Socsim Oversimplified

Carl Mason
carlm@demog.berkeley.edu

March 21, 2016

Contents

1 First Encounter

Originally written in the 1970s, Socsim continues to develop as a research tool constantly changing to meet the goals of new research projects. A slick graphic user interface has never been part of the plan[1]. As a consequence, users must be comfortable editing ascii files in order to use socsim.

Raising the barrier to first use still higher is Socsim's insatiable appetite for input data. Because micro-simulation is about simulating a whole mess of "individuals"' vital demographic events, Socsim needs a whole mess of age, sex, marital status, ***group***[2] and parity specific rates for births, deaths, marriages and change of group membership. There is of course, a default structure – for example if mortality rates are not specified for married males, Socsim will by default use those of single males of the same group. Or if parity specific fertility rates are not specified, Socsim will use the same rates for all parities. Still, a moderately complicated simulation will still require an *astonishing* number of rates.

1.1 The simplest possible simulation

To run Socsim you must provide the following:

A compiled version of Socsim. See `lab.demog.berkeley.edu/socsim` if this has not already been taken care of.

An initial population and marriage file. Socsim reads and writes Population data in a specific format. Population (or ***.opop*** files) are further described in Section 5. In order for Socsim to start, it is must find an ***initial population*** stored in this format. Generally individuals in this population will not be married, if the initial population has a marriage structure, then a marriage file (or ***.omar***) will also be required. The marriage file format is described in Section 2. The initial population need have no particular age or sex distribution[3], since one usually runs Socsim for a hundred (simulated) years or more in order to convert the initial population into a population with a known and stable age structure before simulating the processes that are of real interest. This can be done differently, but not without added complication.

A *supervisory or .sup file*. The supervisory file generally named with `.sup` suffix is passed to Socsim on the command line. It must contain certain ***global parameters*** such as the number of simulation ***segments***, name patterns of initial population and final population files, and either a complete set of vital demographic rates or else ***include*** statements specifying where to find files containing those rates. The full explanation of ***.sup*** file is in Section 4.

The minimal file, (from the `Sample` directory of the source code tree) is shown in Figure 1.

vital demographic rates As noted previously, the vital demographic rates can be included within the supervisory file. But generally it is more convenient to store them in one or more separate files which are referred to in the supervisory file with an `include` directive.

The format in which the rates are given to Socsim is critical. Generally a set of rates for particular demographic event begins with an identifier such as:

birth 1 F married 0

[1] to the extent that there is a "plan"

[2] Groups are a additional category that can be used to extend Socsim in interesting ways. Groups are described in a little more detail in Section 2.1

[3] One **can** start Socsim with an initial population complete with a kin network, but typically the purpose of this whole exercise is to generate a simulated a kinship network.

which indicates that the following several lines will contain a complete set of age specific fertility rates for group 1 married females of parity 0.

Following the rate identifier will be one or more lines each of which contains three space-separated numbers. The first two indicate year and months of the **upper age bound** over which the rate is in effect. The two quantities are added together so "1 1" is equivalent in effect to "0 13". Note that the first age category specified should NOT have "0 0" as its upper bound; an upper age bound of "0 1" would include the first month of life only. Age categories are otherwise flexible; one can specify some age categories that are as short as a single month, while others are many years or decades wide.

The rate, given in per month terms, is the third number. Socsim insists that there be a line specifying an age category with an upper bound of 100 years. Obviously this last rate can be 1 or zero. For more details on specifying rates See Section 8.

When all of the above bits are in place, Socsim can be executed from the Unix shell:

```
/path/to/socsim supervisoryfile.sup 12345
```

The result will be a considerable amount of screen output indicating which options are set and how the simulation is proceeding. Or just a short complaint about an inconsistency in the various input files.

After a successful socsim run, a logfile called /usersupervisoryfile.sup12345.log should be evident in the current directory as well as an output population ***.opop***; marriage, ***.omar*** and perhaps a extra variables, ***.opox*** file whose locations depend on the `output_file` directive in the ***.sup*** file. Socsim also write an event history file

The ***.opop***, ***.omar*** and ***.opox*** files produced by a Socsim run, are of course suitable initial population files(s) for subsequent Socsim simulations. Eventually however, you will probably have to stop simulating and start analyzing those files. See Sections 5and 6 for details on how these files are structured.

1.2 What goes on inside

While all that "information" is scrolling past on the screen and filling the log file, Socsim is busily scheduling and executing vital demographic events for each simulated being in the initial population as well as all of their descendants.

Socsim begins each simulation ***segment***[4] by scheduling an event for each "living" person. At all times during the simulation every non dead simulated being has exactly one scheduled event. Whenever an event is executed or a person's marital status or parity changes, a new event is scheduled for that individual. Thus death of a spouse causes a new event to be scheduled for the widow; a birth causes a new event to be scheduled for both the child and the mother.

To determine which event a person is to be scheduled for, Socsim generates a random waiting time for each event that the person is at risk of having. Obviously the user supplied age, sex, group, and marital status specific rates govern the process of random waiting time generation. Once all of the potential events have randomly generated waiting times associated with them, the earliest one is chosen and that event is scheduled. Information on the other possible events is discarded. In short, it's a competing risks model. The probability of each event is independent of all other events and the earliest event is the only one that counts. See Section 8.8 for more details.

[4] A simulation segment is a period of simulated time during which a single set of vital rates are in effect

Once each person has a scheduled event, Socsim starts to march through time. A list is created of all of the events scheduled for the first month of the simulation segment and each is drawn in random order and executed. When the execution of an event causes a new event to be scheduled (as most do) the resulting new event is either placed on the calendar for execution in a future month or if the waiting time is zero, it is inserted in the list of events waiting for execution in the current month and is drawn in random order.

When all of the events scheduled for the current month are executed, Socsim increments the month and repeats the event execution procedure just out lined. When the last month of the segment is completed, Socsim begins the next segment by reading the rate files and generating new events for all living people. If there is no subsequent segment, then Socsim finishes by writing the population (.opop), marriage (.oar), extra variables (.opox), and population pyramid (.pyr) files.

2 Extending Socsim

Socsim is open source so modifying the code to take account of cultural norms, ethnological truths, or behavioral theories is encouraged. Since many Socsim based research projects require some kind of enhancement, the program is structured so as to encourage modification. It is possible to enhance Socsim without first gaining a deep knowledge of all the intricacies of the program by taking advantage of "hooks" that have been placed in the code and which call functions at key points in the execution. "Stub" versions of these functions can be found in the `enhancementNULL.c` file in the top of the source tree.

