Appendix S1

Alison C. Ketz

11/10/2021

1 Simulation

This appendix provides R code for generating the age-period survival data with two example hazard functions that were used in the manuscript for data generation. Each step in the data generating function will be described and then the full executable function will be provided. Stand alone executable R files for the data generating function are also provided. Then we describe how to format the data for fitting all of these models. The model fitting procedure for a single model is described in detail. We have provided code for each of the 10 models that were fit using NIMBLE, where each model has four R scripts including: 1) A script to execute and run all the additional Rscripts for each simulation as well as for providing necessary libraries. 2) A preliminary script that runs the data generating and formatting scripts, computes the necessary basis functions, and calculates constants needed for model fitting. 3) A script for running the models in nimble with additional nimble functions used for intermediary calculations. 4) An R script for doing model checking and plotting of results. These four R scripts for each model are provided for both hazard function simulations - the R-select or K-select simulation.

1.1 Kselect Data Generation

First we specify necessary constants, including the intercept for the log hazard, i.e. the baseline hazard rate, the number of individuals that are "collared", and the maximum number of age intervals and period intervals.

```
# sample size of individuals from each year
n_yr1 <- 1000
n_yr2 <- 1000
n <- n_yr1 + n_yr2

# intercept
beta0 <- -5

### maximum age interval
nT_age <- 400

### maximum period interval
nT_period <- 104
```

We generate the ages of individuals upon entry (left_age) and we must generate the period of entry (left_period). We specify individuals entering the population assuming a stable age distribution, which means that the left age entry is generated using the same baseline hazard over the ages as the age effect for the generating hazard function. We specified staggered entry of equal probability over 8 interval periods for the period of entry. We also specified staggered entry such that the approximate entry of collared individuals are split between different "years", if we consider that the intervals used for period effects reflect 2 years of a study, despite the fact that these data are simulated and the intervals could ultimately be reflecting any study period of time. The maximum age that an individual in the study can live is the difference between the maximum period and the entry period (maxtimes).

```
#######################################################
### Age at entry with staggered entry,
### drawn from a stable age distribution
### based on the age effect hazard
### for 2 'years', and for the age that
### individuals that go from birth to censoring
#######################################################

#Age effects on log hazard
#baseline is a quadratic function
age <- seq(1, nT_age - 1, by = 1)
age_effect <- 5e-05 * age^2 - 0.02 * age + 1

#Inducing wiggles during the early age intervals
anthro <- .5 * cos(1 / 26 * pi * age)
anthro[1:19] <- 0
anthro[20:38] <- anthro[20:38] * seq(0, 1, length = length(20:38))
anthro[50:170] <- anthro[50:170] * seq(1, 0, length = length(50:170))
anthro[171:(nT_age - 1)] <- 0
age_effect <- age_effect + anthro
age_effect <- age_effect - mean(age_effect)

hazard_se <- -exp((beta0 + age_effect))
stat_se <- rep(NA, nT_age - nT_period)
stat_se[1] <- (1 - exp(hazard_se[1]))
for (j in 2:(nT_age - nT_period)) {
  stat_se[j] <- (1 - exp(hazard_se[j])) * exp(sum(hazard_se[1:(j - 1)]))
}
left_age <- nimble::rcat(n, stat_se)

#########################################
# Staggered entry period effects
#########################################

weeks_entry <- 8
left_yr1 <- nimble::rcat(n_yr1, prob = rep(1 / weeks_entry, weeks_entry))
left_yr2 <- nimble::rcat(n_yr2, prob = rep(1 / weeks_entry, weeks_entry))
```

```
left_period <- c(left_yr1, left_yr2)

# no staggered entry
maxtimes <- nT_period - left_period
```

Then we must specify the hazard functions. These could be any functional (mathematical) form. We have tried to use reasonable variants that would reflect hazards for our study systems. For the age effects for the K-selected species, we used a quadratic shape curve with extra wiggles that are introduced by the additive functional shape (anthro). We allowed the additive effect to gradually start and end over the first half of the ages using the decay sequence. Then we specified the period effects using a periodic curve.

```
########################################
# Age effects for the log hazard
########################################

age <- seq(1, nT_age - 1, by = 1)
age_effect <- 5e-05 * age^2 - 0.02 * age + 1
age_effect <- age_effect + anthro
age_effect <- age_effect - mean(age_effect)

########################################
### Period effects for the log hazard
########################################

period <- seq(1, nT_period - 1, by = 1)
period_effect <- 1 * sin(2/52 * pi * (period) + 1)
period_effect <- period_effect - mean(period_effect)
```

We calculate the log hazard by the assuming the age effects and period effects are additive. Therefore, we add the age effects and period effects to the intercept baseline hazard from the age at entry to the maximum possible age, and across the period effects from left entry to the maximum possible period for each individual.

```
########################################
### Log hazard
########################################
hazard <- matrix(NA, n, nT_age)
for (i in 1:n) {
  hazard[i,] <- c(rep(0,left_age[i] - 1), beta0+
                    age_effect[left_age[i]:(left_age[i] + maxtimes[i] - 1)] +
                    period_effect[left_period[i]:(left_period[i] + maxtimes[i] - 1)],
                    rep(0, nT_age - (left_age[i] - 1 + maxtimes[i])))
}
```

Then we must calculate the probability of mortality in each interval based on the complementary log-log link function. The first period that an individual enters the study has a probability of mortality during that period that is a straightforward exponential link transformation. The subsequent periods depend on the cumulative probability of surviving the previous intervals.

```
########################################
### Probability of mortality
########################################

test_stat <- matrix(0, n, nT_age)
for (i in 1:n) {
  for (j in left_age[i]:(left_age[i] + maxtimes[i] - 1)) {
    if (j==left_age[i]) {
      test_stat[i, j] <- (1 - exp(-exp(hazard[i, j])))
    } else {
      test_stat[i, j] <-
        (1 - exp(-exp(hazard[i, j])))*exp(-sum(exp(hazard[i, left_age[i]:(j - 1)])))
    }
  }
  test_stat[i, left_age[i] + maxtimes[i]] <-
        exp(-sum(exp(hazard[i, left_age[i]:(left_age[i] + maxtimes[i] - 1))))
}
```

