Single studies using the SelfControlledCaseSeries package

Martijn J. Schuemie, Marc A. Suchard and Patrick Ryan

2017-07-05

Contents

1 Introduction

This vignette describes how you can use the `SelfControlledCaseSeries` package to perform a single Self-Controlled Case Series (SCCS) study. We will walk through all the steps needed to perform an exemplar study, and we have selected the well-studied topic of the effect of NSAIDs on gastrointestinal (GI) bleeding-related hospitalization. For simplicity, we focus on one NSAID: diclofenac.

2 Installation instructions

Before installing the `SelfControlledCaseSeries` package make sure you have Java available. Java can be downloaded from www.java.com. For Windows users, RTools is also necessary. RTools can be downloaded from CRAN.

The `SelfControlledCaseSeries` package is currently maintained in a Github repository, and has dependencies on other packages in Github. All of these packages can be downloaded and installed from within R using the `devtools` package:

```
install.packages("devtools")
library(devtools)
install_github("ohdsi/OhdsiRTools")
install_github("ohdsi/SqlRender")
install_github("ohdsi/DatabaseConnector")
install_github("ohdsi/Cyclops")
install_github("ohdsi/SelfControlledCaseSeries")
```

Once installed, you can type `library(SelfControlledCaseSeries)` to load the package.

3 Overview

In the `SelfControlledCaseSeries` package a study requires at least three steps:

1. Loading the necessary data from the database.
2. Transforming the data into a format suitable for an SCCS study. This step includes the creation of covariates based on the variables extracted from the database, such as defining risk windows based on exposures.
3. Fitting the model using conditional Poisson regression.

In the following sections these steps will be demonstrated for increasingly complex studies.

4 Studies with a single drug

4.1 Configuring the connection to the server

We need to tell R how to connect to the server where the data are. `SelfControlledCaseSeries` uses the `DatabaseConnector` package, which provides the `createConnectionDetails` function. Type `?createConnectionDetails` for the specific settings required for the various database management systems (DBMS). For example, one might connect to a PostgreSQL database using this code:

```
connectionDetails <- createConnectionDetails(dbms = "postgresql",
                                             server = "localhost/ohdsi",
                                             user = "joe",
                                             password = "supersecret")

cdmDatabaseSchema <- "my_cdm_data"
cohortDatabaseSchema <- "my_results"
cdmVersion <- "5"
```

The last three lines define the `cdmDatabaseSchema` and `cohortDatabaseSchema` variables,as well as the CDM version. We'll use these later to tell R where the data in CDM format live, where we have stored our cohorts of interest, and what version CDM is used. Note that for Microsoft SQL Server, databaseschemas need to specify both the database and the schema, so for example `cdmDatabaseSchema <- "my_cdm_data.dbo"`.

4.2 Preparing the health outcome of interest

We need to define the exposures and outcomes for our study. One way to do this is by writing SQL statements against the OMOP CDM that populate a table of events in which we are interested. The resulting table should have the same structure as the `cohort` table in the CDM. For CDM v5+, this means it should have

the fields `cohort_definition_id`, `cohort_start_date`, `cohort_end_date`,and `subject_id`. For CDM v4, the `cohort_definition_id` field must be called `cohort_concept_id`.

For our example study, we have created a file called *vignette.sql* with the following contents:

```
/**********************************
File vignette.sql
**********************************/

IF OBJECT_ID('@cohortDatabaseSchema.@outcomeTable', 'U') IS NOT NULL
  DROP TABLE @cohortDatabaseSchema.@outcomeTable;

SELECT 1 AS cohort_definition_id,
    condition_start_date AS cohort_start_date,
    condition_end_date AS cohort_end_date,
    condition_occurrence.person_id AS subject_id
INTO @cohortDatabaseSchema.@outcomeTable
FROM @cdmDatabaseSchema.condition_occurrence
INNER JOIN @cdmDatabaseSchema.visit_occurrence
    ON condition_occurrence.visit_occurrence_id = visit_occurrence.visit_occurrence_id
WHERE condition_concept_id IN (
        SELECT descendant_concept_id
        FROM @cdmDatabaseSchema.concept_ancestor
        WHERE ancestor_concept_id = 192671 -- GI - Gastrointestinal haemorrhage
        )
    AND visit_occurrence.visit_concept_id IN (9201, 9203);
```

Note on CDM V4 'visit_concept_id' should be 'place_of_service_concept_id', and 'cohort_definition_id' should be 'cohort_concept_id'.

This is parameterized SQL which can be used by the `SqlRender` package. We use parameterized SQL so we do not have to pre-specify the names of the CDM and cohort schemas. That way, if we want to run the SQL on a different schema, we only need to change the parameter values; we do not have to change the SQL code. By also making use of translation functionality in `SqlRender`, we can make sure the SQL code can be run in many different environments.

```
library(SqlRender)
sql <- readSql("vignette.sql")
sql <- renderSql(sql,
                 cdmDatabaseSchema = cdmDatabaseSchema,
                 cohortDatabaseSchema = cohortDatabaseSchema
                 outcomeTable = "my_outcomes")$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

connection <- connect(connectionDetails)
executeSql(connection, sql)
```

In this code, we first read the SQL from the file into memory. In the next line, we replace the three parameter names with the actual values. We then translate the SQL into the dialect appropriate for the DBMS we already specified in the `connectionDetails`. Next, we connect to the server, and submit the rendered and translated SQL.