When Socsim is compiled with the proper command line arguments, functions in `enhancement.c` become part of the code. If your modifications can be contained within `enhancement.c`, then improvements and bug fixes to the main trunk of the code will not conflict with your project – probably.

Before you consider enhancing Socsim, however, you should first give a lot of serious thought to how you might manage to accomplish your goal **without writing any C code**. One approach for doing this is is to try to make clever use of the concept of ***groups***, described below.

Another approach to email Carl Mason (carlm@demog.berkeley.edu) and see if maybe he'll do the programming for you.

2.1 Groups

Socsim implements ***groups*** by simply adding a variable to the population list. Every person is a member of exactly one group at all times. Group membership is determined at birth according to the rule given in ***child_inherits_group*** directive, but can change at any time according to age, sex, marital status, and group specific ***transition*** rates that the user specifies.

The group designation can be thought of as ethnicity, location, health status, wealth, education or any characteristic of a person that might or might not change according to a rate schedule.

Members of each group are subject to the same rates for vital events. That is the members of group 7 have fertility, mortality and nuptiality schedules that are independent of those of group 8. When an individual transitions from one group to another, she becomes subject to the new group's rates and therefore must have a new event scheduled.

Socsim determines how many groups are in a simulation at the beginning of each segment by looking at both the starting population and the rates. If Socsim finds a person of group N in the initial population than it knows that there are at least N groups in the simulation. If it finds a mortality rate for members of group Q then it knows that there are at least Q groups.

Because of the default pattern of rates described in Section 8, it is possible to add groups to a simulation and only specify the rates for the group which differ from the those of the existing group.

3 Marriage Market

The marriage event is greatly complicated by the need for two spouses. When a 27 year old female's marriage event comes up, there must be a suitable male at hand[5] or else that marriage cannot be executed on time. The need for two spouses means that randomness alone is inadequate for assuring that the specified marriage rates are achieved.

More generally, one can meet *at most* two of the following three marriage market constraints:

- Female age specific nuptiality rates
- Male age specific nuptiality rates
- Distribution of spousal age differences

Thus compromise is necessary and Socsim offers a choice of two: A "one" queue and a "two" queue system. In the *two queue* system male and female nuptiality rates determine the beginning of a marriage search: only egos who have initiated marriage searches can be married under this system. If no suitable potential spouses are available, the searcher must then wait in a queue. Because a suitable spouse must be waiting in order for a marriage event to be executed, generally one of the two spouses must be getting married *after* a delay of some time since the beginning of her search.

In the *one queue* system, only females have nuptiality rates. When a marriage event is executed under this system, the lucky bride chooses the best match from *all* living unmarried males. Thus under the one queue system, female nuptiality rates are generally met, and marriage markets can be made to optimize selections over even rare criteria. In the unmodified version of Socsim, only the distribution of the age difference between spouses is considered, but in principle the scoring algorithm could be made to optimize over a set of criteria.

The cost of this however are (1) no male marriage rates can be specified – males in the *one queue* scheme marry at the convenience of females – and (2) processing will be slower especially when populations are large.

3.1 The two queue system

Under the legacy "two queue" scheme, the marriage event signals the beginning of a marriage *search*. If a suitable spouse is *already* waiting on the marriage queue, then the marriage is executed. If not, then ego's marriage search fails and she is added to the marriage queue herself–where she will wait (subject to the risk of other events) until she is "selected" by a male suitor who is initiating a marriage search. Under this scheme, the specified nuptiality rates are seldom accomplished because would be spouses often spend time on the marriage queue. This scheme is symmetric with respect to sex. There is a marriage queue for each sex and each sex is treated identically.

If, when a marriage search is initiated, there are potential spouses waiting in the queue then each potential match is fist checked for allowability via the `marriage_allowable()` function and, if the match does not violate incest rules and the potential spouses have not been previously married, then the match is evaluated by the `score3()` function and the highest scoring match is executed.

3.2 The one queue system

The second (newer) scheme is a "one queue" system under which the female marriage event triggers a search across **all** available males and the "best" one is immediately selected. Potential matches are

[5] Socsim, does not yet handle same sex marriage. Sorry

evaluated in the same way as under the two queue system (via the `score3()` function), but many more potential matches are evaluated for each event.

The main advantage of the one queue system is that it generally achieves the specified female age specific marriage rates –even when female rates are high and populations are small. Achieving the female marriage rates allows for overall fertility rates to be achieved when marital and non marital fertility rates are different. This is major headache preventer.

A second advantage of the one queue system is that it allows for a wider range of marriage criteria to be optimized. This of course requires that Socsim be *extended* (See Section 2), but since under the one queue system *every* marriageable male is examined for each female marriage event, even spouses with qualities that are quite rare in the population can be located. Under the two queue scheme one can only choose the best spouse from those who have already initiated but not completed a marriage search. This can be quite unsatisfactory if for example the simulation is of a marriage market in which the optimal spouse is a particular type of cousin.

3.3 Evaluating potential marriages (score3())

Regardless of which marriage queue system is enabled, potential marriages are fist screened by the `marriage_allowable()` and then scored by the `score()` function.

The `marriage_allowable()`function, eliminates consanguineous marriages closer than cousins and remarriage of spouses who have previously been married to each other.

Potential allowable marriages are then evaluated by the `score3()` function. There are two variants built into Socsim, hooks are included in the code to make modification of the evaluation process “easy”.

3.3.1 Age distribution matching

If `marriage_eval` is set to “distribution” then Socsim seeks to select the marriage that most reduces the disparity between the ***target distribution of age differences at marriage*** and the ***observed distribution of age differences at marriage***. The `score3()` function does this by keeping track of the spousal ages at marriage of all marriages in the simulation segment as well as a predetermined target distribution. Each potential marriage is assigned a score that indicates the degree to which the fit between the target and observed distribution would be changed by the marriage in question. The marriage that most reduces the unweighted sum of the differences between the fractions of marriages conducted at each spousal age difference (in single years) is then chosen.

This variant is intended to be used with the one-queue system in which all living unmarried males are potential partners. In this situation then, Only female rates and the parameters of the target spousal age distribution determine marriage events. In other words, the age distribution matching scheme together with the “one queue” marriage market attempts to optimize the distribution of spousal age differences subject to the constraint that females marry according to their rate schedule.

