            tsMin = min(ts), tsMax = max(ts), totDay = length(unique(doy)))

```

```
head(tagSummary)
```

```

## # A tibble: 6 x 6
##   motusTagID  nDet  nRecv tsMin          tsMax      totDay
##   <int>    <int> <int> <dttm>       <dttm>      <int>
## 1 16011     122     5 2015-08-03 06:37:11 2015-08-05 20:41:12 3
## 2 16035     430     5 2015-08-14 17:53:49 2015-09-02 14:06:09 6
## 3 16036      62     1 2015-08-17 21:56:44 2015-08-17 21:58:52 1
## 4 16037    1296     3 2015-08-23 15:13:57 2015-09-08 18:37:16 14
## 5 16038      73     1 2015-08-20 18:42:33 2015-08-22 22:19:37 3
## 6 16039    1050    10 2015-08-23 02:28:45 2015-09-19 06:08:31 8

```

Nous pouvons aussi former des groupements en fonction de multiples variables. En appliquant la même fonction que ci-dessus, mais en groupant par motusTagID et recvDeployName, nous obtiendrons de l'information pour chaque émetteur détecté par chaque récepteur. Comme nous groupons par recvDeployName, il y aura par défaut seulement un recvDeployName dans chaque groupe, de sorte que la variable nRecv sera 1 pour chaque ligne. Cela ne fournit pas beaucoup d'information, mais nous l'incluons pour montrer comment fonctionnent les groupements:

```

tagRecvSummary <- df.alltags.sub %>% group_by(motusTagID,
  recvDeployName) %>% summarize(nDet = n(), nRecv = length(unique(recvDeployName)),
  tsMin = min(ts), tsMax = max(ts), totDay = length(unique(doy)))

```

```
head(tagRecvSummary)
```

```

## # A tibble: 6 x 7
## # Groups:   motusTagID [2]
##   motusTagID recvDeployName  nDet  nRecv tsMin          tsMax      totDay
##   <int> <chr>        <int> <int> <dttm>
## 1 16011 North Bluff     122     1 2015-08-03 06:37:11
## 2 16035 Brier2         41     1 2015-09-02 14:03:19
## 3 16035 D'Estimauville 32     1 2015-09-02 07:58:43
## 4 16035 Netitishi      286     1 2015-08-14 17:53:49
## 5 16035 Southwest Head 65     1 2015-09-02 13:06:13
## 6 16035 Swallowtail     6     1 2015-09-02 13:21:27
## # ... with 2 more variables: tsMax <dttm>, totDay <int>

```

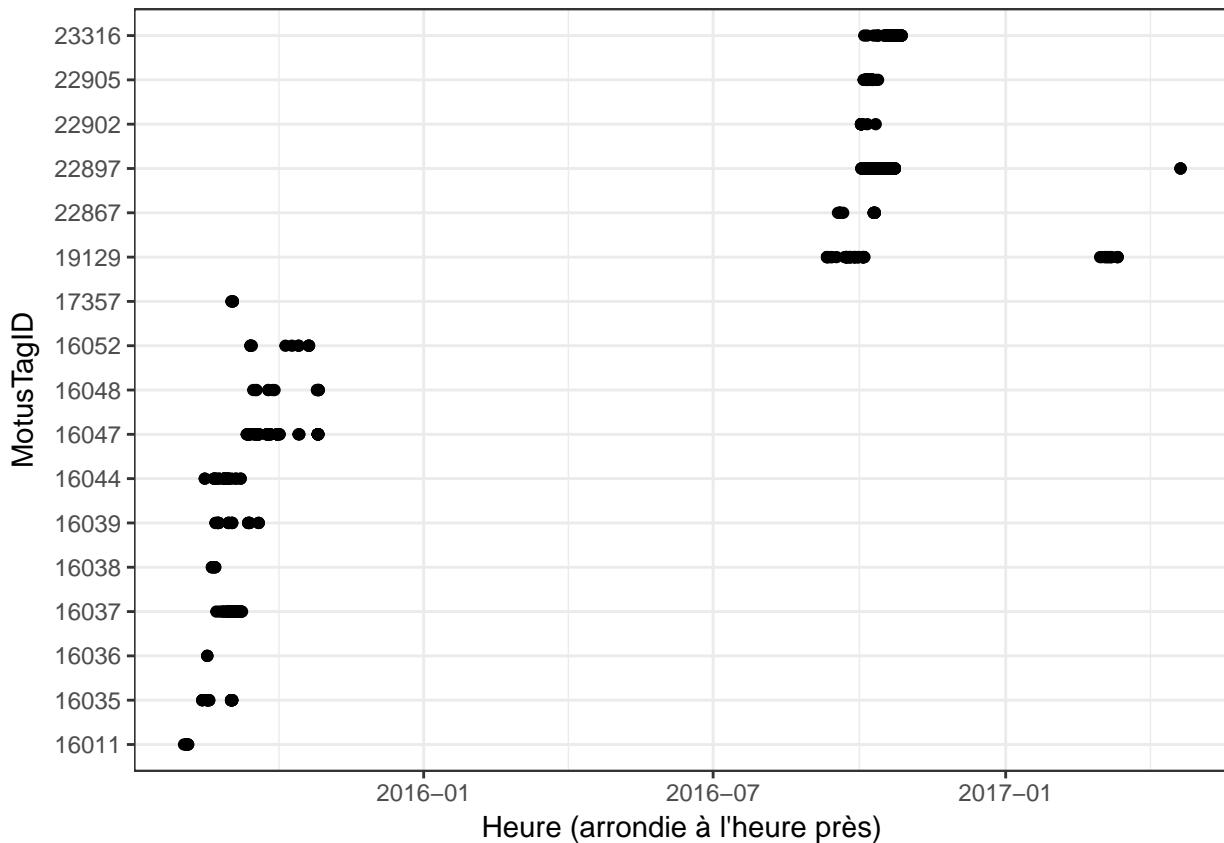
## 6.4 Graphiques de données

Le graphique est un puissant outil de visualisation des profils de détection à grande et à petite échelles. Nous présentons ici une brève introduction à la production de graphiques au moyen de ggplot2. Nous vous recommandons de consulter l'ouvrage intitulé *Cookbook for R* et l'aide-mémoire *ggplot2 cheatsheet* de rstudio pour obtenir de l'information plus détaillée sur les utilisations de ggplot2.

Pour produire des graphiques à échelle grossière avec des fichiers volumineux, nous suggérons de commencer par arrondir les heures de détection à l'heure près ou au jour près afin de réduire le temps de traitement. Ici, nous arrondissons à l'heure près puis nous produisons un graphique de base des détections aux heures par émetteur (motusTagID):

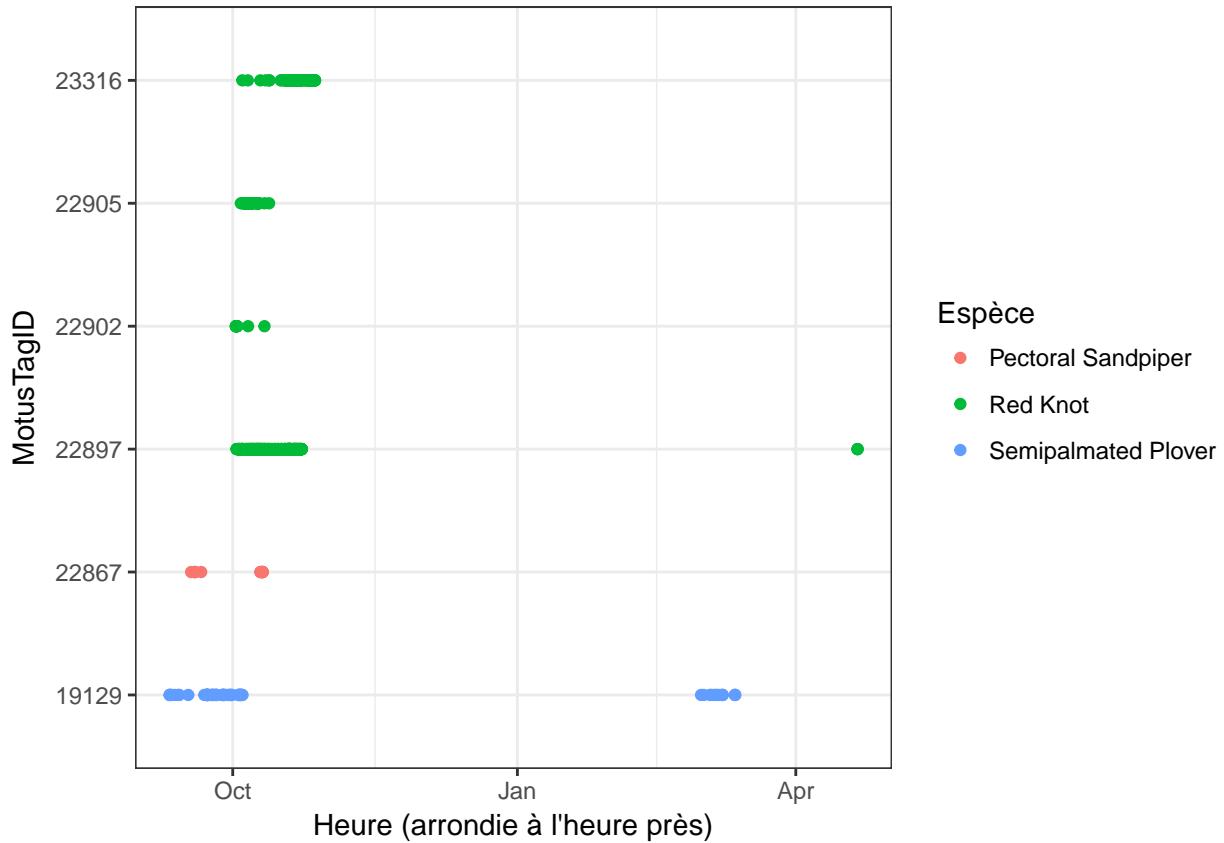
```
df.alltags.sub.2 <- mutate(df.alltags.sub, hour = as.POSIXct(round(ts,
  "hour")))) %>% select(motusTagID, port, tagDeployStart,
  tagDeployLat, tagDeployLon, recvLat, recvLon, recvDeployName,
  antBearing, speciesEN, year, doy, hour) %>% distinct()

p <- ggplot(data = df.alltags.sub.2, aes(hour, as.factor(motusTagID)))
p + geom_point() + ylab("MotusTagID") + xlab("Heure (arrondie à l'heure près)") +
  theme_bw()
```



Concentrons-nous seulement sur les émetteurs déployés en 2016, auxquels nous pouvons attribuer une couleur pour chaque espèce:

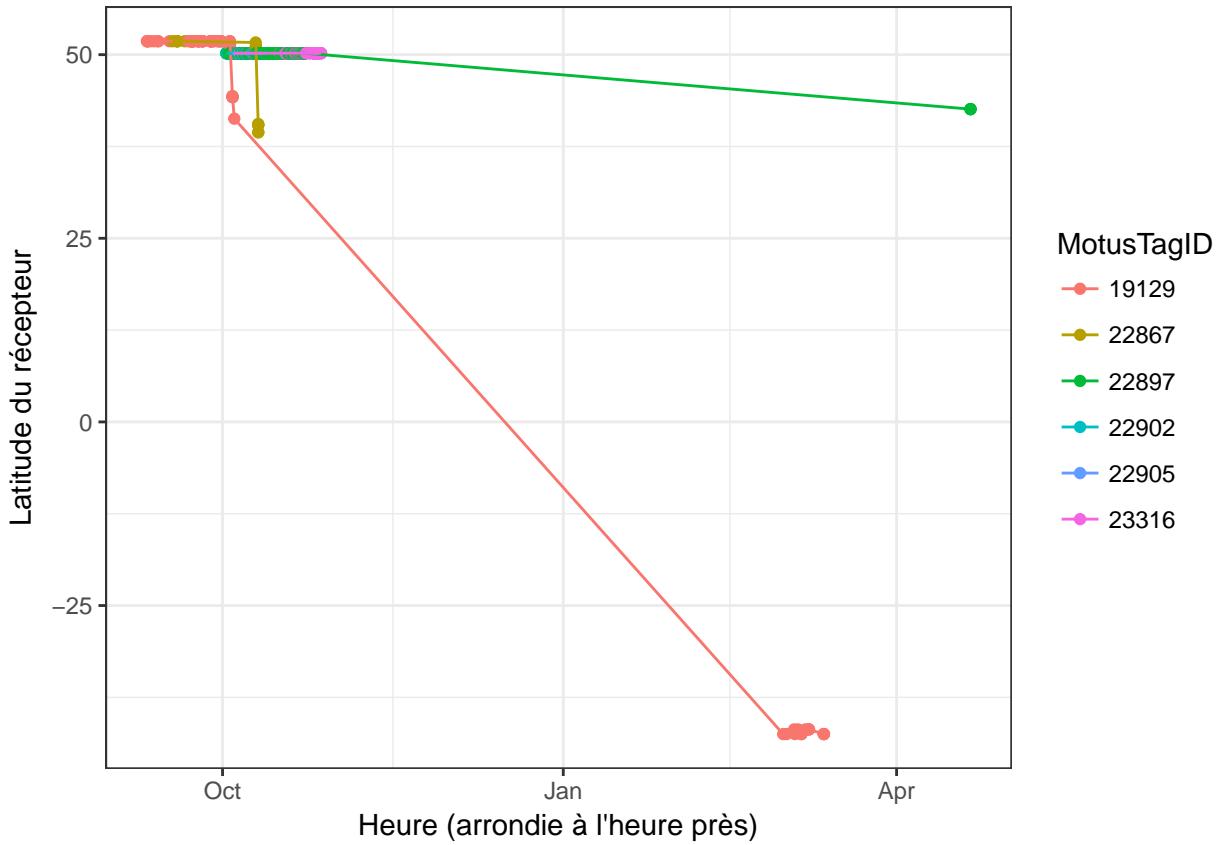
```
p <- ggplot(data = filter(df.alltags.sub.2, year(tagDeployStart) ==
  2016), aes(hour, as.factor(motusTagID), col = speciesEN))
p + geom_point() + ylab("MotusTagID") + xlab("Heure (arrondie à l'heure près)") +
  scale_colour_discrete(name = "Espèce") + theme_bw()
```



En commençant par ordonner en fonction de l'heure et attribuer une couleur aux motusTagID selon l'espèce, nous pouvons voir comment les émetteurs se sont déplacés en latitude:

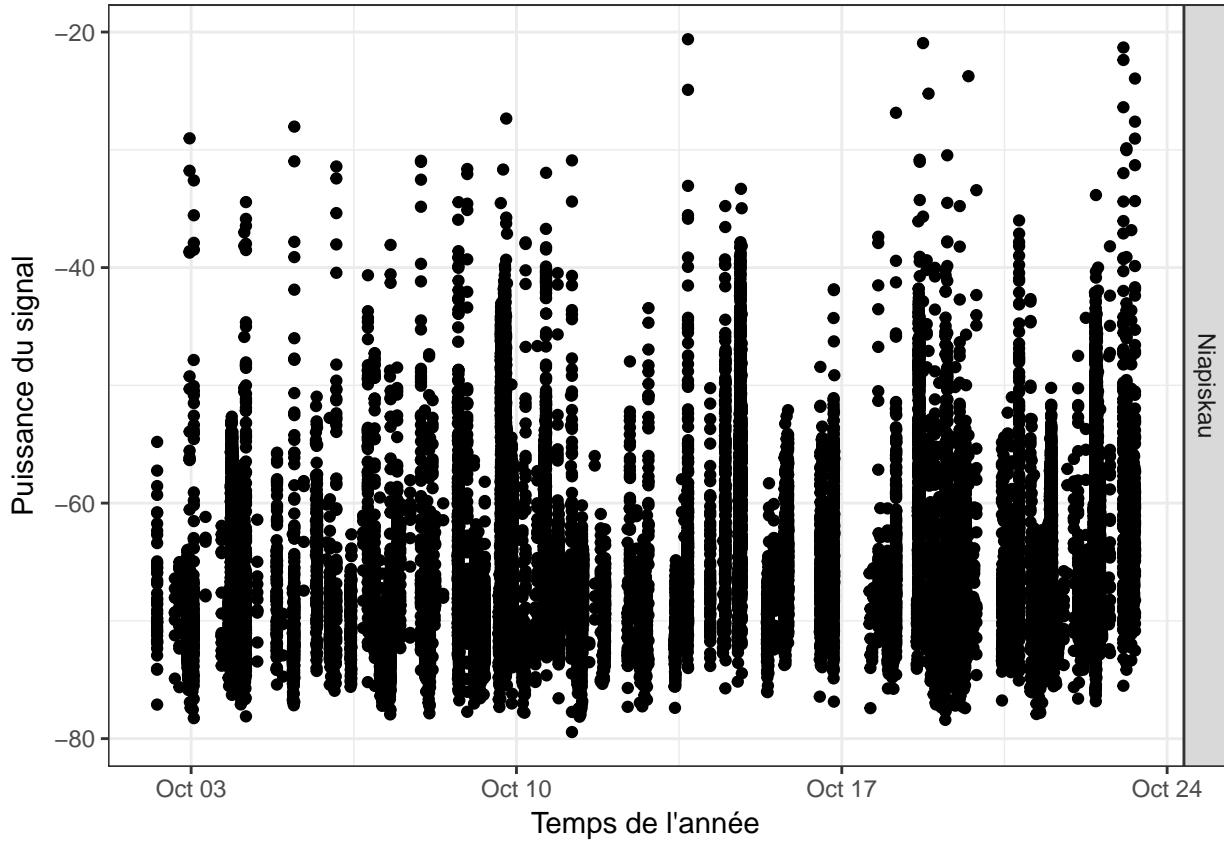
```
df.alltags.sub.2 <- arrange(df.alltags.sub.2, hour)

