DraftOne

Sarah Koehler

June 1, 2016

# Methods

## Generating the simulated data

### Exposure data

There are several different possible ways to generate the simulated exposure data (or to pull in observed data to use for this).

For all of the simulated exposure series, this starts by specifying the number of days to simulate, n:

n <- 1000

#### Binary exposure

For a binary exposure, then it's necessary to specify the probability of exposure, p\_exp. Then it's pretty straightforward to generate the series of exposures:

# Generate a binary exposure  
p\_exp <- 0.1  
x <- sample(c(0, 1), size = n, replace = TRUE, prob = c(1-p\_exp, p\_exp))  
head(x)

## [1] 0 0 0 0 0 0

Here's a function that takes arguments of n and p\_exp and returns a generated series (I'm also adding in a part to put in dates-- unless I specify otherwise, this will default to start at Jan. 1, 2000):

# Function to generate a binary exposure  
binary\_exposure

## function(n, p, start.date = "2000-01-01", ...){  
## start.date <- as.Date(start.date)  
## date <- seq(from = start.date, by = 1, length.out = n)  
## x <- rbinom(n, 1, p)  
## df <- data.frame(date, x)  
## return(df)  
## }  
## <environment: namespace:eesim>

binary\_exposure(n = 5, p = 0.25)

## date x  
## 1 2000-01-01 1  
## 2 2000-01-02 0  
## 3 2000-01-03 0  
## 4 2000-01-04 0  
## 5 2000-01-05 0

#### Continuous exposure

For a continuous exposure (which I'll assume it's okay to assume is normally distributed), it's necessary to specify the mean, mu, and standard deviation, sd, of the distribution of the exposure. Then it's possible to simulate the data:

# Generate a continuous exposure  
mu <- 10 # These values are both from the Chicago NMMAPS values for   
sd <- 10 # temperature in degrees Celcius  
x <- rnorm(n, mean = mu, sd = sd)  
head(x)

## [1] 22.337364 25.925189 27.936190 9.323743 -1.959334 20.094371

# Function to generate a continuous exposure  
continuous\_exposure

## function(n, mu, sd, start.date = "2000-01-01", ...){  
## start.date <- as.Date(start.date)  
## date <- seq(from = start.date, by = 1, length.out = n)  
## x <- rnorm(n, mean = mu, sd = sd)  
## df <- data.frame(date, x)  
## return(df)  
## }  
## <environment: namespace:eesim>

continuous\_exposure(n = 5, mu = 10, sd = 10)

## date x  
## 1 2000-01-01 8.063428  
## 2 2000-01-02 7.632819  
## 3 2000-01-03 -2.577870  
## 4 2000-01-04 1.445170  
## 5 2000-01-05 18.187992

#### Generate exposure data with seasonal pattern

# Get Chicago NMMAPS data and use CVD  
library(dlnm)

## Warning: package 'dlnm' was built under R version 3.2.5

## This is dlnm 2.2.6. For details: help(dlnm) and vignette('dlnmOverview').  
## Important changes: see file.show(system.file('Changesince220',package='dlnm'))

data(chicagoNMMAPS)  
chic <- chicagoNMMAPS  
names(chic)

## [1] "date" "time" "year" "month" "doy" "dow" "death" "cvd"   
## [9] "resp" "temp" "dptp" "rhum" "pm10" "o3"

cvd <- chic$cvd

#### Pull real data

# Pull exposure with a seasonal pattern  
library(dlnm)  
data(chicagoNMMAPS)  
chic <- chicagoNMMAPS  
  
if(n < nrow(chic)){  
 start <- sample(1:(nrow(chic) - n), 1) # randomly choose a starting point  
 x <- chic$temp[start:(start + n -1)]  
}  
head(x)

## [1] 6.944444 3.611111 6.944444 12.500000 11.111111 14.444444

# Function to pull exposure with a seasonal pattern  
# Note: Your `data` must have a column called `date` as well as one with the   
# measure you specify, with that column name.  
pull\_exposure <- function(n, data = "chicagoNMMAPS", measure = "temp", ...){  
 if(data == "chicagoNMMAPS"){  
 require(dlnm)  
 data(chicagoNMMAPS)  
 data <- chicagoNMMAPS  
 }  
   
 data$measure <- data[ , measure]  
   
 if(n < nrow(data)){  
 # randomly choose a starting point  
 start <- sample(1:(nrow(data) - n), 1)   
 df <- data[start:(start + n - 1), c("date", measure)]  
 } else{  
 stop("You asked for too many simulated observations given the dataset you're pulling the data from. Try a smaller value of n.")  
 }  
   
 return(df)  
}  
  
pull\_exposure(n = 3)

## date temp  
## 1569 1991-04-18 7.777778  
## 1570 1991-04-19 8.055556  
## 1571 1991-04-20 4.722222

pull\_exposure(n = 3)

## date temp  
## 4965 2000-08-04 19.44444  
## 4966 2000-08-05 19.44444  
## 4967 2000-08-06 25.55556

pull\_exposure(n = 3, measure = "pm10")

