

## 4 Program Design

In this chapter we detail the design and implementation of the model. We have two primary concerns. The first and most important is to make the design and implementation of the model as transparent as possible by documenting in detail both the final implementation and the considerations that led us to the implementation. The second concern is to minimize the assumptions that are “hard wired” into the model so that it may be used in different ways.

### Overall program design

---

The implementation consists a suite of interrelated programs. The basic projection program implements the model defined in the preceding chapter in very general terms. This program reads from and writes to a large set of arrays, defined in the following section, that may be thought of as one large, complex data structure.

Input data is entered into these arrays using a “input data” routine that calls numerous input utility programs. The input utilities are small, simple programs that may be readily adapted to generate the necessary projection program inputs for different assumptions and data sources. For any given set of projections there will be a single master input program that creates all inputs required for the projection. This program also may be easily adapted to vary the number of age groups, duration groups, and time period, to implement different assumptions, or to capture data from different kinds of sources.

To illustrate the idea of using input utilities, consider the specification of non-HIV-related mortality risks. The input to the basic projection program consists of the arrays  $m_s(a, t)$  giving mortality risks for all single years of age and all projection years for females and males. A useful assumption for many purposes would be that  $m_s(a, t)$  is constant with respect to  $t$  for both sexes and derived from a model life table family. A utility program would take the model life table parameters as input arguments and compute and return the arrays  $m_s(a, t)$ . A projection for which  $m_s(a, t)$  is not invariant with respect to  $t$  would require an alternative but simply programmed utility. No change to the core projection programs would be required to implement this change.

Output data may be obtained directly from the various arrays, but these are numerous, large, and do not include summary statistics of any kind. The number and size of the arrays is a necessary consequence of the generality designed into the core projection program. In practice, output will always be secured via a set of output utility programs that select, process, and present summaries, graphical as well as numerical, of the core projection program output. Unlike the situation for program inputs, many different output routines may be used to examine projection output.

### Variable definitions

---

The core projection programs take as input an initial population state and a set of forces and parameters that govern how the initial state is moved one year forward, how the first projected state is transformed into the second projected state, and so on for as many years as the projection is to be carried out. The core projection programs produce as output the projected population

states and a set of arrays describing the flows between states during each projection cycle. The representation of the population state, the inputs and the outputs are described in the following sub-sections.

### Population state

The simplest way to represent population state over time is to use one or more arrays that include an index for time. The state of the population at each time generated by a projection will be represented by six arrays defined in the preceding chapter, as follows. Here and throughout, variable names and code fragments are rendered in Courier font.

$X_S(a,t)$       `xf[a,t], xm[a,t]`      Not sexually active, not infected

$Y_S(a,t)$       `yf[a,t], ym[a,t]`      Sexually active, not infected

$Z_S(a,d,t)$       `zf[a,d,t], zm[a,d,t]`      Infected

$D_S(t)$       `df[t], dm[t]`      Dead of a non-HIV-related cause

$A_S(t)$       `af[t], am[t]`      Dead of HIV-related cause

Here and throughout, the index *a* for age runs from 1 to A, the index *d* for duration runs from 1 to D, and the index *t* for time runs from 1 to T.

The core projection program will read the values of the state of the population at time *t* from the “*t* slice” of these arrays, produce the projected state for time *t* + 1, and write the values of the projected state to the *t* + 1 slice of the arrays. By a “slice” of the arrays  $X_S(a,t), \dots, A_S(t)$  we mean the set of arrays of one less dimension formed by fixing the value of the index *t*.

The initial state will be written into the time *t* = 0 slice of these arrays.

### Forces of movement, birth rates, sex ratio at birth, vertical transmission

The core projection program requires, in addition to the initial population state, a large number of forces of transition and several parameters. Program variable names are formed by spelling out the Greek letter used in the preceding chapter and appending the letter “f” or “m” to denote sex. The following list shows the representation of the forces and parameters described in the preceding chapter.

$m_S(a,t)$       `mu.f[a,t], mu.m[a,t]`      mortality from non-HIV-related causes

$m_S^B(t)$       `mu.fb[t], mu.mb[t]`      mortality from non-HIV-related causes

$u_S(a,t)$       `upsilon.f[a,t], upsilon.m[a,t]`      becoming sexually active

$t_S(a,t)$       `tau.f[a,t] and tau.m[a,t]`      becoming sexually inactive

$I_S(a,t)$       `lamda.f[a,t], lamda.m[a,t]`      new infections

$a(a,d,t)$       `alpha[a,d,t]`      mortality from HIV-related causes

$\mathbf{a}_S^B(t)$	<code>alpha.fb[t], alpha.mb[t]</code>	mortality from non-HIV-related causes
$\mathbf{b}_Y(a,t)$	<code>beta.y[a,t]</code>	birth rates for sexually active uninfected
$\mathbf{b}_Z(a,t)$	<code>beta.z[a,t]</code>	birth rates for infected
$\mathbf{s}_S$	<code>sigma.f, sigma.m</code>	proportion of female and male births
$\mathbf{z}(t)$	<code>zeta[t]</code>	vertical transmission

In addition to these quantities we require the parameters that determine the forces `lamda.f[a,t]` and `lamda.m[a,t]` (the  $I_S(a,t)$  values). These parameters are as follows.