Based on the probability of mortality during all of the possible periods that an individual is considered part of the study, we must draw when they "fail", i.e. die, using the multinomial distribution (a categorical distribution could also be used). The age interval when the failure event occurs, or mortality event, can then be calculated based on the age at entry. If there is no mortality event, an individual is right censored. Lastly, based on the age of entry, period of entry, and right age, we can calculate the period interval of exit from the study, either through mortality or from right censoring.

```
########################################
### Calculating right censoring
########################################
fail_int <- rep(0, n)
right_age <- rep(0, n)
rt_censor <- rep(0, n)
for(i in 1:n) {
  fail_int[i] <- which(rmultinom(1, 1, test_stat[i,1:nT_age]) == 1)
  right_age[i] <- ifelse(fail_int[i] >= left_age[i] + maxtimes[i],
                         left_age[i] + maxtimes[i], fail_int[i] + 1)
  rt_censor[i] <- ifelse(fail_int[i] < (left_age[i] + maxtimes[i]), 0, 1)
}

right_period <- right_age - left_age + left_period
```

Lastly, we return values that were both specified or generated.

```
return(list(n = n,
            nT_age = nT_age,
            nT_period = nT_period,
            beta0 = beta0,
            age_effect = age_effect,
            period_effect = period_effect,
            hazard = hazard,
```

```
              test_stat = test_stat,
              right_age = right_age,
              left_age = left_age,
              right_period = right_period,
              left_period = left_period,
              rt_censor = rt_censor,
              prop_right_cens = sum(right_age == nT_age)/length(right_age)
              )
  )
```

The following code provides the full function that can be used to generate the age-period survival data. It is also provided in the R file S1_age_period_survival_generate_data_kselect.R.

```
ageperiod_surv_sim_data_kselect <- function() {

  # sample size of individuals from each year
  n_yr1 <- 1000
  n_yr2 <- 1000
  n <- n_yr1 + n_yr2

  # intercept
  beta0 <- -5

  ### maximum age interval
  nT_age <- 400

  ### maximum period interval
  nT_period <- 104

  ########################################################
  ### Age at entry with staggered entry,
  ### drawn from a stable age distribution
  ### based on the age effect hazard
  ### for 2 'years', and for the age that
  ### individuals that go from birth to censoring
  ########################################################

  #Age effects on log hazard
  #baseline is a quadratic function
  age <- seq(1, nT_age - 1, by = 1)
  age_effect <- 5e-05 * age^2 - 0.02 * age + 1

  #Inducing wiggles during the early age intervals
  anthro <- .5 * cos(1 / 26 * pi * age)
  anthro[1:19] <- 0
  anthro[20:38] <- anthro[20:38] * seq(0, 1, length = length(20:38))
  anthro[50:170] <- anthro[50:170] * seq(1, 0, length = length(50:170))
  anthro[171:(nT_age - 1)] <- 0
```

```r
age_effect <- age_effect + anthro
age_effect <- age_effect - mean(age_effect)

hazard_se <- -exp((beta0 + age_effect))
stat_se <- rep(NA, nT_age - nT_period)
stat_se[1] <- (1 - exp(hazard_se[1]))
for (j in 2:(nT_age - nT_period)) {
  stat_se[j] <- (1 - exp(hazard_se[j])) * exp(sum(hazard_se[1:(j - 1)]))
}
left_age <- nimble::rcat(n, stat_se)

##########################################
# Staggered entry period effects
##########################################

weeks_entry <- 8
left_yr1 <- nimble::rcat(n.yr1, prob = rep(1 / weeks_entry, weeks_entry))
left_yr2 <- nimble::rcat(n.yr2, prob = rep(1 / weeks_entry, weeks_entry))
left_period <- c(left_yr1, left_yr2)

# no staggered entry
maxtimes <- nT_period - left_period

########################################
# Age effects for the log hazard
########################################

age <- seq(1, nT_age - 1, by = 1)
age_effect <- 5e-05 * age^2 - 0.02 * age + 1
age_effect <- age_effect + anthro
age_effect <- age_effect - mean(age_effect)

########################################
### Period effects for the log hazard
########################################

period <- seq(1, nT_period - 1, by = 1)
period_effect <- 1 * sin(2/52 * pi * (period) + 1)
period_effect <- period_effect - mean(period_effect)

########################################
### Log hazard
########################################
hazard <- matrix(NA, n, nT_age)
for (i in 1:n) {
  hazard[i,] <- c(rep(0,left_age[i] - 1), beta0+
                    age_effect[left_age[i]:(left_age[i] + maxtimes[i] - 1)] +
```

```
                    period_effect[left_period[i]:(left_period[i] + maxtimes[i] - 1)],
                    rep(0, nT_age - (left_age[i] - 1 + maxtimes[i])))
}

########################################
### Probability of mortality
########################################

test_stat <- matrix(0, n, nT_age)
for (i in 1:n) {
  for (j in left_age[i]:(left_age[i] + maxtimes[i] - 1)) {
    if (j==left_age[i]) {
      test_stat[i, j] <- (1 - exp(-exp(hazard[i, j])))
    } else {
      test_stat[i, j] <-
        (1 - exp(-exp(hazard[i, j])))*exp(-sum(exp(hazard[i, left_age[i]:(j - 1)])))
    }
  }
  test_stat[i, left_age[i] + maxtimes[i]] <-
        exp(-sum(exp(hazard[i, left_age[i]:(left_age[i] + maxtimes[i] - 1)])))
}

########################################
### Calculating right censoring
########################################
fail_int <- rep(0, n)
right_age <- rep(0, n)
rt_censor <- rep(0, n)
for(i in 1:n) {
  fail_int[i] <- which(rmultinom(1, 1, test_stat[i,1:nT_age]) == 1)
  right_age[i] <- ifelse(fail_int[i] >= left_age[i] + maxtimes[i],
                         left_age[i] + maxtimes[i], fail_int[i] + 1)
  rt_censor[i] <- ifelse(fail_int[i] < (left_age[i] + maxtimes[i]), 0, 1)
}

right_period <- right_age - left_age + left_period

########################################
### Return values
########################################

return(list(n = n,
            nT_age = nT_age,
            nT_period = nT_period,
            beta0 = beta0,
            age_effect = age_effect,
            period_effect = period_effect,
```

```
            hazard = hazard,
            test_stat = test_stat,
            right_age = right_age,
            left_age = left_age,
            right_period = right_period,
            left_period = left_period,
            rt_censor = rt_censor,
            prop_right_cens = sum(right_age == nT_age) / length(right_age)
            )
        )
}
```