When the age distribution matching scheme is paired with the two-queue marriage market, the constraint set is much more complicated and the results are uncertain.

Mechanically, the age distribution matching scheme is implemented in the following way:

- At the beginning of each simulation segment, a vector of “target” *proportions* of age distributions is constructed for each simulation group. This vector is derived from the normal distribution with parameters: `agedif_marriage_mean` and `agedif_marriage_sd`. It sums to 1 and represents that desired fraction of marriages to **women** of that group that should have the particular age difference at the time of marriage.

- The marriage queue consists of a linked list of all males(females) who are eligible. Under the one-queue system this would be all males who are not presently married. Under the two-queue system, it consists of males (females) who have had unsuccessful marriage events–a marriage event triggered a marriage search but no suitable spouse was available at that time.

- Note again that this marriage evaluation scheme is intended to be used with the one-queue marriage market under which each male is put on the queue at birth and upon execution of any event that subsequently makes him marriage eligible (e.g. divorce or death of spouse).

- When **under the one-queue system**: a female has a marriage event, or **under the two-queue**: system an ego of *either* sex initiates a marriage search, each potential partnership is given a score based on the age difference between the potential groom and bride. That score is derived from Equation 2. Note that the score can be either positive or negative. The match with the highest score is chosen. If more than one match has the same score, one is chosen at random. The result (intention) is that the match which most reduces the difference between the target and observed distribution of age differences at marriage is the one selected.

  Socsim can be extended to use additional information in this scoring algorithm.

$$\mathrm{agediff} = \mathrm{age}_{\mathrm{groom}} - \mathrm{age}_{\mathrm{bride}} \quad (1)$$

$$\mathrm{score} = \mathrm{target}_{\mathrm{agediff}} - \frac{\mathrm{count}_{\mathrm{agediff}}}{\mathrm{count}_{\mathrm{total}}} \quad (2)$$

- After each marriage is executed, the counts of marriages by female group and age difference is incremented.

- At the end of the simulation, the accrued differences between the target and actual marriage age differences are reported.

3.3.2 In practice ...

While the 'distribution' system of marriage evaluation can in theory match both the female age specific marriage rates *and* the specified distribution of age differences at marriage, "reality" has many ways of defeating the scientist:

Remarriage of widows and divorcees: In high mortality populations where widows remarry, this – particularly old widows, the average age difference at marriage can wind up being too low and the shape of the age difference distribution can diverge from normal.

Large mean age differences are difficult to achieve. If birth cohorts are roughly equal in size it is difficult to sustain a symmetric distribution of age difference at marriage when the mean is far from zero–unless a large fraction of people never marry.

Beware of exogamy If marriages happen between members of different groups and the different groups have significantly different vital rates, then a lot of strange things might happen. While Socsim does manage the age difference distribution separately for each group – it does so from a female perspective. That is the marriage age difference distribution for group N refers to the distribution of age differences at marriage (groom -bride) for marriages involving a *female of group N*. The male's group membership is not considered.

3.3.3 Age preference

If marriage_eval is set to "preference", then score3() implements the "simple" age preference schedule. In this system, potential marriage are evaluated according to a fixed preference scheme. Marriages are preferred according to how close the spouses are in age to the marriage_peak_age no attention is paid to the realized distribution of marriages.

After incest and endogamy/exogamy checks are complete, suitors who have not been eliminated, are assigned a score based on the groom's age - bride's age. If this age difference is greater than marriage_peak_age than the score declines at a constant rate as the difference increases. If the age difference is **less** than marriage_peak_age then the score declines at marriage_slope_ratio* that constant rate as the age difference decreases.

Before giving yourself a bald spot over this, note that only the ordinal rank of this score matters because the spouse is chosen randomly only from among the maximum scoring suitors. The idea is simply to favor a certain optimal age difference and to allow for a preference for older or younger spouses. In most simulations the marriage rates – which determine the age at which people initiate a marriage search, are **far** more important in determining marriage patterns than are these marriage preference parameters. The default value for marriage_peak_age is 36 months.

The marriage_slope_ratio parameter determines the degree to which groom-older marriages are preferred to bride-older marriages. A value of 1.0 implies an equal preference score for an age difference that is x months from marriage_peak_age regardless of whether the groom or bride is older. A value greater than 1 implies a that the preference score will decline faster as the brides age increases relative the the grooms – than it will as the groom's age increases relative to the bride's. The default value, 2, means that preference scores decline twice as fast with distance from marriage_peak_age as the bride's age increases relative the the groom's.

The marriage_agedif_max and marriage_agedif_min simply exclude from consideration marriages where the age difference between the partners (groom age - bride age) is beyond the specified bounds.

While it is possible to use this evaluation scheme with the one-queue marriage market, we have not tested it. This system is intended to be used with the the two-queue system.

4 Supervisory and rate files

The name of the ***supervisory file***, generally with `.sup` suffix, is passed to Socsim on the command line. Socsim expects such a filename followed by a random number seed and will not run without these two command line arguments.

What Socsim expects to find in the supervisory file is a set of parameters and possibly rate specifications that allow it to run the simulation. The supervisory file has to either contain all of the information Socsim needs (aside from the random number seed) or else it must have ***include*** directives that tell Socsim where else to look. The `.sup` file must consist entirely of valid directives or comment lines. Comments are lines that begin with '*'. Comments are ignored by Socsim.

The ***.sup*** file includes both global (affecting the entire simulation) and segment specific simulation parameters. The `run` directive indicates the end of a set of segment specific parameters. When Socsim encounters a `run` directive, it stops reading the `.sup` file and executes the simulation segment. When the segment's execution is complete, Socsim returns to reading the `.sup` file where it left off. This has two important implications:

1. If the directives for a segment are not followed by a `run` directive, the segment will not be executed. Socsim will simply read the instructions for the next segment and execute those – assuming that they end with a `run` directive.

2. Errors in a `.sup` file will not be caught until they are encountered. If there is an error in the specification of the 92^{nd} segment, Socsim will execute the first 91 segments before it exiting abnormally.

