Synthetic Germany

Jon Minton

2022-07-23

Why a 20% East Germany/80% West Germany weighting was used to produce 'Synthetic Germany'

In the main paper we mention that a Synthetic Germany was produced as a comparator series by using a weighted average of East German and West German life expectancies. The weighting used was 20% East Germany and 80% West Germany.

This appendix describes why and how that particular weighting was arrived at.

Core problem and approach taken

The core problem is that we do not have a single Germany data series going back to the 1980s, as Germany was not unified at this time. However, we do have series for East Germany and West Germany. In the HMD separate East and West Germany series were still produced even after reunfication, alongside a complete Germany series.

By having separate East, West and complete Germany series for a common range of years, we can produce a synthetic Germany series as follows:

$$e_x^{syn}(t) = pe_x^e(t) + (1-p)e_x^w(t)$$

Where the superscripts syn, e and w indicate synthetic, east and west respectively.

The parameter p is the share of East Germany in the series.

The optimisation challenge is to select a value of p such that the discrepancy between $e_x^{syn}(t)$ and the observed life expectancies for Germany $e_x^{obs}(t)$ is minimised for all values of t.

This discrepancy is calculated using Root-mean-squared-error (RMSE)

Our comparator period is 1990-2010 inclusive. This is a period in which all three series exist and so can be compared, and before any possible slowdown effects are likely to exist.

Packages

```
library(tidyverse)
```

Data

We start with lifetable data for German populations. We only need e0 and e65, though the code can be adapted for other ages.

```r
# If getting from HMD directly we can use a script
# get_data_from_hmd.R
# as follows
# pacman::p_load(HMDHFDplus)
# source(here("R", "get_data_from_hmd.R"))

# We need life expectancy at age 0 and 65, which we can get from the lifetables

# We will load these lifetables directly (rather than using the script)

# To allow this script to be self-contained we will refer to the github file location directly

hmd_lt <- read_rds("https://github.com/JonMinton/change-in-ex/blob/main/data/lifetables.rds?raw=true")


# We will now just look for German data, and for ages 0 and 65 only

e0_e65 <-
  hmd_lt  %>%
    filter(Age %in% c(0, 65)) %>%
    filter(str_detect(code, "DEU")) %>%
    select(code, sex, year = Year, x = Age, ex) %>%
    arrange(year)
```

