Constructing a Demographic Account, with Lexis Triangles, for Greenland

John Bryant

17/11/2019

Introduction

In this note we show how to arrange data on population, births, deaths, and migration for Greenland into a demographic account. The raw data does not specify Lexis triangles, so the function for constructing the account randomly allocates events to triangles. We show how, thanks to the Lexis triangles, we are able to switch between different views of the data.

To run the code, we need the *R* packages listed below. The package **dembase** can be obtained from GitHub. All the rest can be obtained from CRAN. We are hoping to replace **dembase** with a collection of new packages, with an improved interface, over the course of 2020. The repositories for the new packages under development are here. Once the packages are ready, we will release them on to CRAN.

```
library(dembase)
library(readr)
library(tidyr)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:dembase':
##
##     collapse

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)
```

Preparing Raw Data

We start by reading the raw data, which we downloaded from the Statbank Greenland website on 17 November 2019.

We will use the following age categories for all datasets other than births:

```
levels_age <- c(0:99, "100+")
```

Population

We read in the data on population data, and reformat it. Package **dembase** assumes that year labels are constructed using the "year to" convention, where, for instance, the exact time "2010" refers to 31 December 2010. The Greenland data follow a "year from" convention. We can convert between the two by subtracting 1 from the time variable. (This is horribly confusing: the packages replacing **dembase** will handle year labels better.)

```
popn_df <- read_csv("BEXST1.csv",
                    skip = 2,
                    n_max = 200) %>%
    select(age,
           sex = gender,
           `1977`:`2019`) %>%
    gather(key = time,
           value = count,
           `1977`:`2019`) %>%
    mutate(age = factor(age, levels = levels_age)) %>%
    mutate(sex = factor(sex,
                        levels = c("Women", "Men"),
                        labels = c("Female", "Male"))) %>%
    mutate(time = as.integer(time),
           time = time - 1)
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   residence = col_character(),
##   `place of birth` = col_character(),
##   gender = col_character()
## )

## See spec(...) for full column specifications.
```

We format the data for births, deaths, and international migration in the same way, though we don't have to make any adjustments to the time variable in these cases, since they refer to periods rather than exact times.

```
births_df <- read_csv("BEXBBL3.csv",
                      skip = 2,
                      n_max = 72) %>%
    select(age,
           sex = gender,
           `1973`:`2018`) %>%
    gather(key = time,
           value = count,
           `1973`:`2018`) %>%
    mutate(sex = factor(sex,
                        levels = c("Girls", "Boys"),
                        labels = c("Female", "Male")))
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `district (mother)` = col_character(),
##   gender = col_character()
## )

## See spec(...) for full column specifications.
```

```
deaths_df <- read_csv("BEXBBDM1.csv",
                   skip = 2,
                   n_max = 200) %>%
    select(age,
           sex = gender,
           `1977`:`2018`) %>%
    gather(key = time,
           value = count,
           `1977`:`2018`) %>%
    mutate(age = factor(age, levels = levels_age)) %>%
    mutate(sex = factor(sex,
                        levels = c("Women", "Men"),
                        labels = c("Female", "Male")))
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   `place of birth` = col_character(),
##   gender = col_character(),
##   type = col_character()
## )

## See spec(...) for full column specifications.
```

```
migration_df <- read_csv("BEXBBIU2.csv",
                      skip = 2,
                      n_max = 400) %>%
    select(age,
           sex = gender,
           migration,
           `1993`:`2018`) %>%
    gather(key = time,
           value = count,
           `1993`:`2018`) %>%
    mutate(age = factor(age, levels = levels_age)) %>%
    mutate(sex = factor(sex,
                        levels = c("Women", "Men"),
                        labels = c("Female", "Male")))
```

```
## Parsed with column specification:
## cols(
##   .default = col_double(),
##   municipality = col_character(),
```

```
##   gender = col_character(),
##   migration = col_character()
## )

## See spec(...) for full column specifications.
```

Constructing Demographic Arrays

In package **dembase**, demographic accounts are assembled out of demographic arrays. A demographic array is a standard multiway array, with some extra metadata. Function `Counts`, which is used to construct the demographic arrays, can infer most of the metadata from the dimnames of the input data, but needs some help in deciding whether the times refer to points or intervals. We restrict the demographic arrays and account to the period 1992-2018, since this is the period for which international migration data is available.

```r
popn <- popn_df %>%
    filter(time %in% 1992:2018) %>%
    dtabs(count ~ age + sex + time) %>%
    Counts(dimscales = c(time = "Points"))
```