1.2 Rselect Data Generation

First we have to set constants, including the intercept for the log hazard, i.e. the baseline hazard rate, the number of individuals that are "collared", and the maximum number of age intervals and period intervals.

```
# sample size of individuals
n <- 1000

# intercept of the log hazard
beta0 <- -4

### maximum age interval
nT_age <- 130

### maximum period interval
nT_period <- 130
```

Then we must generate the ages of individuals upon entry (left_age) and we must generate the period of entry (left_period). The maximum age that an individual in the study can live to is the difference between the maximum period and the entry period (maxtimes). An individual that is of age 1 that enters into the population can be alive and a maximum age within the study of (nT_period - left_period). Here we specified all individuals entering the population at age 1, with staggered entry based on equal probability over the first 50 period intervals.

```
### For the rselect simlation, all individuals enter at age 1
left_age <- rep(1, n)

### Staggered entry during the first 50 intervals
interval_entry <- 50
left_period <- nimble::rcat(n, prob = rep( 1 / interval_entry, interval_entry))

### The maximum age that an individual can be alive and in the study
### must be calculated to restrict the right age during generation
maxtimes <- nT_period-left_period
```

Then we specify the hazard functions. These could be any functional (mathematical) form. We have tried to use reasonable variants that would reflect hazards for our study systems. For the age effect for the R-selected species, we used a Weibull shape curve with extra wiggles that are introduced by the additive functional shape (anthro). We allowed the additive effect to gradually start and end over the first half of the ages using the decay sequence. Then we specified the period effects to be a periodic curve.

```
# Age effects for the log hazard
# Baseline Weibull hazard of Age Effects
lam <-.95
age <- seq(1, nT_age - 1, by = 1)
age_effect <- 3 * lam * age^(-(1 - lam))

#Inducing wiggles during the early age intervals
anthro <- .1 * cos(1/8 * pi * age)
anthro[1:50] <- anthro[1:50] * seq(0, 1, length = length(1:50))
anthro[51:103] <- anthro[51:103] * seq(1, 0 , length = length(51:103))
anthro[103:(nT_age-1)] <- 0
age_effect <- age_effect + anthro
age_effect <- age_effect - mean(age_effect)

########################################
### Period effects for the log hazard
########################################
period <- seq(1, nT_period - 1, by = 1)
period_effect <- .5 * sin(5/(120 * pi * period) + 5)
period_effect <- period_effect - mean(period_effect)
```

We calculate the log hazard by the assuming the age effects and period effects are additive. Therefore, we add the age effects and period effects to the intercept baseline hazard from the age at entry to the maximum possible age, and across the period effects from left entry to the maximum possible period.

```
# Calculating the Hazard
hazard <- matrix(NA, n, nT_age)
for (i in 1:n) {
  hazard[i,] <- c(rep(0,left_age[i]-1), beta0 +
                    age_effect[left_age[i]:(left_age[i] + maxtimes[i] - 1)] +
                    period_effect[left_period[i]:(left_period[i] + maxtimes[i] - 1)],
                    rep(0, nT_age - (left_age[i] - 1 + maxtimes[i])))
}
```

Then we calculate the probability of mortality in each interval based on the complementary log-log link function. The first period that an individual enters the study has a probability of mortality during that period that is a straightforward inverse of the complementary log-log link transformation. The subsequent periods depend on the cumulative probability of surviving the previous intervals.

```
# Calculating the probability of mortality in each interval
test_stat <- matrix(0, n, nT_age)
for (i in 1:n) {
```

```
    for (j in left_age[i]:(left_age[i] + maxtimes[i] - 1)) {
      if (j == left_age[i]) {
        test_stat[i, j] <- (1 - exp(-exp(hazard[i, j])))
      } else {
        test_stat[i, j] <-
           (1 - exp(-exp(hazard[i, j]))) *
           exp(-sum(exp(hazard[i, left_age[i]:(j - 1)])))
      }
    }
    test_stat[i, left_age[i] + maxtimes[i]] <-
            exp(-sum(exp(hazard[i, left_age[i]:(left_age[i]
            + maxtimes[i] - 1)])))
  }
```

Based on the probability of mortality during all of the possible periods that an individual is considered part of the study, we must draw when they die using the multinomial distribution (a categorical distribution could also be used). The age when the failure, or mortality event occurs can be calculated based on the age at entry. If there is no mortality event, an individual is right censored. Lastly, based on the age of entry, period of entry, and right age, we calculate the period of exit from the study, either through mortality or from right censoring.

```
# calculating right censoring
  fail_int <- rep(0, n)
  right_age <- rep(0, n)
  rt_censor <- rep(0, n)
  for(i in 1:n){
    fail_int[i] <- which(rmultinom(1, 1, test_stat[i, 1:nT_age]) == 1)
    right_age[i] <- ifelse(fail_int[i] >= left_age[i] + maxtimes[i],
                           left_age[i] + maxtimes[i],
                           fail_int[i] + 1)
    rt_censor[i] <- ifelse(fail_int[i] < (left_age[i] + maxtimes[i]), 0, 1)
  }
  right_period <- right_age - left_age + left_period
```

Lastly, we return on values that were specified or generated.

```
  return(list(n = n,
              nT_age = nT_age,
              nT_period = nT_period,
              beta0 = beta0,
              age_effect = age_effect,
              period_effect = period_effect,
              hazard = hazard,
              test_stat = test_stat,
              right_age = right_age,
              left_age = left_age,
              right_period = right_period,
              left_period = left_period,
              rt_censor = rt_censor,
```

```
            prop_right_cens = sum(right_age == nT_age)/length(right_age)
            )
  )
```