Functions

Commented functions written to produce Synthetic Germany are shown and run below

```r
# Functions for creating a synthetic Germany and selecting the optimal share of East and West Germany t

# Background: The Human Mortality Database (HMD) includes life expectancy data for Germany (DEUT), East
# (DEUTE) and West Germany (DEUTW). Of course there are no population data for Germany prior to reunifi
# but there are separate population data maintained for East and West German populations post-reunifica

# This means DEUTE, DEUTW, and DEUT are reported for a number of common years.

# The aim of this code is to produce a 'Synthetic Germany' for years prior to reunification. This 'Synt
# is based on a weighted average of East and West German population data. The weighting between these t
# is a parameter to determine based on an objective or loss function.
# A objective/loss function is a function that, given one or more numeric inputs (which can be varied),
# a single numeric output. The aim is to select the input that minimises the value returned by the func

# In this case, the input is the share of a value from the east german series (and so by implication th
# of West Germany as well), and the output to try to minimise is the root-mean-square-error (RMSE) betw
# observed life expectancy value from Germany as a whole, and the implied/simulated life expectancy val
# a synthetic Germany with the proposed East/West Germany share.


# The following function makes a synthetic germany series with a particular east germany share (p_east)
# The inputs east_germany and west_germany need to be vectors of the the same length.
# Additionally the proposed value p_east needs to be between 0 and 1.

make_synthetic_population <- function(east_series, west_series, p_east){
```

```
  stopifnot("Series are of different lengths" = length(east_series) == length(west_series))
  stopifnot("Proportion out of possible bounds" = between(p_east, 0, 1))

  east_series * p_east + west_series * (1 - p_east)
}

# The following function compares a proposed synthetic germany series against the referenced/observed v
# The series need to be of the same length to allow them to be pairwise compared.

# The arguments to what can be one of "RMSE', 'abs', and 'rel'
#  - The match.arg function checks that the input to this argument is one of the three valid inputs.
#  - the default argument for what is RMSE

compare_synthetic_to_reference <- function(synthetic, reference, what = c("RMSE", "abs", "rel")){
  stopifnot("Synthetic and reference are different lengths" = length(synthetic) == length(reference))

  what <- match.arg(what)

  if (what == "RMSE"){
    out <- (synthetic - reference)^2 %>%
      mean() %>%
      .^(1/2)
    return(out)
  } else if (what == "abs"){
    return(synthetic - reference)
  } else if (what == "rel"){
    out <- (synthetic - reference)/reference
    return(out)
  } else {
    stop("Wrong what argument (which should have been caught earlier")
  }
  NULL
}


# The following function wraps up the two previous functions, including a default proposed p_east share

# It includes a number of checks that the inputs are of the expected format, and a function within whic
# the columns and rows of interest to populate each series.

compare_series_get_rmse <- function(data, p_east = 0.20,
                                    east_label = "DEUTE", west_label = "DEUTW", ref_label = "DEUTNP",
                                    comp_period = c(1990, 2010))
{

  stopifnot("data is not a dataframe"      = "data.frame" %in% class(e0_e65)         )
  stopifnot("proportion not valid"         = between(p_east, 0, 1)                   )
  stopifnot("comp_period not a range"      = length(comp_period) == 2                )
  stopifnot("comp_period values not valid" = comp_period %>% is.numeric() %>% all()  )

  # Get the series
  extract_series <- function(data = data, code_label, comp_period){
    data %>%
```

```
      filter(code == code_label) %>%
      filter(between(year, comp_period[1], comp_period[2])) %>%
      arrange(year) %>%
      pull(ex)
  }


  message("getting East series")
  east_series <- extract_series(data, east_label, comp_period)
  message("getting West series")
  west_series <- extract_series(data, west_label, comp_period)
  message("getting reference series")
  ref_series  <- extract_series(data, ref_label,  comp_period)

  stopifnot("East/West series lengths differ" = length(east_series) == length(west_series))
  stopifnot("Ref series length differs from EastWest" = length(ref_series) == length(east_series))

  message("creating synthetic population and calculating RMSE")

  rmse <- make_synthetic_population(east_series, west_series, p_east = p_east) %>%
    compare_synthetic_to_reference(reference = ref_series)

  stopifnot("rmse failed: not length 1" = length(rmse) == 1)
  stopifnot("rmse failed: not right class" = class(rmse) == "numeric")

  rmse
}

# The following function (with a function inside it!) runs the optim function and packs the inputs
# in the way that optim expects.


run_optim <- function(data){
  pack_for_optim <- function(par, data){
    p_east <- par["p_east"]
    data %>% compare_series_get_rmse(p_east = p_east)
  }
  optim(
    par = c(p_east = 0.5),
    fn = pack_for_optim, data = data,
    lower = 0, upper = 1,
    method = "L-BFGS-B"
  )
}
```

Extracting series

The following manually extracts the series for females for x = 0 (for e0)

```
# Comparator period 1990-2010

# this pulls out the series of values for East Germany
east_series <-
```

```
  e0_e65 %>%
    filter(x == 0) %>%
    filter(code == "DEUTE") %>%
    filter(between(year, 1990, 2010)) %>%
    filter(sex == "female") %>%
    arrange(year) %>%
    pull(ex)

# This pulls out out for West Germany
west_series <-
  e0_e65 %>%
  filter(x == 0) %>%
  filter(code == "DEUTW") %>%
  filter(between(year, 1990, 2010)) %>%
  filter(sex == "female") %>%
  arrange(year) %>%
  pull(ex)

# And this pulls it out for Germany as a whole
ref_series <-
  e0_e65 %>%
  filter(x == 0) %>%
  filter(code == "DEUTNP") %>%
  filter(between(year, 1990, 2010)) %>%
  filter(sex == "female") %>%
  arrange(year) %>%
  pull(ex)
```

Gridsearch approach

The parameter whose loss function (RMSE) we want to minimise is called `p_share`. This is the proportion of the East Germany series (and so by implication `1 - p_share` is the proportin of the West Germany Series.