4.1 Minimal .sup file

At a minimum the supervisory file must include only a few directives. Figure 1 shows a minimal but sufficient ***.sup*** file.

```
************************************************************************
** This is the simplest possible socsim .sup file. It will run a one
** segment simulation with starting population in test.opop and
** test.omar and ending population in test.out{.opop,.omar} The
** duration of the one and only "segment" is 1200 months; the minimum
** birth interval is 24 months; heterogenous fertility is turned off;
** rate files are in ratefile.Lese.
**
** type [path to socsim] run.sup 12345
** to run socsim
************************************************************************
segments 1
input_file test
output_file test.out
duration 1200

include ratefile.Lese
run
```

Figure 1: Minimal supervisory file

4.2 Global Directives

Global directives can affect the entire simulation and *generally* do not change with each new segment. With the exception of `segments`, `input_file` and `output_file`, however, it is possible to change global directives in a simulation segment. What "global" means is that socsim does **not** reinitialize these directives to their default settings at the beginning of each simulation segment–as it does for the "segment specific" directives described in Section 4.4.

This means that **you can change most global directives within each simulation segment**. If Socsim finds a global directive such as `bint` or `sex_ratio` after the first `run` directive, it **will** change its behavior accordingly and the new behavior will persist until either the end of the simulation or until the directive is encountered again.

Be aware that if you decide to change a global directive for a particular segent, **the new value becomes the default** for subsequent simulation segments. In the case of segment specific directives, the default values are reset at the start of a new segment.

1. `segments` | *int* | def: **0**

   ■ `segments 13`

   The number of simulation ***segments*** in the simulation run. A segment is a period of time under which a given set of vital rates are effective. A simulation segment is executed when `run` directive is encountered. Since the rates are reinitialized at the begininng of each segment, it is necessary to (re)specify all rates in every segment. The `include` directive makes this convenient.

2. `input_file` | *word* | def: **none**

   ■ `input_file initpop`

   Indicates where Socsim should look for the input files. The arguement of this directive is a stem from which complete filename paths are constructed. `input-file-stem`.opop will be the initial population file; `input-file-stem`.omar will be the initial marriage file. The initial marriage file need not exist if the initial population has no kinship structure.

3. `output_file` | *word* | def: **none**

   ■ `output_file resultspop`

   Indicates where Socsim should write output files. Just as with `input_file` directive, Socsim expects the argument to be a file stem, from which Socsim can construct a complete path by simply appending a suffix. By default, the population and marriages files are only written at the end of the final segment and are called `output-file-stem`.opop and `output-file-stem`.omar. However, The `write_output` directive below causes these files to be written at the end of a segment and modifies naming conventions to avoid overwriting.

4. `proportion_male` | *real* | def: **0.5112**

- `proportion_male 0.5112`

Indicates the proportion of births which are male. In order to preserve an early mistake, the directive `sex_ratio` is a synonym for `proportion_male`.

5. `hetfert` | *1/0* | def: **1**

- `hetfert 1`

Enables the heterogeneous fertility feature (0 to disables it). If enabled, each female will have a beta distributed random variable by which her fertility is multiplied before random waiting times are generated. The result is a wider variation in sibling set size than would otherwise result. Note that the degree to which this fertility "multiplier" is inherited by daughters can be set by the user see ***alpha*** and ***beta***.

6. `alpha,beta` | *real* | def: **1.0,0**

- `alpha 1.0`

- `beta 0.0`

Determine the degree to which the fertility multiplier is inherited from each woman's mother. See Section **??** for a complete description.

7. `random_father` | *0/1* | def: **0**

- `random_father 0`

Indicates whether or not births to unmarried women should have a father randomly assigned. The default value 0 means "no". In this case the children of nonmarried/non-cohabiting mothers will have 0 as their father's id.

8. `random_father_min_age` | *int* | def: **15**

- `random_father_min_age 15`

Specifies the minimum age in years that males must be in order to be picked as fathers of children born to unmarried/non-cohabiging women. Constraints on marriageability e.g. incest constraints and endogamy constraints are also enforced with respect to random fathers, but marital status is ignored. The default value is 15. Obviously this is meaningless unless the `random_father` is set to 1

9. `bint` | *int* | def: **9**

- `bint 9`

The minimum birth interval. As a concession to reality, Socsim can impose a minimum number of months between births. For humans 9 months is a good number to use. Socsim adjusts the specified fertility rates upward to compensate for the birth interval.

10. | endogamy | *(-1..1)* | def: **0**

■ `endogamy 0`

Determines how suitors from other ***groups*** are treated in determining suitability for marriage. When ego is having a marriage event executed, s/he inspects everyone on the marriage queue of the opposite sex. A value of `endogamy` between 0 and 1 is taken as the probability that a potential spouse who is a member of a *different* group from that of ego will be rejected. A value of 1 therefore implies *endogamy* while a value of 0 implies that group membership will not matter with respect to marriage. A value of `endogamy` between -1 and 0 is taken as the negative of the probability that a potential spouse from the **same group** will be rejected. A value of -1 therefore enforces complete *exogamy* – all suitors of ego's own group will be rejected with probability 1.
The default value is 0 implies "radnomogamy" meaning that group membership is not considered when evaluating potential spouses.

11. | marriage_queues | *1 or 2* | def: **2**

■ `marriage_queues 2`

Determines which of the two possible marriage market schemes will be used. A "1" indicates that the ***one-queue*** system will be employed. Under this system ***all marriage-eligible males*** are evaluated for each female with a scheduled marriage event. A "2" indicates that the ***two-queue*** marriage market system will be used. In that case, both males and females have stochastically scheduled marriage (search) events and both sexes wait in their respecitve marriage queues if no suitable partner is immediately available. See Section 3 for a longer explanation

12. | marriage_eval | *preference/distribution* | def: **preference**

■ `marriage_eval preference`

Determines the method of evaluating potential marriages. "preference" indicates that the legacy age difference preference schedule be used. That scheme favors marriages based on their closeness to an ideal age difference. (See `marriage_peak_age` and `marriage_slope_raio`). "distribution" indicates that socsim should attempt to match a target distribution of the age difference between spouses at marriage. See `agedif_marriage_mean` and `agedif_marriage_sd`. Also see Section 3.3 for a lengthier explanation of all this.

13. | agedif_marriage_mean | group *real* | def: ***all groups* 2.0**

- `agedif_marriage_mean 1 2`

Determines the mean (in **years**) of the target distribution of spousal age differences for ***women* of the specified group.** This directive consists of the word `agedif_marriage_age_mean` followed by an integer indicating the group to which the directive applies and a real indicating the target mean spousal age difference in years. Currently Socsim uses a normal distribution as the target distribution. We await theoretically robust arguments in favor other parametric distributions. This is **only valid if** `marriage_eval` **is set to "distribution"** See Section 3.3 for more details.