## date pm10  
## 657 1988-10-18 18.16679  
## 658 1988-10-19 20.30745  
## 659 1988-10-20 16.24807

### Expected values for outcome data

#### Constant baseline hazard rate

To generate the expected mean baseline for an outcome with a constant baseline hazard rate over time, you just need to specify the value for the distribution of the outcome (if you're using a Poisson distribution):

# Contant baseline of outcome  
lambda <- 100 # Approximate mean value of daily deaths for chicagoNMMAPS  
exp\_base\_y <- rep(lambda, n)

constant\_baseline <- function(n, lambda, start.date = "2000-01-01", ...){  
 start.date <- as.Date(start.date)  
 date <- seq(from = start.date, by = 1, length.out = n)  
 exp\_base\_y <- rep(lambda, n)  
 df <- data.frame(date = date,  
 exp\_base\_y = exp\_base\_y)  
 return(df)  
}  
constant\_baseline(n = 3, lambda = 100)

## date exp\_base\_y  
## 1 2000-01-01 100  
## 2 2000-01-02 100  
## 3 2000-01-03 100

#### Baseline hazard rate varies by season

A function to calculate the trend variable. Choices for trend are cvd, cos1, cos2, cos3, linear, curvilinear, cos1linear.

calc\_t

## function(n, trend = "no trend", custom\_func = NULL, ...){  
## day <- c(1:n)  
## if (trend == "cos1"){  
## seasont <- 1 + .6 \* cos(2 \* pi \* (day / 365))  
## } else if (trend == "cos2"){  
## seasont <- 1 + .6 \* cos(2 \* pi \* (day / 365)) +  
## ifelse(day < 639 & day > 274, .4 \* cos(2 \* (pi \* (day / 365))), 0)  
## } else if (trend == "cos3"){  
## seasont <- 1 + .75 ^ (day / 365) \* .6 \* cos(2 \* pi \* (day / 365))  
## } else if (trend == "linear"){  
## seasont <- 1 + (day / n)  
## } else if (trend == "curvilinear"){  
## seasont <- 1+ day \* (2 / n) + day^2 \* (-1 / n^2)  
## } else if (trend == "cos1linear"){  
## seasont <- (1 + (day / n)) \* (1 + .6 \* cos(2 \* pi \* (day / 365)))  
## } else if (trend == "no trend"){  
## seasont <- 1  
## } else if (trend == "custom" & !is.null(custom\_func)) {  
## arguments <- list(...)  
## arguments$n <- n  
## seasont <- do.call(custom\_func, arguments)  
## } else {  
## stop(paste0("`trend` value is not a valid choice. Please check the",  
## " function documentation to select a valid option."))  
## }  
## seasont <- seasont / mean(seasont)  
## return(seasont)  
## }  
## <environment: namespace:eesim>

We need to generate the explanatory exposure variable. Here are functions for binary and continuous exposure which varies by season.

# A function for binary exposure which varies by season:  
season\_binexp

## function(n, p, start.date = "2000-01-01", ...){  
## start.date <- as.Date(start.date)  
## date <- seq(from = start.date, by = 1, length.out = n)  
## p <- p #Change this later to reflect probability varying by season using trends  
## x <- sample(c(0, 1), size = n, replace = T, prob = c(1 - p, p))  
## df <- data.frame(date, x)  
## return(df)  
## }  
## <environment: namespace:eesim>

# A function for continuous exposure which varies by season:  
season\_contexp

## function(n, mu, sd, trend, start.date = "2000-01-01", ...){  
## day <- c(1:n)  
## t <- calc\_t(n, trend)  
## start.date <- as.Date(start.date)  
## date <- seq(from = start.date, by = 1, length.out = n)  
## mu <- mu \* t  
## x <- rnorm(n, mean = mu, sd = sd)  
## df <- data.frame(date, x)  
## return(df)  
## }  
## <environment: namespace:eesim>

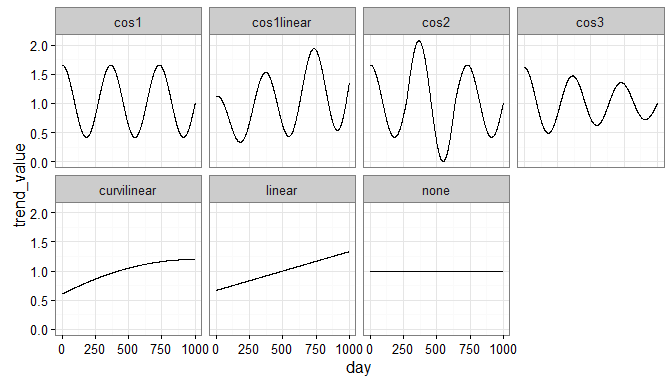
#Seasonal baseline  
seasonal\_baseline <- function(n, lambda, trend, start.date="2000-01-01", ...){  
 start.date <- as.Date(start.date)  
 date <- seq(from = start.date, by=1, length.out=n)  
 t <- calc\_t(n=n, ...)  
 exp\_base\_y <- lambda\*t  
 df <- data.frame(date=date, exp\_base\_y=exp\_base\_y)  
 return(df)  
}

Here's a plot of the trends:

library(dplyr)  
library(tidyr)