p <- ggplot(data = filter(df.alltags.sub.2, year(tagDeployStart) == 2016), aes(hour, recvLat, col = as.factor(motusTagID),
group = as.factor(motusTagID)))
p + geom_point() + geom_path() + theme_bw() + xlab("Heure (arrondie à l'heure près)") +
ylab("Latitude du récepteur") + scale_colour_discrete(name = "MotusTagID")
```

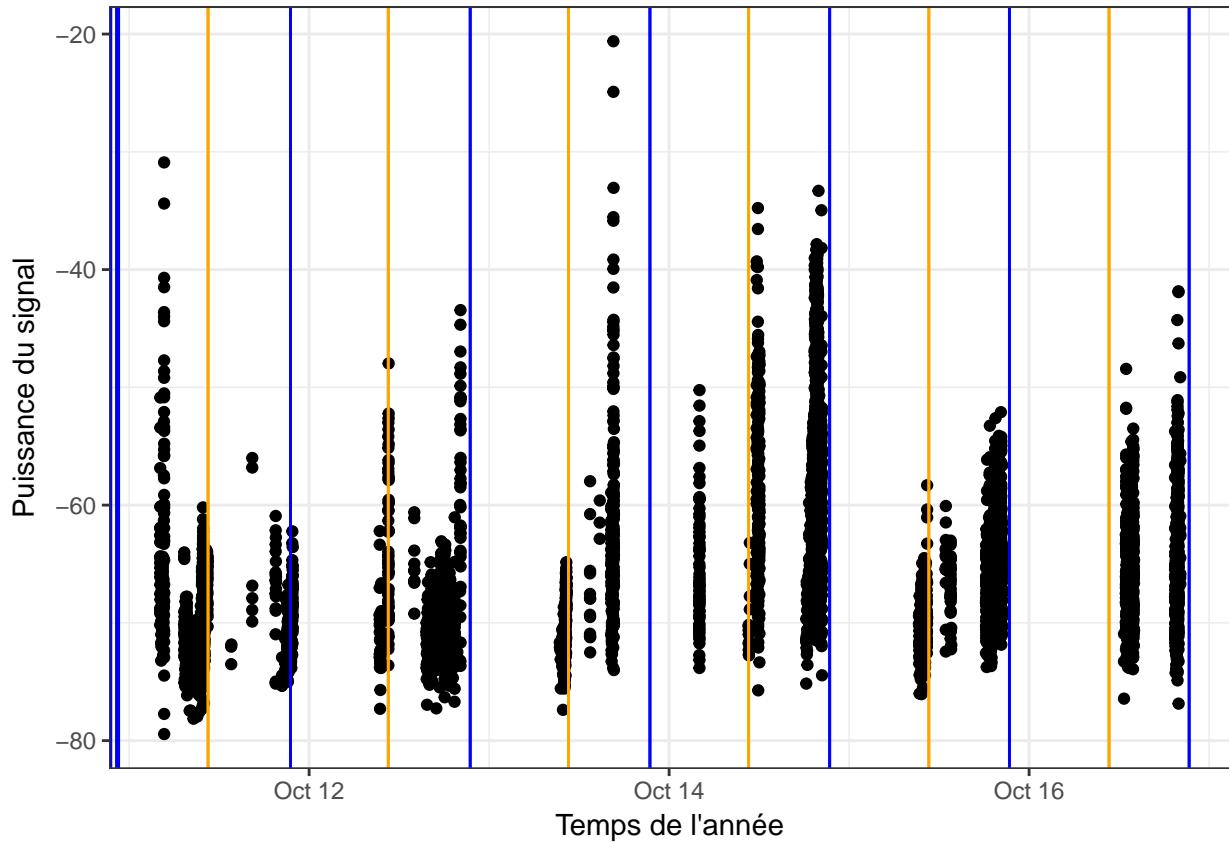


Maintenant, examinons des graphiques plus détaillés de la variation des signaux. Nous utilisons toute la trame de données df.alltags.sub afin d'obtenir la puissance du signal pour chaque détection d'un émetteur particulier. Examinons les détections des signaux de l'émetteur 22897 à Niapiskau effectuées durant l'automne 2016; nous façonnons le graphique par nom de déploiement, dans l'ordre décroissant de la latitude:

```
p <- ggplot(filter(df.alltags.sub, motusTagID == 22897,
  recvDeployName == "Niapiskau"), aes(ts, sig))
p + theme_bw() + geom_point() + xlab("Temps de l'année") +
  ylab("Puissance du signal") + facet_grid(recvDeployName ~
  .)
```



Nous utilisons la fonction sunRiseSet accessible dans le logiciel R de Motus (voir C.2) pour obtenir les heures du lever et du coucher du soleil pour toutes les détections. Ensuite, nous nous concentrerons sur une certaine trame de données et nous ajoutons cette information au graphique ci-dessus en ajoutant une instruction geom\_vline() au code, ce qui a pour effet de produire une ligne jaune pour l'heure du lever du soleil et une ligne bleue pour l'heure du coucher du soleil:



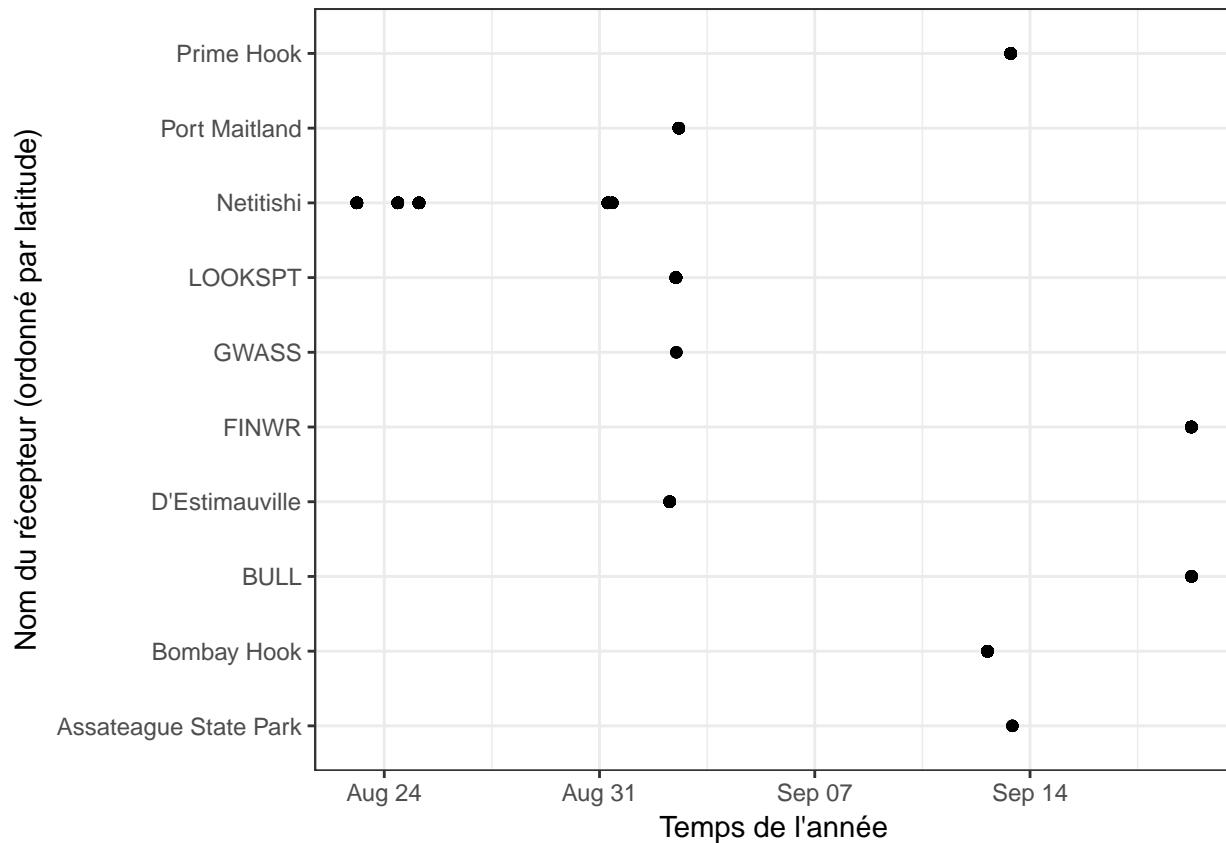
Nous constatons que durant la période en question, c'est pendant le jour que l'émetteur a été détecté le plus souvent, ce qui semble indiquer que l'oiseau était actif à la recherche de nourriture dans le secteur en question durant cette période.

Les mêmes graphiques peuvent fournir de précieux renseignements sur les déplacements lorsque les récepteurs sont ordonnés géographiquement, ce que nous faisons pour l'émetteur 16039:

```
# Ordonnons d'abord sitelat par latitude (pour les
# points)
df.alltags.sub <- mutate(df.alltags.sub, recvDeployName = as.factor(as.character(reorder(recvDeployName
recvLat))))
```

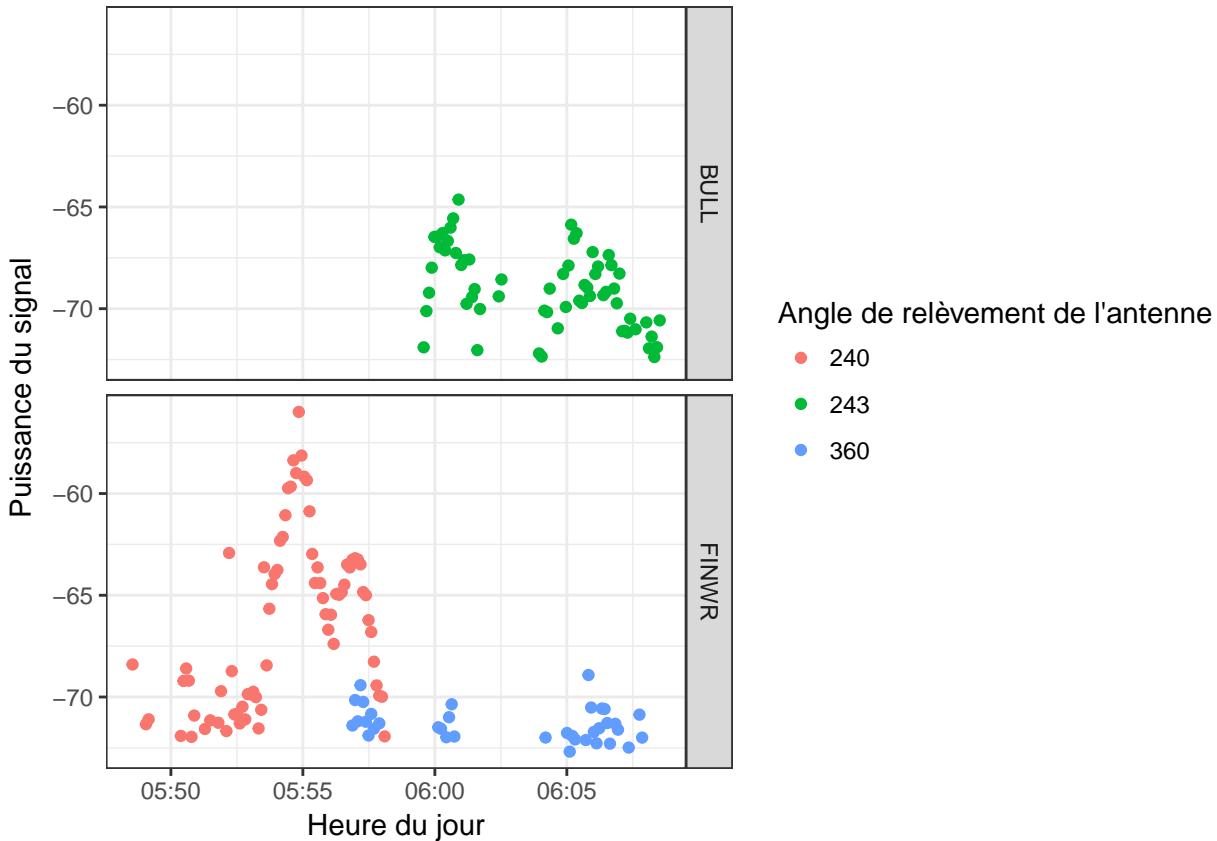
```
p <- ggplot(filter(df.alltags.sub, motusTagID == 16039 &
ts < ymd("2015-10-01")), aes(ts, recvDeployName))
```

```
p + theme_bw() + geom_point() + xlab("Temps de l'année") +
ylab("Nom du récepteur (ordonné par latitude)")
```



Nous nous concentrons sur une section de ce graphique et examinons les angles de relèvement des antennes pour déterminer les directions de vol à la hauteur des stations réceptrices:

```
p <- ggplot(filter(df.alltags.sub, motusTagID == 16039,
  ts > ymd("2015-09-14"), ts < ymd("2015-10-01")),
  aes(ts, sig, col = as.factor(antBearing)))
p + theme_bw() + geom_point() + xlab("Heure du jour") +
  ylab("Puissance du signal") + scale_color_discrete(name = "Angle de relèvement de l'antenne") +
  facet_grid(recvDeployName ~ .)
```



Ce graphique indique le profil type du passage d'un animal en migration: la puissance du signal de l'émetteur augmente puis diminue au moment où l'animal passe dans le faisceau des antennes.

## 6.5 Cartes de données

Pour générer des cartes de trajectoires d'émetteurs, nous utiliserons de nouveau des données résumées de manière à travailler avec une base de données beaucoup plus petite et à accélérer le traitement. Ici, nous résumerons les détections par date. Comme nous l'avons fait au chapitre 5, nous créons une fonction simple pour résumer les données vu que nous devrons probablement répéter cette opération à maintes reprises.

```
# Simplifiez les données en produisant un sommaire en fonction du runID.
# Si vous voulez produire un sommaire à une échelle plus fine ou plus grossière, vous pouvez aussi créer
# L'option de recharge la plus simple est une variable d'estampille temporelle arrondie; par exemple en
# l'appel de fonction mutate(ts.h = plyr::round_any(ts, 3600)).
# Une autre option consiste à utiliser juste la date (p. ex. date = as_date(ts)).