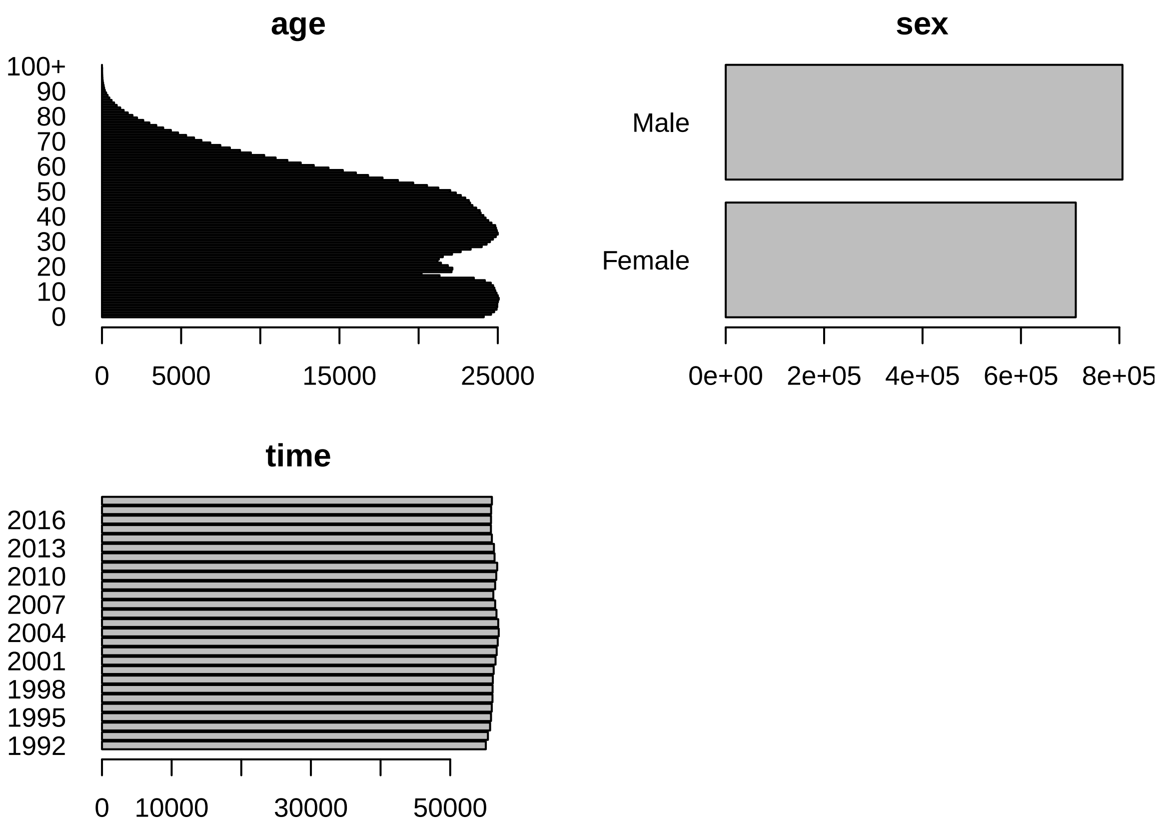
Calling `summary` on demographic array `popn` shows the range of numeric values, plus the metadata.

```r
summary(popn)
```

```
##
## name:     age       sex    time
## length:   101       2      27
## dimtype:  age       sex    time
## dimscale: Intervals Sexes  Points
## first:    0         Female 1992
## last:     100+      Male   2018
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    0.00   66.25  325.00  278.29  433.00  756.00
```

Calling `plot` on `popn` shows bar graphs of the marginal totals.

```r
plot(popn)
```