14. `agedif_marriage_sd` | $int\ real > 0$ | def: **all groups 3**

- `agedif_marriage_sd 1 3`

Determines the standard deviation of the target spousal age distribution. Just as with `agedif_marriage_mean`, this directive is specified separately for each group in the simulation. This is **only valid if** `marriage_eval` **is set to "distribution".** See Section 3.3 for more details.

15. `marriage_peak_age` | int | def: **36**

- `marriage_peak_age 36`

Determines (together with `marriage_slope_ratio`) the preference for spousal age **difference** (in months) among spouses. This is **only valid if** `marriage_eval` **is set to "preference"**. See Section 3.3 for more details.

16. `marriage_slope_ratio` | $real$ | def: **2**

- `marriage_slope_ratio 2.0`

Works with `marriage_peak_age` to determine a marriage preference "score" for potential spouses. This is **only valid if** `marriage_eval` **is set to "preference"**. See Section 3.3 for more details.

17. `marriage_agedif_min` | int | def: **-120**

- `marriage_agedif_min -120`

Determines the *lower* end of the permissible age difference between spouses (groom age - bride age), in months. This is **only valid if** `marriage_eval` **is set to "preference"** See Section 3.3 for more details.

18. `marriage_agedif_max` | int | def: **120**

- `marriage_agedif_max 120`

Determines the *upper* end of the permissible age difference between spouses (groom age - bride age), in months. This is **only valid if** `marriage_eval` **is set to "preference"** See Section 3.3 for more details.

19. `child_inherits_group` | *rule* | def: **from_mother**

■ `child_inherits_group from_mother`

Determines how group membership is assigned at birth. Socsim understands the following rules:

- from_mother
- from_father
- from_same_sex_parent
- from_opposite_sex_parent
- n where n is a group to which all new borns are assigned.

By default, Socsim assigns newborns to mother's group. That is, if no option is specified, Socsim will use "from_mother".

4.3 Directives used in extended versions

A few directives are included for convenience when extending SOCSIM. in the plain version of SOCSIM, none of these directives should be set.

20. `parameter0..parameter5` | *r* | def: **none**

`parameter0` through `parameter5` should be set only if the enhanced version of Socsim that you are using defines what they do.

21. `read_xtra` | *1/0* | def: **0**

Causes Socsim read a file called `input-file-stem`.opox or crash if the file does not exist. The .opox file should contains a set of `extra variable` values for each person in the initial population. This will generally only be used in versions of Socsim that have been modified. See Section **??** for description of the programming hooks available for "easy" modification. The plain vanilla version of Socsim does not use extra variables so reading them will not generally cause anything useful to happen.

22. `size_of_extra` | *int* | def: **0**

Deterines is the number of variables that should be read for each person from the .opox file. This will generally be made to default when one modifies Socsim. In its unmodified state, Socsim does not use extra variables, so this directive is very infrequently what you are looking for.

4.4 Segment specific directives

These directives make sense if specified for each simulation segment.

1. | duration | *int* | def: **no default** |

   Determines the duration in months of the current simulaion segment.

2. | include | *filename* | def: **none** |

   Socsim will read and parse `filename` as if it were part of the current file. It is strongly recommended that include be used to keep rate specifications separate from the rest of the simulation parameters. See Section 8 has details on how vital demographic rates are specified.

3. | execute | *Unix shell command* | def: **none** |

   The specified command is passed to the Unix shell and its output is routed to the screen. After the command exits without error, Socsim returns to processing the current file.
   It is possible with this directive to call an external program that might perhaps generate rate sets on the fly perhaps in response to the previous simulation segment. For example:

```
execute generate_rates 1 5 0 >mortality.seg4
include mortality.seg4
```

5 Population files

Socsim reads and writes everything it knows about people in two files: the population ***.opop*** file and the marriage ***.omar*** file[6]. Both are space delimited files and both contain only numbers.

Since one generally runs Socsim for 200 simulated years in order to start from a population with a known and stable age structure, it is seldom necessary to construct an initial population with any information other than the age and sex of a small number of individuals. Such a file can be easily constructed in a spreadsheet program. The coresponding marriage file is simply an empty file with correct name.

One needs to come to terms with the structure of the .opop and .omar files in much more detail when analyzing simulation output. A snipet of code for reading an .opop file into R is given in Figure 2. In terms of R the .opop file's contents fit naturally into a ***data.frame*** with 14 columns all of which are numerical. In more general terms, the .opop file is a matrix where each row contains information on a single person and each column contains a particular bit of infromation on each person.

Although it's structure suggests that a .opop file might be right at home in a spreadsheet program, this is not so. First the files tend to be too large since they include not only a row for each person who ever lived. Even a modest sized simulation can easily have 100,000 rows. But more important, much of the information in the .opop file consists of identification numbers of other people. In other words the opop file is a multiply linked list. For manipulating linked lists, spreadsheets are profoundly suboptimal.

```
## read .opop into dataframe the .opop file contains one row for each
## simulated person who ever "lived". It generally includes many who
## "died" long ago.

opop<-read.table(file="../SimResults/example.opop",header=F,as.is=T)

## assign names to columns
names(opop)<-c("pid","fem","group",
               "nev","dob","mom","pop","nesibm","nesibp",
               "lborn","marid","mstat","dod","fmult")
```

Figure 2: Rcode for reading simulation output files

In most cases, the .opop file will be sorted in order of person id and since person ids are sequential integers assigned in birth order this means that the .opop file is generally sorted in birth order, with the row number often being the same as the person id. This is not however guaranteed to be the case so check that it is so before relying on it[7].

Table 1 shows which information is in each column.