#
fun.getpath <- function(df)
{
  df %>%
    filter(tagProjID == proj.num, # Conservez seulement les émetteurs enregistrés dans le cadre du projet
          !is.na(recvLat) | !(recvLat == 0)) %>%
    group_by(motusTagID, runID, recvDeployName, ambigID,
             tagDeployLon, tagDeployLat, recvLat, recvLon) %>%
    summarize(max.runLen = max(runLen), ts.h = mean(ts)) %>%
```

```

    arrange(motusTagID, ts.h) %>%
    data.frame()
} # Fin de l'appel de fonction

df.alltags.path <- fun.getpath(df.alltags.sub)

df.alltags.sub.path <- df.alltags.sub %>%
  filter(tagProjID == proj.num) %>% # Conservez seulement les émetteurs enregistrés dans
  arrange(motusTagID, ts) %>%       # Ordonnez les données par estampille temporelle pour
  mutate(date = as_date(ts)) %>%     # Créez une variable date.
  group_by(motusTagID, date, recvDeployName, ambigID,
           tagDeployLon, tagDeployLat, recvLat, recvLon)

df.alltags.path <- fun.getpath(df.alltags.sub.path)

```

### 6.5.1 Utilisation de Google Maps

L'utilisation de Google Maps peut permettre de visualiser rapidement des trajectoires de vol et de choisir parmi plusieurs couches de base.

La première étape consiste à créer une carte ayant un centre précis (types de carte: relief, plan, satellite ou hybride) et à choisir le niveau de zoom (nombre entier de 3 à 21, 3 correspondant au niveau du continent et 10 au niveau de la localité). Ensuite, nous ajoutons des points indiquant les positions des récepteurs et des lignes reliant les stations consécutives où chaque émetteur (motusTagID) a été détecté. Nous pouvons également ajouter des points pour toutes les stations réceptrices qui étaient actives durant une certaine période si nous avons déjà téléchargé toutes les métadonnées.

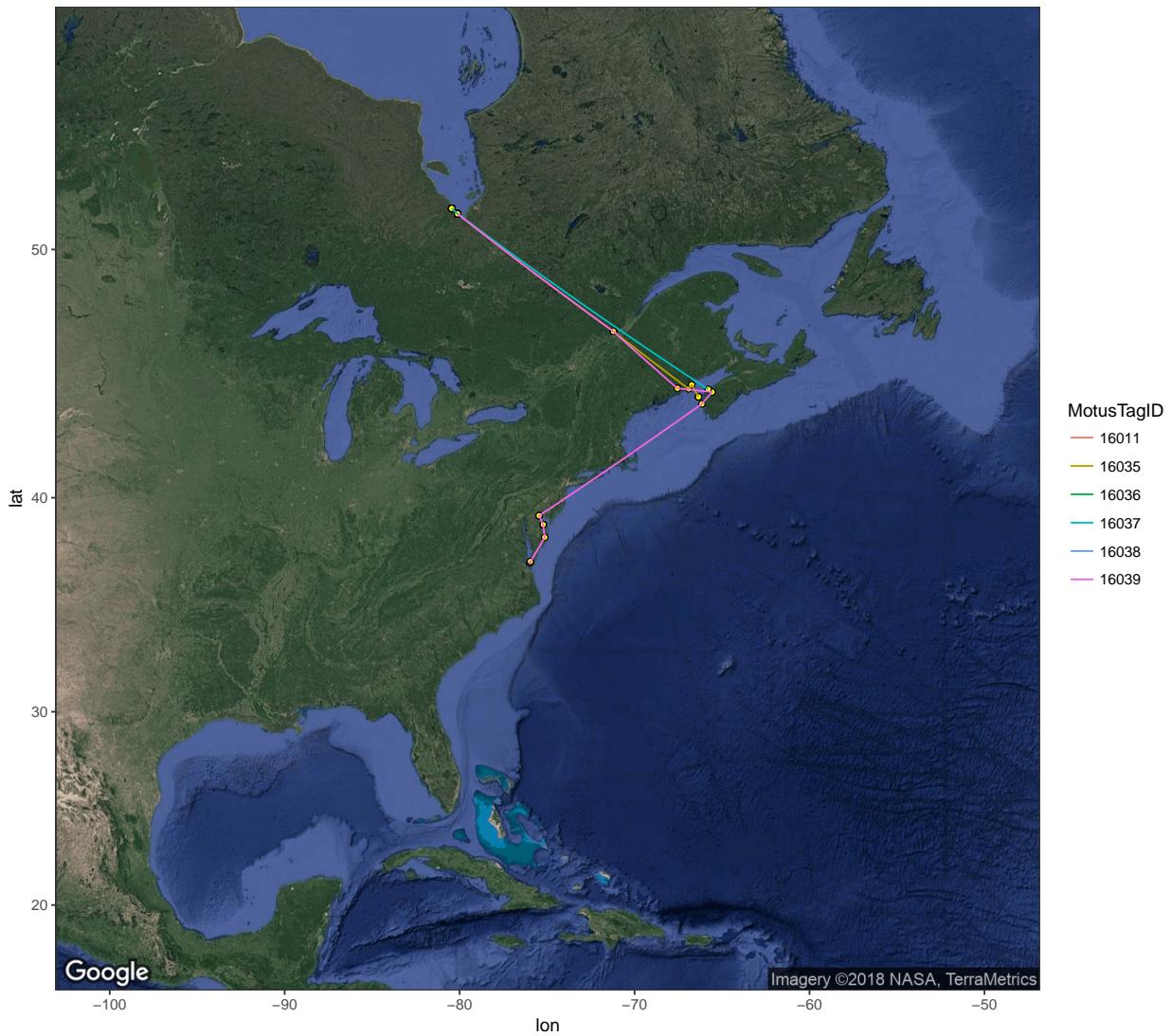
```

gmap <- get_map(location = c(lon = -75, lat = 40), # Centrez la carte (lon/lat).
                 maptype = "satellite", # Choisissez le type de carte.
                 source = "google",
                 zoom = 4) # Entrez le niveau de zoom (nombre entier).

# Utilisez seulement les émetteurs dont nous avons examiné attentivement et filtré les détections (dans
df.tmp <- filter(df.alltags.path,
                  motusTagID %in% c(16011, 16035, 16036, 16037, 16038, 16039))
df.tmp <- arrange(df.tmp, ts.h) # Disposez par heure.
df.tmp <- as.data.frame(df.tmp)

p <- ggmap(gmap)
p + geom_point(data=df.tmp,
                aes(recvLon, recvLat), pch=21, colour = "black", fill = "yellow") +
  geom_path(data=df.tmp,
            aes(recvLon, recvLat, group=motusTagID, col = as.factor(motusTagID))) +
  theme_bw() +
  scale_color_discrete(name="MotusTagID")

```

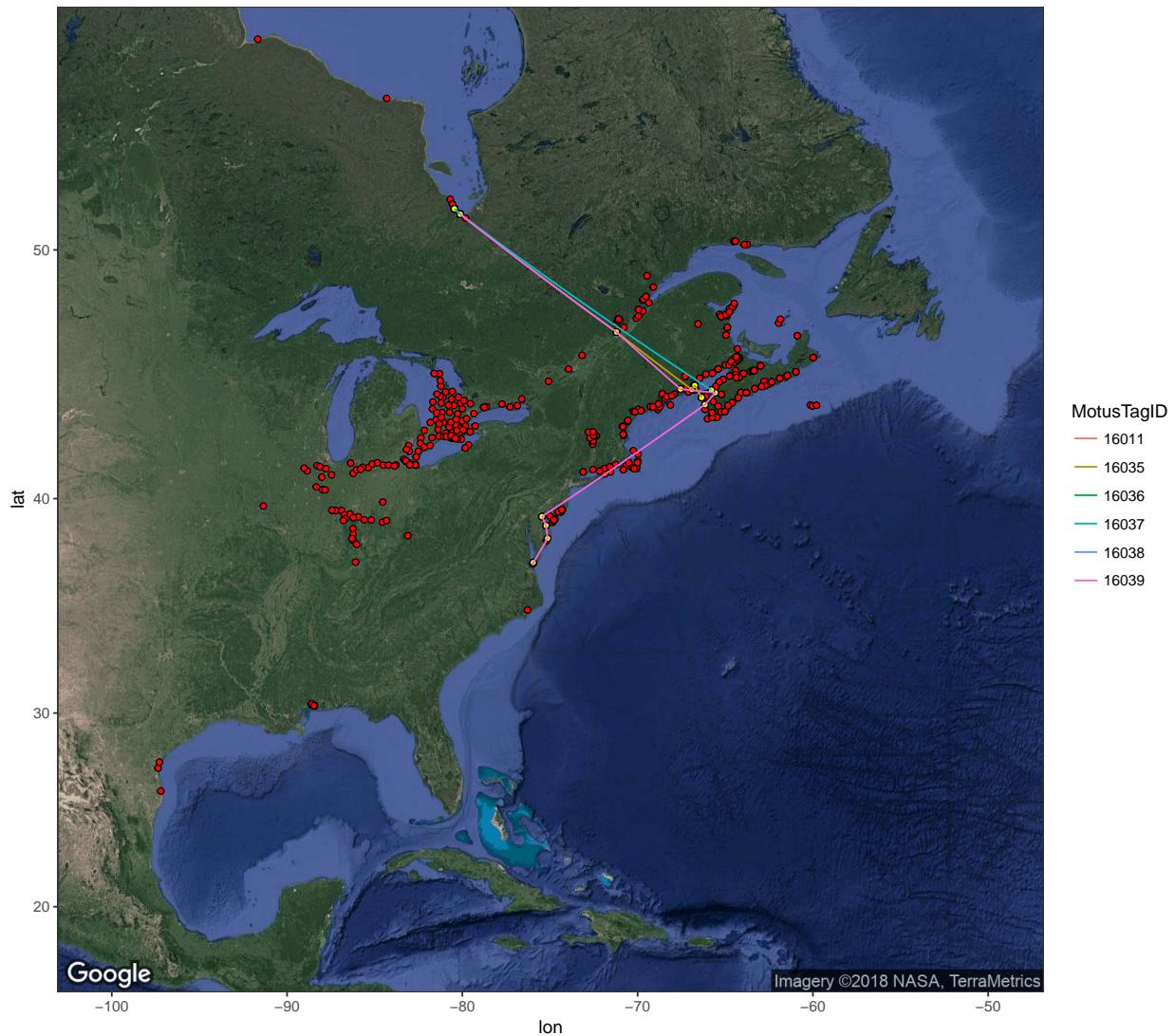


Nous produisons la même carte avec des points supplémentaires correspondant aux stations réceptrices qui étaient actives pendant une période déterminée:

```
# Obtenez les métadonnées sur les stations
# réceptrices.
tbl.recvDeps <- tbl(sql.motus, "recvDeps")
df.recvDeps <-tbl.recvDeps %>% collect %>% as.data.frame() %>%
  mutate(tsStart = as_datetime(tsStart, tz = "UTC",
    origin = "1970-01-01"), tsEnd = as_datetime(tsEnd,
    tz = "UTC", origin = "1970-01-01"))
# Dans le cas des déploiements sans date de fin,
# entrez une date de fin équivalant à un an plus
# tard.
df.recvDeps$tsEnd <- as.POSIXct(ifelse(is.na(df.recvDeps$tsEnd),
  as.POSIXct(format(Sys.time(), "%Y-%m-%d %H:%M:%S")) +
```

```
lubridate::dyears(1), df.recvDeps$tsEnd), tz = "UTC",
origin = "1970-01-01")
# Obtenez les intervalles de fonctionnement pour
# tous les récepteurs déployés.
siteOp <- with(df.recvDeps, lubridate::interval(tsStart,
tsEnd)) # Obtenez les intervalles de fonctionnement pour chaque déploiement.
# Fixez l'intervalle de temps (dates) qui vous
# intéresse.
dateRange <- lubridate::interval(as.POSIXct("2015-08-01"),
as.POSIXct("2016-01-01"))
# Créez une nouvelle variable, «active», qui est
# établie à TRUE si le récepteur était actif à un
# moment donné pendant l'intervalle de temps que
# vous avez fixé, et à FALSE autrement.
df.recvDeps$active <- lubridate::int_overlaps(siteOp,
dateRange)

# Créez une carte montrant en rouge les positions
# des récepteurs qui étaient actifs pendant
# l'intervalle de temps fixé et en jaune les
# positions des récepteurs qui ont détecté des
# émetteurs.
p <- ggmap(gmap)
p + geom_point(data = subset(df.recvDeps, active ==
TRUE), ggplot2::aes(longitude, latitude), pch = 21,
colour = "black", fill = "red") + geom_point(data = df.tmp,
aes(recvLon, recvLat), pch = 21, colour = "black",
fill = "yellow") + geom_path(data = df.tmp, aes(recvLon,
recvLat, group = motusTagID, col = as.factor(motusTagID))) +
theme_bw() + scale_color_discrete(name = "MotusTagID")
```



### 6.5.2 Création de cartes muettes simples

Nous chargeons les cartes de base.

```
na.lakes <- map_data(map = "lakes")
na.lakes <- na.lakes %>% mutate(long = long - 360)

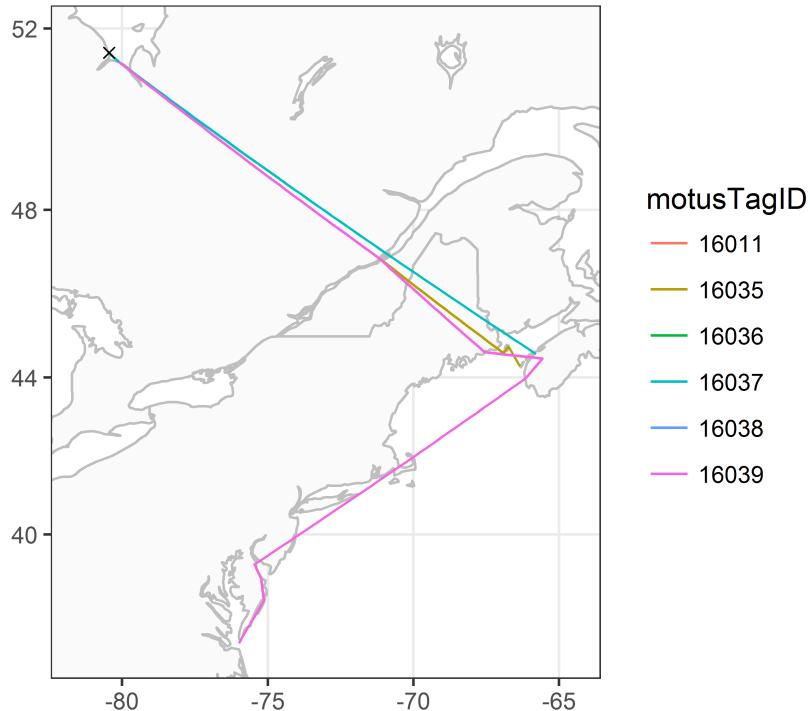
# Incluez toutes les Amériques pour commencer.
na.map <- map_data(map = "world2")
na.map <- na.map %>% filter(region %in% c("Canada",
    "USA")) %>% mutate(long = long - 360)
```

Ensuite, pour tracer les trajectoires, il faut fixer les limites de l'axe des x et de l'axe des y selon la position

des récepteurs qui ont détecté des émetteurs. Selon vos données, il faudra peut-être modifier ces limites pour que les positions de déploiement des émetteurs soient incluses si les émetteurs n'ont pas été déployés à proximité de tours de stations où des émetteurs ont été détectés. Ensuite, à l'aide de ggplot, nous produisons la carte et traçons les trajectoires des émetteurs. Ici, nous utilisons la projection de Mercator et nous colorons les trajectoires différemment pour chaque émetteur (motusTagID), en incluant le point où il a été déployé:

```
# Fixez les limites de la carte selon la position
# des récepteurs qui ont détecté des émetteurs en
# vous assurant que les positions de déploiement
# des émetteurs soient incluses.
xmin <- min(df.tmp$recvLon, na.rm = TRUE) - 2
xmax <- max(df.tmp$recvLon, na.rm = TRUE) + 2
ymin <- min(df.tmp$recvLat, na.rm = TRUE) - 1
ymax <- max(df.tmp$recvLat, na.rm = TRUE) + 1