The following code provides the full function that can be used to generate the age-period survival data. It is also provided in the R file S1_age_period_survival_generate_data_rselect.R.

```
ageperiod_surv_sim_data_rselect <- function() {

  # sample size of individuals
  n <- 1000

  # intercept of the log hazard
  beta0 <- -4

  ### maximum age interval
  nT_age <- 130

  ### maximum period interval
  nT_period <- 130

  ########################################################
  ### Age at entry. np staggered entry
  ########################################################

  ### For the rselect simlation, all individuals enter at age 1
  left_age <- rep(1, n)

  ##########################################
  ### Staggered entry period effects
  ##########################################

  ### Staggered entry during the first 50 intervals
  interval_entry <- 50
  left_period <- nimble::rcat(n, prob = rep( 1 / interval_entry, interval_entry))

  ### The maximum age that an individual can be alive and in the study
  ### must be calculated to restrict the right age during generation
  maxtimes <- nT_period-left_period

  ########################################
  ### Age effects for the log hazard
  ### Baseline Weibull hazard of Age Effects
  ########################################

  lam <-.95
  age <- seq(1, nT_age - 1, by = 1)
  age_effect <- 3 * lam * age^(-(1 - lam))
```

```r
#Inducing wiggles during the early age intervals
anthro <- .1 * cos(1/8 * pi * age)
anthro[1:50] <- anthro[1:50] * seq(0,1,length = length(1:50))
anthro[51:103] <- anthro[51:103] * seq(1,0,length = length(51:103))
anthro[103:(nT_age-1)] <- 0
age_effect <- age_effect + anthro
age_effect <- age_effect - mean(age_effect)


########################################
### Period effects for the log hazard
########################################

period <- seq(1, nT_period - 1, by = 1)
period_effect <- .5 * sin(5/(120 * pi * period) + 5)
period_effect <- period_effect - mean(period_effect)


########################################
### Log hazard
########################################

hazard <- matrix(NA, n, nT_age)

for (i in 1:n) {
  hazard[i,] <- c(rep(0,left_age[i]-1), beta0 +
                    age_effect[left_age[i]:(left_age[i] + maxtimes[i] - 1)] +
                    period_effect[left_period[i]:(left_period[i] + maxtimes[i] - 1)],
                    rep(0, nT_age - (left_age[i] - 1 + maxtimes[i])))
}

########################################
### Probability of mortality
########################################

test_stat <- matrix(0, n, nT_age)
for (i in 1:n) {
  for (j in left_age[i]:(left_age[i] + maxtimes[i] - 1)) {
    if (j == left_age[i]) {
      test_stat[i, j] <- (1 - exp(-exp(hazard[i, j])))
    } else {
      test_stat[i, j] <-
         (1 - exp(-exp(hazard[i, j]))) *
         exp(-sum(exp(hazard[i, left_age[i]:(j - 1)])))
    }
  }
  test_stat[i, left_age[i] + maxtimes[i]] <-
```

```
            exp(-sum(exp(hazard[i, left_age[i]:(left_age[i]
            + maxtimes[i] - 1)])))
  }

  ########################################
  ### Calculating right censoring
  ########################################
  fail_int <- rep(0, n)
  right_age <- rep(0, n)
  rt_censor <- rep(0, n)
  for(i in 1:n){
    fail_int[i] <- which(rmultinom(1, 1, test_stat[i, 1:nT_age]) == 1)
    right_age[i] <- ifelse(fail_int[i] >= left_age[i] + maxtimes[i],
                           left_age[i] + maxtimes[i],
                           fail_int[i] + 1)
    rt_censor[i] <- ifelse(fail_int[i] < (left_age[i] + maxtimes[i]), 0, 1)
  }

  right_period <- right_age - left_age + left_period

  ########################################
  ### Return values
  ########################################

  return(list(n = n,
              nT_age = nT_age,
              nT_period = nT_period,
              beta0 = beta0,
              age_effect = age_effect,
              period_effect = period_effect,
              hazard = hazard,
              test_stat = test_stat,
              right_age = right_age,
              left_age = left_age,
              right_period = right_period,
              left_period = left_period,
              rt_censor = rt_censor,
              prop_right_cens = sum(right_age == nT_age)/length(right_age)
              )
  )
}
```

1.3 Format data

We set a seed so that results can be recovered (set.seed(10000)). When these simulations were run on the CHTC, we used seed from (10000, ..., 10099). We generated the data by sourcing the

previous data generating functions. Then we set the true values of all parameters, and derived survival probabilities and hazards based on those true values returned from the data generating function.

```
set.seed(10000)

###simulating age period data
source("S1_01_age_period_survival_generate_data_kselect.R")
ageperiod_out<-ageperiod_surv_sim_data_kselect()

### Setting constants
nT_age <- max(ageperiod_out$right_age)-1  #ageperiod_out$nT_age-1
nT_period <- max(ageperiod_out$right_period)-1

### Setting true values for log hazard
beta0_true <- ageperiod_out$beta0
age_effect_true <- ageperiod_out$age_effect
period_effect_true <- ageperiod_out$period_effect

#Initialize vectors for sub-calculations of Survival
llambda_age_true <- rep(NA, nT_age)
UCH0_age_true <- rep(NA, nT_age)
S0_age_true <- rep(NA, nT_age)

for (t in 1:nT_age) {
  llambda_age_true[t] <- beta0_true + age_effect_true[t]
  UCH0_age_true[t] <- exp(llambda_age_true[t])
  S0_age_true[t] <- exp(-sum(UCH0_age_true[1:t]))
}

#Initialize vectors for sub-calculations of Survival
llambda_period_true <- rep(NA, nT_period)
UCH0_period_true <- rep(NA, nT_period)
S0_period_true <- rep(NA, nT_period)

for (t in 1:nT_period){
  llambda_period_true[t] <- beta0_true + period_effect_true[t] #female
  UCH0_period_true[t] <- exp(llambda_period_true[t])
  S0_period_true[t] <- exp(-sum(UCH0_period_true[1:t]))
}

### Set data generated from the data generating function
left_age <- ageperiod_out$left_age
right_age <- ageperiod_out$right_age
left_period <- ageperiod_out$left_period
right_period <- ageperiod_out$right_period
rt_censor <- ageperiod_out$rt_censor
n <- ageperiod_out$n
```