If all went well, we now have a table with the outcome of interest. We can see how many events:

```
sql <- paste("SELECT cohort_definition_id, COUNT(*) AS count",
             "FROM @cohortDatabaseSchema.@outcomeTable",
             "GROUP BY cohort_definition_id")
```

```
sql <- renderSql(sql,
                 cohortDatabaseSchema = cohortDatabaseSchema,
                 outcomeTable = "my_outcomes")$sql
sql <- translateSql(sql, targetDialect = connectionDetails$dbms)$sql

querySql(connection, sql)
```

```
#>   cohort_concept_id  count
#> 1                 1 635684
```

4.3 Extracting the data from the server

Now we can tell `SelfControlledCaseSeries` to extract all necessary data for our analysis:

```
diclofenac <- 1124300

sccsData <- getDbSccsData(connectionDetails = connectionDetails,
                          cdmDatabaseSchema = cdmDatabaseSchema,
                          oracleTempSchema = oracleTempSchema,
                          outcomeDatabaseSchema = cohortDatabaseSchema,
                          outcomeTable = outcomeTable,
                          outcomeIds = 1,
                          exposureDatabaseSchema = cdmDatabaseSchema,
                          exposureTable = "drug_era",
                          exposureIds = diclofenac,
                          cdmVersion = cdmVersion)
sccsData
```

```
#> SCCS data object
#>
#> Exposure concept ID(s): 1124300
#> Outcome concept ID(s): 1
```

There are many parameters, but they are all documented in the `SelfControlledCaseSeries` manual. In short, we are pointing the function to the table created earlier and indicating which concept ID in that table identifies the outcome. Note that it is possible to fetch the data for multiple outcomes at once. We further point the function to the `drug_era` table, and specify the concept ID of our exposure of interest: diclofenac. Again, note that it is also possible to fetch data for multiple drugs at once. In fact, when we do not specify any exposure IDs the function will retrieve the data for all the drugs found in the `drug_era` table.

All data about the patients, outcomes and exposures are extracted from the server and stored in the `sccsData` object. This object uses the package `ff` to store information in a way that ensures R does not run out of memory, even when the data are large.

We can use the generic `summary()` function to view some more information of the data we extracted:

```
summary(sccsData)
```

```
#> sccsData object summary
#>
#> Exposure concept ID(s): 1124300
#> Outcome concept ID(s): 1
#>
#> Cases: 304906
#>
#> Outcome counts:
```

```
#>   Event count Case count
#> 1      635684     304906
#>
#> Covariates:
#> Number of covariates: 1
#> Number of covariate eras: 32227
```

4.3.1 Saving the data to file

Creating the `sccsData` file can take considerable computing time, and it is probably a good idea to save it for future sessions. Because `sccsData` uses `ff`, we cannot use R's regular save function. Instead, we'll have to use the `saveSccsData()` function:

```
saveSccsData(sccsData, "diclofenacAndGiBleed")
```

We can use the `loadSccsData()` function to load the data in a future session.

4.4 Defining a simple model

Next, we can use the data to specify a simple model to fit:

```
covarDiclofenac = createCovariateSettings(label = "Exposure of interest",
                                          includeCovariateIds = diclofenac,
                                          start = 0,
                                          end = 0,
                                          addExposedDaysToEnd = TRUE)

sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = covarDiclofenac)

summary(sccsEraData)
```

```
#> sccsEraData object summary
#>
#> Outcome ID: 1
#>
#> Outcome count:
#>   Event count Case count
#> 1       34101      15262
#>
#> Covariates:
#> Number of covariates: 1
#> Number of covariate eras: 15262
#>
#>                                  Original covariate ID
#> Exposure of interest: Diclofenac               1124300
#>                                  Current covariate ID
#> Exposure of interest: Diclofenac                 1000
```

In this example, we use the `createCovariateSettings` to define a single covariate: exposure to diclofenac. We specify that the risk window is from start of exposure to the end by setting start and end to 0, and requiring that the length of exposure is added to the end date.

We then use the covariate definition in the `createSccsEraData`, and also specify that the first 180 days of observation of every person, the so-called 'naive period', will be excluded from the analysis. Note that data in the naive period will be used to determine exposure status at the start of follow-up (after the end of the naive period). We also specify we will use all occurrences of the outcome, not just the first one per person.

4.5 Power calculations

Before we start fitting an outcome model, we might be interested to know whether we have sufficient power to detect a particular effect size. It makes sense to perform these power calculations once the study population has been fully defined, so taking into account loss to the various inclusion and exclusion criteria. This means we will use the sccsEraData object we've just created as the basis for our power calculations. Since the sample size is fixed in retrospective studies (the data has already been collected), and the true effect size is unknown, the SelfControlledCaseSeries package provides a function to compute the minimum detectable relative risk (MDRR) instead:

```
computeMdrr(sccsEraData, exposureCovariateId = 1000, alpha = 0.05, power = 0.8,
    twoSided = TRUE, method = "binomial")
```

```
#>   timeExposed timeTotal propTimeExposed propPopExposued events   mdrr
#> 1     1661649  30008800          0.0554               1  34101 1.0676
```

Note that we have to provide the covariate ID of the exposure of interest, which we learned by calling `summary` on sccsEraData earlier. This is because we may have many covariates in our model, but will likely only be interested in the MDRR of one.