## Warning: package 'tidyr' was built under R version 3.2.5

library(ggplot2)  
  
data.frame(day = 1:1000) %>%  
 mutate(none = calc\_t(n = 1000, trend = "no trend"),  
 cos1 = calc\_t(n = 1000, trend = "cos1"),  
 cos2 = calc\_t(n = 1000, trend = "cos2"),  
 cos3 = calc\_t(n = 1000, trend = "cos3"),  
 linear = calc\_t(n = 1000, trend = "linear"),  
 curvilinear = calc\_t(n = 1000, trend = "curvilinear"),  
 cos1linear = calc\_t(n = 1000, trend = "cos1linear")) %>%  
 gather(trend\_method, trend\_value, -day) %>%  
 ggplot(aes(x = day, y = trend\_value)) +   
 geom\_line() + facet\_wrap(~ trend\_method, ncol = 4) +   
 theme\_bw()



Next, a function to simulate the outcomes.

#Here, exposure would be the output from the season\_binexp function and t would be the output from the calc\_t function.   
sim\_binout <- function(n, mean\_out, t, exposure, rr){  
 day <- c(1:n)  
 rr <- ifelse(exposure == 1, rr, 1)  
 exp\_base\_y <- mean\_out\*t  
 exp\_y <- mean\_out\*t\*rr  
 y <- rpois(n, exp\_y)  
 df <- data.frame(exp\_base\_y=exp\_base\_y, exp\_y=exp\_y, y=y)  
 return(df)  
}  
  
#Here, exposure would be the output from the season\_contexp function and t is from calc\_t.  
sim\_contout<- function(n, mu, t, exposure, rr){  
 day <- c(1:n)  
 exp\_y <- mu\*t\*rr\*(exposure/sd(exposure))  
 y <- rpois(n, exp\_y)  
 df <- data.frame( exp\_y=exp\_y, y=y)  
 return(df)  
}

Putting it all together, here is a function for data with seasonal trends. This function will return a vector of simulated outcomes.

#Put "date" in the data frame like the others?   
sim\_seasdata <- function(n, rr, trend, x\_type="binary", p, mu, sd, mean\_out){  
 if(x\_type == "binary"){  
 x <- season\_binexp(n, p)  
 t <- calc\_t(n, trend)  
 simdata <- data.frame(x, sim\_binout(n, mean\_out, t, x, rr))  
 }   
 else if(x\_type == "continuous"){  
 x <- season\_contexp(n, mu, sd, trend)  
 t <- calc\_t(n, trend)  
 simdata <- sim\_contout(n, mu, t, x, rr)  
 }  
 return(simdata)  
}

Here are some examples of this function:

sim\_seasdata(n = 3, rr = 1.2, trend="cos2", x\_type="binary", p=.2, mean\_out = 100)

## date x exp\_base\_y exp\_y.date exp\_y.x y  
## 1 2000-01-01 0 100.02037 100.02037 100.02037 104  
## 2 2000-01-02 1 100.00370 100.00370 120.00444 95  
## 3 2000-01-03 0 99.97592 99.97592 99.97592 93

# Outcome with mortality displacement  
  
# Outcome with delayed, slightly elevated risk

### Simulated outcome data

Moving from the expected value of the baseline outcome to simulated outcome data requires a few things. First, it's necessary to specify the true effect associated with the exposure:

# Specify true effect  
rr <- 1.2

Then, it's necessary to combine the exposure, x, and expected baseline outcome values, exp\_base\_y, in a dataframe:

n <- 1000  
p <- 0.25  
lambda <- 100  
  
x <- binary\_exposure(n = n, p = p)  
exp\_base\_y <- constant\_baseline(n = n, lambda = lambda)  
df <- full\_join(x, exp\_base\_y, by = "date")

Then you can generate the exp\_y using the equation:

(For this, it will be important that the relative risk is specified for a **one-unit** increase in the exposure value.)

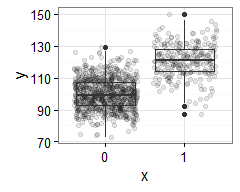
df <- mutate(df, exp\_y = exp(log(exp\_base\_y) + log(rr) \* x))

Now, you need to decide on how the outcome data is distributed around the expected value. This is simulating outcome values from a Poisson distribution using rpois.

# Simulate outcome data  
df$y <- sapply(df$exp\_y, FUN = function(x) rpois(1, x))  
head(df, 3)

## date x exp\_base\_y exp\_y y  
## 1 2000-01-01 0 100 100 101  
## 2 2000-01-02 0 100 100 108  
## 3 2000-01-03 0 100 100 90

to\_plot <- mutate(df, x = as.factor(x))  
ggplot(to\_plot, aes(x = x, y = y)) +   
 geom\_point(position = position\_jitter(), alpha = 0.1) +   
 geom\_boxplot( fill = NA) +   
 theme\_bw()



group\_by(df, exp\_y) %>% summarize(mean\_y = mean(y))