We structure the survival data such that each individual that dies during the study has 2 rows within the data frame, and individuals that are right censored or interval censored have 1 row in the data, where entry (left) and exit (right) for both age intervals and period intervals are specified. When individuals are alive and in the study, the censored variable is 1. Individuals that change state, through mortality, have censored equal to 0 for the second row of their entry. To align the age and period effects within the unit cumulative hazard calculation, i.e. to align the indexes in the for loops for calculating the hazard, a vector aligning the ages to the dates must be calculated (age2date).

```
### Formatting data for entering into the model
base <- rep(0, 5)
for (i in 1:n) {
  if (rt_censor[i] == 0) {
    temp1 <- c(left_age[i],
               right_age[i] - 1,
               1,
               left_period[i],
               right_period[i] - 1)
    temp2 <- c(right_age[i] - 1,
               right_age[i],
               0,
               right_period[i] - 1,
               right_period[i])
    if (left_age[i] == (right_age[i] - 1)) {
      base <- rbind(base, temp2)
    } else {
      base <- rbind(base, temp1, temp2)
    }
  } else {
    base <- rbind(base,c(left_age[i],
                         right_age[i],
                         1,
                         left_period[i],
                         right_period[i]))
  }
}
base <- base[-1,]
rownames(base) <- NULL
colnames(base) <- c("left_age",
                    "right_age",
                    "censored",
                    "left_period",
                    "right_period")
df_fit <- as.data.frame(base)
n_fit <- dim(df_fit)[1]

### Create age to date conversion vector
### This aligns the age/period intervals when
```

```
### looping over the age and period hazards
age2date <- df_fit$left_age - df_fit$left_period
```

The data formatting for both the R-select and K-select species is identical, except that the function that is sourced and executed for generating data should be swapped out. Consequently, we have only described the data formatting here once. We have provided full code for formatting both simulations in S1_02_format_data_kselect.R and S1_02_format_data_rselect.R. These files are sourced in the calls to the model fitting R scripts for the simulations.

1.4 Calculating model preliminaries and constants

Once the data are properly formatted, we must calculate the basis functions and other constants needed to run each model. However, the basis function calculation and calculation of constants is model dependent. Here we will describe extensively the process for calculating the basis function for the model that uses a kernel basis function for period effects, along with the additive combination of constrained generalized additive models with natural splines for the kselect simulation (CSL-K). The code for each of the model variants is provided based on the same model identifier as in Table 1 in the manuscript. The model specific basis functions are denoted in the file names.

1.4.1 K-select simulation model fitting (CSL-K)

First we source the data generating function and the data formatting R code. Note that the uncommented R scripts are for the R-select simulation data generating (S1_01_age_period_survival_generate_data_kselec
) and data formatting (S1_02_format_data_kselect.R).

```
source("../S1_01_age_period_survival_generate_data_kselect.R")
# source("../S1_01_age_period_survival_generate_data_rselect.R")

##########################################################
### Generate the data and format to run in models
##########################################################

source("../S1_02_format_data_kselect.R")
# source("../S1_02_format_data_rselect.R")
```

Then we provide a nimble function that can be used to calculate the kernel convolution process. We constrained the kernel convolutions to sum to 1, so the kernel is calculated by dividing by it's sum. The parameters of this function include the distance matrix (Z), the square root of the smoothing parameter, or precision, for the kernel (stauk), a constant (nconst) that is simply the inverse of the square root of $2 \times \pi$, the precision of the kernel (tauk), the number of knots (nknots), and the white noise process (alphau).

```
kernel_conv <- nimbleFunction(
  run = function(nT = double(0),
                 Z = double(2),
                 stauk = double(0),
                 nconst = double(0),
```

```
                tauk = double(0),
                nknots = double(0),
                alphau = double(1)
  ){
    temp <- nimMatrix(value = 0,nrow = nT, ncol = nknots)
    temp1 <- nimMatrix(value = 0,nrow = nT, ncol = nknots)
    temp2 <- nimNumeric(nknots)
    KA <- nimNumeric(nT)

    for (i in 1:nT) {
      for (j in 1:nknots) {
        temp1[i, j] <- stauk * nconst * exp(-0.5 * Z[i, j]^2 * tauk)
      }
    }
    for (j in 1:nknots) {
      temp2[j] <- sum(temp1[1:nT, j])
    }
    for (i in 1:nT) {
      for (j in 1:nknots) {
        temp[i,j] <- (temp1[i, j] / temp2[j]) * alphau[j]
      }
      KA[i] <- sum(temp[i, 1:nknots])
    }

    muKA <- mean(KA[1:nT])
    KA[1:nT] <- KA[1:nT] - muKA

    returnType(double(1))
    return(KA[1:nT])
  })

ckernel_conv <- compileNimble(kernel_conv)
```