4.6 Model fitting

The `fitSccsModel` function is used to fit the model:

```
model <- fitSccsModel(sccsEraData)
```

We can inspect the resulting model:

```
summary(model)
```

```
#> sccsModel object summary
#>
#> Outcome ID: 1
#>
#> Outcome count:
#>   Event count Case count
#> 1       34101      15262
#>
#> Estimates:
#>                               Name    ID  Estimate  lower .95  upper .95
#>   Exposure of interest: Diclofenac  1000     1.291      1.236      1.349
#>    logRr  seLogRr
#>   0.2556  0.02238
```

This tells us what the estimated relative risk (the incidence rate ratio) is during exposure to diclofenac compared to non-exposed time. Note that we lost some cases due to imposing the 180 day naive period.

4.7 Adding a pre-exposure window

The fact that NSAIDs like diclofenac can cause GI bleeds is well known to doctors, and this knowledge affects prescribing behavior. For example, a patient who has just had a GI bleed is not likely to be prescribed diclofenac. This may lead to underestimation of the rate during unexposed time, because the unexposed time includes time just prior to exposure where observing of the outcome is unlikely because of this behavior. One solution to this problem that is often used is to introduce a separate 'risk window' just prior to exposure, to separate it from the remaining unexposed time. We can add such a 'pre-exposure window' to our analysis:

```
covarPreDiclofenac = createCovariateSettings(label = "Pre-exposure",
                                             includeCovariateIds = diclofenac,
                                             start = -60,
                                             end = -1)

sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = list(covarDiclofenac,
                                                          covarPreDiclofenac))

model <- fitSccsModel(sccsEraData)
```

Here we created a new covariate definition in addition to the first one. We define the risk window to start 60 days prior to exposure, and end on the day just prior to exposure. We combine the two covariate settings in a list for the `createSccsEraData` function. Again, we can take a look at the results:

```
summary(model)
```

```
#> sccsModel object summary
#>
#> Outcome ID: 1
#>
#> Outcome count:
#>   Event count Case count
#> 1       34101      15262
#>
#> Estimates:
#>                                Name    ID  Estimate  lower .95  upper .95
#>   Exposure of interest: Diclofenac  1000    1.2865     1.2306      1.344
#>           Pre-exposure: Diclofenac  1001    0.9624     0.9189      1.007
#>      logRr  seLogRr
#>    0.25195  0.02255
#>   -0.03831  0.02346
```

Here we indeed see a lower relative risk in the time preceding the exposure, indicating the outcome might be a contra-indication for the drug of interest.

4.8 Splitting risk windows

Often we will want to split the risk windows into smaller parts and compute estimates for each part. This can give us insight into the temporal distribution of the risk. We can add this to the model:

```
covarDiclofenacSplit = createCovariateSettings(label = "Exposure of interest",
                                               includeCovariateIds = diclofenac,
                                               start = 0,
                                               end = 0,
```

```
                                                addExposedDaysToEnd = TRUE,
                                                splitPoints = c(7,14))

covarPreDiclofenacSplit = createCovariateSettings(label = "Pre-exposure",
                                                  includeCovariateIds = diclofenac,
                                                  start = -60,
                                                  end = -1,
                                                  splitPoints = c(-30))

sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = list(covarDiclofenacSplit,
                                                          covarPreDiclofenacSplit))
```

Here we've redefined out covariate definitions: We kept the same start and end dates, but enforced split points for the main exposure windows at 7 and 14 days. For the pre-exposure window we divided the window into two, at day 30 before the exposure start. Note that the split point dates indicate the end date of the preceding part, so the exposure is now split into day 0 to (and including) day 7, day 8 to (and including) day 14, and day 15 until the end of exposure. The results are:

```
summary(model)
```

```
#> sccsModel object summary
#>
#> Outcome ID: 1
#>
#> Outcome count:
#>   Event count Case count
#> 1       34101      15262
#>
#> Estimates:
#>                                          Name    ID  Estimate  lower .95
#>    Exposure of interest: Diclofenac, day 0-7  1000    1.3101     1.1828
#>   Exposure of interest: Diclofenac, day 8-14  1001    1.3839     1.2346
#>    Exposure of interest: Diclofenac, day 15-  1002    1.2569     1.1940
#>        Pre-exposure: Diclofenac, day -60--30  1003    1.0505     0.9893
#>           Pre-exposure: Diclofenac, day -29-  1004    0.8797     0.8228
#>   upper .95     logRr  seLogRr
#>      1.4470   0.27014  0.05143
#>      1.5450   0.32488  0.05722
#>      1.3225   0.22865  0.02608
#>      1.1143   0.04924  0.03036
#>      0.9393  -0.12816  0.03377
```

We see that the risk for the three exposure windows is more or less the same, suggesting a constant risk. We also see that the period 60 to 30 days prior to exposure does not seem to show a decreased risk, suggesting the effect of the contra-indication does not extend more than 30 days before the exposure.

4.9 Including age and seasonality

Often both the rate of exposure and the outcome change with age, and can even depend on the season. This may lead to confounding and may bias our estimates. To correct for this we can include age and/or season into the model.

For computational reasons we assume the effect of both age and season are constant within each calendar month. We assume that the rate from one month to the next can be different, but we also assume that subsequent months have somewhat similar rates. This is implemented by using cubic spline functions.