# map
ggplot(na.lakes, aes(long, lat)) + geom_polygon(data = na.map,
  aes(long, lat, group = group), colour = "grey",
  fill = "grey98") + geom_polygon(aes(group = group),
  colour = "grey", fill = "white") + coord_map(projection = "mercator",
  xlim = c(xmin, xmax), ylim = c(ymin, ymax)) + xlab("") +
  ylab("") + theme_bw() + geom_path(data = df.tmp,
  aes(recvLon, recvLat, group = as.factor(motusTagID),
  colour = as.factor(motusTagID))) + geom_point(data = df.tmp,
  aes(tagDeployLon, tagDeployLat), colour = "black",
  shape = 4) + scale_colour_discrete("motusTagID")
```



Les fonctions dont il a été question dans le présent chapitre sont des exemples de la façon dont vous pouvez commencer à explorer vos données; il peut certainement exister d'autres façons de faire. L'annexe B ci-après présente des erreurs fréquentes qui peuvent survenir pendant le téléchargement et l'utilisation des fichiers sql

.motus. ainsi que les solutions pour corriger ces erreurs.

# Appendix A

## Appendix - alltags structure

The following variables are included in each ‘alltags’ view in the SQLite file:

Field	Description
hitID	unique Motus ID for this tag detection
runID	unique Motus ID for the run this detection belongs to
batchID	unique Motus ID for the processing batch this detection came from
ts	timestamp, in seconds since 1 Jan, 1970 GMT; precision: 0.1 ms (SG); 2.5 ms (Lotek).
sig	signal strength in “native units”; for SG: dB (max) (logarithmic, 0 = max possible, -10 = 0.1 * max, etc.); for Lotek: raw value (0...255)
sigsd	std. dev. in signal strength among pulses in this burst. SG Only; NA for Lotek
noise	estimate of background radio noise when tag detected, in dB (max) for SG; NA for Lotek
freq	frequency offset from antenna listening frequency, in kHz for SG only; NA for Lotek
freqsd	std. dev. of freq offset among pulses in this burst, in kHz. Values larger than 0.1 kHz suggest a bogus detection. SG only; NA for Lotek.
slop	total absolute difference (milliseconds) in inter-pulse intervals for this burst between registration and detection. SG only; NA for Lotek
burstSlop	signed difference (seconds) between detection and registration burst intervals. This is always 0 for the first burst in a run (see posInRun)
done	logical: is run finished?
motusTagID	Motus tag ID - unique to each individual tag registered
ambigID	ambiguous ID assigned to ambiguous tags
port	the port number that the detection occurred on
runLen	number of tag bursts in the current run; constant for all records having the same runID
bootnum	boot session of receiver for SG; NA for Lotek
tagProjectID	ID of the project that manages this tag.
mfgID	manufacturer ID
tagType	for coded tags, the name of the codeset (e.g. ‘Lotek-3’)
codeSet	tag manufacturer
mfg	manufacturer’s model name for a tag (e.g. ‘NTQB-3-2’)
tagModel	estimated lifespan of tag (days)
tagLifespan	

Field	Description
nomFreq	nominal tag frequency (MOTUS: Nominal frequency receiver was tuned to, in Mhz. This really only applies to SG, where we usually tune 4 kHz below the nominal tag frequency. So in that case, antFreq = 166.376 while nomFreq = 166.380)
tagBI	burst interval of tag, in seconds (e.g., 5.8984)
pulseLen	tag pulse length (milliseconds), if applicable. This value is only assigned based on the sample recording of the tag.
tagDeployID	Motus tag deployment ID
speciesID	unique numeric Motus ID (integer) for the species on which the tag was deployed
markerNumber	number for any additional marker placed on organism (e.g., bird band #)
markerType	type of additional marker (e.g. metal band)
tagDeployStart	tag deployment start date
tagDeployEnd	tag deployment end date
tagDeployLat	latitude of tag deployment, in decimal degrees N - negative values for Southern hemisphere
tagDeployLon	longitude of tag deployment, in decimal degrees E - negative values for Western hemisphere
tagDeployAlt	altitude of tag deployment, in meters ASL
tagDeployComments	additional comments or unclassified metadata for tag (often in JSON format)
fullID	full tag ID as PROJECT#MFGID:BI@NOMFREQ. Not necessarily unique over time. See motusTagID for a unique tag
deviceID	Motus device ID associated with the receiver serial number
recvDeployID	Receiver deployment ID
recvDeployLat	latitude of receiver deployment, in decimal degrees N - negative values for Southern hemisphere
recvDeployLon	longitude of receiver deployment, in decimal degrees E - negative values for Western hemisphere
recvDeployAlt	altitude of receiver deployment, in meters ASL
recv	serial number of the receiver; e.g., SG-1234BBBK5678 or Lotek-12345
recvDeployName	name assigned to the receiver deployment by the project manager
recvSiteName	name assigned to a site by the project manager (e.g. location name). The same label can be used for multiple deployments.
isRecvMobile	logical; whether the sensor is deployed on a mobile platform (eg. a ship)
recvProjID	unique (numeric) ID of the project that deployed this receiver (e.g., 8)
antType	character; antenna type, e.g. "9-element Yagi", "Omni"
antBearing	numeric; compass direction antenna main axis is pointing at (degrees clockwise from local magnetic North 0-360)
antHeight	numeric; height (meters) of antenna main axis above ground
speciesEN	species English (common) name
speciesFR	species French (common) name
speciesSci	species scientific name
speciesGroup	species group, e.g., BIRDS, BATS
tagProjName	short label of project that deployed the tag, e.g., "HolbSESA"
recvProjName	short label of project that deployed the receiver e.g., "HolbSESA"
gpsLat	latitude of receiver GPS location at time of writing the hourly detections file (degrees North)
gpsLon	longitude of receiver GPS location at time of writing the hourly detections file (degrees East)
gpsAlt	altitude of receiver GPS at time of writing the hourly detections file (meters)

# Appendix B

## Annexe B - Dépannage

Vous devez toujours vous assurer en premier lieu d'utiliser la plus récente version du logiciel motus (voyez la section @ref(checkVersion.B)) et d'avoir installé, chargé et mis à jour tous les logiciels requis (voyez le chapitre 2).

Pendant que vous tentez de télécharger des données avec le logiciel motus, il se peut que vous rencontriez des erreurs, dont un bon nombre sont vraisemblablement causées par une interruption de la connexion Internet. **Assurez-vous toujours d'être connecté à l'Internet lorsque vous utilisez la fonction tagme() avec l'option «update = TRUE».** On peut résoudre la plupart des problèmes en fermant le logiciel motus ou en relançant le logiciel R puis en reprenant le téléchargement avec la fonction tagme(). Si vous rencontrez encore des erreurs et que vous ne pouvez pas télécharger vos données, il se peut que le serveur soit temporairement hors ligne. Au besoin, communiquez avec nous à motus@birdscanada.org.

### B.1 Fermeture du logiciel motus

```
motusLogout()
```

### B.2 Reprise du téléchargement de données

Pour reprendre le téléchargement de vos données, exécutez la fonction tagme() de nouveau, mais sans inclure «new = TRUE»:

```
tagme(project.num, update = TRUE, dir = ...)
```

### B.3 Messages d'erreur courants et solutions

#### B.3.1 Je reçois le message «Auto-disconnecting SQLiteConnection» une ou plusieurs fois après avoir utilisé la fonction tagme().

Si vous recevez ce message une fois que le téléchargement est terminé, n'en tenez pas compte. Si vous le recevez pendant qu'un téléchargement est en cours, la connexion sera généralement maintenue et le téléchargement se poursuivra. Toutefois, si le téléchargement s'arrête, il vous suffit d'exécuter la fonction tagme() de nouveau. Si cela ne donne pas de résultat, nous vous suggérons de fermer le logiciel motus ou de redémarrer le logiciel R (voyez les sections B.1 et B.2).

### B.3.2 Je reçois le message «Internal Server Error» pendant que j'utilise la fonction tagme (... , update = TRUE).

Si vous recevez ce message pendant que vous mettez à jour votre fichier .motus, utilisez la fonction tagme() de nouveau pour poursuivre le téléchargement.

### B.3.3 Je reçois le message «Error: Forbidden» pendant que j'utilise la fonction tagme().

Cette erreur peut se produire si vous tentez de télécharger en même temps des données se rapportant à plusieurs projets à partir du même compte utilisateur. Si vous recevez ce message, fermez le logiciel motus et utilisez la fonction tagme() de nouveau (voyez les sections B.1 et B.2).

### B.3.4 Je reçois le message «Error: Object ‘xxxx’ not found», relatif au nom d'une table ou d'un champ, ou encore certains de vos exemples dans le guide ne fonctionnent pas.

Assurez-vous de suivre les étapes à partir du début du chapitre et de les passer l'une après l'autre dans l'ordre. Il se peut aussi que votre base de données n'ait pas été mise à jour pour accepter la version la plus récente du logiciel motusClient ou motus. Si l'appel de la fonction checkVersion est suivi d'un avertissement, cela peut indiquer que la fonction interne utilisée pour mettre à jour votre base de données n'a pas été enclenchée par la fonction tagme(). Cela peut arriver, par exemple, si vous chargez le logiciel motus Client sans charger aussi le logiciel motus. Le chargement du logiciel motus entraîne aussi le chargement du logiciel motusClient, de sorte que vous devriez avoir besoin de charger seulement le logiciel motus dans votre bibliothèque R.

Pour vérifier que vous avez la plus récente version du fichier .motus:

```
sql.motus <- tagme(project.num, dir = ...)
checkVersion(sql.motus)
```

Si vous recevez un avertissement, vous devez suivre les étapes suivantes:

1. Téléchargez les plus récentes versions des logiciels motusClient et motus (reportez-vous au chapitre 2).
2. Terminez la session R et relancez-la.
3. Chargez la bibliothèque motus en utilisant la fonction «require(Motus)» dans la console R.
4. Chargez votre fichier SQLite. Vérifiez s'il y a des notes sur la console indiquant que la mise à jour de votre base de données est en cours.
5. Vérifiez la version de nouveau.

```
library(motus)
sql <- tagme(project.num, dir = ...)
checkVersion(sql)
```

### B.3.5 Je reçois le message «Error in rssqlite\_connect(dbname, loadable.extensions, flags, vfs): Could not connect to database: unable to open database file» lorsque je tente d'exécuter la fonction tagme().

Si vous recevez ce message, il est probable que vous tentiez de faire un nouveau téléchargement ou une mise à jour en indiquant un répertoire inexistant. Le répertoire est indiqué dans la commande **dir = “”** de la fonction tagme(). Si le répertoire n'est pas indiqué, les fichiers seront enregistrés dans votre répertoire de travail. Utilisez la fonction getwd() pour déterminer quel est votre répertoire de travail courant. Utilisez la fonction setwd() pour établir un nouveau répertoire de travail. Pour spécifier l'endroit où enregistrer les

fichiers à partir de votre répertoire de travail, utilisez la commande «./» suivie de l'indication du chemin du fichier.

```
getwd() # Indique le répertoire de travail, dans le cas présent «C:/Documents».  
tagme(proj.num, new = TRUE, update = TRUE) # Télécharge les données dans votre répertoire de travail.  
tagme(proj.num, new = TRUE, update = TRUE, dir = "./data/") # Télécharge les données dans le dossier d'  
tagme(proj.num, new = TRUE, update = TRUE, dir = "C:/Downloads") # Télécharge les données en suivant l'
```

Bien sûr, il est toujours possible que le guide contienne des erreurs! Si les solutions proposées ici ne fonctionnent pas, communiquez avec motus@birdscanada.org.

## Appendix C

# Annexe C - Le logiciel R de Motus

Le logiciel R de Motus comprend des fonctions qui permettent d'effectuer des calculs communs et de produire des sommaires, des graphiques et des cartes avec les données .motus. Cette annexe présente ces fonctions et des exemples de leur utilisation. Bon nombre des fonctions sont utilisables dans les formats `tbl` et `data.frame`; toutefois, pour certaines, les données doivent être dans le format `sql` tel que spécifié ci-après. Le chapitre 3 présente des marches à suivre détaillées pour accéder aux données et les formater. Les exemples présentés tout au long de ce chapitre sont basés sur les données du projet 176 (Programme de suivi des oiseaux de rivage de la baie James) utilisées comme exemples, auxquelles on peut accéder et qu'on peut convertir dans différents formats par l'intermédiaire du code suivant:

```
# Télécharger les données fournies comme exemples
# dans le format sql et y accéder, nom
# d'utilisateur: motus.sample, mot de passe:
# motus.sample
sql.motus <- tagme(176, new = TRUE, update = TRUE,
  dir = "./data")

# Extraire la table «alltags» du fichier sql
# «sql.motus».
tbl.alltags <- tbl(sql.motus, "alltags")

## Convertir la table «tbl.alltags» en une trame de
## données appelée «df.alltags».
df.alltags <- tbl.alltags %>% collect() %>% as.data.frame()
```

Vous pouvez accéder aux pages d'aide relatives aux fonctions en utilisant «`??sunRiseSet`» dans la console R. Vous pouvez aussi voir le code sous-jacent à la fonction comme suit:

```
sunRiseSet
```

### C.1 checkVersion

#### C.1.1 Description

Lorsque vous appelez la fonction `tagme()` pour charger la base de données sqlite, le système vérifie que la version de votre base de données correspond à la plus récente version du logiciel motus et stocke la version dans une nouvelle table appelée `admInfo`. Au fil du temps, des changements sont apportés qui nécessitent l'ajout de tables, de vues ou de champs à la base de données. La fonction suivante vérifie que la version de

otre base de données a été mise à jour de manière à correspondre à la version courante du logiciel motus. Si le système affiche un avertissement, reportez-vous à l'annexe B pour savoir quoi faire. Si la version de votre base de données n'est pas à jour, reportez-vous au chapitre 2 pour savoir comment mettre à jour les versions des logiciels motus et motusClient.

### C.1.2 Dépendances

**sql.motus** Une base sqlite de données .motus téléchargée au moyen de la fonction tagme()

### C.1.3 Exemple

```
checkVersion(sql.motus)
```

## C.2 sunRiseSet

### C.2.1 Description

Cette fonction crée une variable heure du lever du soleil (sunrise) et heure du coucher du soleil (sunset) et l'ajoute à une trame de données comprenant des valeurs de latitude, de longitude et de date/heure dans le format POSIXct ou numérique.

### C.2.2 Dépendances

**data** Peut être une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables date/heure, latitude et longitude.

**lat** Variable avec valeurs de latitude, par défaut les valeurs de recvDeployLat.

**lon** Variable avec valeurs de longitude, par défaut les valeurs de recvDeployLon.

**ts** Variable de l'heure UTC dans le format numérique ou POSIXct, par défaut les valeurs de ts.

### C.2.3 Exemple

Ajout des variables sunrise/sunset à la trame de données alltags

```
alltags.df.sun <- sunRiseSet(df.alltags)
head(alltags.df.sun)
```

```
##   hitID runID batchID          ts sig sigsd noise freq freqsd
## 1 45107  8886      53 2015-10-26 11:19:49  52     0 -96    4     0
## 2 45108  8886      53 2015-10-26 11:20:28  54     0 -96    4     0
## 3 45109  8886      53 2015-10-26 11:21:17  55     0 -96    4     0
## 4 45110  8886      53 2015-10-26 11:21:55  52     0 -96    4     0
## 5 45111  8886      53 2015-10-26 11:22:44  49     0 -96    4     0
## 6 199885 23305     64 2015-10-26 11:12:04  33     0 -96    4     0
##   slop burstSlop done motusTagID ambigID port runLen bootnum tagProjID
## 1 1e-04    0.0000   1     16047     NA   3     5    11     176
## 2 1e-04   -0.0021   1     16047     NA   3     5    11     176
## 3 1e-04    0.0001   1     16047     NA   3     5    11     176
## 4 1e-04   -0.0010   1     16047     NA   3     5    11     176
## 5 1e-04    0.0001   1     16047     NA   3     5    11     176
```