We start with a gridsearch approach, looking at each full percentage of `p_share` between 0 and 1

```
# And this function looks at each percentage value for share of East Germany
# between 0% and 100%, and pulls out the RMSE given this.

grid_pshare <-
  tibble(
    p_share = seq(0, 1, by = 0.01)
  ) %>%
    mutate(
      rmse = map_dbl(
        p_share,
        ~ make_synthetic_population(east_series, west_series, p_east = .x) %>%
          compare_synthetic_to_reference(., reference = ref_series))
    )
# The contents of the mutate function are complicated. Probably too complicated. ;)
# firstly they evoke map_dbl. This is a particular type of map function that forces the output
# to be a numeric (double) value. This is what we want as RMSE should be a number, and won't be
# a whole number (integer)

# The first argument map_dbl takes is p_share, this is the contents of the p_share column
```

```
# created alongside the tibble in the previous step.

# The next step is a function, make up of two steps joined together with the pipe operator %>%
# The use of the ~ at the start is tidyverse shorthand for 'function'.
# The argument which will be varying is p_share. Unfortunately this isn't the first argument to the
# first function - make_synthetic_population - which it's passed to.
# By default, the map function, like the pipe operator, will pass to the first argument slot in the
# function being passed to. To override this, the .x operator is used.
# This explicitly tells the map function where to put the argument of interest (p_share) in this case.
# (.x is used because there are map2 functions, taking two inputs. The first of these is referred to as
# .x, and the second as .y. This convention is also followed here for a single argument.)
# The result of the first step (before the pipe) of this operation in map_dbl is then passed to
# another function, compare_synthetic_to_reference, and fits in its first slot (marked with a . in this

# This isn't the most clear way of creating the operation. Instead, declaring a single function
# beforehand would likely be easier to understand. However it's a fairly concise shorthand that works!
```

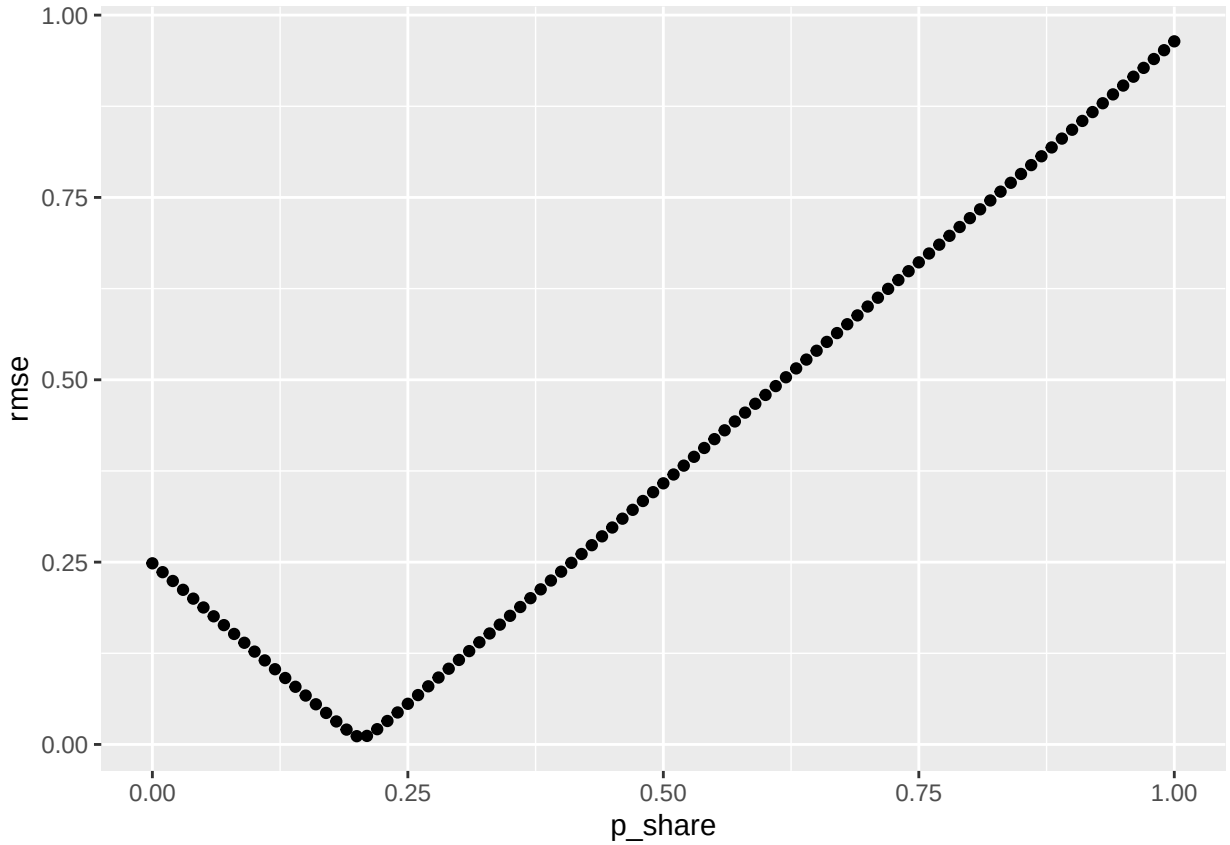
We can see the RMSE for each proposed p_east as follows:

```
grid_pshare
```

```
## # A tibble: 101 x 2
##    p_share  rmse
##      <dbl> <dbl>
##  1    0    0.248
##  2    0.01 0.236
##  3    0.02 0.224
##  4    0.03 0.212
##  5    0.04 0.200
##  6    0.05 0.188
##  7    0.06 0.176
##  8    0.07 0.164
##  9    0.08 0.152
## 10    0.09 0.139
## # ... with 91 more rows
```