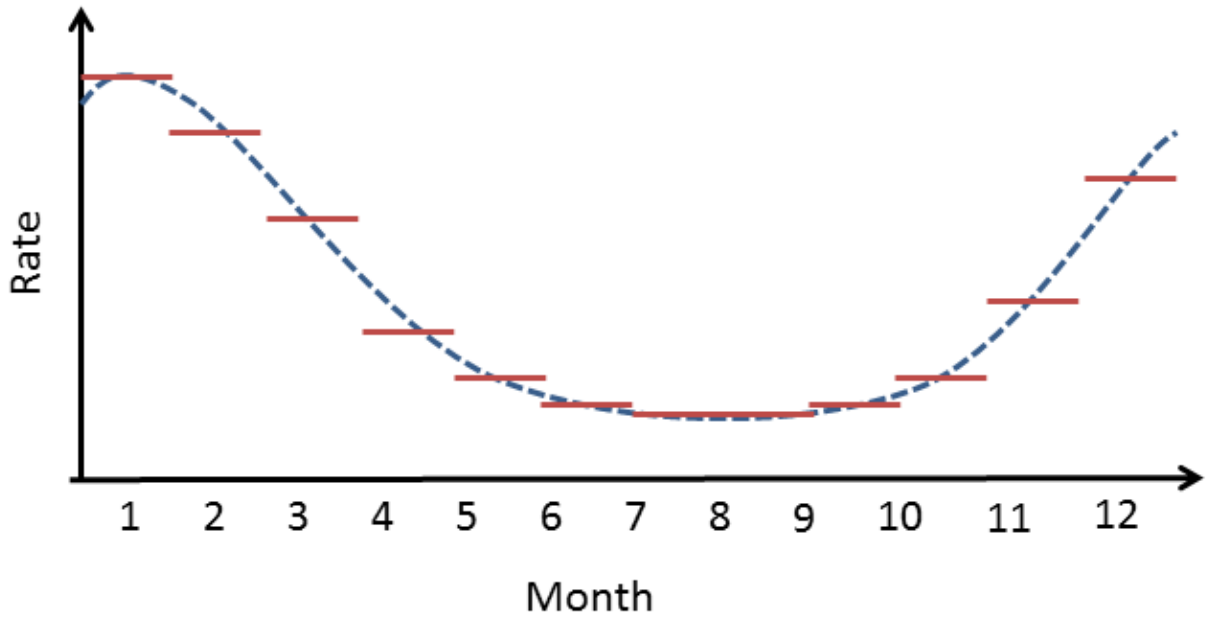


Figure 1. Example of how a spline is used for seasonality: within a month, the risk attributable to seasonality is assumed to be constant, but from month to month the risks are assumed to follow a cyclic cubic spline.

Note that the by default all people that have the outcome will be used to estimate the effect of age and seasonality on the outcome, so not just the people exposed to the drug of interest. We can add age and seasonality like this:

```
ageSettings <- createAgeSettings(includeAge = TRUE,
                                 ageKnots = 5)

seasonalitySettings <- createSeasonalitySettings(includeSeasonality = TRUE,
                                                 seasonKnots = 5)

sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = list(covarDiclofenacSplit,
                                                          covarPreDiclofenacSplit),
                                 ageSettings = ageSettings,
                                 seasonalitySettings = seasonalitySettings)

model <- fitSccsModel(sccsEraData)
```