```

## 6 1e-04    0.0000   1    16047     NA   1    11     4    176
## mfgID tagType codeSet   mfg tagModel tagLifespan nomFreq tagBI pulseLen
## 1    378      ID  Lotek4 Lotek NTQB-3-2          NA  166.38 9.6971    2.5
## 2    378      ID  Lotek4 Lotek NTQB-3-2          NA  166.38 9.6971    2.5
## 3    378      ID  Lotek4 Lotek NTQB-3-2          NA  166.38 9.6971    2.5
## 4    378      ID  Lotek4 Lotek NTQB-3-2          NA  166.38 9.6971    2.5
## 5    378      ID  Lotek4 Lotek NTQB-3-2          NA  166.38 9.6971    2.5
## 6    378      ID  Lotek4 Lotek NTQB-3-2          NA  166.38 9.6971    2.5
## tagDeployID speciesID markerNumber markerType tagDeployStart
## 1      1839    4670 135268103 metal band 1441908000
## 2      1839    4670 135268103 metal band 1441908000
## 3      1839    4670 135268103 metal band 1441908000
## 4      1839    4670 135268103 metal band 1441908000
## 5      1839    4670 135268103 metal band 1441908000
## 6      1839    4670 135268103 metal band 1441908000
## tagDeployEnd tagDeployLat tagDeployLon tagDeployAlt
## 1    1457632800    51.4839    -80.45    NA
## 2    1457632800    51.4839    -80.45    NA
## 3    1457632800    51.4839    -80.45    NA
## 4    1457632800    51.4839    -80.45    NA
## 5    1457632800    51.4839    -80.45    NA
## 6    1457632800    51.4839    -80.45    NA
##
## 1 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployID": "1441908000", "tagDeployID": "1839", "tagDeployLat": 51.4839, "tagDeployLon": -80.45}
## 2 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployID": "1441908000", "tagDeployID": "1839", "tagDeployLat": 51.4839, "tagDeployLon": -80.45}
## 3 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployID": "1441908000", "tagDeployID": "1839", "tagDeployLat": 51.4839, "tagDeployLon": -80.45}
## 4 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployID": "1441908000", "tagDeployID": "1839", "tagDeployLat": 51.4839, "tagDeployLon": -80.45}
## 5 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployID": "1441908000", "tagDeployID": "1839", "tagDeployLat": 51.4839, "tagDeployLon": -80.45}
## 6 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployID": "1441908000", "tagDeployID": "1839", "tagDeployLat": 51.4839, "tagDeployLon": -80.45}
##
## fullID deviceID recvDeployID recvDeployLat
## 1 SampleData#378:9.7@166.38(M.16047)    486    2510    42.60699
## 2 SampleData#378:9.7@166.38(M.16047)    486    2510    42.60699
## 3 SampleData#378:9.7@166.38(M.16047)    486    2510    42.60699
## 4 SampleData#378:9.7@166.38(M.16047)    486    2510    42.60699
## 5 SampleData#378:9.7@166.38(M.16047)    486    2510    42.60699
## 6 SampleData#378:9.7@166.38(M.16047)    515    2512    42.68067
## recvDeployLon recvDeployAlt    recv recvDeployName recvSiteName
## 1    -72.71657    NA Lotek-159    Shelburne    <NA>
## 2    -72.71657    NA Lotek-159    Shelburne    <NA>
## 3    -72.71657    NA Lotek-159    Shelburne    <NA>
## 4    -72.71657    NA Lotek-159    Shelburne    <NA>
## 5    -72.71657    NA Lotek-159    Shelburne    <NA>
## 6    -72.47392    NA Lotek-164    BennettMeadow    <NA>
## isRecvMobile recvProjID antType antBearing antHeight speciesEN
## 1      0      74 yagi-9    127    NA Red Knot
## 2      0      74 yagi-9    127    NA Red Knot
## 3      0      74 yagi-9    127    NA Red Knot
## 4      0      74 yagi-9    127    NA Red Knot
## 5      0      74 yagi-9    127    NA Red Knot
## 6      0      74 yagi-9    243    NA Red Knot
## speciesFR    speciesSci speciesGroup tagProjName
## 1 Bécasseau maubèche Calidris canutus    BIRDS SampleData
## 2 Bécasseau maubèche Calidris canutus    BIRDS SampleData
## 3 Bécasseau maubèche Calidris canutus    BIRDS SampleData

```

```

## 4 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 5 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 6 Bécasseau maubèche Calidris canutus      BIRDS SampleData
##   recvProjName gpsLat gpsLon gpsAlt       sunrise
## 1 <NA>     NA     NA     NA 2015-10-26 11:16:49
## 2 <NA>     NA     NA     NA 2015-10-26 11:16:49
## 3 <NA>     NA     NA     NA 2015-10-26 11:16:49
## 4 <NA>     NA     NA     NA 2015-10-26 11:16:49
## 5 <NA>     NA     NA     NA 2015-10-26 11:16:49
## 6 <NA>     NA     NA     NA 2015-10-26 11:15:58
##           sunset
## 1 2015-10-26 21:52:11
## 2 2015-10-26 21:52:11
## 3 2015-10-26 21:52:11
## 4 2015-10-26 21:52:11
## 5 2015-10-26 21:52:11
## 6 2015-10-26 21:51:06

```

## C.3 plotAllTagsCoord

### C.3.1 Description

Cette fonction pointe sur un graphique la latitude/longitude par rapport à l'heure (UTC arrondie à l'heure près) pour chaque émetteur à partir des données de détection .motus. Par défaut, les coordonnées géographiques sont tirées des enregistrements effectués par les GPS des récepteurs.

### C.3.2 Dépendances

**data** Peut être une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables date/heure et latitude ou longitude.

**tagsPerPanel** Nombre d'émetteurs indiqués dans chaque panneau du graphique, par défaut 5.

**coordinate** Variable avec valeurs de position, par défaut les valeurs de recvDeployLat.

**ts** Variable de l'heure UTC dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

**fullID** Variable de l'identifiant complet de l'émetteur. **mfgID** Variable de l'identifiant de l'émetteur attribué par le fabricant.

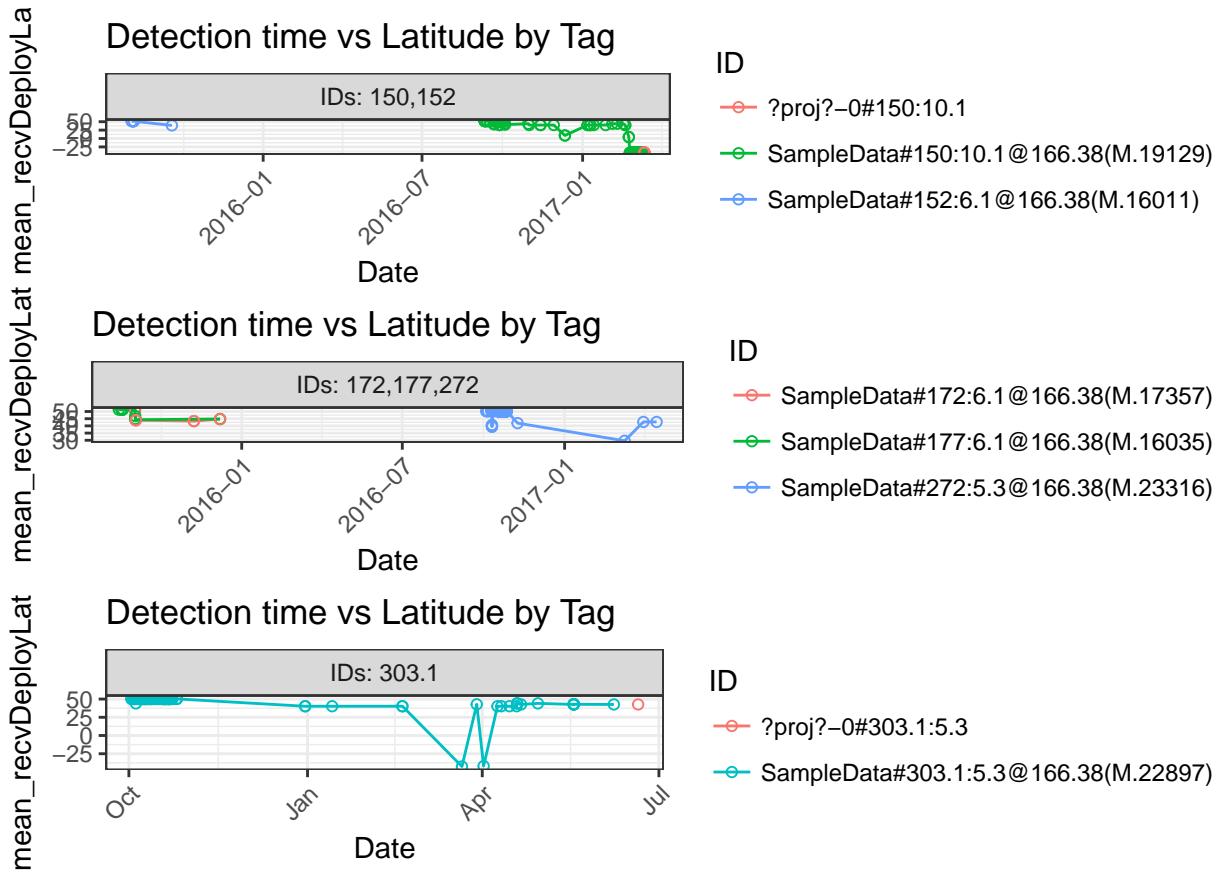
### C.3.3 Exemple

Pointage sur un graphique des coordonnées de certains émetteurs à partir d'une table tbl.alltags, à raison de 3 émetteurs par panneau

```

plotAllTagsCoord(filter(tbl.alltags, motusTagID %in%
c(19129, 16011, 17357, 16035, 22897, 23316)), tagsPerPanel = 3)

```



## C.4 plotAllTagsSite

### C.4.1 Description

Cette fonction pointe sur un graphique la latitude/longitude par rapport à l'heure (UTC arrondie à l'heure près) pour chaque émetteur à partir des données de détection .motus. Par défaut, les coordonnées sont tirées des enregistrements de la latitude effectués par les GPS des récepteurs.

### C.4.2 Dépendances

**data** Peut être une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables date/heure et latitude ou longitude.

**tagsPerPanel** Nombre d'émetteurs indiqués dans chaque panneau du graphique, par défaut 5.

**coordinate** Variable avec valeurs de position, par défaut les valeurs de recvDeployLat.

**ts** Variable de l'heure UTC dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

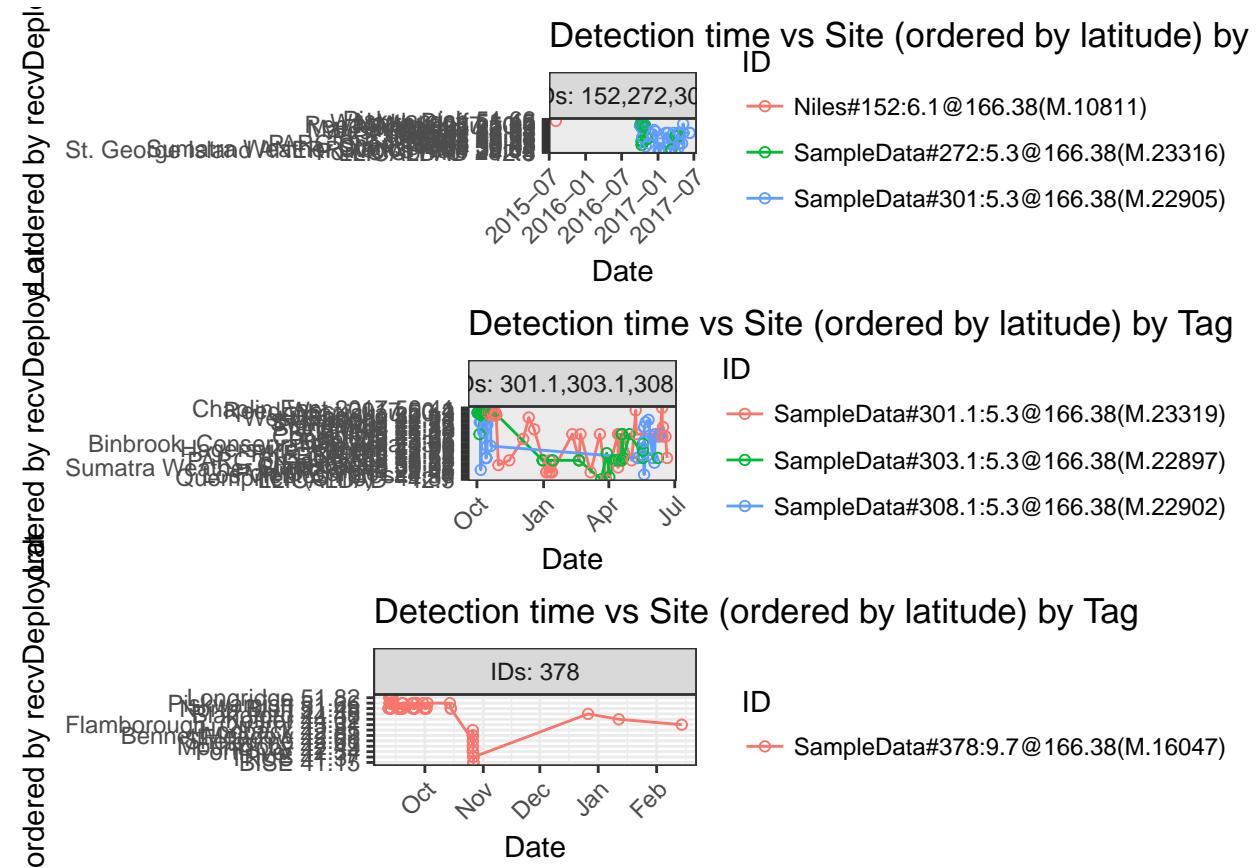
**fullID** Variable de l'identifiant complet de l'émetteur. **mfgID** Variable de l'identifiant de l'émetteur attribué par le fabricant.

### C.4.3 Exemple

Pointage sur un graphique des coordonnées à partir d'une table `tbl.alltags` pour l'espèce choisie, à savoir le Bécasseau maubèche, à raison de 3 émetteurs par panneau.

```
plotAllTagsSite(filter(tbl.alltags, speciesEN == "Red Knot"),
  coordinate = "recvDeployLat", tagsPerPanel = 3)
```

```
## Warning: Missing values are always removed in SQL.
## Use `AVG(x, na.rm = TRUE)` to silence this warning
```



## C.5 plotDailySiteSum

### C.5.1 Description

Cette fonction pointe sur des graphiques le nombre total de détections de tous les émetteurs et le nombre total d'émetteurs détectés, par jour, à un site déterminé. Elle dépend de la fonction `siteSumDaily`.

### C.5.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «`alltags`», ou une trame de données de détection incluant au minimum les variables `motusTagID`, `sig`, `recvDepName` et `ts`.

**motusTagID** Variable consistant en un identifiant d'émetteur motus (motus tag ID).

**sig** Variable de puissance de signal.

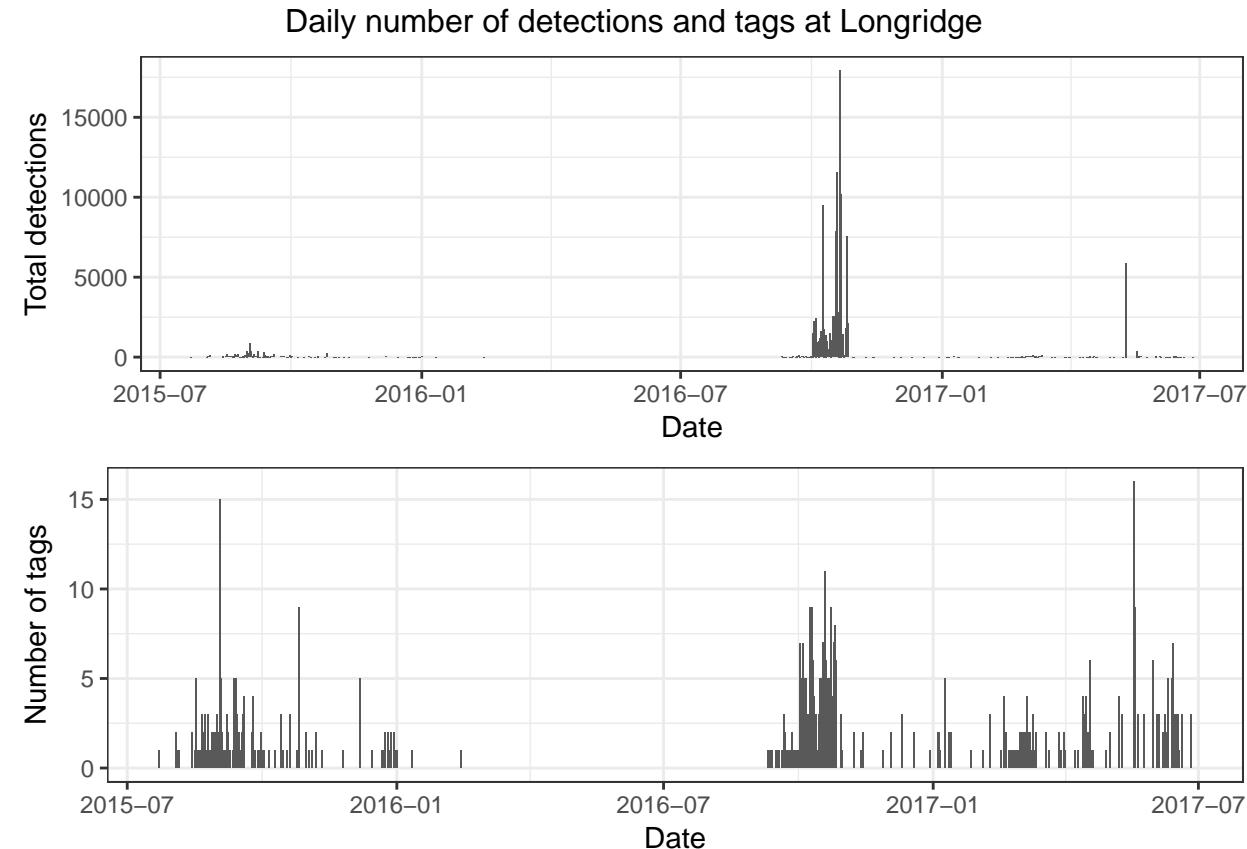
**recvDepName** Variable du nom du lieu de déploiement du récepteur.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

### C.5.3 Exemple

Pointage sur des graphiques du nombre total de détections de tous les émetteurs et du nombre total d'émetteurs détectés au site Longridge à partir de data.frame et df.alltags

```
plotDailySiteSum(df.alltags, recvDeployName = "Longridge")
```



## C.6 plotRouteMap

### C.6.1 Description

Cette fonction pointe sur une carte Google les trajectoires d'émetteurs détectés, la trajectoire de chaque émetteur étant représentée par une couleur distincte. L'utilisateur fixe un intervalle de temps entre deux dates pour que la carte indique la position des récepteurs qui fonctionnaient à un moment donné pendant cet intervalle. ### Dépendances **data** Un fichier sql .motus.

**maptype** Type de carte Google à afficher: relief, plan, satellite ou hybride.

**latCentre** Latitude du point central de la carte.

**lonCentre** Longitude du point central de la carte.

**zoom** Nombre entier de 3 à 21, 3 correspondant au niveau du continent et 10 au niveau de la localité.

**recvStart** Date du début de l'intervalle de temps pour l'indication des récepteurs actifs.

**recvEnd** Date de la fin de l'intervalle de temps pour l'indication des récepteurs actifs.

## C.6.2 Exemple

Pointage sur une carte de trajectoires de type «relief» à partir de toutes les données de détection, avec indication des récepteurs actifs entre le 1er janvier 2016 (2016-01-01) et le 1er janvier 2017 (2017-01-01).