## Source: local data frame [2 x 2]  
##   
## exp\_y mean\_y  
## <dbl> <dbl>  
## 1 100 99.96159  
## 2 120 120.75102

Here is a function to do all this:

sim\_data

## function(n, rr, lambda, x\_type = "binary", trend = "constant",  
## start.date = "2000-01-01", ...){  
##   
## require(dplyr)  
##   
## if(trend=="constant"){  
## if(x\_type == "binary"){  
## x <- binary\_exposure(n=n, ...)  
## }  
## else if(x\_type == "continuous"){  
## x <- continuous\_exposure(n=n, ...)  
## }  
## exp\_base\_y <- constant\_baseline(n=n, lambda = lambda, start.date = start.date)  
## df <- full\_join(x, exp\_base\_y, by="date") %>% mutate(exp\_y =  
## exp(log(exp\_base\_y)+log(rr)\*x))  
## df$y <- sapply(df$exp\_y, FUN = function(x) rpois(1,x))  
## }  
##   
## else if(trend != "constant"){  
## if(x\_type == "binary"){  
## x <- season\_binexp(n = n, ...)  
## }  
## else if(x\_type == "continuous"){  
## x <- season\_contexp(n = n, trend = trend, ...)  
## }  
## exp\_base\_y <- seasonal\_baseline(n = n, lambda = lambda, start.date = start.date,  
## trend = trend)  
## df <- full\_join(x, exp\_base\_y, by="date") %>% mutate(exp\_y =  
## exp(log(exp\_base\_y)+log(rr)\*x))  
## df$y <- sapply(df$exp\_y, FUN = function(x) rpois(1,x))  
## }  
## return(df)  
## }  
## <environment: namespace:eesim>

Here are some examples of this function:

sim\_data(n = 3, rr = 1.2, p = .2, lambda = 100)

## date x exp\_base\_y exp\_y y  
## 1 2000-01-01 0 100 100 87  
## 2 2000-01-02 0 100 100 125  
## 3 2000-01-03 0 100 100 93

sim\_data(n = 3, rr = 1.02, x\_type = "continuous", mu = 10, sd = 10,  
 lambda = 100)

## date x exp\_base\_y exp\_y y  
## 1 2000-01-01 11.05650 100 124.4766 126  
## 2 2000-01-02 12.90733 100 129.1235 122  
## 3 2000-01-03 10.17513 100 122.3229 120

## Fitting the models

### Generalized linear model

The formula for the generalized linear model (GLM) is:

where:

* : Count of outcomes on day , dispersed as a quasi-Poisson distribution
* : Intercept (expected log count of health outcome when equals 0 and for middle of the timeframe [so time = 0])
* : log relative risk for a 1-unit increase in exposure
* : Exposure on day
* : Spline of time for degrees of freedom per year

# Fit a generalized linear model  
spline\_mod

## function(df, df\_year = 7){  
## dgrs\_free <- df\_year \* as.numeric(diff(df[c(1, nrow(df)), "date"])) / 365.4  
## df$time <- scale(df$date, center = TRUE, scale = FALSE)  
## mod <- glm(y ~ x + ns(time, round(dgrs\_free)),  
## data = df,  
## family = quasipoisson(link = "log"))  
##   
## out\_1 <- summary(mod)$coef[2, ]  
## out\_2 <- confint.default(mod)[2, ]  
## out <- c(out\_1, out\_2)  
## return(out)  
## }  
## <environment: namespace:eesim>

Here are some examples of fitting this model:

df <- sim\_data(n = 5 \* 365, rr = 1.2, p = .2, lambda = 100)  
spline\_mod(df)

## Estimate Std. Error t value Pr(>|t|) 2.5 %   
## 1.724798e-01 5.226114e-03 3.300345e+01 6.022625e-187 1.622368e-01   
## 97.5 %   
## 1.827228e-01

exp(spline\_mod(df)[1])

## Estimate   
## 1.188248

df <- sim\_data(n = 5 \* 365, rr = 1.02, x\_type = "continuous",  
 mu = 10, sd = 10, lambda = 100)  
spline\_mod(df)

## Estimate Std. Error t value Pr(>|t|) 2.5 %   
## 1.980932e-02 2.129918e-04 9.300508e+01 0.000000e+00 1.939186e-02   
## 97.5 %   
## 2.022677e-02

exp(spline\_mod(df)[1])

## Estimate   
## 1.020007

### Case-crossover model

Here is the formula for the case-crossover model. It's actually using a quasi-Poisson GLM with a factor included identifying stratum (month and year):

where:

* : Count of outcomes on day , assuming these outcomes follow a quasi-Poisson distribution
* : Intercept (expected log count of health outcome when equals 0 and for the baseline level of the time strata
* : log relative risk for a 1-unit increase in exposure
* : Exposure on day
* : offset for log baseline counts for each stratum
* : Time stratum (month and year)