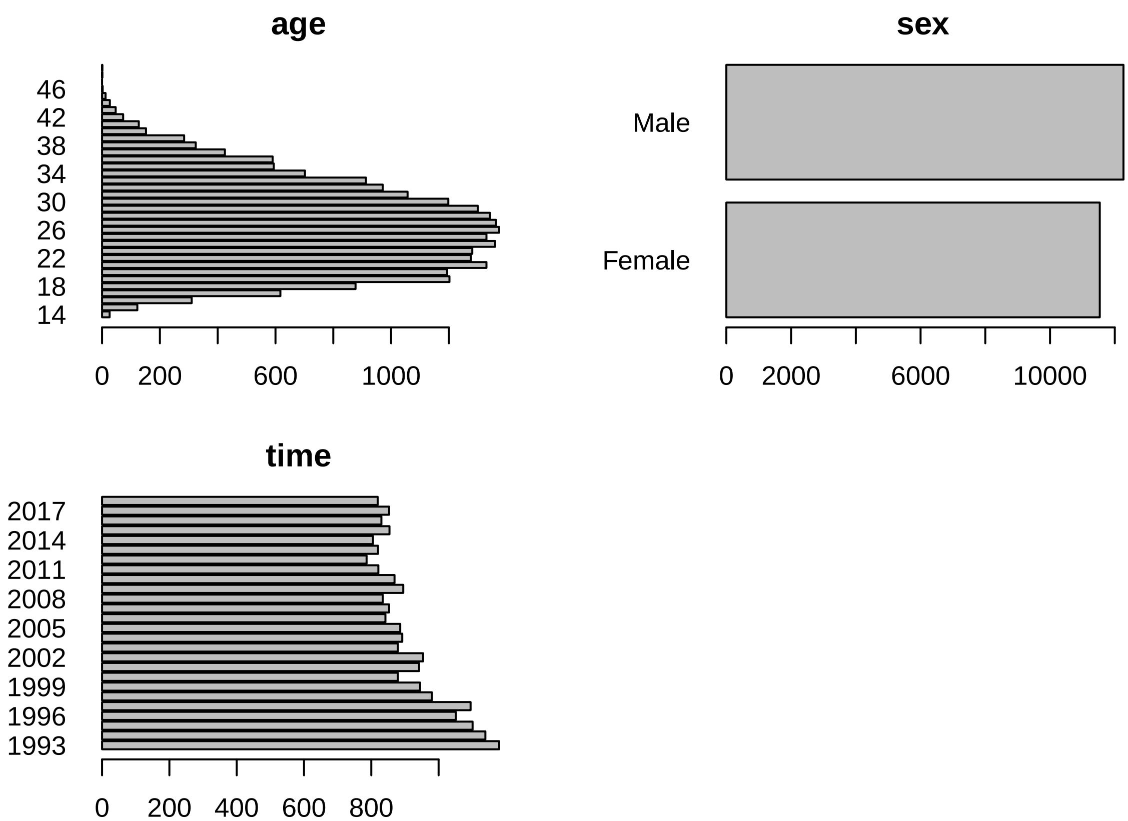


The remaining demographic arrays are constructed in a similar way to `popn`.

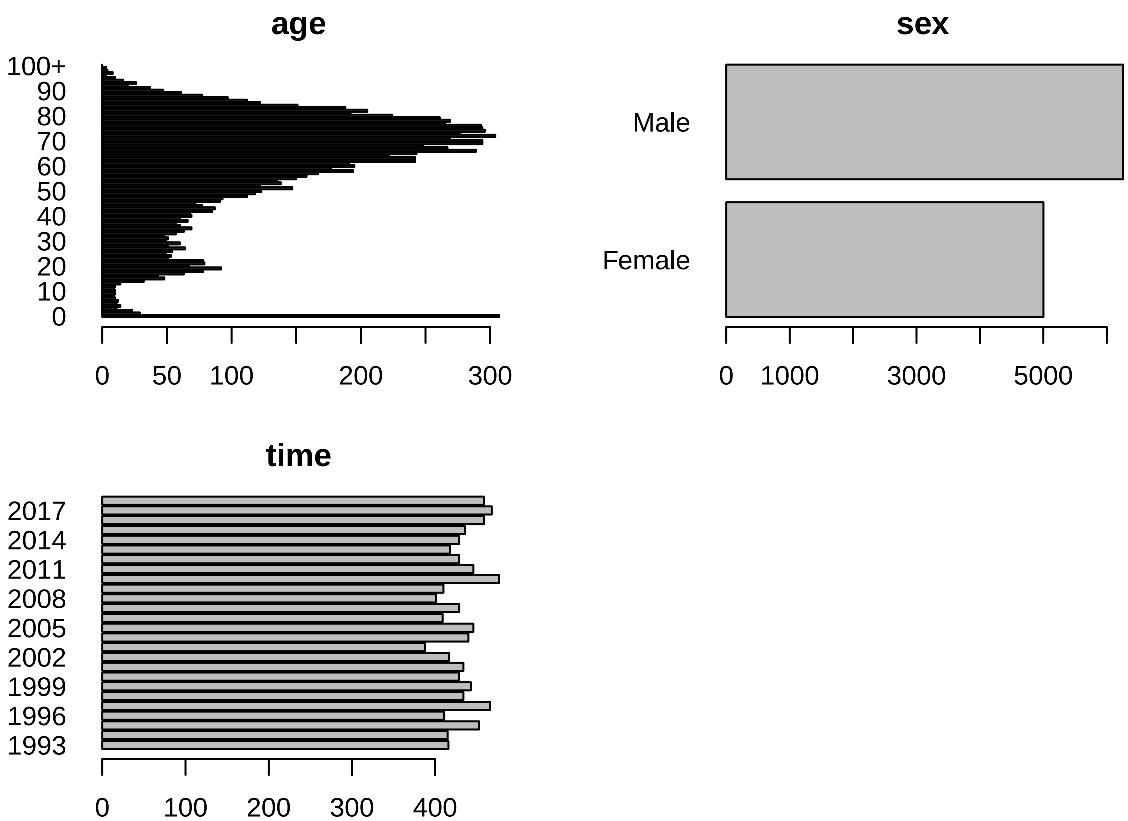
```
births <- births_df %>%
    filter(time %in% 1993:2018) %>%
    dtabs(count ~ age + sex + time) %>%
    Counts(dimscales = c(time = "Intervals"))
```

```
## assuming dimension "age" with dimtype "age" has dimscale "Intervals"
```