Again, we can inspect the model:

```
summary(model)
```

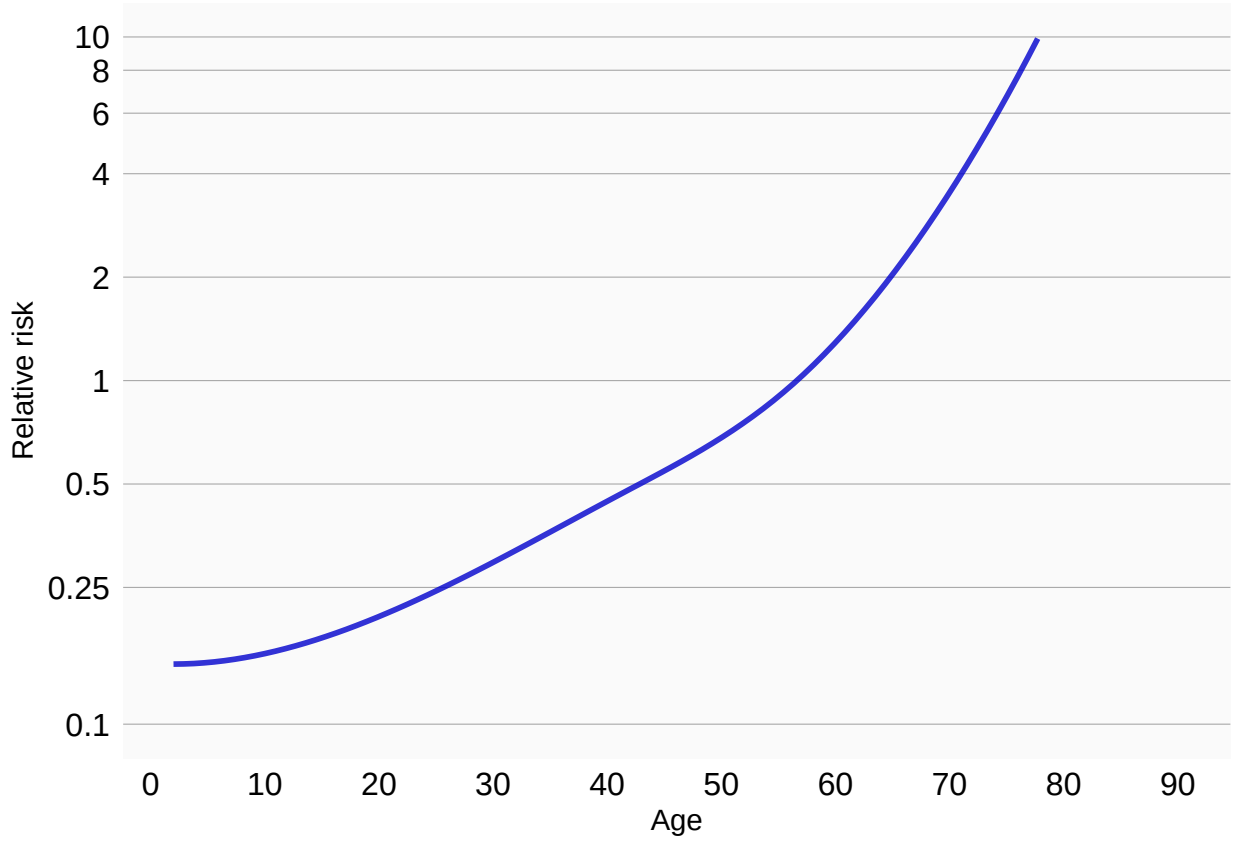
```
#> sccsModel object summary
#>
#> Outcome ID: 1
#>
#> Outcome count:
```

```
#>   Event count Case count
#> 1      477652     221883
#>
#> Estimates:
#>                                          Name    ID  Estimate  lower .95
#>                       Age spline component 1   100    2.3080     1.9211
#>                       Age spline component 2   101    5.4204     4.6248
#>                       Age spline component 3   102   23.5825    19.7502
#>                       Age spline component 4   103  138.7980   115.3770
#>                       Age spline component 5   104  567.8032   473.2041
#>               Seasonality spline component 1   200    0.9335     0.9109
#>               Seasonality spline component 2   201    1.2590     1.2427
#>               Seasonality spline component 3   202    1.0833     1.0555
#>    Exposure of interest: Diclofenac, day 0-7  1000    1.3075     1.1803
#>   Exposure of interest: Diclofenac, day 8-14  1001    1.3824     1.2333
#>    Exposure of interest: Diclofenac, day 15-  1002    1.2563     1.1933
#>        Pre-exposure: Diclofenac, day -60--30  1003    1.0503     0.9892
#>           Pre-exposure: Diclofenac, day -29-  1004    0.8783     0.8216
#>   upper .95     logRr   seLogRr
#>      2.7734   0.83639  0.093667
#>      6.3541   1.69017  0.081040
#>     28.1638   3.16050  0.090530
#>    167.0090   4.93302  0.094349
#>    681.4509   6.34177  0.093037
#>      0.9566  -0.06881  0.012485
#>      1.2754   0.23028  0.006613
#>      1.1118   0.08001  0.013266
#>      1.4441   0.26810  0.051459
#>      1.5434   0.32385  0.057220
#>      1.3219   0.22816  0.026108
#>      1.1142   0.04910  0.030365
#>      0.9379  -0.12971  0.033774
```

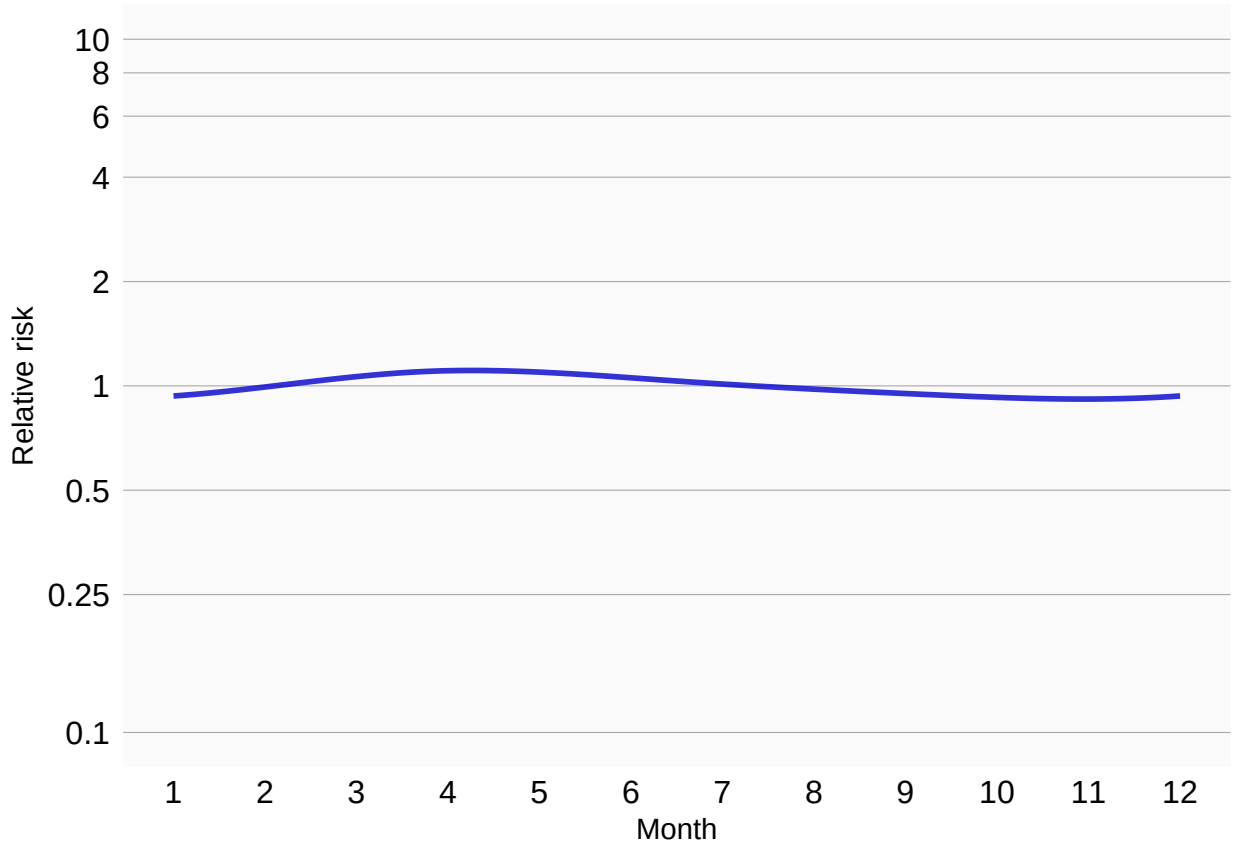
We see that our estimates for exposed and pre-exposure time have not changes much. We can plot the spline curves for age and season to learn more:

```
plotAgeEffect(model)
```

```
#> Warning: Removed 14 rows containing missing values (geom_path).
```




```
plotSeasonality(model)
```



