week08-00–population

March 3, 2025

0.1 Example: aging and population growth

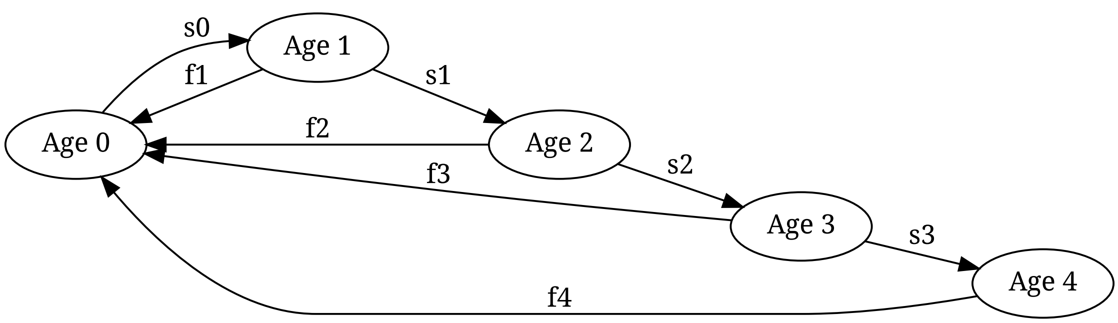
The nodes of the diagram will represent the age of an individual in a population.

The transitions corresponding to labels edges $s_i = (i \to i+1)$ represent probability of survival from age i to age $i+1$.

And the transitions $f_i : (i \to 0)$ represent probability of having an offspring at age i.

[1]:
```
from graphviz import Digraph
elev = Digraph()

pop = Digraph("pop")
pop.attr(rankdir='LR')

p = list(range(5))
with pop.subgraph() as c:
#    c.attr(rank='same')
    for i in p:
        c.node(f"Age {i}")

for i in p:
    if i+1 in p:
        pop.edge(f"Age {i}",f"Age {i+1}",f"s{i}")
    if i != 0:
        pop.edge(f"Age {i}","Age 0",f"f{i}")

pop
```

[1]:



0.2 Interpreting this model

- at time 0, suppose that the size of the population which is of age 0 is equal to p_0, the size of the population of age 1 is equal to p_1, etc.

  More succinctly, the population is described by the sequence $(p_0, p_1, \dots)$.

  Note that the total population is equal to the sum $\sum_{i=0}^{\infty} p_i$, which looks a bit odd! But, the infinite sum isn't really infinite – p_i must be equal to 0 for all sufficiently large values of i).

- at time 1, the size of the population of age 0 is given by

  $$f_1 p_1 + f_2 p_2 + \cdots = \sum_{i=1}^{\infty} f_i p_i.$$

  The size of the population of age 1 is given by the product $s_0 \cdot p_0$, and more generally for $i \geq 1$ the size of the population of age i is given by $s_{i-1} p_{i-1}$.

  Thus the population at time 1 is described by the sequence

  $$(\sum_{i=1}^{\infty} f_i p_i,\ s_0 p_0,\ s_1 p_1,\ \dots)$$

  And in particular the total population at time 1 is given by

  $$\sum_{i=1}^{\infty} f_i p_i + \sum_{j=0}^{\infty} s_j p_j.$$

- at time 2,

  it is easy to see that for $i > 1$, the size of the population of age i is equal to $s_{i-1} s_{i-2} p_{i-2}$

  The sizes of the populations having age 0 and 1 have a more complicated description!!

Given a "better" description of the population(s) at all times $t \geq 0$, we might hope to answer questions such as: "is the population decaying or growing?"

0.3 Matrix description

We are now going to give a more compact description of the preceding example, under an additional assumption.

Let's suppose that the lifespan of the populace is no more than 7 time units – i.e. we suppose that $s_7 = 0$.