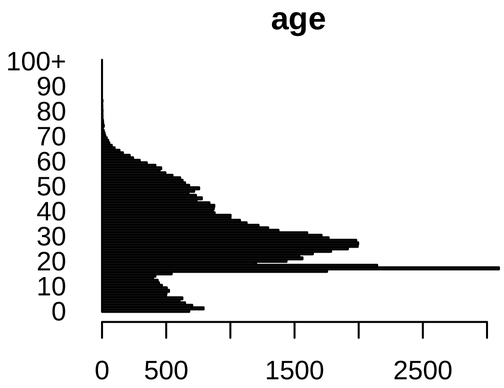
```
plot(births)
```

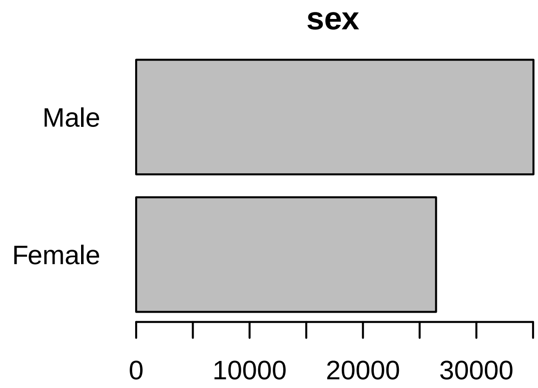


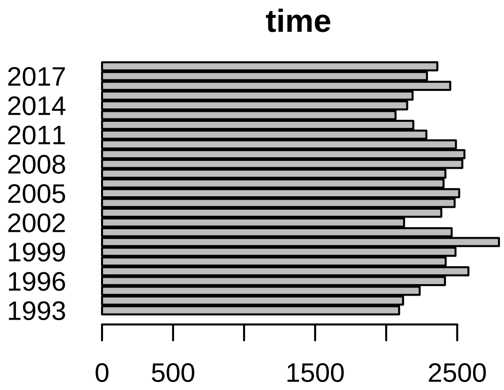

```r
deaths <- deaths_df %>%
    filter(time %in% 1993:2018) %>%
    dtabs(count ~ age + sex + time) %>%
    Counts(dimscales = c(time = "Intervals"))
plot(deaths)
```



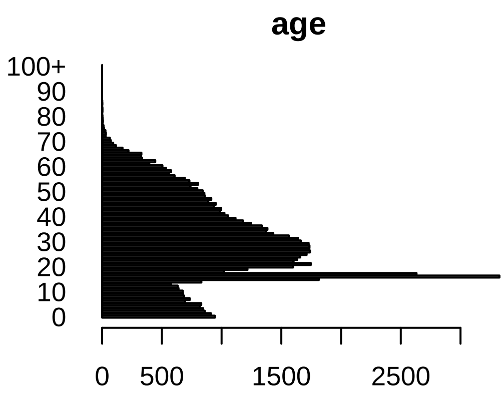

```r
immigration <- migration_df %>%
    filter(migration == "Immigrations") %>%
    dtabs(count ~ age + sex + time) %>%
    Counts(dimscales = c(time = "Intervals"))
plot(immigration)
```

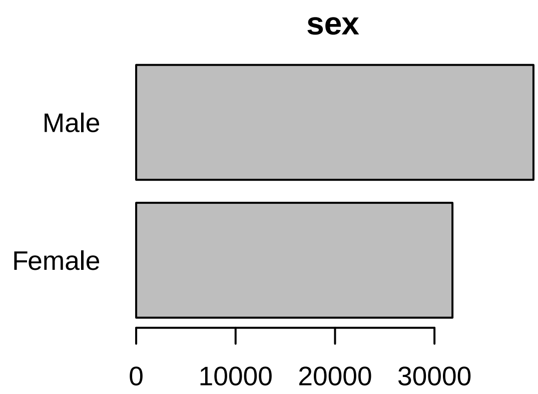


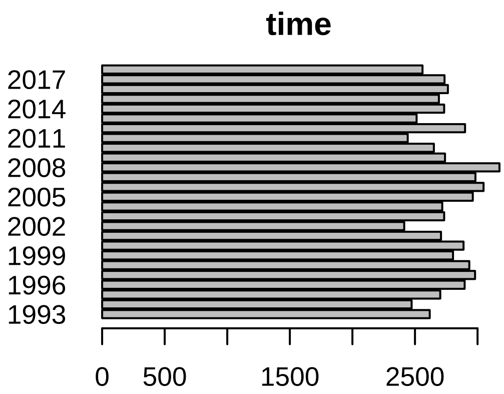





```r
emigration <- migration_df %>%
    filter(migration == "Emigrations") %>%
    dtabs(count ~ age + sex + time) %>%
    Counts(dimscales = c(time = "Intervals"))
plot(emigration)
```