Next, we provide the function that is used to calculate the basis function for the constrained generalized additive model (cgam) that is taken from Meyer et al (2008) and the bcgam R package.

```
convex <- function(x, t, pred.new=TRUE) {
  n = length(x)
  k = length(t)-2
  m = k + 2
  sigma = matrix(1:m*n, nrow = m, ncol = n)
  for(j in 1:(k-1)){
    i1=x<=t[j]
    sigma[j,i1] = 0
    i2=x>t[j]&x<=t[j+1]
    sigma[j,i2] = (x[i2]-t[j])^3 / (t[j+2]-t[j]) / (t[j+1]-t[j])/3
    i3=x>t[j+1]&x<=t[j+2]
    sigma[j,i3] = x[i3]-t[j+1]-(x[i3]-t[j+2])^3/(t[j+2]-t[j])/(t[j+2]-t[j+1])/3+(t[j+1]-t[j])^
```

```
    i4=x>t[j+2]
    sigma[j,i4]=(x[i4]-t[j+1])+(t[j+1]-t[j])^2/3/(t[j+2]-t[j])-(t[j+2]-t[j+1])^2/3/(t[j+2]-t[j
  }
  i1=x<=t[k]
  sigma[k,i1] = 0
  i2=x>t[k]&x<=t[k+1]
  sigma[k,i2] = (x[i2]-t[k])^3 / (t[k+2]-t[k]) / (t[k+1]-t[k])/3
  i3=x>t[k+1]
  sigma[k,i3] = x[i3]-t[k+1]-(x[i3]-t[k+2])^3/(t[k+2]-t[k])/(t[k+2]-t[k+1])/3+(t[k+1]-t[k])^2/
  i1=x<=t[2]
  sigma[k+1,i1]=x[i1]-t[1]+(t[2]-x[i1])^3/(t[2]-t[1])^2/3
  i2=x>t[2]
  sigma[k+1,i2]=x[i2]-t[1]
  i1=x<=t[k+1]
  sigma[k+2,i1]=0
  i2=x>t[k+1]
  sigma[k+2,i2]=(x[i2]-t[k+1])^3/(t[k+2]-t[k+1])^2/3

  v1=1:n*0+1
  v2=x
  x.mat=cbind(v1,v2)

  if(pred.new==TRUE){
    list(sigma=sigma,x.mat=x.mat)}

  else{
    if(pred.new==FALSE){
      coef=solve(t(x.mat)%*%x.mat)%*%t(x.mat)%*%t(sigma)
      list(sigma=sigma, x.mat=x.mat, center.vector=coef)}
  }

}
```

For the cgam basis, we specify 6 knots using quantiles of the right_age vector. Using the above convex function, we calculate the basis function for the age effects so that they are neither decreasing nor increasing. We re-scaled the basis function by dividing the function values by the maximum value to aid in convergence.

```
quant_age <- .25

knots_age <- c(1,
               round(quantile(df.fit$right_age - 1,
                              c(seq(quant_age, .99, by = quant_age), .99))))
knots_age <- unique(knots_age)

delta_i <- convex(1:nT_age,knots_age, pred.new=FALSE)
delta <- t(rbind(delta_i$sigma - t(delta_i$x.mat %*% delta_i$center.vector)))
delta <- delta / max(delta)
```

```
Z_age_cgam <- delta
nknots_age_cgam <- dim(Z_age_cgam)[2]
```

Next, we calculate the basis function for the natural spline for the age effects. First we set knots based on the quantiles of the right_age vector such that there are at least 20 knots. Then we calculate the natural spline basis from the splines R package function. Then we use QR factorization to obtain the null spce of the spline basis, so that we can impose the sum to zero constraint for the age effects.

```
### Knots
quant_age <- .05
knots_age_spline <- c(1,
                      round(quantile(df.fit$right_age-1,
                                     c(seq(quant_age,.99, by=quant_age),
                                     .99))))
knots_age_spline <- unique(knots_age_spline)
nknots_age_spline <- length(knots_age_spline)

### Basis for age hazard
splinebasis <- ns(1:nT_age, knots = knots_age_spline)

### A constraint matrix so period_effects == 0
constr_sumzero <- matrix(1, 1, nrow(splinebasis)) %*% splinebasis

### QR factorization for null space of constraint
qrc <- qr(t(constr_sumzero))
Z <- qr.Q(qrc,
          complete = TRUE)[, (nrow(constr_sumzero) + 1):ncol(constr_sumzero)]
Z_age_spline <- splinebasis %*% Z
nknots_age_spline <- dim(Z_age_spline)[2]
```

We plot the basis functions.

```
pdf("figures/basis_function_age.pdf")
plot(1:nT_age,
     Z_age_spline[, 1],
     ylim = c(-1, 1),
     type = "l",
     main = "Basis Function Age Effect Spline")
for (i in 2:nknots_age_spline) {
  lines(1:nT_age, Z_age_spline[, i])
}
plot(1:nT_age,
     Z_age_cgam[, 1],
     ylim = c(-1, 1),
     type = "l",
     main = "Basis Function Age Effect CGAM")
for (i in 2:nknots_age_cgam) {
```

```
  lines(1:nT_age, Z_age_cgam[, i])
}
dev.off()
```

The we calculate the basis functions for the period effects for the kernel convolution process. First we set the knots, where we used dense equal space knots on each period interval. Then the basis function is just the absolute value of the distance between each period and the neighboring periods.

```
intvl_period <- 1
knots_period <- seq(1, nT_period, by = intvl_period)
knots_period <- unique(knots_period)
nknots_period <- length(knots_period)

Z_period <- matrix(0, nT_period, nknots_period)
for (i in 1:nrow(Z_period)) {
  for (j in 1:nknots_period) {
    Z_period[i, j] <- abs(i - knots_period[j])
  }
}
```

Lastly, we specify the number of Markov chain Monte-Carlo iterations (reps), the burn-in (bin), the number of MCMC chains (n_chains), and the thinning interval (n_thin).

```
reps <- 50000
bin <- reps * .5
n_chains <- 3
n_thin <- 1
```

The full preliminary function to run all of these basis function calculations is provided in S1_03_prelim_constants_kselect_CSL_K.R.

1.5 Run the model using the NIMBLE package

We provide a function that can calculate the probability of a mortality for each individual across the intervals (age and period) that the individual is collared and alive in the study. The function is based on indexing over the age effects (age_effect), and uses the age2date vector to calculate the proper index for the period intervals (period_effect). The intercept (beta0) is added to the log hazard. The summation from left to right provides the approximation of the integration of the log hazard for obtaining the cumulative probability of survival during each age interval and each period interval.

```
state_transition <- nimbleFunction(
  run = function(records = double(0),
                 left = double(1),
                 right = double(1),
                 beta0 = double(0)
                 age_effect = double(1),
                 period_effect = double(1),
                 age2date = double(1),
```

```
                nT_age = double(0)
){

  SLR <- nimNumeric(records)
  UCH <-nimMatrix(value = 0,nrow = records, ncol = nT_age)
  for (j in 1:records) {
    for (k in left[j]:(right[j] - 1)) {
      UCH[j, k] <- exp(beta0 +
                       age_effect[k] +
                       period_effect[k - age2date[j]])
    }
    SLR[j] <- exp(-sum(UCH[j, left[j]:(right[j] - 1)]))
  }
  returnType(double(1))
  return(SLR[1:records])
})

cstate_transition <- compileNimble(state_transition)
```