# Fit a case-crossover model  
casecross\_mod

## function(df){  
## df$stratum <- factor(format(df$date, "%Y.%m"))  
##   
## if (sum(df$x == 0 | df$x == 1) == length(df$x)){  
## event.check <- as.matrix(table(df$stratum, df$x))  
## informative.strata <- rownames(event.check)[apply(event.check,  
## 1, prod) > 0]  
## df <- subset(df, stratum %in% informative.strata)  
##   
## if(length(informative.strata) > 1){  
## mod <- glm(y ~ x + stratum,  
## data = df,  
## family = quasipoisson(link = "log"))  
## } else {  
## mod <- glm(y ~ x,  
## data = df,  
## family = quasipoisson(link = "log"))  
## }  
## } else {  
## mod <- glm(y ~ x + stratum,  
## data = df,  
## family = quasipoisson(link = "log"))  
## }  
##   
## out\_1 <- summary(mod)$coef[2, ]  
## out\_2 <- confint.default(mod)[2, ]  
## out <- c(out\_1, out\_2)  
## return(out)  
## }  
## <environment: namespace:eesim>

Here are some examples of fitting this model:

df <- sim\_data(n = 5 \* 365, rr = 1.2, p = .2, lambda = 100)  
casecross\_mod(df)

## Estimate Std. Error t value Pr(>|t|) 2.5 %   
## 1.752843e-01 5.643283e-03 3.106070e+01 2.473356e-169 1.642237e-01   
## 97.5 %   
## 1.863450e-01

exp(casecross\_mod(df)[1])

## Estimate   
## 1.191585

df <- sim\_data(n = 5 \* 365, rr = 1.02, x\_type = "continuous",  
 mu = 10, sd = 10, lambda = 100)  
casecross\_mod(df)

## Estimate Std. Error t value Pr(>|t|) 2.5 %   
## 2.010146e-02 2.178805e-04 9.225910e+01 0.000000e+00 1.967442e-02   
## 97.5 %   
## 2.052849e-02

exp(casecross\_mod(df)[1])

## Estimate   
## 1.020305

### Cross-year model

# Fit a cross-year model  
crossyear\_mod

## function(df){  
## df$stratum <- factor(format(df$date, "%j"))  
## df$year <- factor(format(df$date, "%Y"))  
##   
## if (sum(df$x == 0 | df$x == 1) == length(df$x)){  
## event.check <- as.matrix(table(df$stratum, df$x))  
## informative.strata <- rownames(event.check)[apply(event.check,  
## 1, prod) > 0]  
## df <- subset(df, stratum %in% informative.strata)  
##   
## if(length(informative.strata) > 1){  
## mod <- glm(y ~ x + stratum,  
## data = df,  
## family = quasipoisson(link = "log"))  
## } else {  
## mod <- glm(y ~ x,  
## data = df,  
## family = quasipoisson(link = "log"))  
## }  
## } else {  
## mod <- glm(y ~ x + stratum,  
## data = df,  
## family = quasipoisson(link = "log"))  
## }  
##   
## out\_1 <- summary(mod)$coef[2, ]  
## out\_2 <- confint.default(mod)[2, ]  
## out <- c(out\_1, out\_2)  
## return(out)  
## }  
## <environment: namespace:eesim>

Here are some examples of applying this model:

df <- sim\_data(n = 5 \* 365, rr = 1.2, p = .2, lambda = 100)  
crossyear\_mod(df)

## Estimate Std. Error t value Pr(>|t|) 2.5 %   
## 1.852599e-01 6.259327e-03 2.959741e+01 2.449549e-136 1.729918e-01   
## 97.5 %   
## 1.975279e-01

exp(crossyear\_mod(df)[1])

## Estimate   
## 1.203531

df <- sim\_data(n = 5 \* 365, rr = 1.02, x\_type = "continuous",  
 mu = 10, sd = 10, lambda = 100)  
crossyear\_mod(df)

## Estimate Std. Error t value Pr(>|t|) 2.5 %   
## 1.984833e-02 2.546966e-04 7.792931e+01 0.000000e+00 1.934914e-02   
## 97.5 %   
## 2.034753e-02

exp(crossyear\_mod(df)[1])

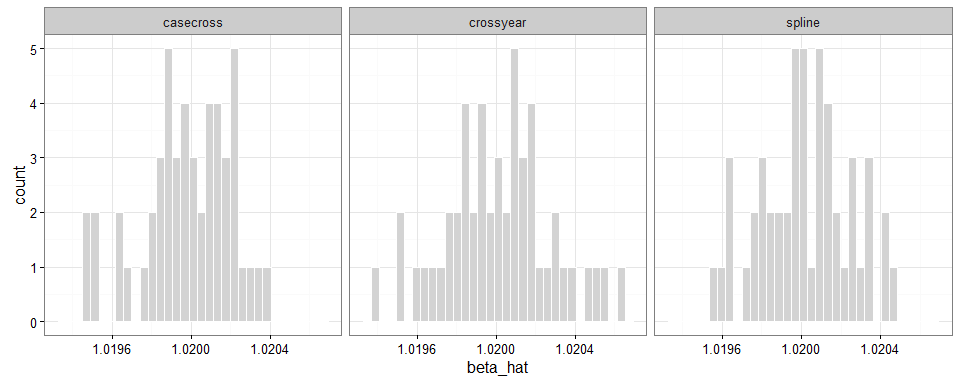
## Estimate   
## 1.020047

## Run lots of simulations

Now it's pretty easy to run lots of simulations using replicate:

n\_sim <- 50  
ex\_spline <- replicate(n\_sim, exp(spline\_mod(sim\_data(n = 5 \* 365,  
 rr = 1.02, x\_type = "continuous",  
 mu = 10, sd = 10, lambda = 100))[1]))  
  