Constructing a Demographic Account

Now that we have all the demographic arrays, we can construct a demographic account. There are two types of demographic accounts: "movements" accounts and "transitions" accounts. Our data are events, rather than transitions, so we construct a movements account.

```
account <- Movements(population = popn,
                     births = births,
                     entries = list(immigration = immigration),
                     exits = list(deaths = deaths,
                                  emigration = emigration))
```

Calling `summary` on the account shows annual population numbers and associated increments and decrements to the population. As can be seen at the bottom of the summary, the account is not internally consistent. The account contains cells that do not conform to the basic accounting identity that population at the end of each period equals population at the beginning plus entries minus exits. We could try to adjust the numbers to make the account consistent, using, for instance, function `estimateAccount` in package **demest**. We don't attempt to do that here, however, as our main focus is on Lexis triangles.

```
summary(account)
```

```
## An object of class Movements
##
## name:     age       sex    time
## length:   101       2      26
```

```
## dimtype:  age       sex    time
## dimscale: Intervals Sexes  Intervals
## first:    0         Female 1993
## last:     100+      Male   2018
##
##             1992  1993  1994  1995  1996  1997  1998  1999  2000  2001
## population 55113 55415 55728 55859 55967 56072 56084 56121 56242 56512
##             2002  2003  2004  2005  2006  2007  2008  2009  2010  2011
## population 56675 56825 56969 56899 56645 56458 56193 56452 56615 56749
##             2012  2013  2014  2015  2016  2017  2018
## population 56370 56282 55983 55847 55860 55877 55992
##
##                 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003
##      births (+) 1180 1139 1101 1051 1095  980  945  879  942  954  879
## immigration (+) 2091 2119 2236 2414 2579 2419 2488 2794 2461 2126 2388
##      deaths (-)  416  415  453  411  466  434  443  429  434  417  388
##  emigration (-) 2618 2473 2702 2897 2980 2934 2804 2888 2708 2414 2733
##                 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014
##      births (+)  892  886  842  853  834  895  869  821  786  820  805
## immigration (+) 2482 2514 2404 2417 2536 2551 2491 2283 2191 2066 2148
##      deaths (-)  440  446  409  429  401  410  477  446  429  418  429
##  emigration (-) 2718 2962 3048 2983 3175 2740 2651 2442 2900 2513 2733
##                 2015 2016 2017 2018
##      births (+)  854  830  853  819
## immigration (+) 2186 2451 2287 2360
##      deaths (-)  436  459  468  459
##  emigration (-) 2691 2763 2736 2560
##
## all cells consistent : FALSE
```

Lexis Triangles and Different Views of the Events Data

We can extract deaths from the account using function `components`.

```
deaths_tri <- components(account, "deaths")
```

When we summarise `deaths_tri` we find that it has acquired a new “triangle” dimension:

```
summary(deaths_tri)
```

```
##
## name:     age       sex    time      triangle
## length:   101       2      26        2
## dimtype:  age       sex    time      triangle
## dimscale: Intervals Sexes  Intervals Triangles
## first:    0         Female 1993      Lower
## last:     100+      Male   2018      Upper
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.000   1.000   1.072   2.000  11.000
```

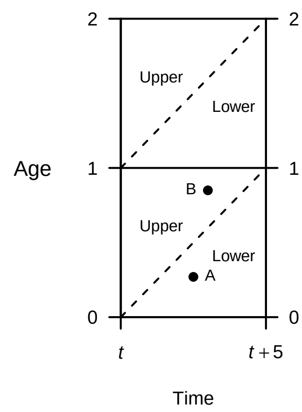


Figure 1: Lexis triangles

Lexis triangles can be used to distinguish between cohorts, in data on events that is cross-classified by age and time. An event belongs to the upper Lexis triangle if the person experiencing the event was already aged a at the start of the period; otherwise the event belongs to the lower triangle. In Figure 1, for instance, event B belongs to the upper triangle, and event A belongs to the lower triangle.