We see a strong effect for age on the outcome, but this effect is spread out over many years and so it less likely to affect the estimates for any individual, since most people are only observed for a few years in the database. We do not see a strong effect for season.

4.10 Considering event-dependent observation time

The SCCS method requires that observation periods are independent of outcome times. This requirement is violated when outcomes increase the mortality rate, since censoring of the observation periods is then event-dependent. A modification to the SCCS has been proposed that attempts to correct for this. First, several models are fitted to estimate the amount and shape of the event-dependent censoring, and the best fitting model is selected. Next, this model is used to reweigh various parts of the observation time. This approach is also implemented in this package, and can be turned on using the `eventDependentObservation` argument of the `createSccsEraData` function:

```
sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = list(covarDiclofenacSplit,
                                                          covarPreDiclofenacSplit),
                                 ageSettings = ageSettings,
                                 seasonalitySettings = seasonalitySettings,
                                 eventDependentObservation = TRUE)

model <- fitSccsModel(sccsEraData)
```

Again, we can inspect the model:

```
summary(model)
```

```
#> sccsModel object summary
#>
#> Outcome ID: 1
#>
#> Outcome count:
#>   Event count Case count
#> 1      477652     221883
#>
#> Estimates:
#>                                          Name    ID       Estimate
#>                        Age spline component 1   100  0.00144182841
#>                        Age spline component 2   101  0.00001338050
#>                        Age spline component 3   102  0.00000047041
#>                        Age spline component 4   103  0.00000013375
#>                        Age spline component 5   104  0.00000006638
#>                Seasonality spline component 1   200  0.93393312697
#>                Seasonality spline component 2   201  1.26203707132
#>                Seasonality spline component 3   202  1.08377465617
#>    Exposure of interest: Diclofenac, day 0-7  1000  1.30672447217
#>   Exposure of interest: Diclofenac, day 8-14  1001  1.38117721934
#>    Exposure of interest: Diclofenac, day 15-  1002  1.25599408549
#>        Pre-exposure: Diclofenac, day -60--30  1003  1.04934811388
#>           Pre-exposure: Diclofenac, day -29-  1004  0.87762735714
#>      lower .95      upper .95      logRr   seLogRr
#>   0.0011994346  0.00173296015   -6.54184  0.093874
#>   0.0000114179  0.00001567801  -11.22171  0.080888
#>   0.0000003938  0.00000056177  -14.56966  0.090597
#>   0.0000001112  0.00000016092  -15.82728  0.094385
#>   0.0000000553  0.00000007966  -16.52794  0.093129
#>   0.9113429890  0.95708699447   -0.06835  0.012494
#>   1.2457895980  1.27849492265    0.23273  0.006611
#>   1.0559583046  1.11232490830    0.08045  0.013266
#>   1.1796344074  1.44331308108    0.26752  0.051464
#>   1.2322138384  1.54203332742    0.32294  0.057218
#>   1.1930389639  1.32160792743    0.22793  0.026109
#>   0.9882584365  1.11312293384    0.04817  0.030353
#>   0.8208905194  0.93707786113   -0.13053  0.033770
```

5 Studies with more than one drug

Although we are usually interested in the effect of a single drug or drug class, it could be beneficial to add exposure to other drugs to the analysis if we believe those drugs represent time-varying confounders that we wish to correct for.

5.1 Adding a class of drugs

For example, oftentimes diclofenac is co-prescribed with proton-pump inhibitors (PPIs) to mitigate the risk of GI bleeding. We would like our estimate to represent just the effect of the diclofenac, so we need to keep the effect of the PPIs separate. First we have to retrieve the information on PPI exposure from the database:

```
diclofenac <- 1124300
ppis <- c(911735, 929887, 923645, 904453, 948078, 19039926)

sccsData <- getDbSccsData(connectionDetails = connectionDetails,
                          cdmDatabaseSchema = cdmDatabaseSchema,
                          oracleTempSchema = oracleTempSchema,
                          outcomeDatabaseSchema = cohortDatabaseSchema,
                          outcomeTable = outcomeTable,
                          outcomeIds = 1,
                          exposureDatabaseSchema = cdmDatabaseSchema,
                          exposureTable = "drug_era",
                          exposureIds = c(diclofenac, ppis),
                          cdmVersion = cdmVersion)
sccsData
```

```
#> SCCS data object
#>
#> Exposure concept ID(s): 1124300,911735,929887,923645,904453,948078,19039926
#> Outcome concept ID(s): 1
```

