China-btfr-superfast.R

Carl Schmertmann

2020-01-29

```
# Carl Schmertmann
# created 07 May 2018
# edited  11 Jul 2019
#
# superfast_bTFR function (does NOT require Stan)
#
# inputs: integer number of children C [may be an n-vector]
#         counts of women W = (W15...W45) [may be an nx7 matrix]
#         scalar or vector of any available q5 estimates [may be a nx(#est) matrix]
#         lowest possible TFR value under consideration (LTFR)
#         highest possible TFR value under consideration (HTFR)
#
# output: if n=1, a single 4-component list containing
#            $ a density object for TFR
#            $ the 10,50,90%iles of the posterior distribution
#         if n>1, a list of 4-component lists, one per population
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(LearnBayes)

superfast_bTFR = function(C , W , q5_est ,
                          LTFR = 0, HTFR = 20, delta_TFR=.02,
                          Lq5 = .0001, Hq5  =.0999,
                          rownames=NULL) {
  require(assertthat)
  require(LearnBayes)
  n = length(C)
  W_matrix  = matrix(W, nrow=n)
  q5_matrix = matrix(q5_est, nrow=n)

  if (n==1) {
    assert_that( length(W)==7,
                 msg='There is only one C value, but W does not have 7 elements')
  } else if (n>1) {
    assert_that( nrow(W)==n, ncol(W)==7,
                 msg='C and W are incompatible')
    if (class(q5_est)=='matrix') {
```

```
    assert_that( nrow(q5_est)==n,
                 msg='C and matrix of q5_est are incompatible')
  }
  if (class(q5_est)=='numeric') {
    assert_that( length(q5_est)==n,
                 msg='C and vector of q5_est are incompatible')
  }

}

result = vector('list',n)  # container for results
if (!is.null(rownames)) names(result) = rownames

#--------------------------
# fertility: phi10...phi45
# phi_a is the proportion of
# lifetime fertility during
# age group [a,a+5)
#--------------------------

m = c(0, 1.39212, 1.59237, 1.22909, 0.4523, -0.88676, -3.43722)
names(m) = seq(15,45,5)

X = matrix( c(0, 0.27443, 0.54078, 0.73193, 0.87739, 1.04049,  1.51719,
              0, 0.31782, 0.51079, 0.51072, 0.34831, 0.05289, -0.72364),
            nrow=7,ncol=2)
rownames(X) = seq(15,45,5)

phi = function(beta1,beta2) {

  gamma = as.vector( m + X %*% c(beta1,beta2))  # log odds rel to F15
  phi   = exp(gamma) / sum(exp(gamma))          # proportion of lifetime fertil by age group

  result = c(0, phi)
  names(result) = seq(10,45,5)

  return(result)

} # phi

#--------------------------
# mortality: L0,L5,...L45
#--------------------------
## function to convert (q5,k) into life table L column in Wilmoth et al. model

bigL = function(q5,k) {

  ## mortality constants for females, ages 0,1,...45
  ## Wilmoth et al. coefficients from Pop Studies
  ## J Wilmoth, S Zureick, V Canudas-Romo, M Inoue & C Sawyer (2011)
  ## A flexible two-dimensional mortality model for use in indirect estimation,
  ## Population Studies, 66:1, 1-28, DOI: 10.1080/00324728.2011.611411
```

```
  af = c(-0.6619, -99, -2.5608, -3.2435, -3.1099, -2.9789, -3.0185,