Under this assumption, we can represent the population at time t by a vector

$$\mathbf{p}^{(t)} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_7 \end{bmatrix} \in \mathbb{R}^8$$

If the population at time t is described by $\mathbf{p}^{(t)} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_7 \end{bmatrix}$ then the population at time $t+1$ is given by

$$\mathbf{p}^{(t+1)} = \begin{bmatrix} \sum_{i=0}^{7} f_i p_i \\ s_0 p_0 \\ \vdots \\ s_6 p_6 \end{bmatrix} = A\mathbf{p}^{(t)}$$

where

$$A = \begin{bmatrix} f_0 & f_1 & f_2 & \cdots & f_6 & f_7 \\ s_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & s_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & s_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & s_6 & 0 \end{bmatrix}.$$

Thus if we begin with population $\mathbf{p}^{(0)} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_9 \end{bmatrix}$, then

$$\mathbf{p}^{(1)} = A\mathbf{p}^{(0)}$$

and

$$\mathbf{p}^{(2)} = A\mathbf{p}^{(1)} = A \cdot A \cdot \mathbf{p}^{(0)} = A^2\mathbf{p}^{(0)}$$

.

where A^2 denotes the $A \cdot A$, the *square* or *second power* of the matrix A.

In general, for $j \geq 0$ we have

$$\mathbf{p}^{(j)} = A^j\mathbf{p}^{(0)}$$

Thus computing the long-range behaviour of the system amounts to understanding the powers A^j of the 8×8 matrix A.

In particular, we find the total population at time t by computing

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \cdot A \cdot \mathbf{p}^{(t)}.$$

0.4 Case `fA,sA`

Let's compute several $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \cdot A^j$ for a particular A, namely when we make the following assumptions on the f_i and s_i:

```
fA = [.30,.50,.35,.25,.25,.15,.15,.5]
sA = [.30,.60,.55,.50,.30,.15,.05,0]
```

Remeber that for a given population vector $\mathbf{p}$, the resulting population at time j is given by $\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \cdot A^j \cdot \mathbf{p}$.

```
[13]: import numpy as np
      from pprint import pprint

      float_formatter = "{:.4f}".format
      np.set_printoptions(formatter={'float_kind':float_formatter})


      def sbv(index,size):
          return np.array([1.0 if i == index else 0.0 for i in range(size)])

      # can concatenate `numpy` arrays. For example

      l = np.array([1,2,3,4])

      B = np.array([np.zeros(4),np.zeros(4),np.ones(4)])

      l,B
```

```
[13]: (array([1, 2, 3, 4]),
       array([[0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000],
              [1.0000, 1.0000, 1.0000, 1.0000]]))
```

```
[10]: np.concatenate([[l],B])
```

```
[10]: array([[1.0000, 2.0000, 3.0000, 4.0000],
             [0.0000, 0.0000, 0.0000, 0.0000],
             [0.0000, 0.0000, 0.0000, 0.0000],
             [1.0000, 1.0000, 1.0000, 1.0000]])
```

```
[23]: fA = np.array([.30,.50,.35,.25,.25,.15,.15,.5])
      sA = np.array([.30,.60,.55,.50,.30,.15,.05])

      # we use numpy.linalg.matrix_power to compute powers of a matrix

      def onePowers(f,s,iter=20,skip=1):
          # create the "all ones vector" of the appropriate length
          ones = np.ones(len(f))

          # create the matrix `A` -- initial row is the vector `f`; subsequent rows␣
       ↪are multiples of
          # standard basis vectors

          A = np.concatenate([ [f], [ s[i]*sbv(i,len(f)) for i in range(len(sA))] ],␣
       ↪axis = 0)

          # computes the product of `ones` and the jth power of the matrix `A`
```

```python
    # returns the results as a `dictionary` with key `j` and value
    # `ones @ A^j`

    s ={ j : ones @ np.linalg.matrix_power(A,j)
         for j in range(0,iter,skip) }

    return s


onePowers(f=fA,s=sA,iter=35,skip=2)
```