And we can visualise how the RMSE varies with different `p_share` values as follows:

```
grid_pshare %>%
  ggplot(aes(p_share, rmse)) +
  geom_point()
```



So, there's a clear minimal RMSE when around 20% (p = 0.20) is proposed.

Is it exactly 20%? Let's check

```
grid_pshare %>%
  filter(rmse == min(rmse))
```

```
## # A tibble: 1 x 2
##   p_share   rmse
##     <dbl>  <dbl>
## 1     0.2 0.0114
```

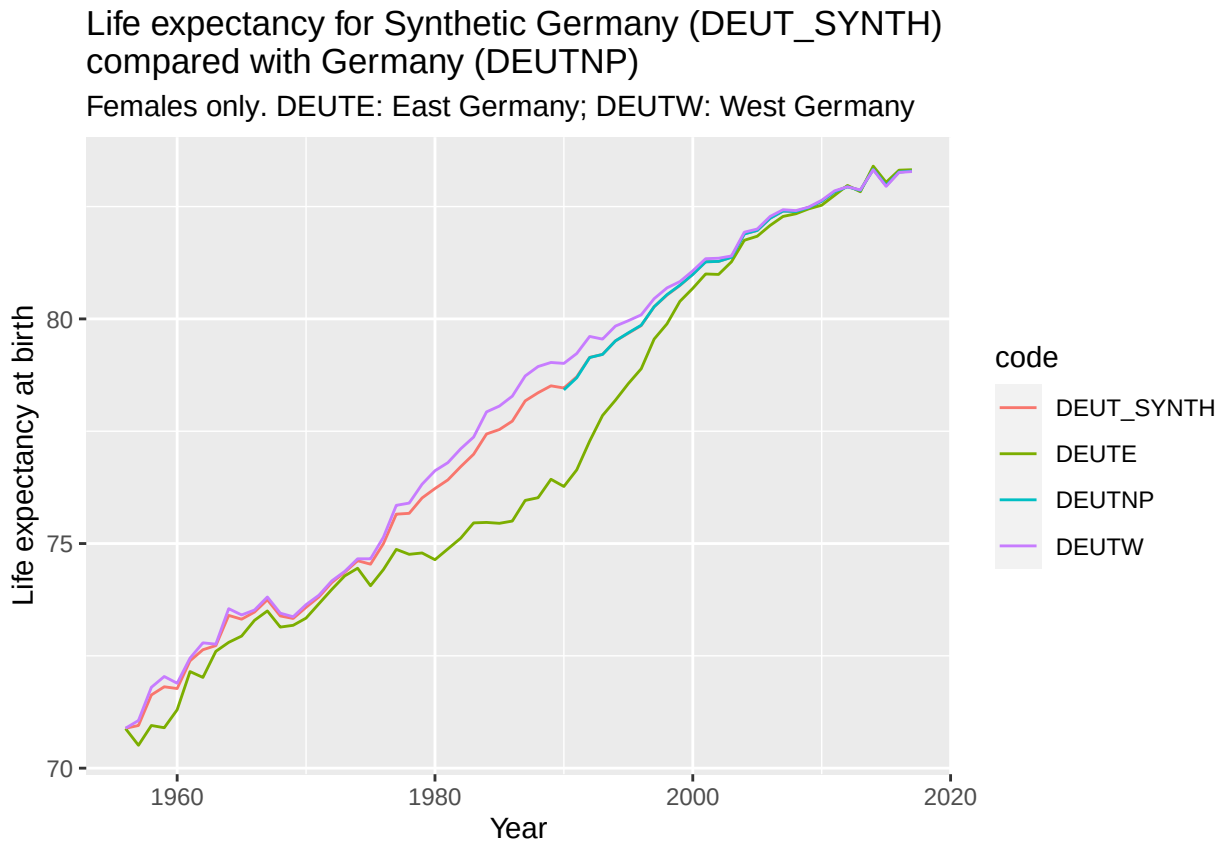
Yes, it's at 20% exactly (to the nearest %)