[6] It may also read and write a file of extra variables and a file of transition history

[7] It is best practice not to rely on the .opop file's pid ordering because this convention can be broken in the initial population file

position	name	description
1	pid	Person id unique identifier assigned as integer in birth order
2	fem	1 if female 0 if male
3	group	Group identifier 1..60 current group membership of individual
4	nev	Next scheduled event
5	dob	Date of birth integer month number
6	mom	Person id of mother
7	pop	Person id of father
8	nesibm	Person id of next eldest sibling through mother
9	nesibp	Person id of next eldest sibling through father
10	lborn	Person id of last born child
11	marid	Id of marriage in .omar file
12	mstat	Marital status at end of simulation integer 1=single;2=divorced; 3=widowed; 4=married
13	dod	Date of death or 0 if alive at end of simulation
14	fmult	Fertility multiplier

Table 1: contents and format of the .opop file

5.1 Reckoning kinship

In analyzing Socsim output, one is often interested in reckonning kinship. Since with the possible exception of the initial population, everyone in socsim is related to everyone else, it is possible to find nearly any two people's relationship by following the chain of parents and siblings. To find ego's maternal grandmother, one simply finds the ego's mother's person id in the 6^{th} column of ego's row in the opop file. Moving then to the row of the opop file coresponding to ego's mother's person id, one look's again in the 6^{th} column to find egos' mother's mother's person id.

To find all of ego's children, one starts with the person id of ego's last born child (stored in column 10) of egos' row of opop. In the row coresponding to ego's last born child's row of opop, we find, in column 8 (9), ego's last born child's next eldest sibling through her mother (father). In that person's row of the .opop file, we can find yet another next eldest sibling and so on until we find ego's first born child, whose next eldest sibling through mother (assuming that ego is female) is necessarily zero.

Alternatively, one could simply collect all the rows of opop which have that same value in column 6 (mom) and or 7(dad) as ego has.

The R computing environment (`http://www.r-project.org`is particularly well suited for doing this kind of analysis of kinship.

5.2 Reference to marriages

Since individuals can be married more than once (simultaneously in some cases) reckoning marriage information is trickier than working with kinship alone. See Section **??** for more details on how to work with Socsim's .omar file. For the present purpose note that column 11 of the .opop file contains a

pointer, in the form of a marriage id number, to ego's most recent marriage. If column 11 is zero, then ego has never been married.

5.3 Reference to transition history

If the simulation includes group transitions, than socsim will write a file with the same path as the population and marriage files but with the suffix ***.otx***. The transition events do not have unique ids as do marriages, but each transition record contains the person id of the protagonist. Consequently it is much more natural to link from the transition history to the population file.

Section **??** describes the ***otx*** file in detail.

6 The marriage file

The marriage (.omar) file is similar to the population (.opop) file in that much of the information it maintains is in the form of unique id numbers that corespond to rows of the .opop file or to the .omar file itself. In other words, the .omar file is another *linked list*

Figure **??** shows a snippet of R code suitable for reading a .omar file. Table 2 shows the meaning of each column of the file.

6.1 Reckoning marriages

As with the .opop file, reckoning marriage histories requires following a list of integers from one record (row) to another. The marriage id of the wife's *most recent prior* marriage is stored in column 7. The coresponding pointer for the husband is stored in column 8. The other six columns of the marriage record hold information on the marriage itself. Note that marriages are created sequentially, but monogamy is not assumed. So for any particular marriage record, the wife's prior marriage (pointed to in column 7) must have a start date (column 4) that is no larger (later) than that of the particular marriage. A "prior marriage" must start before (or in socsim, at least in the same month) as the subsequent marriage. But the same is not true for the marriage end date, which is stored in column 5.

Also, be careful of zeros. Marriages that remained intact when the simulation ended have end dates of zero.

To find a female ego's first husband, one begins with the .opop file. The 11^{th} column of ego's row holds the marriage id of egos most recent marriage. The row in the .omar file whose first column entry matches that number is the record of egos' most recent or "last" marriage. In order to find her **first** marriage, we must locate the row of the .omar file wherein the entry in column 1 matches the the marriage id of the wifes prior marriage which is stored in column 7 of the .omar file. We repeat this proces until we locate a record of a marriage for which ego's person id is stored in the second column and column 7 holds a zero.

In R, a more efficient way of finding first marriages is to select the subset of marriages for which either the husband's prior or wife's prior marriage (column 7 or column 8) are zero, and then use the `match()` function to link the marriage id to the husband or wife's opop record. Figure **??** shows a snippet of R code that performs this task.

```
omar<-read.table(file="../SimResults/example.omar",header=F,as.is=T)
names(omar)<-c("mid","wpid","hpid","dstart","dend",
               "rend","wprior","hprior")

rownames(omar)<-omar$mid
```

Figure 3: R code for reading a .omar file
labelfig:readOmar

position	name	description
1	mid	Marriage id number (unique sequential integer)
2	wpid	Wife's person id
3	hpid	Husband's person id
4	dstart	Date marriage began
5	dend	Date marriage ended or zero if still in force at end of simulation
6	rend	Reason marriage ended 2 = divorce; 3 = death of one partner
7	wprior	Marriage id of wife's next most recent prior marriage
8	hprior	Marriage id of husband's next most recent prior marriage

Table 2: Structure of Socsim marriage file

```
## get first marriage id -- socsim stores marriage ids as linked list
## headed by most recent marriage.  The (h/w)prior field stores the id
## of each spouses prior marriage

## select a subset of marriages which are first for at least one partner
fomar<-omar[omar$hprior == 0 | omar$wprior == 0,]
opop$fmid<-NA

## use match() to lookup the marriage id of each person (in opop)'s
## first marriage id
opop[opop$fem==0,"fmid"]<-
  fomar[match(opop[opop$fem==0,"pid"],fomar$hpid),"mid"]
opop[opop$fem==1,"fmid"]<-
  fomar[match(opop[opop$fem==1,"pid"],fomar$wpid),"mid"]
```

Figure 4: R code for finding first marriages
labelfig:firstMarriages

7 Transition history files

If group transitions are part of the simulation then socsim will write (and may read) a transition history file. In spirit, the ***otx*** file is much like the marriage file, the main differences are that the ***file*** is linked to only one person and there is no unique identifier for a transition history record.

Like all other Socsim output files, the ***otx*** file is space delimited and contains only numbers. Table 3 shows the structure of the ***otx*** file. Figure 5 contains R code for reading an otx file into a data frame.

position	name	description	
1	pid	Person id to who the transition event occurred	
2	date	Month in which the transtion occurred	
3	fromg	group from which the person transitions	
4	tog	group into which the person transitions	
5	sequence	a non positive number indicating the order of the event. A zero indicates that the current record refers to the most recent transition event; a -7 indicates that seven transitions have occurred to this person subsequent to that of the current record.	