Demographic accounts in package **dembase** need to specify which Lexis triangle each event belongs to. Since we did not have a "triangle" dimension in the original data, function `Movements` has randomly assigned events to triangles. Here is a small subset of values, with the Lexis triangles:

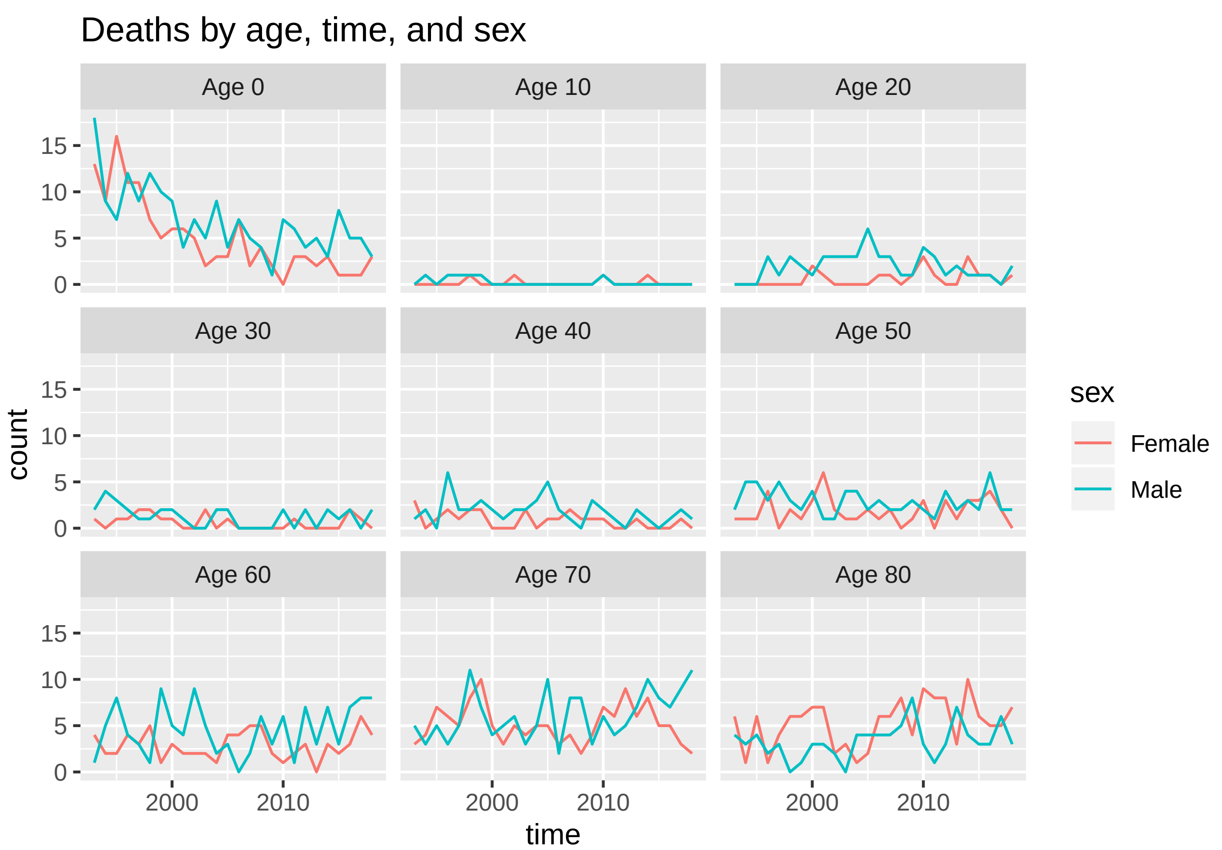
```
deaths_tri %>%
    subarray(sex == "Female") %>%
    subarray(age < 5) %>%
    subarray(time < 1998)
```

```
## An object of class "Counts"
##
## name:     age       time      triangle
## length:   5         6         2
## dimtype:  age       time      triangle
## dimscale: Intervals Intervals Triangles
## first:    0         1993      Lower
## last:     4         1998      Upper
##
## , , triangle = Lower
##
##    time
## age 1993 1994 1995 1996 1997 1998
##   0   10    3    5   10    3    6
##   1    1    2    2    0    0    0
##   2    0    0    0    2    0    0
##   3    0    0    0    0    0    0
##   4    0    0    1    0    0    0
##
## , , triangle = Upper
##
##    time
## age 1993 1994 1995 1996 1997 1998
```

```
##    0     3     6    11     1     8     1
##    1     0     1     0     0     1     0
##    2     0     1     0     0     0     0
##    3     0     1     0     0     0     0
##    4     0     0     0     0     0     0
```

When our data contains Lexis triangles, in addition to age and time, we can "rotate" the age time plan, to get different views of the data. We start with the original view, where deaths are arranged by age and time.

```
deaths_tri %>%
    subarray(age %in% seq(0, 80, 10)) %>%
    collapseDimension(dimension = "triangle") %>%
    as.data.frame() %>%
    mutate(age = paste("Age", age)) %>%
    ggplot(aes(x = time, y = count, color = sex)) +
    facet_wrap(vars(age)) +
    geom_line() +
    ggtitle("Deaths by age, time, and sex")
```