And what does the Synthetic Germany look like when this proposed mix is used?

```
e0_e65 %>%
  filter(code %in% c("DEUTE", "DEUTW", "DEUTNP")) %>%
  filter(sex == "female") %>%
  filter(x == 0) %>%
  pivot_wider(names_from = code, values_from = ex, values_fill = NA)  %>%
  arrange(year) %>%
  mutate(DEUT_SYNTH = 0.2 * DEUTE + (1 - 0.2) * DEUTW) %>%
  pivot_longer(cols = c("DEUTNP", "DEUTE", "DEUTW", "DEUT_SYNTH"), names_to = "code", values_to = "e0") %>%
  ggplot(aes(x = year, y = e0, group = code, colour = code)) +
  geom_line() +
  labs(
    x = "Year",
    y = "Life expectancy at birth",
```

```
    title = "Life expectancy for Synthetic Germany (DEUT_SYNTH)\ncompared with Germany (DEUTNP)",
    subtitle = "Females only. DEUTE: East Germany; DEUTW: West Germany"
  )
```

```
## Warning: Removed 34 row(s) containing missing values (geom_path).
```



We can see here that over the reference period `DEUT_SYNTH` (red line) matches `DEUTNP` (blue line) quite closely, as we would expect and hope for.

However the above only completes the calculation for one of the four populations of interest: We also need to do this for e65, and for males.

The following code starts to automate the process for these other three populations:

```
# This is a slightly laborious process, however, involving quite a bit of manual copying and
# pasting of bits of code. For example the above is only for females, and we might want to know
# the equivalent value for males. We might also want to calculate life expectancy for different ages.

# To start to automate further we can look at where the code above is repetitive, work out the
# common patterns in the repeated sections of code, and build functions around them.
# We can also look for where we've hard-coded values, and replace them with objects/references.

# Within the functions developed, a function called get_series() within the compare_series_get_rmse
# standardises the extraction of series. It looks as follows:

# extract_series <- function(data = data, code_label, comp_period){
#   data %>%
#     filter(code == code_label) %>%
```

```
#     filter(between(year, comp_period[1], comp_period[2])) %>%
#     arrange(year) %>%
#     pull(ex)
# }

# We can use this extract_series function within the broader function to get the RMSE for a given p_eas
# in fewer steps (for us)

e0_e65 %>%
  filter(x == 0) %>%
  filter(sex == "female") %>%
  compare_series_get_rmse(data =., p_east = 0.2)
```

```
## [1] 0.01138085
```

```
# This makes it easier to look for another manual combination of starting age (x) and sex

e0_e65 %>%
  filter(x == 65) %>%
  filter(sex == "male") %>%
  compare_series_get_rmse(data =., p_east = 0.2)
```

```
## [1] 0.01245946
```

```
# We can also use map_dbl and nesting to pass different datasets to this function

e0_e65 %>%
  group_by(sex, x) %>%
  nest() %>%
  mutate(rmse = map_dbl(data, compare_series_get_rmse, p_east = 0.2))
```

```
## # A tibble: 4 x 4
## # Groups:   sex, x [4]
##   sex        x data                 rmse
##   <chr>  <int> <list>              <dbl>
## 1 female     0 <tibble [152 x 3]> 0.0114
## 2 female    65 <tibble [152 x 3]> 0.00656
## 3 male       0 <tibble [152 x 3]> 0.0121
## 4 male      65 <tibble [152 x 3]> 0.0125
```

```
# We can repeat the grid search approach by adding a third column, p_share, containing
# a sequence of possible values of the p_east share. We can get all permutations of
# data and p_share using expand_grid (or expand.grid) as follows:

expand_grid(
  e0_e65 %>%
    group_by(sex, x) %>%
    nest(),
  p_share = seq(0, 1, by = 0.01)
) %>%
  select(sex, x, p_share, data)
```

```
## # A tibble: 404 x 4
##    sex        x p_share data
##    <chr>  <int>   <dbl> <list>
##  1 female     0    0    <tibble [152 x 3]>
```