```
plotRouteMap(sql.motus, maptype = "terrain", latCentre = 44,
  lonCentre = -70, zoom = 5, recvStart = "2016-01-01",
  recvEnd = "2016-12-31")

## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=44,-70&zoom=5&size=640x640&scale=1

## Warning: Removed 120 rows containing missing values (geom_point).

## Warning: Removed 685 rows containing missing values (geom_path).
```



## C.7 plotSite

### C.7.1 Description

### C.7.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables ts, antBearing, fullID et recvDepName.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**antBearing** Variable de l'angle de relèvement de l'antenne.

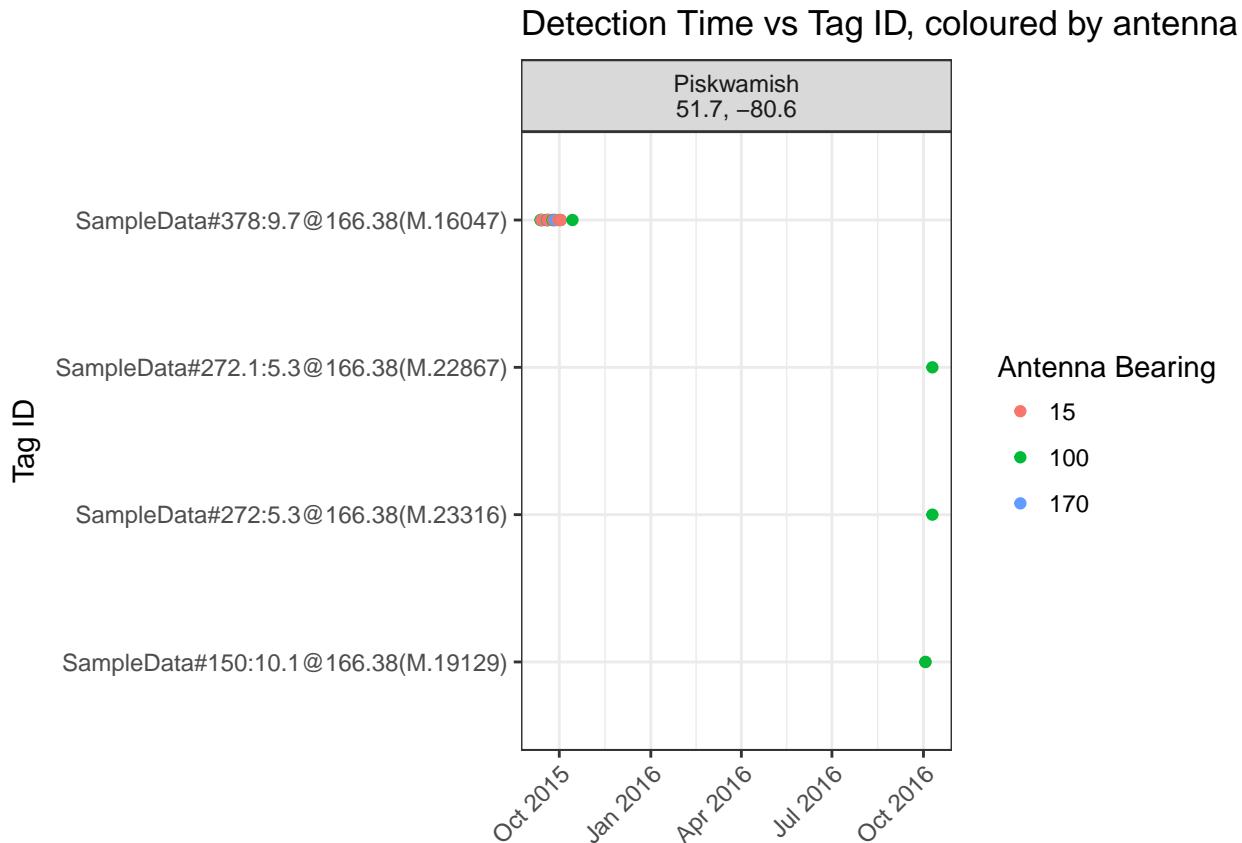
**fullID** Variable de l'identifiant complet de l'émetteur.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

### C.7.3 Exemple

Pointage seulement des données de détection à un site particulier (Piskwamish) à partir de data.frame et de df.alltags.

```
plotSite(filter(df.alltags, recvDeployName == "Piskwamish"))
```



## C.8 plotSiteSig

### C.8.1 Description

Cette fonction pointe sur un graphique la puissance du signal par rapport au temps de l'année pour tous les émetteurs détectés à un site particulier, les couleurs variant selon la gamme des angles de relèvement des antennes.

### C.8.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables antBearing, ts, lat, sig, fullID et recvDepName.

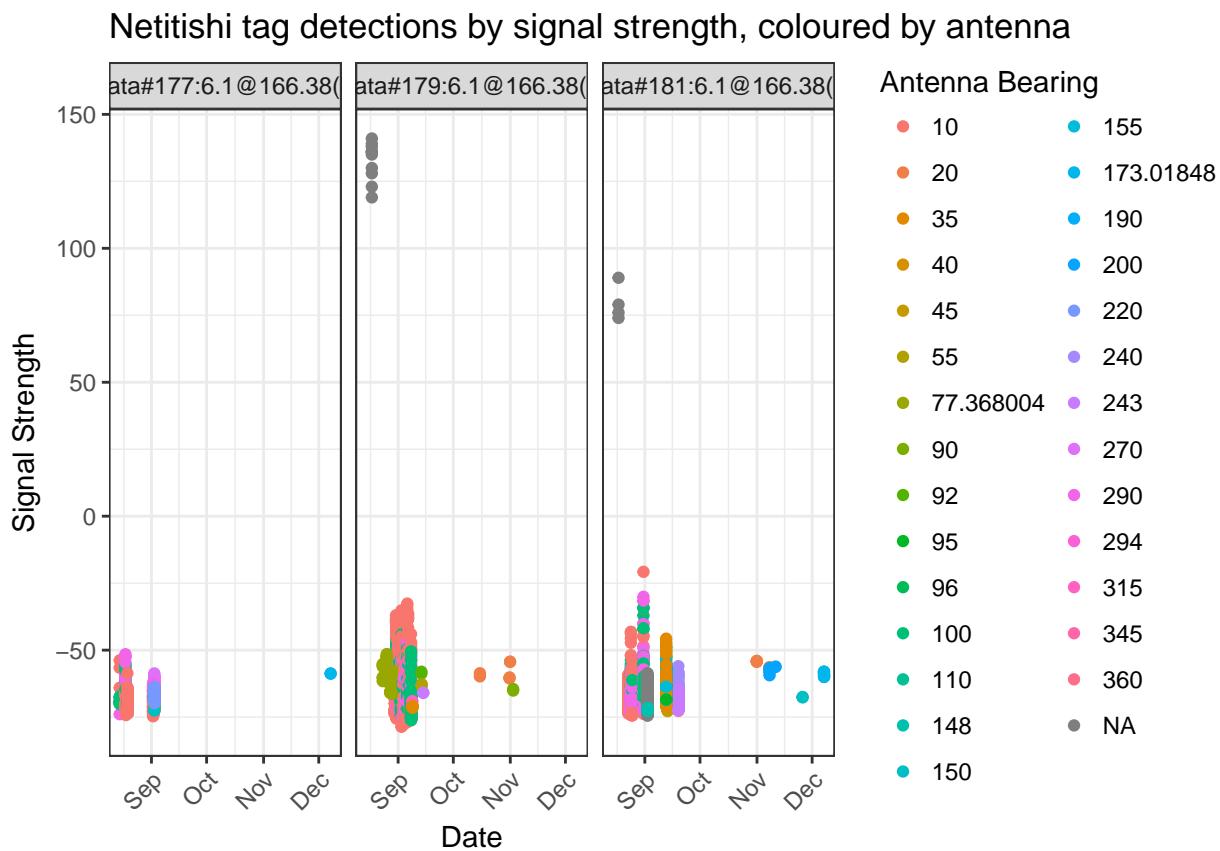
**antBearing** Variable de l'angle de relèvement de l'antenne.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.  
**recvDeployLat** Variable de la latitude de la position du récepteur.  
**sig** Variable de la puissance du signal de l'émetteur.  
**fullID** Variable de l'identifiant complet de l'émetteur.  
**recvDepName** Variable du nom du lieu de déploiement du récepteur.

### C.8.3 Exemple

Pointage de la puissance du signal de certains émetteurs détectés au site de Piskwamish.

```
plotSiteSig(filter(df.alltags, motusTagID %in% c(16037,
  16039, 16035)), recvDeployName = "Netitishi")
```



## C.9 plotTagSig

### C.9.1 Description

Cette fonction pointe sur un graphique la puissance du signal d'un émetteur particulier par rapport au temps de l'année, à raison d'un panneau de graphique par site (dans l'ordre de la latitude), les couleurs variant selon la gamme des angles de relèvement des antennes.

### C.9.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables motusTagID, sig, ts, antBearing, recvDeployLat, fullID et recvDepName.

**motusTagID** Variable de l'identifiant de l'émetteur Motus.

**antBearing** Variable de l'angle de relèvement de l'antenne.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**recvDeployLat** Variable de la latitude de la position du récepteur.

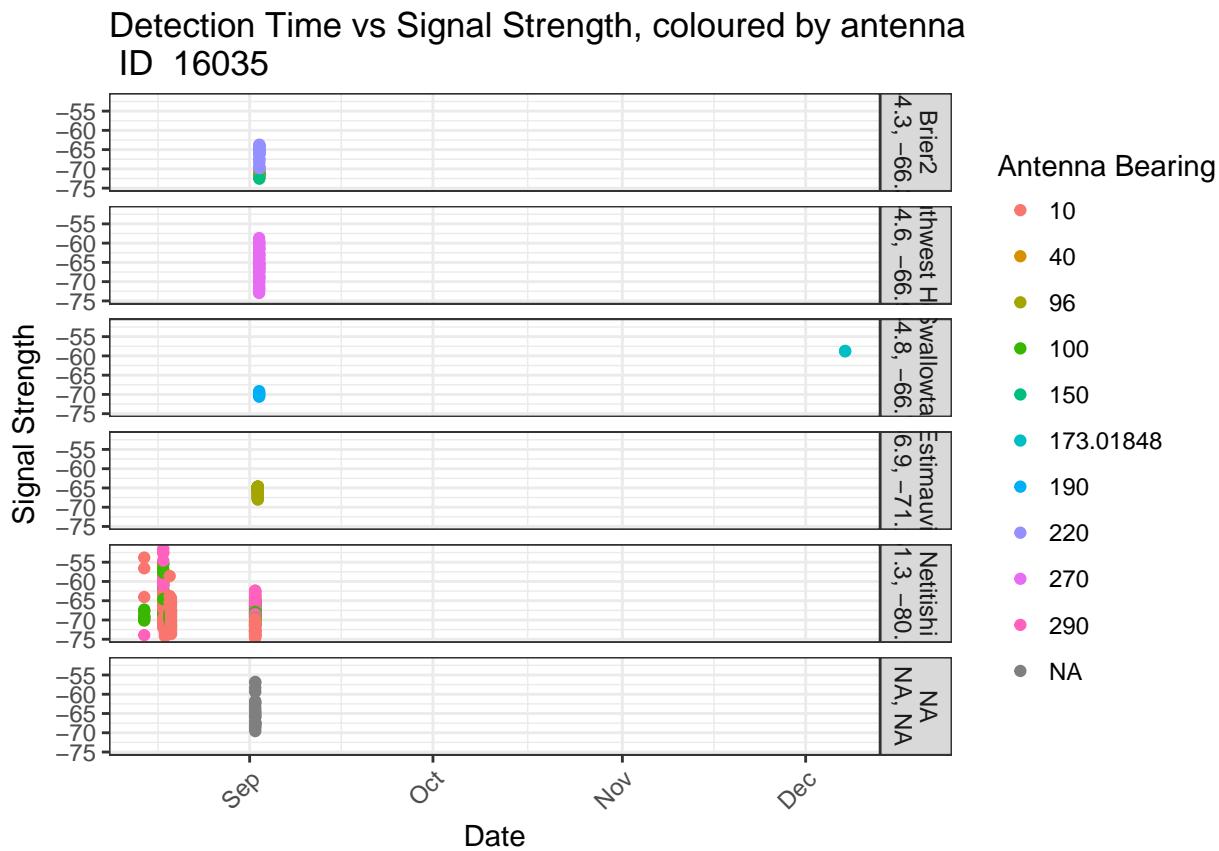
**sig** Variable de la puissance du signal de l'émetteur. **fullID** Variable de l'identifiant complet de l'émetteur.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

### C.9.3 Exemple

Pointage de la puissance du signal d'un émetteur particulier à partir de la table de données tbl.alltags.

```
plotTagSig(tbl.alltags, motusTagID = 16035)
```



## C.10 simSiteDet

### C.10.1 Description

Cette fonction crée une trame de données comprenant seulement les données de détection des émetteurs détectés par deux récepteurs ou plus en même temps.

## C.10.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables ts, motusTagID et recvDepName.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**motusTagID** Variable de l'identifiant de l'émetteur Motus.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

## C.10.3 Exemple

Obtention, à partir d'une trame de données df.alltags, d'une trame de données appelée «simSites» comprenant seulement les données de détection simultanée d'émetteurs.