ex\_casecross <- replicate(n\_sim, exp(casecross\_mod(sim\_data(n = 5 \* 365,  
 rr = 1.02, x\_type = "continuous",  
 mu = 10, sd = 10, lambda = 100))[1]))  
  
ex\_crossyear <- replicate(n\_sim, exp(crossyear\_mod(sim\_data(n = 5 \* 365,  
 rr = 1.02, x\_type = "continuous",  
 mu = 10, sd = 10, lambda = 100))[1]))  
  
ex <- data.frame(beta\_hat = c(ex\_spline, ex\_casecross, ex\_crossyear),  
 model = rep(c("spline", "casecross", "crossyear"),  
 each = n\_sim)) %>%  
 mutate(model = factor(model))  
  
ggplot(ex, aes(x = beta\_hat)) +   
 geom\_histogram(fill = "lightgray", color = "white") +   
 facet\_wrap(~ model, ncol = 3) +   
 theme\_bw()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



group\_by(ex, model) %>%   
 summarize(mean\_beta = mean(beta\_hat), sd\_beta = sd(beta\_hat))

## Source: local data frame [3 x 3]  
##   
## model mean\_beta sd\_beta  
## <fctr> <dbl> <dbl>  
## 1 casecross 1.019977 0.0002268384  
## 2 crossyear 1.020009 0.0002724583  
## 3 spline 1.020020 0.0002291082

Here is a function that will run a lot of simulations for a given dataset and model and will output a dataframe with the , , and p-values for the for each replication:

rep\_sims

## function(n\_sims, model, n = 5 \* 365, rr = 1.01,  
## x\_type = "continuous", mu = 10, sd = 2, lambda = 100){  
## out <- replicate(n\_sims, eval(call(model,  
## sim\_data(n = n, rr = rr,  
## x\_type = x\_type,  
## mu = mu, sd = sd,  
## lambda = lambda))))  
## out <- as.data.frame(t(out))  
## names(out) <- c("est", "se", "t", "p", "lower\_ci", "upper\_ci")  
## return(out)  
## }  
## <environment: namespace:eesim>

Here are some examples of using this function:

rep\_sims(n\_sims = 3, model = "spline\_mod")

## est se t p lower\_ci upper\_ci  
## 1 0.010682090 0.001143685 9.340061 2.763367e-20 0.008440508 0.01292367  
## 2 0.009469526 0.001155204 8.197274 4.636737e-16 0.007205367 0.01173369  
## 3 0.008783506 0.001127893 7.787534 1.148710e-14 0.006572876 0.01099414

rep\_sims(n\_sims = 3, model = "casecross\_mod")

## est se t p lower\_ci upper\_ci  
## 1 0.010228241 0.001183484 8.642486 1.211505e-17 0.007908655 0.01254783  
## 2 0.008783038 0.001151687 7.626238 3.926462e-14 0.006525773 0.01104030  
## 3 0.010579407 0.001102457 9.596211 2.717723e-21 0.008418632 0.01274018

rep\_sims(n\_sims = 3, model = "crossyear\_mod")

## est se t p lower\_ci upper\_ci  
## 1 0.009971133 0.001260758 7.908838 5.093037e-15 0.007500092 0.01244217  
## 2 0.010473754 0.001243904 8.420068 8.878486e-17 0.008035748 0.01291176  
## 3 0.011201481 0.001267740 8.835787 2.798569e-18 0.008716756 0.01368621

## Assess performance based on simulations

All of these functions will take, as input, a dataframe of replicated simulations:

rep\_df <- rep\_sims(n\_sims = 50, model = "spline\_mod", rr = 1.02)  
head(rep\_df, 4)

## est se t p lower\_ci upper\_ci  
## 1 0.02108064 0.001090133 19.33769 8.333180e-76 0.01894402 0.02321726  
## 2 0.01987391 0.001051283 18.90443 8.239720e-73 0.01781343 0.02193438  
## 3 0.01949124 0.001075214 18.12778 1.483673e-67 0.01738386 0.02159862  
## 4 0.01978757 0.001084033 18.25366 2.136337e-68 0.01766290 0.02191224

### Average estimate coefficient

This is the mean value of all the s over simulations [@figueiras2005analysis]. I'm also having this function calculate the mean of the values calculated by all the simulations.

# Measure mean of estimated coefficient  
mean\_beta

## function(df){  
## beta\_hat <- mean(df$est)  
## rr\_hat <- mean(exp(df$est))  
## out <- data.frame(beta\_hat, rr\_hat)  
## return(out)  
## }  
## <environment: namespace:eesim>

mean\_beta(rep\_df)

## beta\_hat rr\_hat  
## 1 0.01982714 1.020026

### Standard deviation of estimated coefficients

This is the standard deviation of all the s over simulations [@figueiras2005analysis]. (In other words, I think they took all the s and took the standard deviation of all those point estimates.)