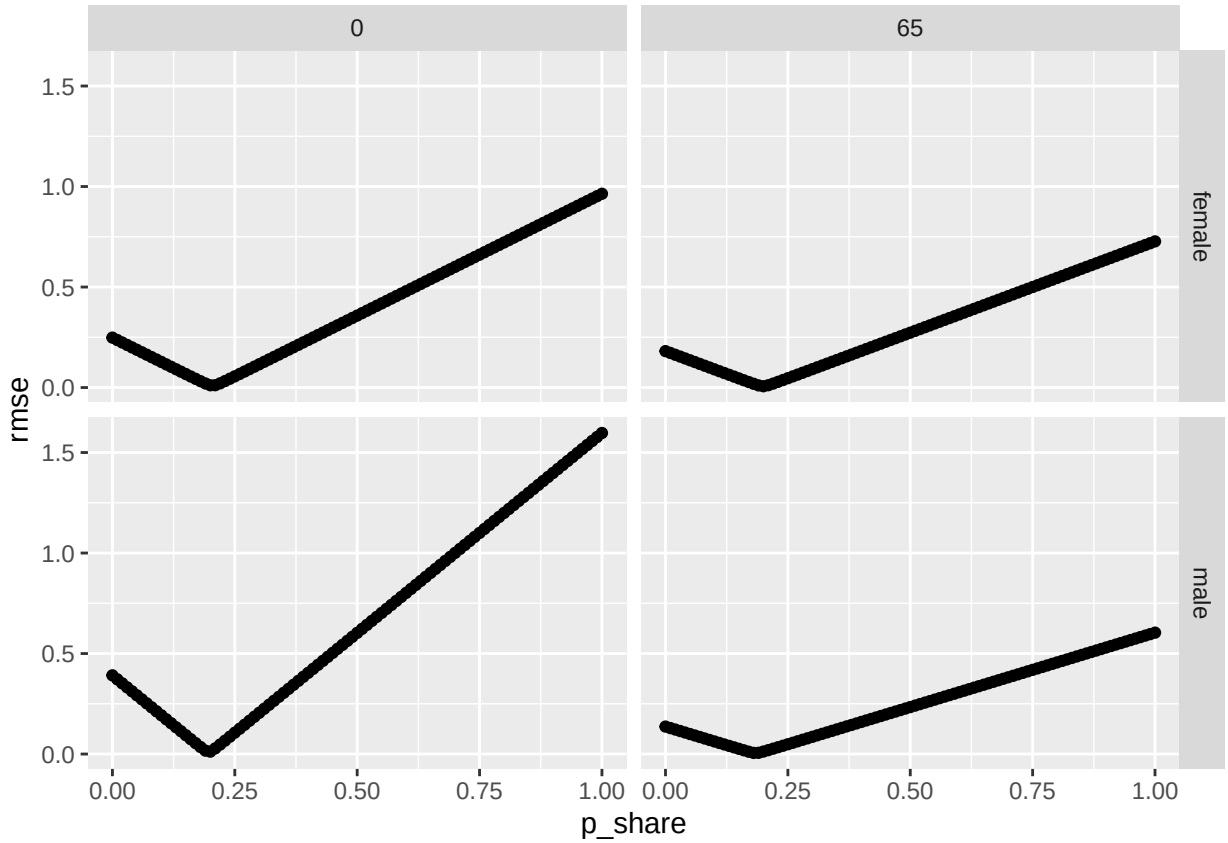
```
##  2 female     0    0.01 <tibble [152 x 3]>
##  3 female     0    0.02 <tibble [152 x 3]>
##  4 female     0    0.03 <tibble [152 x 3]>
##  5 female     0    0.04 <tibble [152 x 3]>
##  6 female     0    0.05 <tibble [152 x 3]>
##  7 female     0    0.06 <tibble [152 x 3]>
##  8 female     0    0.07 <tibble [152 x 3]>
##  9 female     0    0.08 <tibble [152 x 3]>
## 10 female     0    0.09 <tibble [152 x 3]>
## # ... with 394 more rows
```

```
# Then we can pass to the function we used above to get the RMSE, as well as save to a new object

grid_search_pshares <-
  expand_grid(
    e0_e65 %>%
      group_by(sex, x) %>%
      nest(),
    p_share = seq(0, 1, by = 0.01)
  ) %>%
  select(sex, x, p_share, data) %>%
  mutate(rmse = map2_dbl(data, p_share, ~compare_series_get_rmse(data = .x, p_east = .y)))

grid_search_pshares
```

```
## # A tibble: 404 x 5
##    sex        x p_share data               rmse
##    <chr>  <int>   <dbl> <list>            <dbl>
##  1 female     0    0    <tibble [152 x 3]> 0.248
##  2 female     0    0.01 <tibble [152 x 3]> 0.236
##  3 female     0    0.02 <tibble [152 x 3]> 0.224
##  4 female     0    0.03 <tibble [152 x 3]> 0.212
##  5 female     0    0.04 <tibble [152 x 3]> 0.200
##  6 female     0    0.05 <tibble [152 x 3]> 0.188
##  7 female     0    0.06 <tibble [152 x 3]> 0.176
##  8 female     0    0.07 <tibble [152 x 3]> 0.164
##  9 female     0    0.08 <tibble [152 x 3]> 0.152
## 10 female     0    0.09 <tibble [152 x 3]> 0.139
## # ... with 394 more rows
```