Then we specify the model statement using the NIMBLE syntax, which is based on declarative BUGS code. We specify the prior distribution for the intercept (beta0) using parameter expansion.

```
beta0_temp ~ dnorm(0, .01)
mix ~ dunif(-1, 1)
beta0 <- beta0_temp * mix
```

Then we specify the prior distribution for the effects for the cgam model, and calculate the age effects. We centered the age effects for the cgam model by subtracting the mean of the age effects and then used the sum-to-zero age effects. We could not use QR factorization to obtain the null space of the basis function when calculating the basis function, because this mapped the basis functions that prevented the non-negative constraint for the coefficients to ensure the concave shape.

```
for (k in 1:nknots_age_cgam) {
  ln_b_age_cgam[k] ~ dnorm(0, tau_age_cgam)
  b_age_cgam[k] <- exp(ln_b_age_cgam[k])
}
tau_age_cgam ~ dgamma(1, 1)
for (t in 1:nT_age) {
  age_effect_temp[t] <- inprod(b_age_cgam[1:nknots_age_cgam],
                               Z_age_cgam[t, 1:nknots_age_cgam])
  age_effect_cgam[t] <- age_effect_temp[t] - mu_age_cgam
}
mu_age_cgam <- mean(age_effect_temp[1:nT_age])
```

The prior for spline model was specified using the double exponential, or Laplace prior rather than a normal prior distribution. The QR factorization that was implemented on the basis function ensures these effects are summing to zero. We combined the cgam and spline models by adding them.

```
for (k in 1:nknots_age_spline) {
  b_age_spline[k] ~ ddexp(0, tau_age_spline)
}
tau_age_spline ~ dgamma(.01, .01)
for (t in 1:nT_age) {
  age_effect_spline[t] <- inprod(b_age_spline[1:nknots_age_spline],
                                 Z_age_spline[t, 1:nknots_age_spline])
}

########################
### Age effect combine
########################

age_effect[1:nT_age] <- age_effect_cgam[1:nT_age] + age_effect_spline[1:nT_age]
```

We calculate the period effects using the kernel convolution function provided earlier (period_effect). We specify the prior for the kernel smoother on the log scale (ln_sk_period), along with a parameter expansion mixing parameter to aid in mixing (mix2). We generate the Gaussian white noise process (alphau_period) by generating the standard normal (alpha_period) and then multiply this by the standard deviation (sda_period) that has a truncated normal prior.

```
mix2 ~ dunif(-1, 1)
ln_sk_period ~ dnorm(0, sd = 1)
sdk_period <- exp(mix2 * ln_sk_period)
tauk_period <- 1 / sdk_period^2
stauk_period <- sqrt(tauk_period)
sda_period ~ T(dnorm(0, sd = 1),0, Inf)
taua_period <- 1 / sda_period^2
for (i in 1:(nknots_period)) {
  alpha_period[i] ~ dnorm(0, 1)
  alphau_period[i] <- sda_period * alpha_period[i]
}
ratioinf_period <- sdk_period / sda_period #ratio of variability/smoothing

period_effect[1:nT_period] <- kernel.conv(
  nT = nT_period,
  Z = Z_period[1:nT_period, 1:nknots_period],
  stauk = stauk_period,
  nconst = nconst,
  tauk = tauk_period,
  nknots = nknots_period,
  alphau = alphau_period[1:nknots_period]
)
```

Then we compute the cumulative probability of mortality within each period using the state_transition function. The likelihood is just a Bernoulli distribution for each individual given whether they are alive and in the study (censor=1), or whether they have died (censor=0).

```
  SLR[1:records] <- state_transition(records = records,
                                     left = left_age[1:records],
                                     right = right_age[1:records],
                                     nT_age = nT_age,
                                     age_effect = age_effect[1:nT_age],
                                     period_effect = period_effect[1:nT_period],
                                     age2date = age2date[1:records],
                                     beta0 = beta0)

  ###
  ### Likelihood
  ###
  for (j in 1:records) {
    censor[j] ~ dbern(SLR[j])
  }
```

We calculate derived parameters consisting of the cumulative probability of survival over the age intervals and period intervals separately. These could be calculated differently, where the could be combined by adding over specific intervals, or a full survival surface could also be calculated.

```
  for (t in 1:nT_age) {
    llambda_age[t] <- beta0 + age_effect[t]
    UCH0_age[t] <- exp(llambda_age[t])
    S0_age[t] <- exp(-sum(UCH0_age[1:t]))
  }
  for (t in 1:nT_period) {
    llambda_period[t] <- beta0 + period_effect[t]
    UCH0_period[t] <- exp(llambda_period[t])
    S0_period[t] <- exp(-sum(UCH0_period[1:t]))
  }
```