```
simSites <- simSiteDet(df.alltags)
head(simSites)

##   motusTagID                 ts num.dup    hitID runID batchID
## 1      16047 2015-10-26 11:20:28      2    45108   8886     53
## 2      16047 2015-10-26 11:20:28      2   199896  23305     64
## 3      16047 2015-10-26 11:22:44      2   199902  23306     64
## 4      16047 2015-10-26 11:22:44      2    45111   8886     53
## 5     22897 2017-05-18 02:27:59      2 159449264 9005196  64491
## 6     22897 2017-05-18 02:27:59      2 201832550 10191953  66308
##   sig   sigsd noise freq freqsd slop burstSlop done ambigID
## 1 54.0000 0.0000 -96.0000 4.0000 0.0000 1e-04 -0.0021     1     NA
## 2 20.0000 0.0000 -96.0000 4.0000 0.0000 1e-04  0.0012     1     NA
## 3 28.0000 0.0000 -96.0000 4.0000 0.0000 1e-04 -0.0006     1     NA
## 4 49.0000 0.0000 -96.0000 4.0000 0.0000 1e-04  0.0001     1     NA
## 5 -63.7422 14.6880 -76.3434 2.7883 0.0196 1e-04  0.0003     1   -114
## 6 -67.5413 14.6268 -75.6038 2.6942 0.0061 1e-04  0.0000     1   -114
##   port runLen bootnum tagProjID mfgID tagType codeSet   mfg tagModel
## 1     3      5       11      176    378     ID Lotek4 Lotek NTQB-3-2
## 2     1     11       4      176    378     ID Lotek4 Lotek NTQB-3-2
## 3     3      6       4      176    378     ID Lotek4 Lotek NTQB-3-2
## 4     3      5      11      176    378     ID Lotek4 Lotek NTQB-3-2
## 5     3      5     145      176  303.1     ID Lotek4 Lotek NTQB-4-2
## 6     1      6     374      176  303.1     ID Lotek4 Lotek NTQB-4-2
##   tagLifespan nomFreq tagBI pulseLen tagDeployID speciesID markerNumber
## 1          NA 166.38 9.6971      2.5     1839     4670 135268103
## 2          NA 166.38 9.6971      2.5     1839     4670 135268103
## 3          NA 166.38 9.6971      2.5     1839     4670 135268103
## 4          NA 166.38 9.6971      2.5     1839     4670 135268103
## 5          NA 166.38 5.2978      2.5    10485     4670 9822-53123
## 6          NA 166.38 5.2978      2.5    10485     4670 9822-53123
##   markerType tagDeployStart tagDeployEnd tagDeployLat tagDeployLon
## 1 metal band     1441908000 1457632800 51.48390 -80.45000
## 2 metal band     1441908000 1457632800 51.48390 -80.45000
## 3 metal band     1441908000 1457632800 51.48390 -80.45000
## 4 metal band     1441908000 1457632800 51.48390 -80.45000
## 5 metal band    1475337600 1497283200 50.19278 -63.74528
## 6 metal band    1475337600 1497283200 50.19278 -63.74528
##   tagDeployAlt
## 1          NA
## 2          NA
```

```

## 3      NA
## 4      NA
## 5      NA
## 6      NA
##
## 1 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployLat": 42.60699, "recvDeployLon": -72.71657, "recvDeployName": "Shelburne", "recvSiteName": "<NA>"}
## 2 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployLat": 42.68067, "recvDeployLon": -72.47392, "recvDeployName": "BennettMeadow", "recvSiteName": "<NA>"}
## 3 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployLat": 42.68067, "recvDeployLon": -72.47392, "recvDeployName": "BennettMeadow", "recvSiteName": "<NA>"}
## 4 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "recvDeployLat": 42.60699, "recvDeployLon": -72.71657, "recvDeployName": "Shelburne", "recvSiteName": "<NA>"}
## 5 SampleData#303.1:5.3@166.38(M.22897) 297 2146 42.61540
## 6 SampleData#303.1:5.3@166.38(M.22897) 304 2549 42.58290
##   recvDeployLon recvDeployAlt          recv recvDeployName recvSiteName
## 1      -72.71657           NA Lotek-159 Shelburne      <NA>
## 2      -72.47392           NA Lotek-164 BennettMeadow <NA>
## 3      -72.47392           NA Lotek-164 BennettMeadow <NA>
## 4      -72.71657           NA Lotek-159 Shelburne      <NA>
## 5      -80.45810           NA SG-5113BBBK2853     BSC HQ      <NA>
## 6      -80.39840           NA SG-5113BBBK2972     Old Cut     <NA>
##   isRecvMobile recvProjID      antType antBearing antHeight speciesEN
## 1          0        74    yagi-9       127      NA Red Knot
## 2          0        74    yagi-9       243      NA Red Knot
## 3          0        74    yagi-9       120      NA Red Knot
## 4          0        74    yagi-9       127      NA Red Knot
## 5          0        1 yagi-9-laird     180      NA Red Knot
## 6          0        1 yagi-9-laird     290       25 Red Knot
##   speciesFR      speciesSci speciesGroup tagProjName
## 1 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 2 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 3 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 4 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 5 Bécasseau maubèche Calidris canutus      BIRDS SampleData
## 6 Bécasseau maubèche Calidris canutus      BIRDS SampleData
##   recvProjName gpsLat gpsLon gpsAlt
## 1      <NA>     NA     NA     NA
## 2      <NA>     NA     NA     NA
## 3      <NA>     NA     NA     NA
## 4      <NA>     NA     NA     NA
## 5      <NA>     NA     NA     NA
## 6      <NA>     NA     NA     NA

```

## C.11 siteSum

### C.11.1 Description

Cette fonction crée un sommaire des données sur la première et la dernière détections à un site déterminé, avec indications de l'intervalle de temps entre celles-ci, du nombre d'émetteurs détectés et du nombre total de détections. Elle pointe sur des graphiques le nombre total de détections de tous les émetteurs et le nombre total d'émetteurs détectés à chaque site.

### C.11.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables motusTagID, sig, recvDeploLat, recvDepName et ts.

**motusTagID** Variable de l'identifiant de l'émetteur Motus.

**sig** Variable de la puissance du signal.

**recvDeployLat** Variable de la latitude du lieu de déploiement du récepteur.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**units** Unités de temps indiquant l'intervalle de temps. Options: secondes, minutes, heures, jours et semaines. Par défaut en heures.

### C.11.3 Exemple

Création de sommaires par site pour des sites sélectionnés, avec indication du temps en minutes.

```
site_summary <- siteSum(filter(df.alltags, recvDeployName %in%
  c("Niapiskau", "Netitishi", "Old Cur", "Washkaugou")),
  units = "mins")
```



```
head(site_summary)
```

```
## # A tibble: 3 x 6
##   recvDeployName first_ts           last_ts          tot_ts num.tags
##   <fct>        <dttm>        <dttm>        <time>      <int>
## 1 Niapiskau_50.2~ 2016-10-01 23:47:57 2016-10-27 00:03:21 36015.~      6
## 2 Washkaugou_51.~ 2016-10-09 23:52:45 2016-10-10 00:00:42 7.9468~      2
## 3 Netitishi_51.3~ 2015-08-14 17:53:49 2015-09-08 01:10:13 34996.~      7
## # ... with 1 more variable: num.det <int>
```

## C.12 siteSumDaily

### C.12.1 Description

Cette fonction crée un sommaire des données sur la première et la dernière détections à un site déterminé, avec indications de l'intervalle de temps entre celles-ci, du nombre d'émetteurs détectés et du nombre total de détections pour chaque jour. Elle produit le même sommaire que la fonction siteSum, mais pour chaque jour par site.

### C.12.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables motusTagID, sig, recvDepName et ts.

**motusTagID** Variable de l'identifiant de l'émetteur Motus.

**sig** Variable de la puissance du signal.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**units** Unités de temps indiquant l'intervalle de temps. Options: secondes, minutes, heures, jours et semaines. Par défaut en heures.

### C.12.3 Exemple

Création de sommaires par site à partir des données de détection des émetteurs de l'utilisateur, pour tous les sites pour chaque jour, avec indication du temps en minutes, en utilisant la table de données `tbl.alltags`.

```
daily_site_summary <- siteSumDaily(tbl.alltags, units = "mins")
head(daily_site_summary)
```

```
##                               recvDeployName      date      first_ts
## 1      Assateague State Park\n38.2, -75.1 2015-09-13 2015-09-13 10:12:50
## 2                  Baccaro\n43.5, -65.5 2017-05-19 2017-05-19 16:01:21
## 3      BennettMeadow\n42.7, -72.5 2015-10-26 2015-10-26 11:12:04
## 4 Binbrook_Conservation_Area\n43.1, -79.8 2017-05-18 2017-05-18 03:13:25
## 5                  BISE\n41.2, -71.6 2015-10-26 2015-10-26 17:55:47
## 6      Blandford\n44.5, -64.1 2015-12-26 2015-12-26 14:58:27
##                               last_ts      tot_ts num_tags num_det
## 1 2015-09-13 10:14:40 1.828847 mins      1       6
## 2 2017-05-19 16:02:40 1.323297 mins      3       6
## 3 2015-10-26 11:30:49 18.747692 mins      1      27
## 4 2017-05-18 03:13:46 0.353195 mins      2       6
## 5 2015-10-26 19:16:55 81.132212 mins      1      44
## 6 2015-12-26 14:58:47 0.323150 mins      1       2
```

## C.13 siteTrans

### C.13.1 Description

Cette fonction crée une trame de données sur les transitions entre des sites; les données de détection sont ordonnées en fonction de l'heure de détection. Par transition on entend l'intervalle de temps entre la dernière détection au site x («départ» possible) et la première détection au site y («arrivée» possible); les données sont présentées dans l'ordre chronologique. Chaque ligne contient les indications suivantes: heure de la dernière détection et de la latitude/longitude au site x; heure de la première détection et de la latitude/longitude au site y; distance entre les deux sites; intervalle de temps entre les deux détections; vitesse du déplacement entre les détections; et angle de relèvement (cap) entre les deux sites.

### C.13.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables ts, motusTagID, tagDeployID, recvDeployLat, recvDeployLon et recvDepName.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**motusTagID** Variable de l'identifiant de l'émetteur Motus.

**tagDeployID** Variable de l'identifiant du lieu de déploiement de l'émetteur Motus.

**recvDeployLat** Variable de la latitude du lieu de déploiement du récepteur.

**recvDeployLon** Variable de la longitude du lieu de déploiement du récepteur.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

### C.13.3 Exemple

Visualisation des transitions entre des sites seulement pour l'émetteur 16037 à partir de la trame de données df.alltags.

```
transitions <- siteTrans(filter(df.alltags, motusTagID ==  
  16037), latCoord = "recvDeployLat", lonCoord = "recvDeployLon")  
head(transitions)  
  
## # A tibble: 6 x 14  
## # Groups: motusTagID, tagDeployID [1]  
##   motusTagID tagDeployID ts.x           lat.x lon.x recvDeployName.x  
##   <int>        <int> <dttm>       <dbl> <dbl> <chr>  
## 1    16037        1825 2015-08-17 17:02:39  NA     NA  NP mobile_NA, NA  
## 2    16037        1825 2015-08-28 16:40:18  51.5  -80.4 North Bluff_51.5~  
## 3    16037        1825 2015-09-08 01:10:13  51.3  -80.1 Netitishi_51.3, ~  
## 4    16037        1825 2015-09-08 18:37:16  44.6  -65.8 Comeau (Marshall~  
## 5    16037        1825 2015-09-13 19:46:27  39.0  -74.8 NWW_39, -74.8  
## 6    16037        1825 2015-09-14 15:56:49  37.1  -76.0 BULL_37.1, -76  
## # ... with 8 more variables: ts.y <dttm>, lat.y <dbl>, lon.y <dbl>,  
## #   recvDeployName.y <chr>, tot_ts <time>, dist <dbl>, rate <dbl>,  
## #   bearing <dbl>
```

## C.14 tagSum

### C.14.1 Description

Cette fonction crée un sommaire, pour chaque émetteur, des données sur les heures de la première et de la dernière détections, les sites de la première et de la dernière détections, l'intervalle de temps entre la première et la dernière détections, la distance en ligne droite entre les sites de la première et de la dernière détections, la vitesse de déplacement de l'émetteur et l'angle de relèvement entre les deux sites.

### C.14.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables motusTagID, fullID, recvDeployLat, recvDeployLon, recvDepName et ts.

**motusTagID** Variable de l'identifiant de l'émetteur Motus. **fullID** Variable de l'identifiant complet de l'émetteur.

**recvDeployLat** Variable de la latitude du lieu de déploiement du récepteur.

**recvDeployLon** Variable de la longitude du lieu de déploiement du récepteur.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

### C.14.3 Exemple

Création d'un sommaire de données pour tous les émetteurs à partir de l'ensemble des données de détection en utilisant la trame de données df.alltags.

```

tag_summary <- tagSum(tbl.alltags)
head(tag_summary)

##                                     fullID      first_ts
## 1 Niles#152:6.1@166.38(M.10811) 2015-08-03 06:37:11
## 2 ?proj?-0#395:9.7 2015-07-23 10:10:54
## 3 Selva#172:6.1@166.38(M.17021) 2015-09-02 04:06:07
## 4 SampleData#152:6.1@166.38(M.16011) 2015-08-03 06:37:11
## 5 SampleData#395:9.7@166.38(M.16052) 2015-09-12 17:38:04
## 6 SampleData#179:6.1@166.38(M.16037) 2015-08-17 17:01:38
##           last_ts      first_site      last_site
## 1 2015-08-03 06:37:35 North Bluff_51.5, -80.5 North Bluff_51.5, -80.5
## 2 2015-09-02 20:06:13 Machias_44.5, -67.1 Ruby's_62.9, -143.7
## 3 2015-09-03 00:27:16 Netitishi_51.3, -80.1 MDR_44, -68.1
## 4 2015-09-18 09:37:39 North Bluff_51.5, -80.5 NW_39, -74.8
## 5 2015-10-20 20:43:43 Longridge_51.8, -80.7 Mount Thom_45.6, -63
## 6 2015-11-02 13:21:42 NP mobile_NA, NA Hillman_Marsh_42, -82.5
##   recvLat.x recvLon.x recvLat.y recvLon.y      tot_ts      dist
## 1    51.4839   -80.4500   51.48390   -80.45000  24.386 secs     0
## 2    44.5023   -67.1018   62.89072  -143.68170 3578118.674 secs 5087724
## 3    51.2913   -80.1168   43.96893   -68.12830  73268.958 secs 1211577
## 4    51.4839   -80.4500   39.02827   -74.81004 3985228.228 secs 1452237
## 5    51.8231   -80.6912   45.55480   -62.98590 3294338.784 secs 1472844
## 6        NA         NA   42.04270   -82.51440 6639604.044 secs     NA
##   rate      bearing num_det
## 1 0.0000000 -180.00000     4
## 2 1.4218993  -38.36268    12
## 3 16.5360200 127.53144   279
## 4 0.3644051 160.20437   127
## 5 0.4470832 111.25011   147
## 6        NA         NA    1353

```

## C.15 tagSumSite

### C.15.1 Description

Cette fonction crée un sommaire de données pour chaque émetteur indiquant les heures de la première et de la dernière détections à chaque site, l'intervalle de temps entre les première et dernière détections à chaque site et le nombre total de détections à chaque site.

### C.15.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables motusTagID, fullID, recvDepName et ts.

**motusTagID** Variable de l'identifiant de l'émetteur Motus.

**fullID** Variable de l'identifiant complet de l'émetteur.

**recvDepName** Variable du nom du lieu de déploiement du récepteur.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

### C.15.3 Exemple

Création de sommaires de données pour seulement certains émetteurs avec indication du temps en heures par défaut, à partir de la trame de données df.alltags.

```
tag_site_summary <- tagSumSite(filter(df.alltags, motusTagID %in%
  c(16047, 16037, 16039)))
head(tag_site_summary)
```

```
##               fullID          recvDeployName
## 1 SampleData#179:6.1@166.38(M.16037)      BULL\n37.1, -76
## 2 SampleData#179:6.1@166.38(M.16037) Comeau (Marshalltown)\n44.6, -65.8
## 3 SampleData#179:6.1@166.38(M.16037)      Hillman_Marsh\n42, -82.5
## 4 SampleData#179:6.1@166.38(M.16037)      Netitishi\n51.3, -80.1
## 5 SampleData#179:6.1@166.38(M.16037)      North Bluff\n51.5, -80.5
## 6 SampleData#179:6.1@166.38(M.16037)      NP mobile\nNA, NA
##           first_ts       last_ts      tot_ts num_det
## 1 2015-09-14 15:55:48 2015-09-14 15:56:49 0.01693000 hours     2
## 2 2015-09-08 18:29:57 2015-09-08 18:37:16 0.12192881 hours     6
## 3 2015-11-02 13:20:47 2015-11-02 13:21:42 0.01525056 hours     2
## 4 2015-08-30 01:36:08 2015-09-08 01:10:13 215.56819269 hours 1166
## 5 2015-08-23 15:13:57 2015-08-28 16:40:18 121.43902892 hours     26
## 6 2015-08-17 17:01:38 2015-08-17 17:02:39 0.01693344 hours     11
```

## C.16 timeToSunriset

### C.16.1 Description

Cette fonction crée et ajoute des variables du temps jusqu'au lever/coucher du soleil et du temps depuis le lever/coucher du soleil à partir d'une trame de données comprenant des valeurs de latitude, de longitude et de date/heure dans le format POSIXct.

### C.16.2 Dépendances

**data** Une table choisie à partir de données de détection .motus, p. ex., «alltags», ou une trame de données de détection incluant au minimum les variables date/heure, latitude et longitude.

**lat** Variable de la latitude, par défaut les valeurs de recvDeployLat.

**lon** Variable de la longitude, par défaut les valeurs de recvDeployLon.

**ts** Variable de la date et de l'heure dans le format numérique ou POSIXct, par défaut les valeurs de ts.

**units** Unités de temps indiquant l'intervalle de temps. Options: secondes, minutes, heures, jours et semaines. Par défaut en heures.

### C.16.3 Exemple

Obtention d'information sur le temps jusqu'au lever/coucher du soleil et du temps depuis le lever/coucher du soleil, en minutes, à partir de la table tbl.alltags.