This has now produced a lot of operations! What does it show?

```
grid_search_pshares %>%
  ggplot(aes(x=p_share, y = rmse)) +
  geom_line() + geom_point() +
  facet_grid(sex ~ factor(x))
```



So, it looks like approximately the same answer for each of the four groups. Let's check

```
grid_search_pshares %>%
  group_by(x, sex) %>%
  filter(rmse == min(rmse))
```

```
## # A tibble: 4 x 5
## # Groups:   x, sex [4]
##   sex        x p_share data                 rmse
##   <chr>  <int>   <dbl> <list>               <dbl>
## 1 female     0    0.2  <tibble [152 x 3]> 0.0114
## 2 female    65    0.2  <tibble [152 x 3]> 0.00656
## 3 male       0    0.2  <tibble [152 x 3]> 0.0121
## 4 male      65    0.18 <tibble [152 x 3]> 0.00575
```

So, to the nearest whole %, the proposed share is 20% except for males at age 65, where it's 18%.

The following code chunk (not run) discusses how the answer could be made more precise by reducing the mesh size on the grid search.

```
# Using the grid search approach above, we could get more precise answers to the best share by
# making the grid search 'mesh' smaller. For example, we could include the resolution 10% by changing
# the by= argument in seq from 0.01 to 0.001.
# i.e.
# grid_search_pshares <-
#   expand_grid(
#     e0_e65 %>%
#       group_by(sex, x) %>%
```

```
#       nest(),
#     p_share = seq(0, 1, by = 0.001) # change from by=0.01 here
#   ) %>%
#   select(sex, x, p_share, data) %>%
#   mutate(rmse = map2_dbl(data, p_share, ~compare_series_get_rmse(data = .x, p_east = .y)))

# But this would take ten times as long to run, as there are ten times as many values to consider.

# We could also narrow the search space, to say between 0.15 and 0.25, and then narrow the mesh size to

# grid_search_pshares <-
#   expand_grid(
#     e0_e65 %>%
#       group_by(sex, x) %>%
#       nest(),
#     p_share = seq(0.15, 0.25, by = 0.001) # change from by=0.01, and start and end to 0.15 and 0.25 r
#   ) %>%
#   select(sex, x, p_share, data) %>%
#   mutate(rmse = map2_dbl(data, p_share, ~compare_series_get_rmse(data = .x, p_east = .y)))

# But again, we only know to do this because we first did the coarser search over the full range of val

# And now imagine if we wanted to do this for each individual age, or for many other countries
# with similar East/West (or North/South) population breakdowns. Or for many different reference period
```