Table 3: Contents and format of the .otx file

```
otx<-read.table(file="../SimResults/test.otx",header=F,as.is=T)
names(otx)<-c("pid","month","fromg","tog","pnum")
```

Figure 5: R code for reading an ***otx*** file

8 Specifying demographic rates

It is most convenient to store demographic rates for each simulation segment in distinct files and use the `include` directive to reference them from the supervisory file (See Section 4). But regardless of how you choose to organize your rate files, you will need to assemble a large collection of rates. How Socsim expects those rates to be formatted is described in Section 8.2, but before we get to that Section 8.1 describes the rules Socsim follows when it encounters incomplete rate sets. These are important to understand, because Socsim does not warn you when for example you leave out fertility rates for parity 1 divorced females of group 3 (birth 3 F divorced 1). Instead, it "defaults" to rates for party 0 divorced females of group 3 (birth 3 F divorced 0). Similarly if the birth 3 F divorced 0 rates are missing, Socsim uses birth 3 F single 0) in their place. The rules are fairly intuitive, but its important to understand that one can err by not specifying that certain rates are zero.

8.1 Rate default rules

To run a moderately realistic simulation, Socsim requires age specific fertility rates for females and mortality rates for both males and females of each ***group*** and marital status. If rates do not differ by marital status or group then you can use Socsim's default rules to avoid entering the same blocks of rates repeatedly. When Socsim encounters incomplete rates sets, it follows a set of rules to determine how the blanks are to be filled in. Figure 8.1 shows the rules that Socsim uses when it encounters incomplete rate sets. The "==>" symbol in Figure 8.1 means "defaults to" so for example,

```
widowed                          ==> divorced; parity 0; group 1
```

which appears in the "Fertility Rates" section under the heading "For parity zero women in group 1" indicates that if Socsim does not find fertility rates for parity zero, widowed females in group 1 it will "default to" the rates for divorced women of parity zero and group 1.

Where Figure 8.1 indicates that a rate block defaults to "Zero", Socsim does not default to anything leaving such events with zero probability of occurring at any age. So for example, unless you think that single males in group 1 should live forever, you must specify mortality rates for such "people".

8.2 Structure of vital rates

The format in which Socsim expects to find rates is simple. Each ***block*** of rates begins with a set of keywords which indicate the event for which the rates apply and the marital status, sex, group membership and possibly parity of the people who are at risk of experiencing the event. That information is followed on subsequent lines of rate values and the **upper bound** of the age group for which the rate is in force.

A ***rate block*** is a complete set of age specific rates governing a demographic event for people of a particular sex,group and marital status. An example of a rate block is shown in Figure 8.2. The first line after the comment line, indicates which event (death); group (1); sex (M=male); and marital status (single) this rate block pertains to. The order matters and is always, Event then group then sex then marital status. In the case of birth rates this may be followed by number indicating parity. In the case of transition rates, the line must end with a number indicating the destination group.

Each subsequent line contains a one month rate (in the case of fertility) or a one month probability in the case of all other events, and the age interval over which the monthly rate (probability) holds. The first two numbers in the line are years and months of the upper age bound. These are added together so a 1 and 11 would mean 23 months. The third number is the rate. In the case of fertility it represents the expected number of births **per month** to a woman who survives to end of the given age interval. Specifically – The interval that includes upper age bound given in the previous line and ends just before the upper age bound given on the **current line**.

Figure 8.2 shows an example of rate block. The first line indicates that that rates which follow refer to mortality of group 1 single males (death 1 M single).

The first first rate line (`0 1 .0460940`) indicates that the probability of death in the first month of life for males (technically for never married males) is .0460940. Note that a rate line with an upper age bound of "0 0" is meaningless and is ignored. So be careful when specifying infant mortality rates.

Taking another line from Figure 8.2, the probability that a single male dies between the ages of 1141 and 1200 months, conditional on having survived to the beginning of the age interval is $1-(1-.08326)^{60}$.

8.3 Mortality rates

Mortality rates are the most straight forward of the rates that Socsim uses. The example in Figure 8.2 is typical. The event identifier is `death`, the "1" refers to the group, "M" to male and "single" is of course marital status.

8.4 Marriage rates

Marriage rates are specified with the event identifier of `marriage`, but it should be born in mind the event which these rates regulate is not really marriage but rather the commencement of a marriage search. Since marriage requires two participants Socsim cannot simply execute a marriage when the event is scheduled. Marriage requires two participants and such it is very difficult to achieve arbitrarily specified marriage rates for both males and females. If a marriage age distribution is also part of the simulation, it gets even harder. Socsim deals with this in a variety of ways which are described in Section 3 and Section 4.

As a consequence, of all this excuse making is that marriage rates often need to be "tuned" in order to achieve the desired result.

Figure 6: Rate default rules

Fertility Rates

```
For parity zero women in group 1
--------------------------------
single                          ==> Zero
married                         ==> Zero
divorced                        ==> single; parity 0; group 1
widowed                         ==> divorced; parity 0; group 1
cohabiting                      ==>  married; parity 0; group 1

For women in group 1 with higher parity
---------------------------------------
mstatus m; PARITY P; group 1     ==> mstatus m; PARITY P-1; group 1

For women of any parity and any group > 1

mstatus m; parity  p; GROUP  G   ==>  mstatus m; parity p; GROUP G-1
```

Marriage, Divorce and Mortality Rates

```
For men and women in group
----------------------------------
DEATH    for single; sex s; group 1       ==> Zero
MARRIAGE for single; sex s; group 1       ==> Zero
DIVORCE  for mstatus m; sex s; group 1    ==> Zero

  (Events except for divorce)
----------------------------
event e for divorced;sex s    ==>  event e; for single;sex s;group 1
event e for widowed;sex s     ==>  event e; for divorced;sex s;group 1
event e for married;sex s     ==>  event e; for widowed;sex s;group 1
event e for cohabitting;sex s ==>   event e; for married;sex s;group 1

For Groups > 1
--------------
event e for mstatus m;sex s; group g ==> event e for mstatus m;sex s; group g-1
```

Group Transition Rates

```
TRANSITTION to group H for mstatus m;sex s; group g == Zero
```

Figure 7: Example of a block of mortality rates

```
*Mortality, single Male (lines beginning with * are comments)
death 1 M single
0       1       .0460940
0       12      .0057540
0       60      .0008730
0       120     .0002600
.
.
.
0       1140    .0832630
0       1200    .0832630
```

8.5 Divorce rates

Divorce rates are specified with the event identifier `divorce`. Divorce is unusual among Socsim events in that it's rates do not apply to the age of one spouse or the other but rather by age of the marriage. It is thus generally not necessary to specify divorce rates for both sexes.

