Incorporating time-to-death processes in age-structured prevalence models

Tim Riffe[1]

[1]Max Planck Institute for Demographic Research

February 20, 2016

1 Morbidity as a function of time-to-death

Some morbidity conditions vary as a function of time to death much more than as a function of age itself (Riffe, T. et al. 2015). For such conditions, the age-pattern (Sulivan curve) is itself a function of both mortality and the time-to-death pattern of the condition. That is to say, changing mortality will change the Sullivan curve independently of changes in the time-to-death process. As Hal mentioned, morbidity projections are often made assuming a fixed Sullivan curve. Such projections usually force improving mortality. When morbidity is assumed fixed, a projection based on the Sullivan curve will deviate non-trivially from a projection based on the time-to-death pattern of the same morbidity condition. It is straightforward to describe this phenomenon in continuous math, but it was my goal to implement the underlying calculations for the resulting health expectancies using matrices. I have used a combination of real matrix calculus and cheap hacks to attain this goal.

2 Method

Rather than embed the time-to-death process in a much-expanded transition matrix and matching reward matrix, I proceed by making the age pattern of prevalence a function of time-to-death, prior to creation of the rewards matrix. This is a cheap workaround for the sake of results.

Say we have a time to death pattern of prevalence, $g(y)$, something like this: Prevalence is essentially zero until very close to death, where it accelerates rapidly.

Matrix calculations proceed by manipulating an object created by what is possibly one of the coolest tricks I learned in Hal's course. Recall the matrix, $\mathbf{N}$, created by taking the inverse of the matrix that contains conditional

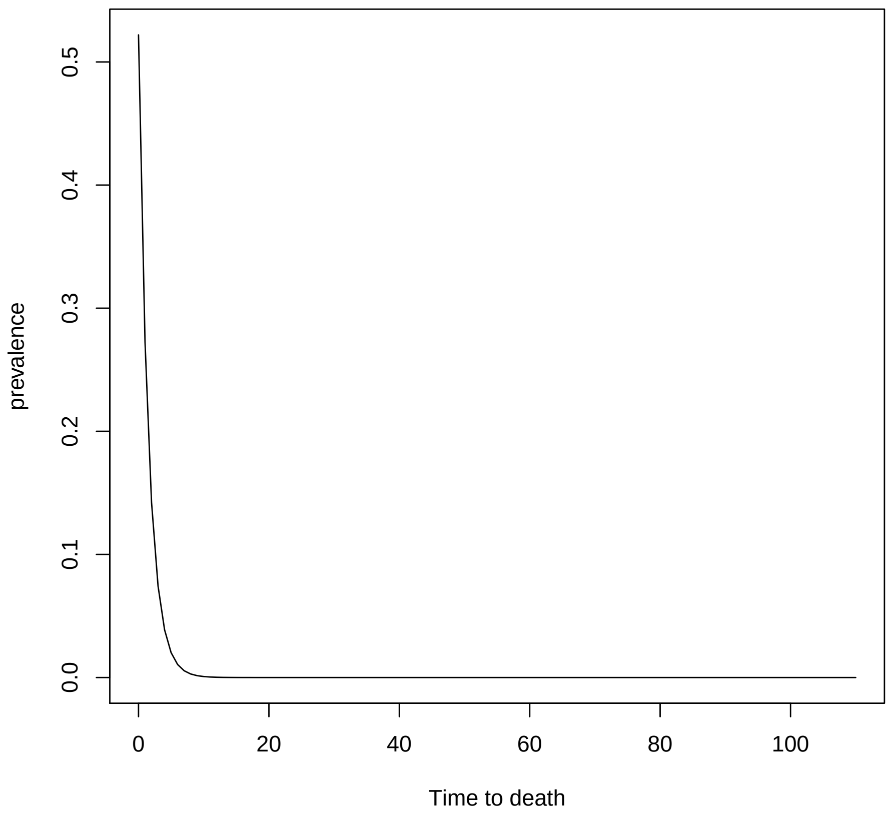

Figure 1: A fake time-to-death process

death probabilities in the subdiagonal. Understanding the demographic interpretation of **N** is key: Each column of **N** gives the conditional remaining survival curve, identical to $\frac{l(x+y)}{l(x)}$. That a matrix inverse can produce for us this useful and meaningful object is black magic to me, but we will exploit it.