The following code shows how the above answers can be arrived at using numeric optimisation using the `optim` function.

```
# The grid search approach can quickly become too computationally intensive. That's where numerical
# optimisation algorithms can be used instead.

# Within R, the standard function to use for numerical optimisation is optim.
# Optim passes a value or values to a function, and tries to get the smallest or largest value
# out of this function, rejigging the value/values passed to the function multiple times until it can't
# get a substantially improved (as in minimised/maximised) value from the function.
#

# To make the optim function work well with the functional programming approach used by the map functio
# I made a small wrapper function for optim:


# run_optim <- function(data){
#   pack_for_optim <- function(par, data){
#     p_east <- par["p_east"]
#     data %>% compare_series_get_rmse(p_east = p_east)
#   }
#   optim(
#     par = c(p_east = 0.5),
#     fn = pack_for_optim, data = data,
#     lower = 0, upper = 1,
#     method = "L-BFGS-B"
#   )
# }
```

```
# This function also specifies that a particular method, 'L-BFGS-B', which
# gives performs bounded optimisation be used, along with the lower and upper bounds

# I've also given the algorithm a starting value of p = 0.5, which we might assume if we had
# not looked at the data and performed the analysis previously.

# Having done this packing, we now can use optim for each combination of age and sex to get the
# best estimate for p_east in far fewer computational steps, and with much higher resolution

synth_germany_best_shares_via_optim <-
  e0_e65 %>%
  group_by(sex, x) %>%
  nest() %>%
  mutate(
    optim_outputs = map(data, run_optim)
  ) %>%
  mutate(
    optim_share = map_dbl(optim_outputs, pluck, "par")
  )

synth_germany_best_shares_via_optim
```

```
## # A tibble: 4 x 5
## # Groups:   sex, x [4]
##   sex        x data              optim_outputs    optim_share
##   <chr>  <int> <list>            <list>                 <dbl>
## 1 female     0 <tibble [152 x 3]> <named list [5]>       0.205
## 2 female    65 <tibble [152 x 3]> <named list [5]>       0.199
## 3 male       0 <tibble [152 x 3]> <named list [5]>       0.197
## 4 male      65 <tibble [152 x 3]> <named list [5]>       0.184
```

The optim function contains a lot of additional information. For example, we could get the best RMSE by interrogating the 'value' element as follows:

```
synth_germany_best_shares_via_optim %>%
  mutate(
    rmse_at_optim_pshare = map_dbl(optim_outputs, pluck, "value")
  )
```

```
## # A tibble: 4 x 6
## # Groups:   sex, x [4]
##   sex        x data              optim_outputs    optim_share rmse_at_optim_ps~
##   <chr>  <int> <list>            <list>                 <dbl>             <dbl>
## 1 female     0 <tibble [152 x 3]> <named list [5]>       0.205           0.00983
## 2 female    65 <tibble [152 x 3]> <named list [5]>       0.199           0.00654
## 3 male       0 <tibble [152 x 3]> <named list [5]>       0.197           0.0103
## 4 male      65 <tibble [152 x 3]> <named list [5]>       0.184           0.00472
```

```
# So, the best value for RMSE was higher (worse) for males at birth, and lowest (better) for
# males at age 65.
```

Conclusion

For all populations, a proposed share of around 20% East Germany/80% West Germany appears reasonable, even though slightly different shares are slightly more optimal for some populations.

Rather than use the exact `p_share`s calculated by `optim` as optimal for each of the populations, we decided to use 20% for all populations for ease of explanation within the main text. The precise optim values given above could be used instead, if the 20% assumption were felt to be too coarse.