Automatically Computing Turnover for Steady-State Population Distributions

Jesse Knight

July 16, 2018
Revision 1: October 9, 2018

1 Background

When modelling sexually transmitted diseases, it is important to consider heterogeneity in levels of sexual activity among the population, as transmission dynamics are profoundly impacted by these characteristics. Moreover, it is rarely sufficient to model static groups, since people's sexual behaviour typically changes over time. Therefore, transfer of individuals between activity groups should be included in the model, which we call "turnover".

The most important implications of this feature are the transfer of *infected* and *susceptible* individuals from one activity group to another, which provides another mode of transmission besides direct sexual contact between the activity groups.[1] For example, in a perfectly assortative population, activity level turnover is the only way transmission can occur *between* activity groups.

From the modelling perspective, it is possible to consider turnover from any group to any other group. However, the definition of turnover rates among each of the groups will impact the steady-state population distribution. So too will the rates of births and deaths, and the distribution of new individuals. Our problem, then, is as follows:

Problem: How do we choose turnover rates, in conjunction with rates of birth and death, and the distribution of entering individuals, in order to yield a specific steady-state distribution of activity levels in the population?

Hypothesis: There exists a closed-form equation relating these rates and distributions.

Application: This equation can be used to define the appropriate rates of population turnover, in order to match the other observed parameters.

2 Maths

We denote the state variable representing the proportion of the population in activity group $i \in [1 \dots \mathrm{N}]$ as x_i and the vector of all x_i as $\boldsymbol{x}$. Since turnover transitions can occur between any two groups, in either direction, we denote the turnover rates as an $\mathrm{N} \times \mathrm{N}$ matrix ζ, where ζ_{ij} corresponds to the transition $x_i \to x_j$. A verbose definition is given in Eq. (1), where the diagonal elements are denoted $*$ since they are not useful.

$$\zeta = \begin{bmatrix} * & x_1 \to x_2 & \cdots & x_1 \to x_{\mathrm{N}} \\ x_2 \to x_1 & * & \cdots & x_2 \to x_{\mathrm{N}} \\ \vdots & \vdots & \ddots & \vdots \\ x_{\mathrm{N}} \to x_1 & x_{\mathrm{N}} \to x_2 & \cdots & * \end{bmatrix} \tag{1}$$

The rate of population entry is denoted ν, and exit as μ. The proportion of the entering population who are activity level i is denoted p_i, while the distribution of exiting population is simply x_i.

These transitions and related rates are summarized for $\mathrm{N} = 3$ in Figure 1.

Our aim now is to relate these quantities to a target steady-state distribution for the vector $\boldsymbol{x}$.

[1] An interesting parallel can be found with the modes of heat transfer: conduction – direct transfer of heat by particles in contact (contact within an activity group), versus convection – movement of hot particles from one area to another (turnover).

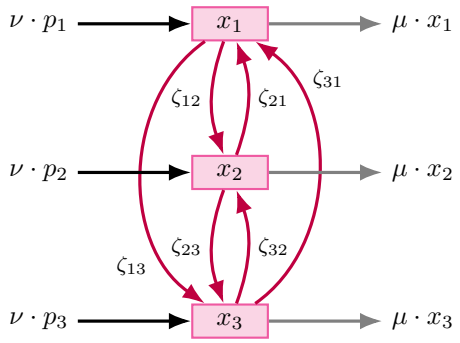

Figure 1: Schematic of demographic transitions among 3 activity groups: high x_1, medium x_2, low x_3; entry is shown in black, exit is shown in grey, turnover is shown in purple.

2.1 System Equations

For a given activity level, the rate of change of the group is defined by the net flows, as in:

$$\frac{d}{dt}x_i = \nu\, p_i + \sum_j \zeta_{ji}\, x_j - x_i \left(\mu + \sum_j \zeta_{ij}\right) \tag{2}$$

In matrix form, we can write the entire system as:

$$\frac{d}{dt}\boldsymbol{x} = \nu\, \boldsymbol{p} + \left(\boldsymbol{x}^{\mathsf{T}}\zeta\right) - \boldsymbol{x} \left(\mu + \sum_j \zeta_{ij}\right) \tag{3}$$

Let's assume that ν, μ, and $\boldsymbol{p}$ are known, and that we aim to find ζ which yields the observed distribution of activity levels $\boldsymbol{x}$ at equilibrium. At equilibrium, the derivative may not actually be zero, however, since populations typically grow over time at a rate $g = \nu - \mu$. The desired equilibrium rate of change is therefore given by $\nu\,\boldsymbol{x} - \mu\,\boldsymbol{x}$, which we can substitute for the left hand side of Eq. (3) and rearrange to give:[2]

$$\nu(\boldsymbol{x} - \boldsymbol{p}) = \left(\boldsymbol{x}^{\mathsf{T}}\zeta\right) - \boldsymbol{x} \sum_j \zeta_{ij} \tag{4}$$

The right hand side can then be factored to give a linear system of $\boldsymbol{z} = \mathrm{vec}(\zeta)$, parametrized by $\boldsymbol{x}$:

$$\nu(\boldsymbol{x} - \boldsymbol{p}) = A\,\boldsymbol{z}, \qquad A \in \mathbb{R}^{\mathrm{N}\times\mathrm{N}^2}, \;\; A_{ij} = f(x, i, j) \tag{5}$$

Some example systems showing the linear factorization A and $\boldsymbol{z}$ are given in Appendix A: Example Systems, while Appendix B: Python Code gives an algorithm for defining A for any N.