Once retrieved, we can use the data to build and fit our model:

```
covarPpis = createCovariateSettings(label = "PPIs",
                                    includeCovariateIds = ppis,
                                    stratifyById = FALSE,
                                    start = 1,
                                    end = 0,
                                    addExposedDaysToEnd = TRUE)

sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = list(covarDiclofenacSplit,
                                                          covarPreDiclofenacSplit,
                                                          covarPpis),
                                 ageSettings = ageSettings,
                                 seasonalitySettings = seasonalitySettings,
                                 eventDependentObservation = TRUE)

model <- fitSccsModel(sccsEraData)
```

Here, we added a new covariate based on the list of concept IDs for the various PPIs. In this example we set `stratifyById` to FALSE, meaning that we will estimate a single incidence rate ratio for all PPIs, so one estimate for the entire class of drugs. Note that duplicates will be removed: if a person is exposed to two PPIs on the same day, this will be counted only once when fitting the model. Furthermore, we have set the `start` day to 1 instead of 0. The reason for this is that PPIs will also be used to treat GI bleeds, and are likely to be prescribed on the same day as the event. If we would include day 0, the risk of the outcome would be attributed to the PPI used for treatment, not the other factors that caused the GI bleed such as any exposure to our drug of interest. Again, we can inspect the model:

```
summary(model)
```

```
#> sccsModel object summary
#>
#> Outcome ID: 1
#>
```

```
#> Outcome count:
#>   Event count Case count
#> 1      477656     221884
#>
#> Estimates:
#>                                          Name    ID       Estimate
#>                        Age spline component 1   100  0.00148267108
#>                        Age spline component 2   101  0.00001451524
#>                        Age spline component 3   102  0.00000051986
#>                        Age spline component 4   103  0.00000014701
#>                        Age spline component 5   104  0.00000007313
#>                Seasonality spline component 1   200  0.93248483577
#>                Seasonality spline component 2   201  1.26264480643
#>                Seasonality spline component 3   202  1.08299048177
#>    Exposure of interest: Diclofenac, day 0-7  1000  1.31746486297
#>   Exposure of interest: Diclofenac, day 8-14  1001  1.39395384538
#>    Exposure of interest: Diclofenac, day 15-  1002  1.27228249951
#>        Pre-exposure: Diclofenac, day -60--30  1003  1.05239240243
#>           Pre-exposure: Diclofenac, day -29-  1004  0.88178136355
#>                                         PPIs  1005  0.86949649231
#>       lower .95      upper .95      logRr   seLogRr
#>   0.00123348791  0.00178196006   -6.51391  0.093846
#>   0.00001238511  0.00001700929  -11.14031  0.080936
#>   0.00000043518  0.00000062092  -14.46971  0.090678
#>   0.00000012216  0.00000017689  -15.73275  0.094445
#>   0.00000006091  0.00000008778  -16.43107  0.093197
#>   0.90998675144  0.95554302671   -0.06990  0.012462
#>   1.24638982714  1.27911025241    0.23321  0.006611
#>   1.05519525236  1.11151903260    0.07973  0.013266
#>   1.18892816556  1.45490394068    0.27571  0.051503
#>   1.24359731631  1.55633675881    0.33214  0.057227
#>   1.20801129404  1.33928887162    0.24081  0.026318
#>   0.99105910656  1.11645870362    0.05107  0.030394
#>   0.82475484633  0.94154240972   -0.12581  0.033785
#>   0.85884359717  0.88026751799   -0.13984  0.006286
```

We do see a decrease in risk when people are exposed to PPIs.

5.2 Adding all drugs

Another approach could be to add all drugs into the model. Again, the first step is to get all the relevant data from the database:

```
sccsData <- getDbSccsData(connectionDetails = connectionDetails,
                          cdmDatabaseSchema = cdmDatabaseSchema,
                          oracleTempSchema = oracleTempSchema,
                          outcomeDatabaseSchema = cohortDatabaseSchema,
                          outcomeTable = outcomeTable,
                          outcomeIds = 1,
                          exposureDatabaseSchema = cdmDatabaseSchema,
                          exposureTable = "drug_era",
                          exposureIds = c(),
                          cdmVersion = cdmVersion)
```

Note that the `exposureIds` argument is left empty. This will cause data for all concepts in the exposure table to be retrieved. Next, we simply create a new set of covariates, and fit the model:

```
covarAllDrugs = createCovariateSettings(label = "All other exposures",
                                        excludeCovariateIds = diclofenac,
                                        stratifyById = TRUE,
                                        start = 1,
                                        end = 0,
                                        addExposedDaysToEnd = TRUE,
                                        allowRegularization = TRUE)

sccsEraData <- createSccsEraData(sccsData,
                                 naivePeriod = 180,
                                 firstOutcomeOnly = FALSE,
                                 covariateSettings = list(covarDiclofenacSplit,
                                                          covarPreDiclofenacSplit,
                                                          covarAllDrugs),
                                 ageSettings = ageSettings,
                                 seasonalitySettings = seasonalitySettings,
                                 eventDependentObservation = TRUE)

model <- fitSccsModel(sccsEraData)
```

The first thing to note is that we have defined the new covariates to be all drugs except diclofenac by not specifying the `includeCovariateIds` and setting the `excludeCovariateIds` to the concept ID of diclofenac. Furthermore, we have specified that `stratifyById` is TRUE, meaning an estimate will be produced for each drug.