8.6 Fertility rates

Fertility rates are specified with the identifier `birth` and are different from other rates in two ways:

1. They are parity specific. But they default to parity $n-1$ so it is only necessary to specify rates for parity zero.
2. They are rates rather than probabilities. So multiplying a rate by the number of months in the age category gives the expected number of births that a woman who lives through the age category will experience.

8.7 Transition rates

Transition rates give rates of "transition" from one ***group*** to another. By default, no transitions occur, however, if the initial population contains more than one group then the group inheritance rule determines the group identity of new borns.

Transition rates are specified with the identifier `transit` and are different from other rates in that both the group to which the rate applies **and** the group to which the event will cause a person to belong must be specified.

To specify transition rates from group 1 to group 2 for single males, one would write the following:

trasit 1 M single 2

As noted in Figure 8.1, transition rates have no defaults. All rates have to be specified in order to take effect.

8.7.1 Duration specific transition rates

It is often useful for transition rates to be duration rather than age specific. In other words, the probability of a transition event occurring can depend on the time since the individual transitioned into the current group rather than the individual's age. In order for this distinction to matter, a person must have experienced at least one transition other wise the time spent in the group is equivalent to the person's age.

Divorce rates are always duration specific, but transition rates may vary with either age or duration. The format for specifying transition rate blocks is the same regardless of whether they are age or duration specific. To tell Socsim that a particular rate block is meant to be duration specific, add the directive: `duration_specific` as in:

```
duration_specific transit 4 F married 5
```

when this directive is encountered, the log file will indicate that transition rates from group 4 to group 5 for married females will be duration rather than age specific. The default is for transition rates to be age specific so no indication is given in the log file of that condition. It is possible to have both age and duration specific transition rates in effect in the same simulation, however, only one transition rate block is allowed per pair of groups. So transitions from say group 3 to group 5 for single males must be *either* age or duration specific, but whichever it is, has no effect on transitions from group 3 to group 5 for ***married*** males. One may be age specific while the other may be duration specific.

As noted in Section 4.4, the `duration_specific` directive does not replace the keywords that define the rate block.

8.8 Event competition

At the start of every month in the simulation, every living person has exactly one event scheduled for some future date. In the course of this month, all scheduled events are executed. Events affect the individuals for whom they are scheduled but may also affect spouses and others to whom they are connected. All births, deaths, marriages and group transitions that are scheduled for the current month are executed in random order. After a person's event is executed, unless that event was death, a new event must be scheduled. New events are scheduled by an "event competition." This event competition is also held once for each living person at the beginning of each simulation segment (that is, every time the demographic rates or societal constants change).

Each event for which the individual is at risk (e.g., men rarely give birth) can be modeled as a piecewise exponential distribution. A random number is used to generate a waiting time until this event occurs (which is bounded by the individual's maximum possible age at death). The individual's next event is the one with the shortest randomly generated waiting time. The event competition thus follows a competing risk framework wherein the probability of each event is independent of all others.

8.9 Generating potential waiting times

The waiting time algorithm is conceptually equivalent to drawing a random number u, from a uniform (0,1) distribution, calling u the probability that the event will not yet have occurred, then finding the first month by which the probability of non-occurrence is less than or equal to u. The probability that an event will not have occurred by a particular month T is given by the expression

$$\prod_{t=0..T} (1 - p_t) \tag{3}$$

Where p_t is the probability of the event's occurrence in period t conditioned on it not having occurred at any time before t. Since $(1 - p_t)$ is always between 0 and 1, the expression given above is nonincreasing in T. Consequently, beginning with $t = 0$ we can successively multiply the $(1 - p_t)$ terms together until the value of the product falls below u. What Socsim does is mathematically equivalent to this procedure, however, the implementation in function `datev` takes advantage of fact that the probabilities can be the same over months or years and works with powers of $(1 - p_t)$.

9 Heterogeneity multipliers

Heterogeneity beyond that which follows from the algorithm described in Section 8.9 is often desirable in microsimulation. Socsim increases heterogeneity of fertility for example, in order to create more realistic sibling set sizes and to allow for heritability of fertility.

Heterogeneity of mortality and group transitions are not included by default but much code is in place to allow users to add these features easily in a way that makes sense for a particular simulation.

The general principle is that if each person at risk of an event is given a value θ drawn from a distribution with mean=1 and the hazard rates used for generating each individual's waiting time for the event are multiplied by θ, then $E(h\theta) = h$. Where h is the original hazard of the event. Thus the overall population's event history should still reflect the rate structure but with greater variance/heterogeneity.

9.1 Fertility multiplier (fmult)

When the heterogeneous fertility option,(***hetfert***) is enabled, each female in the population is assigned a fertility multiplier. For the initial population these multipliers may be read in the ***.opop*** file or they

may be generated by Socsim. Females born during the simulation will have multipliers generated by Socsim. Once assigned, fertility multipliers remain with the woman for her entire life increasing or reducing proportionally the hazard of giving birth at each.

In the current implementation, individual fertility multipliers, ***fmult***s, are pseudorandomly distributed as a cubic approximation to the beta distribution with mean 1.0, variance 0.416, and a range of 0 to 2.4. Fertility heterogeneity is also heritable to a degree determined by the user specified directives `alpha` and `beta`.

9.2 Inheritance of Fertility Multipliers

The degree to which heterogeneous fertility is inherited is determined by the `alpha` (α) and `beta` (β) directives. Equation 5 defines the algorithm used to effect the inheritance. At birth females are first assigned a fertility multiplier as according to the beta distribution described in Section 9.1 this is γ in Equation 5. The temporary variable x gets the α weighted average of γ and ego's mother's fertility multiplier. If $\beta = 1$ then x is the daughter's fertility multiplier. Otherwise it is further modified by the second equation.

$$x = \alpha * \mathrm{fmult}_{\mathrm{mother}} + (1 - \alpha) * \gamma \quad (4)$$

$$\mathrm{fmult}_{\mathrm{daughter}} = 2.5 * \exp^{\beta * log(\frac{x}{2.5})} \quad (5)$$