We need to do two basic things to **N** in order to make progess: First, we need a matrix of the condtional remaining lifespan distribution, **D**, rather than the conditional survival curve. This is just negative the first row difference of **N**, or (roughly):

$$\mathbf{D}(m,i) = \mathbf{N}(m,i) - \mathbf{N}(m+1,i) \tag{1}$$

note, you need to append a row of zeros to **N** before this in order to close out the lifetable and ensure that the dimensions of **D** are identical to **N**. Since the subdiagonal of **N** was 1 before, and the upper triangle of **N** is populated with zeros, we get a -1 in the diagonal of **D**. We can erase this by taking the Hadamard product of **D** with the lower triangle matrix of **D** (not inclusive of the diagonal).

$$\mathcal{D} = \mathbf{D} \odot lower(\mathbf{D}) \tag{2}$$

You could get the same result by populating the upper triangle of **N** with 1s and then just taking minus the row differences...

Now we have **D**, where each column contains the conditional remaining lifetime distribution, or $\frac{d(x+y)}{l(x)}$, so we have the numbers we need, but they're in the the wrong places in the matrix for multiplying! Note, we have values descending down columns starting from the matrix diagonal. The diagonal is now interpreted as zero remaining years of life, and the subdiagonal means 1 remaining year of life, and so forth. We want all "zero remaining years" to be in the same row, and all the "one remaining year" to be in the next row, and so forth. That is to say the lower triangle needs to "lift" to the top of the matrix, leaving zeros below it. We'll call this matrix $\mathcal{D}^{up}$. This is not the same as a simple transpose, and it was the crux of this problem...

First take $vec(\mathcal{D})$, which is easy to program. Now define the vec^{-1} function, which let's us jump back to a matrix from a vector

$$vec^{-1}(vec(\mathcal{D})) = \mathcal{D} \tag{3}$$

Our goal is to create the matrix $\mathcal{D}^{up}$ from $\mathcal{D}$ via first turning $\mathcal{D}$ into a vector, then permuting its positions, then reforming to $\mathcal{D}^{up}$ via the vec^{-1} operator. To do this, imagine the commutation matrix, $Perm^{up}$, that moves between the two intermediate vectors (it's an interesting programming problem, but not worth describing in detail how to construct)

$$\mathcal{D}^{up} = vec^{-1}(vec(\mathcal{D}) \times Perm^{up}) \tag{4}$$

Now that we have $\mathcal{D}^{up}$, we can multiply this directly with $g(y)$ from the above example, which we now call $\mathbf{g}$, and it is $s+1$ elements long (with a zero tacked on to the end). Then the new prevalence vector, $\mathbf{p}$ is produced with:

$$\mathbf{p} = \mathbf{g} \times \mathcal{D}^{up} \tag{5}$$

One then procedes to make the rewards matrix as described in the course, and can calculate, business as usual. The above is not necessarily computationally efficient, due to the creation of the commutation matrix, $Perm^{up}$, but that is perhaps an area for improvement. For the time being it appears that any alternative implementation would also require a matrix of similar dimensions: If there are ω age classes, then $Perm^{up}$, or alternatively an expanded version of the transition and rewards matrices, implied ω^4 matrix elements. Even if sparse matrices are used, this is far from being computationally convenient. This works for now, but for this particular scenario matrix calculations do not seem to beat using conventional arithmetic to do the same thing. I don't rule out there being a more efficient way to do this, though.

References

Riffe, T., P. H. Chung, J. Spijker, and J. MacInnes (2015). Time-to-death patterns in markers of age and dependency. *MPIDR Working Papers WP-2015*(3), 25.