# Quick Start

The following corresponds to Appendix 1 of the paper *poptimesimul: R Programs for Generating a Longitudinal Synthetic Population for Educational and Research Purposes.* It describes the `R` programs that make up `poptimesimul` and how they should be run. We also briefly describe the `R` code for the examples that are given in the paper. The source code uses base `R` only. The example programs have extensive comments to help the user understand the steps.

The latest version of the programs is available in the GitHub repository `https://github.com/Jack-Gambino/poptimesimul`.

While reading this, it is useful to refer to Figure 1. Users who are only interested in creating a population, without changing any of the parameters, need to run just one file and change a single value depending on whether they want (1) just one month of data or (2) many months of data:

1. *One month of data*: To create an output file (or an environment) containing values for a population at one point in time, simply run the program `makepop.R` with the value `L=1`. By default, it will call other files automatically, as described below. This will create a file `onemonth.RData` containing the important variables. For more information, see the `poptimesimul` and `makepop` help files in the `man` or `html` directories.

2. *Many months of data*: To create an output file containing many months of data, run `makepop.R` with the value `L=120` (120 is the default number of months but other values can be used). It will call `makemonth()`, as described below. This will create a file `allmonths.RData`. For more information, see the `poptimesimul` and `makepop` help files in the `man` or `html` directories.

The basic parameters for the synthetic population are set in the file `parameters.R`. These include population values such as the number of strata, the number of clusters in each stratum and the average cluster size (number of people). The initial employment and unemployment probabilities (by age-sex group) are set here, as are the relative frequencies for education level (which vary by stratum). The parameters for income and `y_icc`, which are similar, are also set here. For the latter variable, a correlation parameter, which leads to an intracluster correlation, is set. Finally, basic transition probabilities for labour force status are defined.

The file `functions.R` defines a few small utility functions. These include functions that may be of general use (such as one to generate equi-correlated multivariate normal deviates) and functions that may only be useful for this particular project.

We describe the other programs in their logical sequence:

- The main program is `makepop.R`. The number of months $L$ determines what happens next. If $L = 1$ then `onemonth.R` is sourced. If $L > 1$ then `manymonths.R` is sourced. Much of the work to prepare the first month of data is done within `makepop.R`.

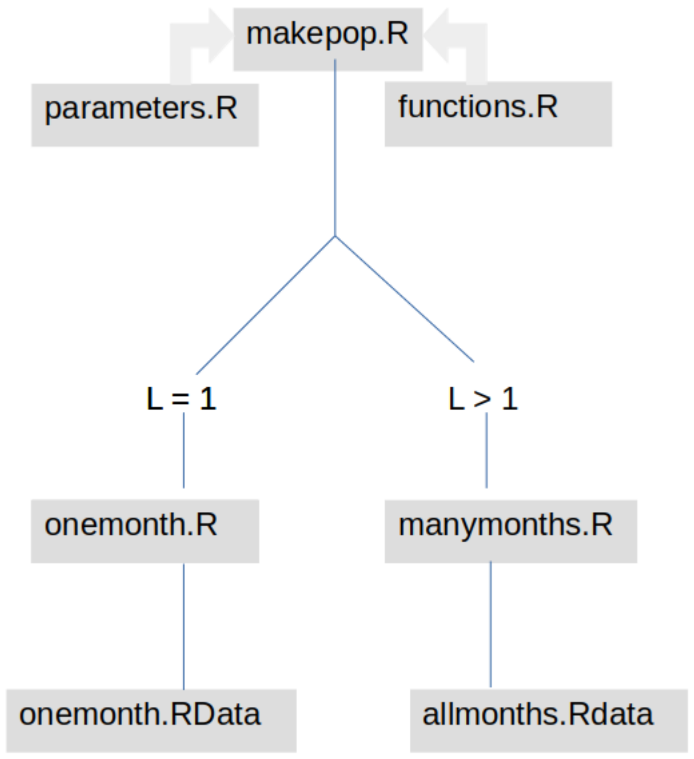

Figure 1: Functions, parameters and output files

- $L = 1$: `onemonth.R` is sourced by `makepop.R`. It simply saves the key population variables in `onemonth.RData`.
- `analyze.R` is an optional program that can be run (or better still, stepped through) to become familiar with the one-month version of the population. It can be run once the output file `onemonth.RData` has been created in the previous step.
- $L > 1$: `manymonths.R` is sourced by `makepop.R`. Its main job is to adjust transition probabilities month by month and call `makemonth()` each time. At the end, it saves the key population variables in `manymonths.RData`.
- `makemonth()` (in `makemonth.R`) is called by `makepop.R` via `manymonths.R`. For each month, once its parameters (transition probabilities) for that month are set in `makepop.R`, the function `makemonth()` is called with those parameters. `makemonth()` produces new population values for the current month. In this way, the initial population is "aged" month by month for $L$ months. The main idea is to adjust the transition probabilities month by month to make the population evolve in a realistic fashion (see Appendix 2). Once the probabilities are adjusted, the "aging" is done by calling `makemonth()`. Once all months have been generated, the user will have the full $L$ months of population data as described in Sections 2 and 3.
- `analyze_time_series.R` is an optional program that can be stepped through to become familiar with the full longitudinal population.

**Example programs**

There are several example programs that illustrate the use of the synthetic population. The programs were designed to have pedagogical value by implementing methods that are useful to students of sample survey theory. They also suggest how the population can be used by researchers. The package also includes two `.Rmd` files (vignettes), which correspond to the first and third example below.

- `example_stratification_inc.R` is the most elementary example (and not discussed further in this paper—but see its vignette or `example_stratification.pdf`). It simply illustrates the benefits of stratified random sampling. It produces both simulation-based results and exact ones from sampling theory (which are available because we have the whole population). In the stratified case, the sample is allocated using Neyman allocation.
- `example_var_with_replacement.R` uses the initial (month 1) population to illustrate that, in multistage sampling, the assumption that first stage units (clusters in our case) are selected *with* replacement when they are, in fact, selected *without* replacement is conservative, i.e., it overestimates the variance—slightly if the sampling fraction is low and more noticeably as the sampling fraction increases. There are two versions of the program: the one mentioned here and another one that uses just one stratum. For additional details about this example, see Section 4.1.1.

- `example_stratification_deterioration.R` is used to show how a good stratification of the population deteriorates over time as the characteristics of the members of the population evolve over the years. In addition to showing the deterioration of a fixed stratification-allocation combination it also shows what would happen if we could (unrealistically) reallocate the sample to strata each time. See Section 4.2.1 or the corresponding vignette (or example_stratification_deterioration.pdf) for additional details.