Bateson and Schwartz measure the *variance* of the estimated coefficients, instead, and they took the mean value of the variance of each of the estimated coefficients (rather than the variance across all the point estimates) [-@bateson1999control]. In other words, I think they measured:

I'll write a function that will estimate both, var\_across\_betas to correspond with Figueiras and mean\_beta\_var to correspond with Bateson and Schwartz:

# Measure standard deviation across estimated coefficient  
beta\_var

## function(df){  
## var\_across\_betas <- var(df$est)  
## mean\_beta\_var <- mean(df$se^2)  
## out <- data.frame(var\_across\_betas, mean\_beta\_var)  
## return(out)  
## }  
## <environment: namespace:eesim>

beta\_var(rep\_df)

## var\_across\_betas mean\_beta\_var  
## 1 1.204097e-06 1.152494e-06

### % bias in estimated coefficient

Based on Figueiras and coauthors, this is measured as [-@figueiras2005analysis]:

where is the mean estimated coefficient (e.g., log relative risk) over simulations.

Wang and co-authors measure *relative bias* instead [-@wang2011potential]:

I'll write a function that measures both (fig\_bias and wang\_bias):

# Measure percent bias  
beta\_bias <- function(df, true\_rr){  
 fig\_bias <- 100 \* (mean(df$est) - true\_rr) / true\_rr  
 wang\_bias <- 100 \* (true\_rr - mean(df$est)) / true\_rr  
 out <- data.frame(fig\_bias, wang\_bias)  
 return(out)  
}  
beta\_bias(rep\_df, true\_rr = 1.02)

## fig\_bias wang\_bias  
## 1 -98.05616 98.05616

As a note, one of these will just be the negative of the other... Does it make more sense to just take the absolute value of the difference between the two estimates and divide it by ?

### % coverage of estimated coefficients

where is the true value of the coefficient (e.g., log relative risk) [@figueiras2005analysis].

From Butland and coauthors [@butland2013measurement]:

"An estimate of coverage probability records the percentage of simulations where the 95% confidence interval contains the 'true' value of ."

From Bateson and Schwartz [@bateson1999control]:

"Coverage probabilities are the percentages of the 1,000 confidence intervals that included the true relative risk of exposure."

These all seem to be saying the same thing.

# Measure coverage  
coverage\_beta <- function(df, true\_rr){  
 true\_beta <- log(true\_rr)  
 coverage <- df$lower\_ci <= true\_beta & df$upper\_ci >= true\_beta  
 out <- data.frame(coverage = sum(coverage) / nrow(df))  
 return(out)  
}  
  
coverage\_beta(rep\_df, true\_rr = 1.02)

## coverage  
## 1 0.94

### Power

This is the percent of all estimated coefficients for the simulations where the lower 95% confidence interval is above 0 [@figueiras2005analysis]. This is a metric of statistical efficiency.

From Butland and coauthors [@butland2013measurement]:

"An estimate of power records the percentage of simulations that would have detected the health effect estimate as statistically significant at the 5% significance level."

These also seem to be saying the same thing.

# Measure power  
power\_beta <- function(df){  
 no\_zero <- df$lower\_ci >= 0 | df$upper\_ci <= 0  
 out <- data.frame(power = sum(no\_zero) / nrow(df))  
 return(out)  
}  
  
power\_beta(rep\_df)

## power  
## 1 1

### Relative efficiency

From Bateson and Schwartz [@bateson1999control]:

"Relative efficiency is the ratio of the to the ."

Where is the means of 1,000 parameter variances from the regressions of simulated values and is the means of 1,000 parameter variances from a case-crossover model.

This looks like it is a comparative metric. I have already measured the mean variance of the s with the beta\_var function I defined above.

### Combined function to assess performance

I created a function that will perform all these checks:

check\_sims <- function(df, true\_rr){  
 a <- mean\_beta(df)  
 b <- beta\_var(df)  
 c <- beta\_bias(df, true\_rr = true\_rr)  
 d <- coverage\_beta(df, true\_rr = true\_rr)  
 e <- power\_beta(df)  
   
 out <- cbind(a, b, c, d, e)  
 return(out)  
}  
check\_sims(rep\_df, true\_rr = 1.02)

## beta\_hat rr\_hat var\_across\_betas mean\_beta\_var fig\_bias wang\_bias  
## 1 0.01982714 1.020026 1.204097e-06 1.152494e-06 -98.05616 98.05616  
## coverage power  
## 1 0.94 1

## Power calculations

Power can vary with the following elements of the analysis:

* Size of the dataset
* Effect size
* Baseline mortality rate (here, )
* Patterns in exposure (variance for continuous exposure, percent of exposed days for binary exposure)