Now we rotate to a cohort-time format. (We need to drop the oldest age group, since function `rotateAgeTime` does not allow open age groups.)

```
deaths_cohort_time <- deaths_tri %>%
    subarray(age < 100) %>%
    rotateAgeTime(to = "cohort-time")
```

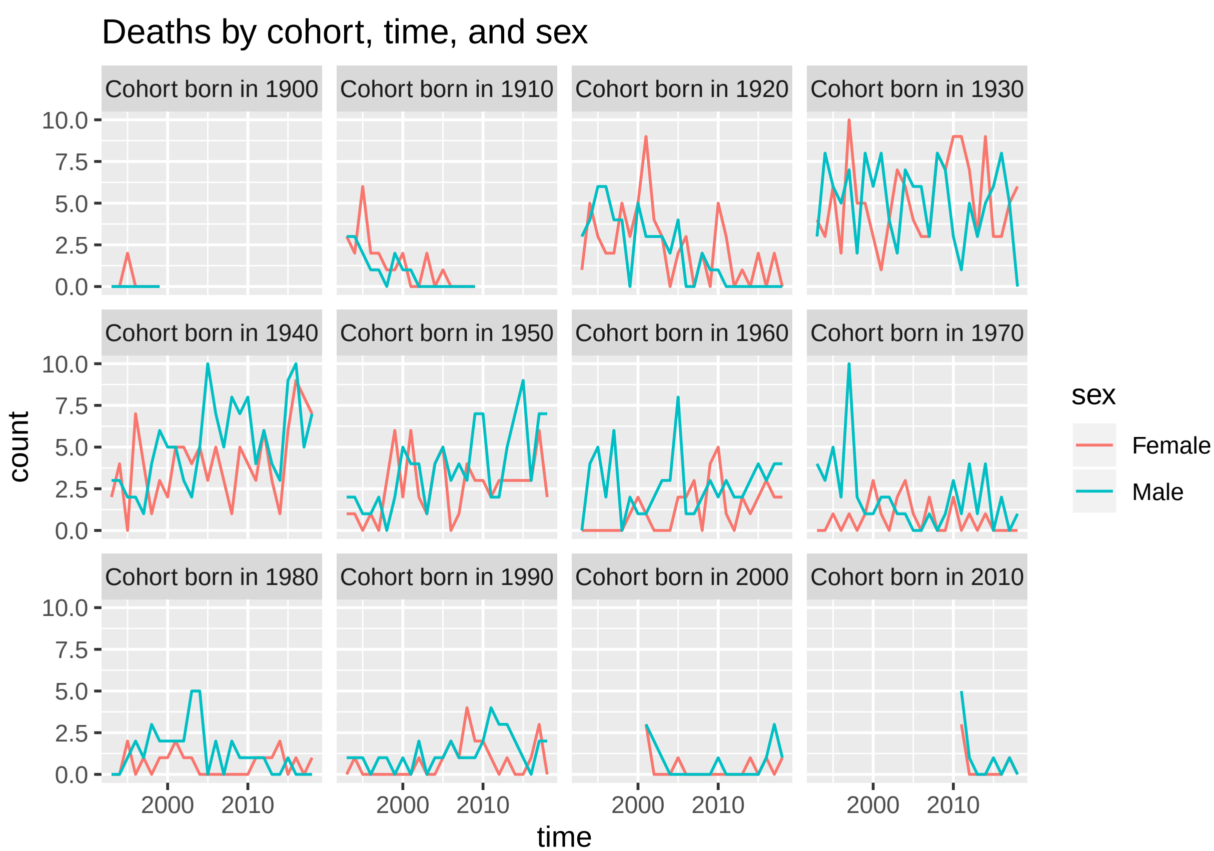
When we call `summary` on the rotated dataset, we find that the "age" dimension has been replaced by a "cohort" dimension.

```r
summary(deaths_cohort_time)
```

```
##
## name:      cohort    sex    time      triangle
## length:    126       2      26        2
## dimtype:   cohort    sex    time      triangle
## dimscale: Intervals Sexes  Intervals Triangles
## first:     1893      Female 1993      Lower
## last:      2018      Male   2018      Upper
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   0.000   1.000   1.083   2.000  11.000    2704
```

We can now graph deaths over time, for selected birth cohorts.

```r
deaths_cohort_time %>%
    subarray(cohort %in% seq(1900, 2010, 10)) %>%
    collapseDimension(dimension = "triangle") %>%
    as.data.frame() %>%
    mutate(cohort = paste("Cohort born in", cohort)) %>%
    ggplot(aes(x = time, y = count, color = sex)) +
    facet_wrap(vars(cohort)) +
    geom_line() +
    ggtitle("Deaths by cohort, time, and sex")
```



Next we rotate the original data to a cohort-age view.