We have set `allowRegularization` to TRUE, meaning we will use regularization for all estimates in this new covariate set. Regularization means we will impose a prior distribution on the effect size, effectually penalizing large estimates. This helps fit the model, for example when some drugs are rare, and when drugs are almost often prescribed together and their individual effects are difficult to untangle.

Because there are now so many estimates, we will not use the `summary()` function but instead export all estimates to a data frame using `getModel()`:

```
estimates <- getModel(model)
estimates[estimates$originalCovariateId == diclofenac, ]
```

```
#>                                         name   id  estimate    lb95Ci
#> 9   Exposure of interest: Diclofenac, day 0-7 1000 1.2186165 1.0998782
#> 10 Exposure of interest: Diclofenac, day 8-14 1001 1.3125335 1.1707281
#> 11  Exposure of interest: Diclofenac, day 15- 1002 1.2029109 1.1413948
#> 12      Pre-exposure: Diclofenac, day -60--30 1003 1.0281745 0.9682846
#> 13         Pre-exposure: Diclofenac, day -29- 1004 0.8492762 0.7941782
#>       ub95Ci       logRr    seLogRr originalCovariateId
#> 9  1.3458385  0.19771621 0.05148507             1124300
#> 10 1.4656947  0.27195923 0.05732336             1124300
#> 11 1.2666515  0.18474438 0.02656319             1124300
#> 12 1.0906634  0.02778491 0.03036163             1124300
#> 13 0.9069971 -0.16337086 0.03388618             1124300
#>    originalCovariateName
#> 9             Diclofenac
#> 10            Diclofenac
#> 11            Diclofenac
#> 12            Diclofenac
```

```
#> 13            Diclofenac
```

Here we see that despite the extensive adjustments that are made in the model, the effect estimates for diclofenac have remained nearly the same.

In case we're interested, we can also look at the effect sizes for the PPIs:

```
estimates[estimates$originalCovariateId %in% ppis, ]
```

```
#>                                name   id  estimate lb95Ci ub95Ci
#> 89     Other exposures: Esomeprazole 1151 0.7068353     NA     NA
#> 96      Other exposures: rabeprazole 1174 0.8532116     NA     NA
#> 113      Other exposures: Omeprazole 1207 0.8018277     NA     NA
#> 120    Other exposures: lansoprazole 1224 0.8678528     NA     NA
#> 141    Other exposures: pantoprazole 1261 0.7008323     NA     NA
#> 446 Other exposures: dexlansoprazole 1858 0.6608101     NA     NA
#>          logRr seLogRr originalCovariateId originalCovariateName
#> 89  -0.3469576      NA               904453          Esomeprazole
#> 96  -0.1587477      NA               911735           rabeprazole
#> 113 -0.2208615      NA               923645            Omeprazole
#> 120 -0.1417331      NA               929887          lansoprazole
#> 141 -0.3554867      NA               948078          pantoprazole
#> 446 -0.4142887      NA             19039926       dexlansoprazole
```

Note that because we used regularization, we are not able to compute the confidence intervals for these estimates. We do again see that PPIs all have relative risks lower than 1 as we would expect.

6 Acknowledgments

Considerable work has been dedicated to provide the `SelfControlledCaseSeries` package.

```
citation("SelfControlledCaseSeries")
```

```
#>
#> To cite package 'SelfControlledCaseSeries' in publications use:
#>
#>   Martijn Schuemie, Patrick Ryan, Trevor Shaddox and Marc A.
#>   Suchard (2017). SelfControlledCaseSeries: Self-Controlled Case
#>   Series. R package version 1.1.1.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Manual{,
#>     title = {SelfControlledCaseSeries: Self-Controlled Case Series},
#>     author = {Martijn Schuemie and Patrick Ryan and Trevor Shaddox and Marc A. Suchard},
#>     year = {2017},
#>     note = {R package version 1.1.1},
#>   }
#>
#> ATTENTION: This citation information has been auto-generated from
#> the package DESCRIPTION file and may need manual editing, see
#> 'help("citation")'.
```

Furthermore, `SelfControlledCaseSeries` makes extensive use of the `Cyclops` package.

```
citation("Cyclops")
```

```
#>
#> To cite Cyclops in publications use:
#>
#> Suchard MA, Simpson SE, Zorych I, Ryan P and Madigan D (2013).
#> "Massive parallelization of serial inference algorithms for
#> complex generalized linear models." _ACM Transactions on Modeling
#> and Computer Simulation_, *23*, pp. 10. <URL:
#> http://dl.acm.org/citation.cfm?id=2414791>.
#>
#> A BibTeX entry for LaTeX users is
#>
#>   @Article{,
#>     author = {M. A. Suchard and S. E. Simpson and I. Zorych and P. Ryan and D. Madigan},
#>     title = {Massive parallelization of serial inference algorithms for complex generalized linear m
#>     journal = {ACM Transactions on Modeling and Computer Simulation},
#>     volume = {23},
#>     pages = {10},
#>     year = {2013},
#>     url = {http://dl.acm.org/citation.cfm?id=2414791},
#>   }
```

Part of the code (related to event-dependent observation periods) is based on the SCCS package by Yonas Ghebremichael-Weldeselassie, Heather Whitaker, and Paddy Farrington.

This work is supported in part through the National Science Foundation grant IIS 1251151.