$\mathbf{k}_S(a)$	<code>kappa.f[a], kappa.m[a]</code>	time-independent endogenous component
$\mathbf{q}_S(a,t)$	<code>theta.f[a,t], theta.m[a,t]</code>	time-dependent endogenous component
$\mathbf{i}_S(a,t)$	<code>iota.f[a,t], iota.m[a,t]</code>	exogenous component
$\mathbf{f}_S(a,t)$	<code>phi.f[a,t], phi.m[a,t]</code>	age pattern for $\mathbf{k}_S(a)$ , $\mathbf{q}_S(a,t)$ and $\mathbf{i}_S(a,t)$

## Flows

Output of the core projection programs consists of the projected population states, contained in the population state arrays defined above, and of the flows between different categories for each projection cycle. The notation used for these arrays is derived from the notation of the preceding chapter in the same manner as the notation described immediately above. Flows are denoted by catenating two letters, the letter to the left identifying the origin of the flow, the letter to the right identifying the destination. Note that these flows are numbers of events.

$\overline{XD}_S(a,t)$	<code>xdf[a,t], xdm[a,t]</code>	deaths of inactive, uninfected persons
$\overline{XY}_S(a,t)$	<code>xyf[a,t], xym[a,t]</code>	movements to sexually active status
$\overline{YD}_S(a,t)$	<code>ydf[a,t], ydm[a,t]</code>	deaths of active, uninfected persons
$\overline{YX}_S(a,t)$	<code>yxf[a,t], yxm[a,t]</code>	movements to sexually inactive status
$\overline{YZ}_S(a,t)$	<code>yzf[a,t], yzm[a,t]</code>	new infections
$\overline{ZD}_S(a,d,t)$	<code>zdf[a,t], zdm[a,t]</code>	non-HIV-related deaths of infected persons
$\overline{ZA}_S(a,d,t)$	<code>zaf[a,t], zam[a,t]</code>	HIV-related deaths of infected persons
$B_S^U(t)$	<code>ubf[t], ubm[t]</code>	uninfected births

$B_s^I(t)$	<code>ibf[t], ibm[t]</code>	infected births
$BD_s^U(t)$	<code>ubdf[t], ubdm[t]</code>	deaths (non-HIV-related) of infected births
$BD_s^I(t)$	<code>ibdf[t], ibdm[t]</code>	deaths (non-HIV-related) of infected births
$BA_s^I(t)$	<code>ibaf[t], ibam[t]</code>	non-HIV-related deaths of infected births

The last five flows refer to births to the population during each year of the projection, which may be uninfected or infected via vertical transmission, and to deaths of these births that occur before the end of the year in which they occur. Uninfected births necessarily die of non-HIV-related causes, but infected births may die either of non-HIV-related or of HIV-related causes.

## The core projection programs

---

The projection results described in the following chapter require three main programs, `input.routine.1()`, `project()`, and `output.routine.1()`. The input and output routines would be modified to implement different assumptions or input data.

Following the formulas of the preceding chapter, the core projection program `project()` first calculates the flows between states during the first projection year and uses these to compute the first projected population state distribution. It then calculates flows and the projected state for the next year of the projection, and so on until all specified years are computed.

`input.routine.1()` begins by calling the program `initialize(aa, dd, tt)`, which takes as input arguments the range of the age, duration and time indexes, i.e., the constants A, D and T, and initializes the arrays defined in the preceding section. The `initialize()` program is little more than a complete list of the various program variables. This will be very useful, however, for re-implementing the model in a different language should that prove to be desirable.

`input.routine.1()` then calls various input utility programs to compute or read input data of various types, process this data appropriately, and write the results to the appropriate positions of the appropriate arrays. The program `compute.mu(a, b)`, for example, takes as input arguments the additive and multiplicative parameters for a relational model life table, computes the life table using the Brass General standard mortality pattern, derives the corresponding forces of mortality, and writes copies of these into the appropriate time slices of the appropriate arrays. This program is written for projection in which non-HIV-related mortality risks are constant over time. Should it be desired to implement a projection with changing risks, an input utility would be written for this purpose. The modular design of the input utilities means that they are each simple, short programs that may be easily recast to embody alternative assumptions or input data.

`output.routine.1()` reads various information from the output arrays, e.g., the numbers of uninfected and infected persons of each sex, and computes summary information, e.g., infected males and females as a proportion of, respectively, total males and females.

## Input utility programs

---

An essential concept of the overall program design is to modularize components in such a way that the model is easily “extensible”, *i.e.*, so that supplementary input and output utilities may be simply written to give the core programs new functionality. The core projection programs incorporate a high level of generality and detail to allow for many different specific applications within the framework of a single model.