         -3.0201, -3.1487, -3.269, -3.5202)
  bf = c(0.7684, -99, 1.7937, 1.6653, 1.5797, 1.5053, 1.3729, 1.2879,
         1.1071, 0.9339, 0.6642)
  cf = c(-0.0277, -99, 0.1082, 0.1088, 0.1147, 0.1011, 0.0815, 0.0778,
         0.0637, 0.0533, 0.0289)
  vf = c(0, -99, 0.2788, 0.3423, 0.4007, 0.4133, 0.3884, 0.3391, 0.2829,
         0.2246, 0.1774)

  # log mortality rates for age intervals starting at x=0,1,5,10,...,45
  logmx = af + bf * log(q5) + cf * (log(q5))^2 + vf * k
  names(logmx) = c(0,1,seq(5,45,5))

  # mortality rates (special treatment for 4m1, because q5 has to match input)
  mx = exp(logmx)
  mx['1'] = -1/4 * (mx['0'] + log(1-q5))

  # survival (little_l)
  n = c(1,4,rep(5,9))
  little_l = c(1, exp( -cumsum( n*mx )))
  names(little_l) = c(0,1,seq(5,50,5))

  # person-years (bigL, trapezoidal approximation)
  # [0,1) [1,5) [5,10) ... [45,50)
  L = n/2 * ( head(little_l, -1) + tail(little_l, -1) )

  # collapse the first 2 intervals to make 5-year groups [0,5) [5,10) ... [45,50)
  result = c(L['0'] + L['1'], tail(L,-2))

  return(result)

} # bigL

#------------------------------------------------
# Expected children at TFR=1
# eta_a is the expected number
# of surviving 0-4 year old children per woman currently
# in age group [a,a+5) if TFR=1
#------------------------------------------------

eta = function( q5,k, beta1, beta2) {

  this.phi = phi( beta1, beta2)
  this.L   = bigL(q5,k)

  surv = this.L[paste(seq(10,40,5))]/this.L[paste(seq(15,45,5))]

  eta = this.L['0']/5 * 1/2 * (surv * this.phi[paste(seq(10,40,5))]  + this.phi[paste(seq(15,45,5))])
  names(eta) = seq(15,45,5)

  return(eta)
}
```

```r
#-------------------------------------------------
# create a large grid of (q5,k,beta1,beta2) values
#-------------------------------------------------
q5_vals = Lq5 + (Hq5-Lq5)*1:4/5
z_vals = seq(-2, +2, length=5)

G = expand.grid( q5=q5_vals, k=z_vals, b1=z_vals, b2=z_vals)

#-------------------------------------------------
# calculate eta15...eta45 for each (q5,k,beta1,beta2)
# combination in the grid. eta_a is the expected number
# of surviving 0-4 year old children per woman currently
# in age group [a,a+5) AT TFR=1
#-------------------------------------------------

this.eta = t(apply(G,1, function(x) eta(x[1], x[2], x[3], x[4])))
colnames(this.eta) = paste0('eta', seq(15,45,5))

#-------------------------------------------------
# array calculations
# there are I combinations of (q5,k,beta1,beta2), one per row of G
# there are J populations (J=n= length of C = rows of W)
# there are K TFR values for which we want marginal posterior probs
#-------------------------------------------------


TFR_vals = seq(from = LTFR+delta_TFR/2,
               to   = HTFR-delta_TFR/2,
               by   = delta_TFR)


# W_eta will be an IxJ matrix of expected numbers of children
# when TFR=1. For example, W_eta[63,12] is the number of expected
# children in population 12 at the 63rd parameter combination if TFR=1

W_eta = this.eta %*% t(matrix(W, ncol=7))