```
sunrise <- timeToSunriset(tbl.alltags, units = "mins")
head(sunrise)
```

```
##   hitID runID batchID          ts sig sigsd noise freq freqsd
## 1  45107   8886      53 2015-10-26 11:19:49  52      0   -96     4      0
```

```

## 2 45108 8886      53 2015-10-26 11:20:28 54     0   -96    4    0
## 3 45109 8886      53 2015-10-26 11:21:17 55     0   -96    4    0
## 4 45110 8886      53 2015-10-26 11:21:55 52     0   -96    4    0
## 5 45111 8886      53 2015-10-26 11:22:44 49     0   -96    4    0
## 6 199885 23305     64 2015-10-26 11:12:04 33     0   -96    4    0
##   slop burstSlop done motusTagID ambigID port runLen bootnum tagProjID
## 1 1e-04   0.0000   1   16047    NA   3     5    11    176
## 2 1e-04   -0.0021  1   16047    NA   3     5    11    176
## 3 1e-04   0.0001   1   16047    NA   3     5    11    176
## 4 1e-04   -0.0010  1   16047    NA   3     5    11    176
## 5 1e-04   0.0001   1   16047    NA   3     5    11    176
## 6 1e-04   0.0000   1   16047    NA   1     11    4    176
##   mfgID tagType codeSet   mfg tagModel tagLifespan nomFreq tagBI pulseLen
## 1 378      ID  Lotek4 Lotek NTQB-3-2          NA 166.38 9.6971    2.5
## 2 378      ID  Lotek4 Lotek NTQB-3-2          NA 166.38 9.6971    2.5
## 3 378      ID  Lotek4 Lotek NTQB-3-2          NA 166.38 9.6971    2.5
## 4 378      ID  Lotek4 Lotek NTQB-3-2          NA 166.38 9.6971    2.5
## 5 378      ID  Lotek4 Lotek NTQB-3-2          NA 166.38 9.6971    2.5
## 6 378      ID  Lotek4 Lotek NTQB-3-2          NA 166.38 9.6971    2.5
##   tagDeployID speciesID markerNumber markerType tagDeployStart
## 1           1839     4670 135268103 metal band 1441908000
## 2           1839     4670 135268103 metal band 1441908000
## 3           1839     4670 135268103 metal band 1441908000
## 4           1839     4670 135268103 metal band 1441908000
## 5           1839     4670 135268103 metal band 1441908000
## 6           1839     4670 135268103 metal band 1441908000
##   tagDeployEnd tagDeployLat tagDeployLon tagDeployAlt
## 1 1457632800     51.4839   -80.45    NA
## 2 1457632800     51.4839   -80.45    NA
## 3 1457632800     51.4839   -80.45    NA
## 4 1457632800     51.4839   -80.45    NA
## 5 1457632800     51.4839   -80.45    NA
## 6 1457632800     51.4839   -80.45    NA
##
## 1 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "fullID": "SampleData#378:9.7@166.38(M.16047)"}
## 2 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "fullID": "SampleData#378:9.7@166.38(M.16047)"}
## 3 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "fullID": "SampleData#378:9.7@166.38(M.16047)"}
## 4 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "fullID": "SampleData#378:9.7@166.38(M.16047)"}
## 5 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "fullID": "SampleData#378:9.7@166.38(M.16047)"}
## 6 {"ageID": "HY", "bill": 36.5, "blood": "Y", "country": "Canada", "culmen": 36.5, "fatScore": 3, "locationID": "M.16047", "fullID": "SampleData#378:9.7@166.38(M.16047)"}
##   deviceID recvDeployID recvDeployLat
## 1 SampleData#378:9.7@166.38(M.16047)     486     2510 42.60699
## 2 SampleData#378:9.7@166.38(M.16047)     486     2510 42.60699
## 3 SampleData#378:9.7@166.38(M.16047)     486     2510 42.60699
## 4 SampleData#378:9.7@166.38(M.16047)     486     2510 42.60699
## 5 SampleData#378:9.7@166.38(M.16047)     486     2510 42.60699
## 6 SampleData#378:9.7@166.38(M.16047)     515     2512 42.68067
##   recvDeployLon recvDeployAlt      recv recvDeployName recvSiteName
## 1 -72.71657          NA Lotek-159  Shelburne <NA>
## 2 -72.71657          NA Lotek-159  Shelburne <NA>
## 3 -72.71657          NA Lotek-159  Shelburne <NA>
## 4 -72.71657          NA Lotek-159  Shelburne <NA>
## 5 -72.71657          NA Lotek-159  Shelburne <NA>
## 6 -72.47392          NA Lotek-164 BennettMeadow <NA>

```

```

##   isRecvMobile recvProjID antType antBearing antHeight speciesEN
## 1          0    74  yagi-9       127      NA  Red Knot
## 2          0    74  yagi-9       127      NA  Red Knot
## 3          0    74  yagi-9       127      NA  Red Knot
## 4          0    74  yagi-9       127      NA  Red Knot
## 5          0    74  yagi-9       127      NA  Red Knot
## 6          0    74  yagi-9      243      NA  Red Knot
##           speciesFR      speciesSci speciesGroup tagProjName
## 1 Bécasseau maubèche Calidris canutus        BIRDS SampleData
## 2 Bécasseau maubèche Calidris canutus        BIRDS SampleData
## 3 Bécasseau maubèche Calidris canutus        BIRDS SampleData
## 4 Bécasseau maubèche Calidris canutus        BIRDS SampleData
## 5 Bécasseau maubèche Calidris canutus        BIRDS SampleData
## 6 Bécasseau maubèche Calidris canutus        BIRDS SampleData
##   recvProjName gpsLat gpsLon gpsAlt           sunrise
## 1 <NA>       NA     NA      NA 2015-10-26 11:16:49
## 2 <NA>       NA     NA      NA 2015-10-26 11:16:49
## 3 <NA>       NA     NA      NA 2015-10-26 11:16:49
## 4 <NA>       NA     NA      NA 2015-10-26 11:16:49
## 5 <NA>       NA     NA      NA 2015-10-26 11:16:49
## 6 <NA>       NA     NA      NA 2015-10-26 11:15:58
##           sunset ts_to_set ts_since_set ts_to_rise ts_since_rise
## 1 2015-10-26 21:52:11   632.3533     806.2056 1438.215974     3.013824
## 2 2015-10-26 21:52:11   631.7104     806.8485 1437.573097     3.656701
## 3 2015-10-26 21:52:11   630.9109     807.6480 1436.773577     4.456221
## 4 2015-10-26 21:52:11   630.2513     808.3076 1436.113997     5.115801
## 5 2015-10-26 21:52:11   629.4518     809.1071 1435.314477     5.915321
## 6 2015-10-26 21:51:06   639.0310     799.5242   3.896393 1437.332308

```

## Appendix D

# Annexe D - Le logiciel motusClient - Fonctions de filtrage de données

Le logiciel R motusClient intègre des fonctions qu'on peut utiliser pour attribuer des probabilités aux données de détection d'émetteurs et filtrer ces données en fonction de ces probabilités. Par exemple, alors que vous explorez vos données pour éliminer les faux positifs et les détections ambiguës (voyez le chapitre 5), il se peut que vous constatiez que certaines détections ne se rapportent pas à votre émetteur ou vos émetteurs. Au lieu de simplement utiliser un script R pour éliminer ces détections, vous pouvez faire appel aux fonctions de filtrage présentées ici pour créer un filtre sur mesure et l'enregistrer dans votre fichier .motus; ce filtre attribue une valeur de probabilité entre 0 et 1 aux runID fournis dans le filtre.

Les fonctions de filtrage de données du logiciel R opèrent au niveau de la séquence. Une séquence est un groupe de détections consécutives des signaux d'un émetteur par un récepteur. En général, le risque est élevé qu'une séquence d'au moins 2 détections (c.-à-d. 2 pulsations) représente un faux positif. Il existe de multiples façons d'attribuer des probabilités aux données associées à chaque runID, entre autres au niveau le plus simple, à savoir générer une liste de 0 et de 1 pour les enregistrements que vous aimeriez exclure ou inclure. Vous pourriez aussi élaborer un modèle qui attribue une probabilité à chaque runID dans vos données.

## D.1 listRunsFilters

### D.1.1 Description

Cette fonction renvoie une trame de données contenant les filterID, login, noms, projectID et descriptions pour le projectID propre à un émetteur ou un récepteur déterminé qui se trouve dans la base de données locale.

## D.2 Dépendances

**src** L'objet SQLite que vous obtenez lorsque vous chargez un fichier .motus dans R, p. ex., le fichier «sql.motus» dans le chapitre 3.

## D.3 Exemple

```
filt.df <- listRunsFilters(src = sql.motus)
```

## D.4 createRunsFilter

### D.4.1 Description

Cette fonction peut servir principalement à modifier les propriétés de filtres existants, comme la description du filtre ou l'identifiant du projet (projectID), mais elle est également appelée à l'interne par l'instruction «writeRunsFilter» (section D.6) pour générer un nouvel identifiant de filtre (filterID). Pour sauvegarder les enregistrements de filtres, vous devez utiliser «writeRunsFilter» (section D.6). La fonction renvoie dans la base de données locale l'identifiant de filtre (nombre entier) qui correspond au nouveau filtre ou au filtre existant portant le nom de filtre (filterName) fourni. S'il existe déjà un filtre du même nom, la fonction génère un avertissement et renvoie l'identifiant du filtre existant.

### D.4.2 Dépendances

**src** L'objet SQLite que vous obtenez lorsque vous chargez un fichier .motus dans R, p. ex., le fichier «sql.motus» dans le chapitre 3.

**filterName** Le nom que vous voudriez attribuer au filtre. La fonction crée un nouveau filtre seulement s'il n'existe pas de filtre du même nom dans la base de données locale.

**motusProjID** L'identifiant numérique associé à un projet. Par exemple, les données utilisées à titre d'exemples tout au long du présent guide sont celles du projet 176 (Programme de suivi des oiseaux de rivage de la baie James). Par défaut, l'identifiant du projet est «NA» lorsqu'il n'existe pas déjà d'identifiant; c'est la valeur recommandée pour le moment. L'identifiant de projet attribué à un filtre sera utile surtout pour la synchronisation future des filtres avec le serveur de Motus. Il n'est pas nécessaire que les enregistrements de détections contenus dans le filtre portent l'identifiant attribué au filtre. **descr** Description du filtre (facultatif). Par défaut = NA. **update** Expression booléenne (par défaut = FALSE). Si le filtre existe déjà, détermine si les propriétés (p. ex. descr) sont conservées ou mises à jour.

### D.4.3 Exemple

Création pour la base de données sql.motus d'un nouveau filtre appelé «myfilter» qui n'est pas relié à un projet spécifique:

```
createRunsFilter(sql.motus, "myfilter")

# OU ajout d'une attribution à un projet

createRunsFilter(sql.motus, "myfilter", motusProjID = 176)

# OU ajout d'un projet et d'une description, ce qui
# peut entraîner la mise à jour d'une possible
# version antérieure appelée myfilter.

createRunsFilter(sql.motus, "myfilter", motusProjID = 176,
                descr = "assign probability of 0 to false positives",
                update = TRUE)
```

## D.5 getRunsFilters

### D.5.1 Description

Cette fonction renvoie une référence à une table SQLite aux enregistrements runsFilters enregistrés dans la base de données (runID, motusTagID et probabilité) associés à un nom spécifique (et, facultativement, à un projet spécifique) à partir de la base de données locale. La section ?? du chapitre 5 présente des exemples de façons d'utiliser la table renvoyée pour la fusionner avec des données de détection.

### D.5.2 Dépendances

**src** L'objet SQLite que vous obtenez lorsque vous chargez un fichier .motus dans R, p. ex., le fichier «sql.motus» dans le chapitre 3.

**filterName** Le nom que vous avez attribué au filtre que vous avez créé ou enregistré. La fonction renvoie un avertissement si le nom du filtre n'existe pas.

**motusProjID** L'identifiant numérique associé à un projet. Par exemple, les données utilisées à titre d'exemples tout au long du présent guide sont celles du projet 176 (Programme de suivi des oiseaux de rivage de la baie James). Par défaut, l'identifiant du projet est «NA» lorsqu'il n'existe pas déjà d'identifiant.

### D.5.3 Exemple

```
tblfilt <- getRunsFilters(src = sql.motus, filterName = "myfilter")
tblfilt2 <- getRunsFilters(sql.motus, "myfilter2")

# Filtrez les enregistrements contenus dans la
# trame de données qui sont dans la table tblfilt.
df <- left_join(df, tblfilt, by = c("runID", "motusTagID")) %>%
  mutate(probability = ifelse(is.na(probability),
    1, probability)) %>% filter(probability > 0)

# Vous pouvez appliquer un deuxième filtre,
# tblfilt2, au résultat de l'utilisation du filtre
# précédent.
df <- left_join(df, tblfilt2, by = c("runID", "motusTagID")) %>%
  mutate(probability = ifelse(is.na(probability),
    1, probability)) %>% filter(probability > 0)
```

## D.6 writeRunsFilter

### D.6.1 Description

Cette fonction écrit dans la base de données locale (fichier SQLite) le contenu d'une trame de données contenant runID, motusTagID et une probabilité attribuée. Si le nom de filtre («filterName») n'existe pas, elle appellera la fonction «createRunsFilter» (section D.4) pour en créer un dans votre base de données. Par défaut, cette fonction crée la situation suivante: tout nouvel enregistrement dans la trame de données est annexé au filtre existant ou au nouveau filtre appelé «filterName» et les enregistrements déjà existants (mêmes runID et motusTagID) sont remplacés (overwrite = TRUE), mais ceux qui ne sont pas dans la trame de données sont retenus dans la table de filtre existante (delete = FALSE). Pour remplacer la totalité des valeurs de filtre existantes par celles de la nouvelle trame de données, utilisez delete = TRUE. La fonction

renvoie une référence de table SQLite au filtre, de la même manière que si la fonction «getRunsFilter» (section D.5) était appelée.

## D.6.2 Dépendances

**src** L'objet SQLite que vous obtenez lorsque vous chargez un fichier .motus dans R, p. ex., le fichier «sql.motus» dans le chapitre 3.

**filterName** Le nom du filtre auquel vous voudriez affecter la base de données.

**motusProjID** L'identifiant numérique associé à un projet. Par exemple, les données utilisées à titre d'exemples tout au long du présent guide sont celles du projet 176 (Programme de suivi des oiseaux de rivage de la baie James). Par défaut, l'identifiant du projet est «NA» lorsqu'il n'existe pas déjà d'identifiant.

**df** La trame de données qui contient le runID (nombre entier), le motusTagID (nombre entier) et la probabilité (nombre à virgule flottante) de détections auxquels vous aimeriez affecter un filtre. MotusTagID devrait correspondre à l'identifiant de l'émetteur (tag ID) et non à l'identifiant ambigID négatif associé aux détections ambiguës. **overwrite** Par défaut = “TRUE”. Lorsque TRUE est choisi, les enregistrements existants (mêmes runID et motusTagID) qui correspondent aux mêmes filterName et runID sont remplacés dans la base de données locale. **delete** Par défaut = “FALSE”. Lorsque TRUE est choisi, tous les enregistrements de filtres existants associés au nom de filtre (filterName) sont éliminés et ceux qui se trouvent dans la trame de données (df) sont réinsérés. Vous devriez utiliser cette option si la trame de données contient l'ensemble des filtres que vous voulez enregistrer.

## D.6.3 Exemples

```
# Écrire sur «myfilter» une trame de données
# contenant les enregistrements de filtres (runID,
# motusTagID et probabilité).
writeRunsFilter(src = sql.motus, filterName = "myfilter",
  df = filter.df)

# Écrire sur «myfilter» une trame de données
# contenant les enregistrements de filtres (runID,
# motusTagID et probabilité) en écrasant la version
# précédente dans sa totalité.
writeRunsFilter(src = sql.motus, fileName = "myfilter",
  df = filter.df, delete = TRUE)

# Écrire sur «myfilter» une trame de données
# contenant les enregistrements de filtres (runID,
# motusTagID et probabilité), mais seulement en
# annexant les nouveaux enregistrements, sans
# supprimer ceux qui ont été créés précédemment.
writeRunsFilter(src = sql.motus, "myfilter", df = filter.df,
  overwrite = FALSE)
```