More specific assumptions are implemented using a set of input utility programs that produce the inputs needed by the core programs from a simpler set of inputs. In this way the core programs may be used to implement many variations on the basic model. This report does not try to anticipate such uses in any detail, only to provide a small number of basic utilities directed at model testing and research applications.

This section provides a set of very simple, preliminary input utilities. It should be emphasized that when, in this context, we refer to assumptions made, these are specific to these simplified input utilities, not to the general model. All of these assumptions may be relaxed by writing more general input utilities.

### Mortality from non-HIV-related causes

Forces of mortality from non-HIV-related causes are contained in the arrays `mu.f[a,t]` and `mu.m[a,t]`. A force of mortality input utility will take a relatively small number of parameters as arguments and fill these arrays with corresponding computed values. The program `brass.general.mltf(a,b)` takes additive and multiplicative relational model life table parameters and computes and returns a life table based on the Brass General standard age pattern of mortality (United Nations, 1983). The program `compute.mu(a,b)` calls this program to generate a life table for the argument parameter values and then compute and returns `mu[a]`. The master input program `input.routine.1()` will then, along with its numerous other tasks, call `compute.mu(a,b)` to compute suitable forces of mortality for males and females and write them to each time slice of the `mu.f(a,t)` and `mu.m(a,t)` arrays.

### Force of becoming sexually active

The input utility `compute.upsilon(v.scale, gamma, peak)` for these forces takes the three indicated gamma function parameters and returns the corresponding gamma function values at ages 0.5, 1.5, ...,  $A - 0.5$ , scaled so that the maximum value is one. Different values of the parameters are used for setting the male and female schedules.

### Force of becoming sexually inactive

Because empirical data for these forces is not readily available, a simple model will be employed in which the force of becoming sexually inactive is zero until some age  $a$ , rises linearly to  $b$  at age 50 (the end of the reproductive age span), and remains constant thereafter. The input utility `compute.tau(begin.age, slope)` takes the arguments `begin.age` ( $a$ ) and `slope` ( $b/(50 - a)$ ) and returns values of this function for ages 0.5, 1.5, ...,  $A - 0.5$ . Different values of the parameters are used for setting the male and female schedules.

### Force of new infection

The parameters used to compute the force of new infection can be set to any values desired. For simple initial specification, however, the age pattern has been set by a function  $f_s(a)$ , scaled so that its maximum value is one. Thus  $i_s(a,t)$  is calculated as a suitably chosen constant times

$f_s(a)$ , and similarly for  $k_s(a)$  and  $q_s(a,t)$ . The input utility for  $f_s(a)$  is `compute.phi(v.scale, gamma, h.scale)`, takes the usual gamma function parameters and returns scaled values. The values of  $i_s(a,t)$ ,  $k_s(a)$  and  $q_s(a,t)$  are computed from  $f_s(a)$  and specified constant multipliers in `input.routine.1()`.

### **Mortality from HIV-related causes**

The input utility for the force of mortality due to HIV-related causes reads a specified text file containing values for the array `alpha[a,d,t]`.

### **Initial population state**

The input utility `compute.proportion.active(mu, epsilon, tau)` takes force schedules for mortality, the force of becoming sexually active (for inactive persons) and the force of becoming sexually inactive (for active persons) and returns a schedule of proportions of persons active at each age, using a multistate life table calculation. The input utility `compute.stable.ad(growth.rate, mu)` takes a stable population growth rate and a force schedule for mortality and computes the corresponding stable age distribution. `input.routine.1()` calls these programs with suitable chosen arguments and computes the initial distributions from their results.

### **Output utility programs**

---

The core projection programs produce as output many very large arrays, far too many numbers to easily scan or absorb. We therefore define a set of default output programs to extract useful summaries from these arrays.

For the present, we consider a very simple default output program `output1()` that takes as input the entire output of the `project()` program and produces a table showing total population, new infections, HIV-related deaths and non-HIV-related deaths for each projection cycle.

### **Choice of language**

---

The computer language in which the model is implemented should be a high level language to minimize the labor of coding. In particular, it should support multidimensional arrays and array operations. It must incorporate good support for graphics, since it will nearly always be desirable to study results in graphical as well as numerical form.

An interpreted language is preferable to a compiled language, since this simplifies development. Efficiency of implementation, the main argument in favor of a compiled language is a tertiary concern for us. Should a more efficient implementation be necessary, our design and documentation will facilitate recoding in a compiled language at reasonable cost of time and effort.

The language should be widely available and widely used, since this will make the model more accessible. Availability for different operating systems and free availability are advantages. Finally, of course, the language must be sufficiently familiar to at least one of the authors to make implementation of the model possible on a very tight schedule.

S-PLUS does an excellent job of meeting these requirements (including the last!). It is a widely used commercial superset of the data analysis language S developed at AT&T's Bell Laboratories (Venables and Ripley 1994). S-PLUS is available from the StatSci Division of MathSoft. It is complete, high level, interpreted computer language with excellent support for multi-dimensional arrays and graphics, available for both unix and windows platforms. It is not freely available, but there is a freely available language R that reproduces most of the functionality required for our purposes.