## ab is a J x 2 matrix of beta parameters, for under-5 mortality in each
## population
ab    = t( sapply( q5_est,
                   function(x) {
                     LearnBayes::beta.select(
                       list(x=min(x)/2, p=.05),
                       list(x=max(x)*2, p=.95)
                     )
                   }))

## f_ij is an IxJ matrix containing the PRIOR probs of each parameter
## combination in each population

f_ij = outer(1:nrow(G), 1:n,
             function(i,j) {dbeta(G$q5[i], ab[j,1], ab[j,2]) *
                 dnorm(G$k[i]) *
```

```
                    dnorm(G$b1[i]) *
                    dnorm(G$b2[i]) }
  )

  ## Cstar_ijk is an IxJxK array of expected numbers of children for
  ## each (i=parameter set, j=population, k=TFR level)

  Cstar_ijk = outer( W_eta, TFR_vals, '*')

  ## Cobs_ijk is an (IJK)-vector of the observed # of children,
  ## in exactly the same order as Cstar_ijk
  Cobs_ijk  = C[ arrayInd( seq(Cstar_ijk), dim(Cstar_ijk))[,2] ]

  ## L_ijk is an (IJK)-vector of the likelihoods for
  ## each (i=parameter set, j=population, k=TFR level)

  L_ijk = dpois(Cobs_ijk, Cstar_ijk)  # long vector

  ## posterior is an IxJxK array of prior * likelihood, for
  ## each (i=parameter set, j=population, k=TFR level)

  posterior = array( as.vector(f_ij) * L_ijk,
                     dim=c(nrow(G), n, length(TFR_vals)),
                     dimnames=list(NULL,rownames,TFR_vals))

  ## pp is a J x K matrix of (relative) marginal posterior probs for
  ## each (j=population, k=TFR level)

  pp = apply(posterior,2:3, sum)

  ## simulate the density function for a row (=a population)
  ## by using the cumulative (rel) probabilties

  simdens = function(pprow) {
    inv_cdf = approxfun( x= cumsum(pprow)/sum(pprow),
                         y=TFR_vals + delta_TFR/2)

    d  = density( inv_cdf( seq(.005,.995,.01)), adjust=1.5)
    return( list( dens= d,
                  Q10 = inv_cdf(.10),
                  Q50 = inv_cdf(.50),
                  Q90 = inv_cdf(.90)))
  }

  result = apply(pp, 1, simdens)

  if (n==1) return(result[[1]])
  if (n>1) return(result)
} # superfast_bTFR


# FROM
# http://worldpopulationreview.com/countries/china-population/
```

```
library(wpp2019)
data(pop)

CHNF = popF %>%
        filter(name == 'China')

CHNM = popM %>%
        filter(name == 'China')

W_CHN  = 1000 * (CHNF[4:10, '2020'])
C_CHN  = 1000* (CHNF[   1, '2020'] + CHNM[1, '2020'])
q5_CHN = .009  #approx

est = superfast_bTFR(C=C_CHN,
                     W=W_CHN,
                     q5_est=q5_CHN,
                     LTFR = 1, HTFR = 2, Hq5=.05, delta_TFR=.01)
```

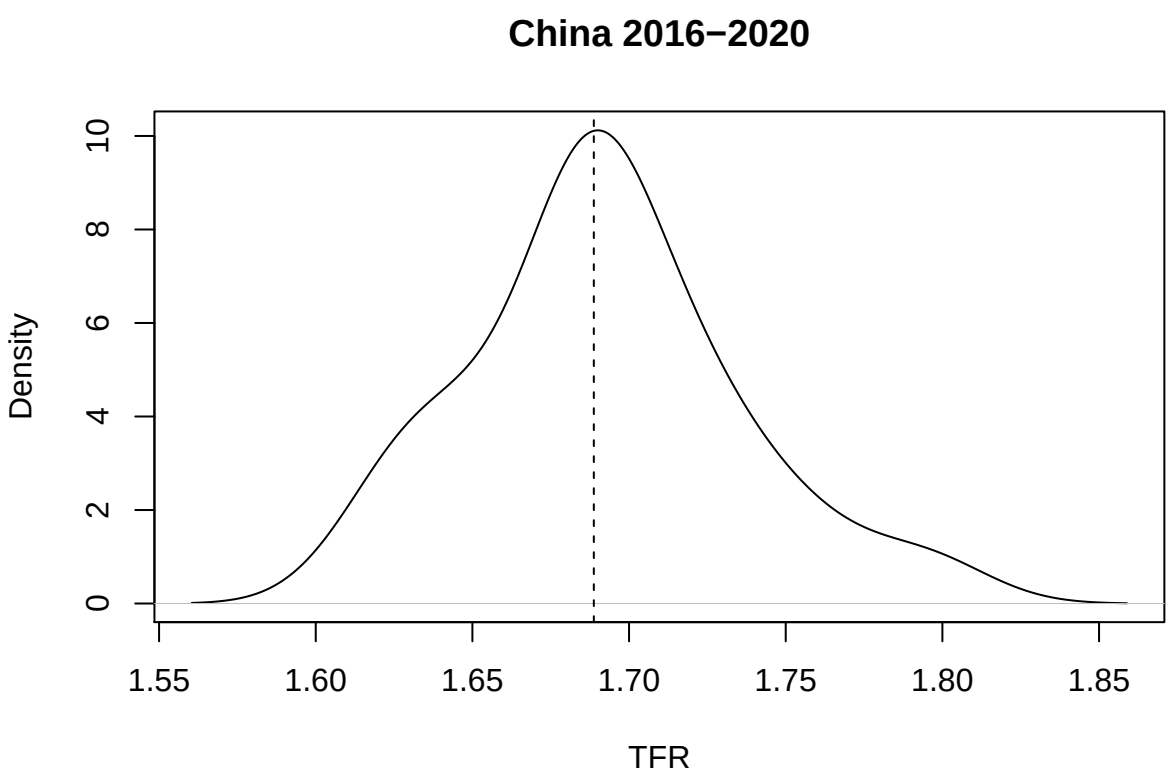
```
## Loading required package: assertthat
```

```
##
## Attaching package: 'assertthat'
```

```
## The following object is masked from 'package:tibble':
##
##     has_name
```

```
## Warning in regularize.values(x, y, ties, missing(ties)): collapsing to unique
## 'x' values
```

```
plot( est$dens, main='China 2016-2020', xlab='TFR')
abline(v=est$Q50, lty=2)
```


China 2016−2020
Density
0
2
4
6
8
10
1.55
1.60
1.65
1.70
1.75
1.80
1.85
TFR