2.2 Solving the System

While it is possible to solve Eq. (5), a number of problems emerge.

1. **Diagonal elements** – The diagonal elements of ζ have no impact on the system. Therefore in any solution, they will be undefined.
2. **Underdetermined system** – For $\mathrm{N} > 2$, the system is underdetermined, since $\mathrm{Rank}(A) = \mathrm{N} \leq (\mathrm{N} \times \mathrm{N} - \mathrm{N})$. This means that A is not invertible and there are many possible solutions to $\boldsymbol{b} = A\boldsymbol{z}$.
3. **Bounding $\boldsymbol{\zeta}$** – Exact solution methods to this problem do not permit consideration of nonlinear constraints. Importantly, this includes bounds on the values of ζ, which must not be negative or excessively large.

[2] Note that this system does not depend on μ.

Problem 1 is actually trivial, as the diagonal values of ζ can simply be removed from the system before solving, then set to zero post-hoc, as in Appendix B: Python Code.

Problem 2 can also be solved easily using $\boldsymbol{z} = A^{+}\boldsymbol{b}$, where A^{+} denotes the pseudo-inverse of A:

$$A^{+} = A^{\mathsf{T}}\left(A\,A^{\mathsf{T}}\right)^{-1} \tag{6}$$

This approach additionally minimizes the L2-norm of $\boldsymbol{z}$, which provides the necessary constraints to find a unique solution directly. In this case, smaller values of ζ are also desirable, since high levels of turnover are not expected, and smaller values afford stability to the model.

Problem 3, however, unfortunately precludes the use of direct solution methods like $\boldsymbol{z} = A^{+}\boldsymbol{b}$, since bounds on x_i are not enforceable in these approaches. Still, many other (iterative) techniques for solving the system exist. In Appendix B: Python Code, the Limited-memory BFGS algorithm for bounded problems[3] is used to minimize $\mathcal{J}(\boldsymbol{z}) = ||A\boldsymbol{z} - \boldsymbol{b}||_2$, subject to bounds $l \leq z_i \leq u, \;\; \forall i$, providing a solution $\boldsymbol{z}$, which is reshaped to give ζ.

It should be noted that such optimizations should converge to zero, not just a small number, since the system is still underdetermined. Any other final values of $\mathcal{J}$ likely indicate convergence problems, and may result in model instability due to unbalanced transitions.

2.3 Additional Constraints

One advantage of the proposed framework is that it is simple to add additional constraints on ζ of the form $\boldsymbol{b}' = A'\,\boldsymbol{z}$; these constraints can simply be appended to $\boldsymbol{b}$ and the rows of A as in:

$$\begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{b}' \end{bmatrix} = \begin{bmatrix} A \\ A' \end{bmatrix} \tag{7}$$

Using the notation n_{ij} to represent the vectorized index ij so that $z_{n_{ij}} = \zeta_{ij}$, here are some examples:

- If one specific transition rate $\zeta_{ij} = r$ is known, append:

$$\begin{bmatrix} r \end{bmatrix} = \begin{bmatrix} e_1 & \cdots & e_{\mathrm{N}^2} \end{bmatrix} \tag{8}$$

  where

$$e_n = \begin{cases} 1 & n = n_{ij} \\ 0 & \text{else} \end{cases} \tag{9}$$

- If the average duration in a given activity class d_i is known, append:

$$\begin{bmatrix} {d_i}^{-1} - \mu \end{bmatrix} = \begin{bmatrix} e_1 & \cdots & e_{\mathrm{N}^2} \end{bmatrix} \tag{10}$$

  where

$$e_n = \begin{cases} 1 & n = n_{ij}, \;\; j \in [1, \ldots, \mathrm{N}] \\ 0 & \text{else} \end{cases}$$

  since

$$d_k = \left(\mu + \sum_j \zeta_{kj}\right)^{-1} \tag{11}$$

- If a crude control over the overall magnitude of turnover is desired (L1-norm of $\zeta = Z$), append:

$$\begin{bmatrix} Z \end{bmatrix} = \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix} \tag{12}$$

The necessary number of additional constraints to yield a fully determined system is $\mathrm{N} \times \mathrm{N} - 2\,\mathrm{N}$. While it is possible to construct such a system for a given N, most sets of constraints won't easily fill this gap for any N. For example, specifying the durations for all activity levels $\boldsymbol{d}$ yields N additional constraints. For $\mathrm{N} = 3$, this happily yields a unique solution, but for $\mathrm{N} = 4$, we will still have $(\mathrm{N} \times \mathrm{N} - 2\,\mathrm{N}) - \mathrm{N} = 4$ degrees of freedom, and so on. Still, this flexibility can certainly be useful to better match any available data.