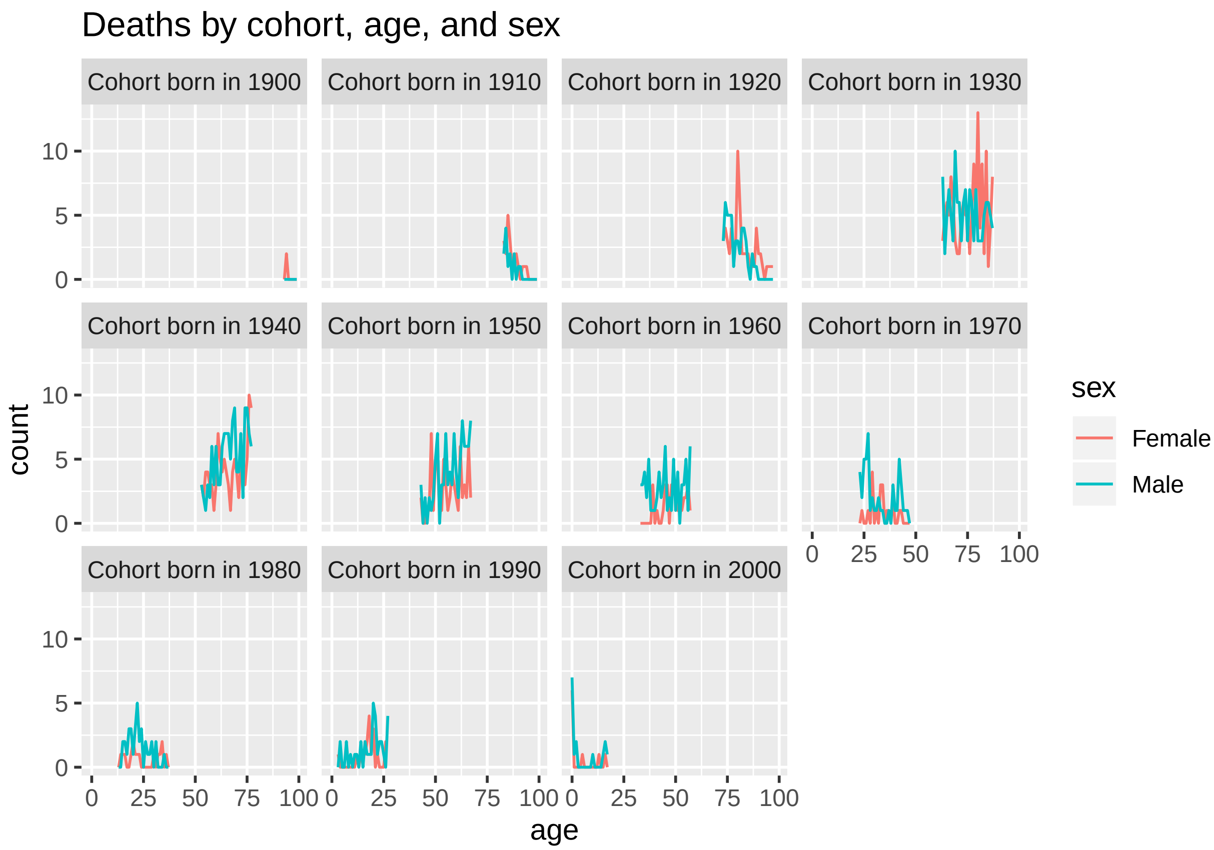
```r
deaths_cohort_age <- deaths_tri %>%
    subarray(age < 100) %>%
    rotateAgeTime(to = "cohort-age")
summary(deaths_cohort_age)
```

```
##
## name:     age       sex    cohort    triangle
## length:   100       2      126       2
## dimtype:  age       sex    cohort    triangle
## dimscale: Intervals Sexes  Intervals Triangles
## first:    0         Female 1893      Lower
## last:     99        Male   2018      Upper
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##    0.00    0.00    1.00    1.08    2.00   11.00   40000
```

We graph deaths over age, for selected cohorts.

```r
deaths_cohort_age %>%
    subarray(cohort %in% seq(1900, 2000, 10)) %>%
    collapseDimension(dimension = "triangle") %>%
    as.data.frame() %>%
    mutate(cohort = paste("Cohort born in", cohort)) %>%
    ggplot(aes(x = age, y = count, color = sex)) +
    facet_wrap(vars(cohort)) +
    geom_line() +
    ggtitle("Deaths by cohort, age, and sex")
```

```
## Warning: Removed 350 rows containing missing values (geom_path).
```



Function `Movements` has also added triangle dimensions to the births, immigration, and emigration series, and similar transformations can be carried out with them.