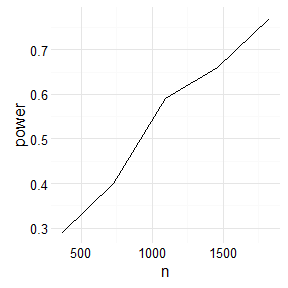
power\_calc <- function(varying, values, plot = FALSE, ...){  
 out <- data.frame(x = values, power = NA)  
 if(varying == "n"){  
 for(i in 1:nrow(out)){  
 rep\_df <- rep\_sims(n = out$x[i], ...)  
 out$power[i] <- power\_beta(rep\_df)[1,1]  
 }  
 } else if(varying == "rr"){  
 for(i in 1:nrow(out)){  
 rep\_df <- rep\_sims(rr = out$x[i], ...)  
 out$power[i] <- power\_beta(rep\_df)[1,1]  
 }  
 } else if(varying == "lambda"){  
 for(i in 1:nrow(out)){  
 rep\_df <- rep\_sims(lambda = out$x[i], ...)  
 out$power[i] <- power\_beta(rep\_df)[1,1]  
 }  
 }  
   
 if(plot == TRUE){  
 library(ggplot2)  
 my\_plot <- ggplot(out, aes(x = x, y = power)) +   
 geom\_line() + theme\_minimal() +   
 xlab(varying)  
 print(my\_plot)  
 }  
   
 colnames(out)[1] <- varying  
 return(out)  
}

Here are some examples of varying the sample size, n:

power\_calc(varying = "n", values = c(50 \* (1:5)),  
 n\_sims = 50, model = "spline\_mod", rr = 1.02)

## n power  
## 1 50 0.82  
## 2 100 0.98  
## 3 150 1.00  
## 4 200 1.00  
## 5 250 1.00

power\_calc(varying = "n", values = c(365 \* (1:5)),  
 n\_sims = 100, model = "spline\_mod", rr = 1.003, plot = TRUE)



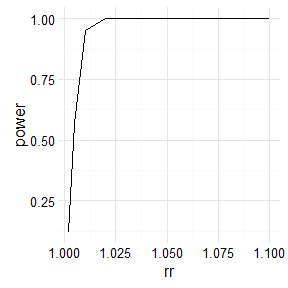
## n power  
## 1 365 0.29  
## 2 730 0.40  
## 3 1095 0.59  
## 4 1460 0.66  
## 5 1825 0.77

Here are some examples of varying the relative risk, r:

power\_calc(varying = "rr", values = c(1.002, 1.005, 1.01, 1.02, 1.03, 1.05, 1.1),  
 n\_sims = 50, model = "spline\_mod", n = 365 \* 5)

## rr power  
## 1 1.002 0.28  
## 2 1.005 1.00  
## 3 1.010 1.00  
## 4 1.020 1.00  
## 5 1.030 1.00  
## 6 1.050 1.00  
## 7 1.100 1.00

power\_calc(varying = "rr", values = c(1.002, 1.005, 1.01, 1.02, 1.03, 1.05, 1.1),  
 n\_sims = 100, model = "spline\_mod", n = 365, plot = TRUE)



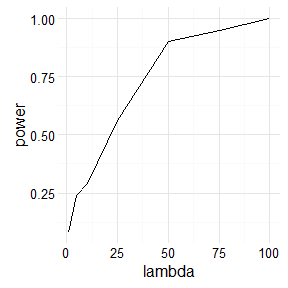
## rr power  
## 1 1.002 0.12  
## 2 1.005 0.57  
## 3 1.010 0.95  
## 4 1.020 1.00  
## 5 1.030 1.00  
## 6 1.050 1.00  
## 7 1.100 1.00

Here are some examples of varying the baseline mortality rate, lambda:

power\_calc(varying = "lambda", values = c(1, 5, 10, 25, 50, 75, 100),  
 n\_sims = 50, model = "spline\_mod", n = 365 \* 5, rr = 1.02)

## lambda power  
## 1 1 0.44  
## 2 5 1.00  
## 3 10 1.00  
## 4 25 1.00  
## 5 50 1.00  
## 6 75 1.00  
## 7 100 1.00

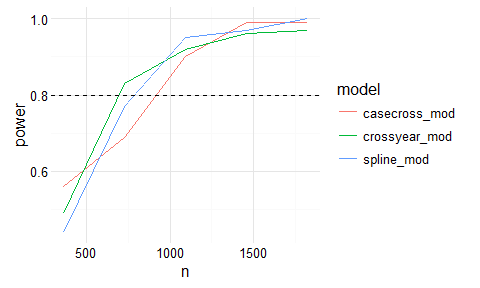
power\_calc(varying = "lambda", values = c(1, 5, 10, 25, 50, 75, 100),  
 n\_sims = 100, model = "spline\_mod", n = 365 \* 5, rr = 1.005, plot = TRUE)



## lambda power  
## 1 1 0.08  
## 2 5 0.24  
## 3 10 0.29  
## 4 25 0.56  
## 5 50 0.90  
## 6 75 0.95  
## 7 100 1.00

I can compare power from different models using the same data parameters (i.e., sample size, relative risk, etc.) for these:

for(model in c("spline\_mod", "casecross\_mod", "crossyear\_mod")){  
 df\_mod <- power\_calc(varying = "n", values = c(365 \* (1:5)),  
 n\_sims = 100, model = "spline\_mod", rr = 1.005)  
 df\_mod$model <- model  
 if(model == "spline\_mod"){  
 df <- df\_mod  
 } else {  
 df <- rbind(df, df\_mod)  
 }  
}  
df$model <- factor(df$model)  
  
ggplot(df, aes(x = n, y = power, color = model)) +  
 geom\_line() +   
 theme\_minimal() +  
 geom\_hline(aes(yintercept = 0.8), linetype = 2)



## References