[3] Courtesy of the `optimize.minimize` function from SciPy.

3 Conclusion

We have resolved a relationship between some typically available demographic parameters and rates of group turnover ζ which yield a particular steady-state distribution. This allows us to compute ζ on the fly, as shown in the `zetafun` function in Appendix B: Python Code. Additional linear constraints can be considered when solving for ζ, represented by the optional arguments `bprime` and `Aprime` to `zetafun`. Bounds on ζ can also be considered (e.g. `bounds`), depending on the method used to solve the system.

Finally, it should be noted that the analysis here has not considered disease-attributable death, a potentially serious limitation. Methods to ensure demographic stability in the face of significant disease-attributable death, or even if it is suitable to do so, should be the subject of future work.

Appendix A: Example Systems

Here we show examples of the system:

$$\nu(\boldsymbol{x} - \boldsymbol{p}) = A\,\boldsymbol{z} \tag{13}$$

for $\mathrm{N} = 2$ and $\mathrm{N} = 3$.

N = 2

$$\begin{bmatrix} \nu(x_1 - p_1) \\ \nu(x_2 - p_2) \end{bmatrix} = \begin{bmatrix} \cdot & -x_1 & x_2 & \cdot \\ \cdot & x_1 & -x_2 & \cdot \end{bmatrix} \begin{bmatrix} \zeta_{11} \\ \zeta_{12} \\ \zeta_{21} \\ \zeta_{22} \end{bmatrix} \tag{14}$$

N = 3

$$\begin{bmatrix} \nu(x_1 - p_1) \\ \nu(x_2 - p_2) \\ \nu(x_3 - p_3) \end{bmatrix} = \begin{bmatrix} \cdot & -x_1 & -x_1 & x_2 & \cdot & \cdot & x_3 & \cdot & \cdot \\ \cdot & x_1 & \cdot & -x_2 & \cdot & -x_2 & \cdot & x_3 & \cdot \\ \cdot & \cdot & x_1 & \cdot & \cdot & x_2 & -x_3 & -x_3 & \cdot \end{bmatrix} \begin{bmatrix} \zeta_{11} \\ \zeta_{12} \\ \zeta_{13} \\ \zeta_{21} \\ \zeta_{22} \\ \zeta_{23} \\ \zeta_{31} \\ \zeta_{32} \\ \zeta_{33} \end{bmatrix} \tag{15}$$

Appendix B: Python Code

```python
import numpy as np
import json
from scipy.optimize import minimize

def zetafun(ri,ro,xi,xo,bounds=None,bprime=None,Aprime=None):
  r"""Compute the turnover matrix to yield a steady-state population distribution

  Args:
    ri: entry rate
    ro: exit rate
    xi: entry population distrubution
    xo: target population distribution
    bprime: additional constraints LHS
    Aprime: additional rows of A (RHS)

  Returns:
    a matrix $\\zeta$
  """
  N = len(xo)
  # define the LHS
  b = ri*(xo-xi).T
  # vectors of indices for convenience
  iz = [(i,j) for i in range(N) for j in range(N) if i is not j]
  iv = [i*N+j for i in range(N) for j in range(N) if i is not j]
  # define the RHS system matrix
  A = np.array(\
    [ [ xo[zi[1]] if zi[0] == i else
       -xo[zi[1]] if zi[1] == i else 0
        for zi in iz
      ] for i in range(N)
    ])
  # append any additional constraints
  if (bprime is not None) and (Aprime is not None):
    b = np.concatenate(( b, np.atleast_1d(bprime) ), axis=0)
    A = np.concatenate(( A, np.atleast_2d(Aprime) ), axis=0)
  # solve the system
  jfun = lambda z: np.linalg.norm((np.dot(A,z)-b),2)
  z0   = np.dot(np.linalg.pinv(A),b)
  out  = minimize(jfun, z0, bounds = [bounds for zi in z0], method = 'L-BFGS-B')
  z    = out['x']
  if not out['success']:
    print('Warning: turnoverfun did not converge.')
  # return zeta as a matrix
  zeta = np.array(\
    [ [ z[iv.index(i*N+j)] if i*N+j in iv else 0
        for i in range(N)
      ] for j in range(N)
    ])
  return zeta

if __name__ == '__main__':
  nu =  0.05;  # births
  mu =  0.04;  # deaths
  xi = np.array([0.15, 0.20, 0.65]); # entry distribution
  xo = np.array([0.05, 0.10, 0.85]); # target distribution
  N  = len(xi)
  Z12 = 1.0
  A1  = [[1 for i in range(N**2-N)]]
  zeta = zetafun(nu,mu,xi,xo,(0.01,0.5),Z12,A1)
  with open('result.txt','w') as f:
    f.write(str(zeta))
```

Result:

```
[[0.         0.27509795 0.28726165]
 [0.14596417 0.         0.25638075]
 [0.01002537 0.02527011 0.        ]]
```