We put this altogether in the modelcode statement.

```
modelcode <- nimbleCode({
  ### Prior for intercept
  ### using parameter expansion for convergence
  beta0_temp ~ dnorm(0, .01)
  mix ~ dunif(-1, 1)
  beta0 <- beta0_temp * mix

  ######################
  ### Age effect cgam
  ######################

  for (k in 1:nknots_age_cgam) {
    ln_b_age_cgam[k] ~ dnorm(0, tau_age_cgam)
    b_age_cgam[k] <- exp(ln_b_age_cgam[k])
  }
  tau_age_cgam ~ dgamma(1, 1)
```

```
for (t in 1:nT_age) {
  age_effect_temp[t] <- inprod(b_age_cgam[1:nknots_age_cgam],
                               Z_age_cgam[t, 1:nknots_age_cgam])
  age_effect_cgam[t] <- age_effect_temp[t] - mu_age_cgam
}
mu_age_cgam <- mean(age_effect_temp[1:nT_age])

######################
### Age effect spline
######################

for (k in 1:nknots_age_spline) {
  b_age_spline[k] ~ ddexp(0, tau_age_spline)
}
tau_age_spline ~ dgamma(.01, .01)
for (t in 1:nT_age) {
  age_effect_spline[t] <- inprod(b_age_spline[1:nknots_age_spline],
                                 Z_age_spline[t, 1:nknots_age_spline])
}

########################
### Age effect combine
########################

age_effect[1:nT_age] <- age_effect_cgam[1:nT_age] + age_effect_spline[1:nT_age]

####################################
### Period effect kernel convolution
####################################

mix2 ~ dunif(-1, 1)
ln_sk_period ~ dnorm(0, sd = 1)
sdk_period <- exp(mix2 * ln_sk_period)
tauk_period <- 1 / sdk_period^2
stauk_period <- sqrt(tauk_period)
sda_period ~ T(dnorm(0, sd = 1),0, Inf)
taua_period <- 1 / sda_period^2
for (i in 1:(nknots_period)) {
  alpha_period[i] ~ dnorm(0, 1)
  alphau_period[i] <- sda_period * alpha_period[i]
}
ratioinf_period <- sdk_period / sda_period #ratio of variability/smoothing

period_effect[1:nT_period] <- kernel.conv(
  nT = nT_period,
  Z = Z_period[1:nT_period, 1:nknots_period],
  stauk = stauk_period,
```

```
    nconst = nconst,
    tauk = tauk_period,
    nknots = nknots_period,
    alphau = alphau_period[1:nknots_period]
  )

  ###
  ### Computing state transition probability
  ###

  SLR[1:records] <- state_transition(records = records,
                                    left = left_age[1:records],
                                    right = right_age[1:records],
                                    nT_age = nT_age,
                                    age_effect = age_effect[1:nT_age],
                                    period_effect = period_effect[1:nT_period],
                                    age2date = age2date[1:records],
                                    beta0 = beta0)

  ###
  ### Likelihood
  ###
  for (j in 1:records) {
    censor[j] ~ dbern(SLR[j])
  }

  ############################
  ### Derived parameters
  ############################

  for (t in 1:nT_age) {
    llambda_age[t] <- beta0 + age_effect[t]
    UCH0_age[t] <- exp(llambda_age[t])
    S0_age[t] <- exp(-sum(UCH0_age[1:t]))
  }
  for (t in 1:nT_period) {
    llambda_period[t] <- beta0 + period_effect[t]
    UCH0_period[t] <- exp(llambda_period[t])
    S0_period[t] <- exp(-sum(UCH0_period[1:t]))
  }

})#end model statement
```

We specify data, constants and initial values for all of the parameters in the model.

```
###Data
nimData <- list(censor = df_fit[, 3],
                Z_period = Z_period,
```

```
                Z_age_cgam = Z_age_cgam,
                Z_age_spline = Z_age_spline,
                left_age = df_fit[, 1],
                right_age = df_fit[, 2],
                age2date = age2date
                )
###Constants
nimConsts <- list(records = n_fit,
                  nT_age = nT_age,
                  nT_period = nT_period,
                  nknots_age_cgam = nknots_age_cgam,
                  nknots_age_spline = nknots_age_spline,
                  nknots_period = nknots_period,
                  nconst = 1 / sqrt(2 * pi))

### Initial values
initsFun <- function()list(tau_age_cgam = runif(1, .1, 1),
                           tau_age_spline = runif(1, .1, 1),
                           beta0_temp = rnorm(1, beta0_true, .0001),
                           mix = 1,
                           mix2 = 1,
                           sda_period = runif(1, 0, 5),
                           ln_sk_period = rnorm(1, 0, 1),
                           alpha_period = rep(0, nknots_period),
                           ln_b_age_cgam = runif(nknots_age_cgam, -10, -5),
                           b_age_spline = rnorm(nknots_period) * 10^-4
                           )
nimInits <- initsFun()
```

Then we build the model, MCMC object, and run the MCMC approximation. We save the MCMC iterations.

```
Rmodel <- nimbleModel(code = modelcode,
                      constants = nimConsts,
                      data = nimData,
                      inits = initsFun()
                      )

#identify params to monitor
parameters <- c(
               "beta0",
               "S0_age",
               "age_effect",
               "age_effect_cgam",
               "age_effect_spline",
               "b_age_spline",
               "b_age_cgam",
               "tau_age_cgam",
```

```
                "tau_age_spline",
                "mu_age_cgam",
                "period_effect",
                "S0_period",
                "llambda_period",
                "sdk_period",
                "sda_period",
                "alpha_period",
                "ratioinf_period"
                )

starttime<-Sys.time()
confMCMC <- configureMCMC(Rmodel,
                          monitors = parameters,
                          thin = n_thin,
                          useConjugacy = FALSE)
nimMCMC <- buildMCMC(confMCMC, enableWAIC = TRUE)
Cnim <- compileNimble(Rmodel)
CnimMCMC <- compileNimble(nimMCMC, project=Rmodel)
mcmcout <- runMCMC(CnimMCMC,
                   niter = reps,
                   nburnin = bin,
                   nchains = n_chains,
                   inits = initsFun,
                   samplesAsCodaMCMC = TRUE,
                   summary = TRUE,
                   WAIC = TRUE)

runtime <- diffperiod_(Sys.time(), starttime, units = "min")

save(runtime, file = "results/runtime.Rdata")
save(mcmcout, file = "results/mcmcout.Rdata")
```

All of the code in this section is aggregated and combined in the file S1_04_run_model_kselect_CSL_K.R. Simulations were run on the University of Wisconsin, Madison's center for high though-put computing cluster. For each of the simulations, a different seed was used to run the models. Code for plotting this results of the simulation from a single seed, as well as for obtaining convergence diagnostics will be provided in the file S1_05_post_plots_kselect_CSL_K.R.

It is not possible to extensively describe all of the models from both the simulations and from the case studies. However, they were all fit using similar code/methods. A key for fitting these models in in having to calculate the basis functions. In the model described above, several of the basis functions were used. I have aggregated all of the basis function calculations from all of the models in the document S1_03_basis_functions.R.

Code for running the kselect simulation for a single seed has been provided. To avoid redundant code, these same models were used for the rselect simulation for the single seed, as well as for both case studies. Data sets from the white-tailed deer and Columbian sharp-tailed grouse case studies

were used rather than generated data. Code for loading these data are provided in the corresponding case study directories.