[23]:
```
{0: array([1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000, 1.0000]),
 2: array([0.5100, 0.8400, 0.6225, 0.4250, 0.2400, 0.1200, 0.1150, 0.3000]),
 4: array([0.3100, 0.4498, 0.2779, 0.1830, 0.1294, 0.0745, 0.0735, 0.2025]),
 6: array([0.1649, 0.2395, 0.1580, 0.1069, 0.0743, 0.0427, 0.0419, 0.1140]),
 8: array([0.0896, 0.1306, 0.0856, 0.0573, 0.0396, 0.0228, 0.0223, 0.0607]),
 10: array([0.0486, 0.0708, 0.0463, 0.0311, 0.0215, 0.0124, 0.0121, 0.0330]),
 12: array([0.0264, 0.0384, 0.0252, 0.0169, 0.0117, 0.0067, 0.0066, 0.0179]),
 14: array([0.0143, 0.0208, 0.0136, 0.0092, 0.0063, 0.0036, 0.0036, 0.0097]),
 16: array([0.0078, 0.0113, 0.0074, 0.0050, 0.0034, 0.0020, 0.0019, 0.0053]),
 18: array([0.0042, 0.0061, 0.0040, 0.0027, 0.0019, 0.0011, 0.0011, 0.0029]),
 20: array([0.0023, 0.0033, 0.0022, 0.0015, 0.0010, 0.0006, 0.0006, 0.0016]),
 22: array([0.0012, 0.0018, 0.0012, 0.0008, 0.0005, 0.0003, 0.0003, 0.0008]),
 24: array([0.0007, 0.0010, 0.0006, 0.0004, 0.0003, 0.0002, 0.0002, 0.0005]),
 26: array([0.0004, 0.0005, 0.0003, 0.0002, 0.0002, 0.0001, 0.0001, 0.0002]),
 28: array([0.0002, 0.0003, 0.0002, 0.0001, 0.0001, 0.0001, 0.0000, 0.0001]),
 30: array([0.0001, 0.0002, 0.0001, 0.0001, 0.0000, 0.0000, 0.0000, 0.0001]),
 32: array([0.0001, 0.0001, 0.0001, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000]),
 34: array([0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000])}
```

The calculations above suggests that

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \cdot A^j = \mathbf{0} \quad \text{for} \quad j \geq 34.$$

More precisely, it suggests that the entries of `[1,1,...,1] @ A^j` are 0 to 4 decimal places for `j >= 20`.

Thus with the given matrix A, the model suggests that the total population will decay to 0.

[31]:
```python
def computePops(apowers,pop):
    # arguments: apowers should be a dictionary. keys: natural numbers `j`
    #                                            vals: powers of a matrix
    #             pop should be a population vector
    return {j: float(apowers[j] @ pop) for j in apowers.keys()}

p = 10.0*sbv(0,8)


pprint(computePops(onePowers(f=fA,s=sA,iter=50,skip=5),p))
```

```
{0: 10.0,
 5: 2.2797,
 10: 0.48649829174999987,
 15: 0.1054446047670881,
 20: 0.02286189433640572,
 25: 0.004956817513272183,
 30: 0.001074715966150294,
 35: 0.0002330153182348078,
 40: 5.0521384480990156e-05,
 45: 1.0953830457210062e-05}
```

1 Case fB,sB

Now let's consider different probabilities, as follows:

```
fB = [.50,.70,.55,.35,.35,.15,.15,.5]
sB = [.40,.70,.55,.50,.35,.15,.05]
```

[10]:
```
fB = [.50,.70,.55,.35,.35,.15,.15,.5]
sB = [.40,.70,.55,.50,.35,.15,.05,0]

pprint(onePowers(f=fB,s=sB,iter=20))
```

```
{0: array([1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00, 1.00]),
 1: array([1.20, 1.25, 1.05, 0.70, 0.50, 0.20, 0.15, 0.50]),
 2: array([1.33, 1.23, 0.91, 0.49, 0.44, 0.21, 0.18, 0.60]),
 3: array([1.30, 1.20, 0.96, 0.54, 0.49, 0.23, 0.20, 0.67]),
 4: array([1.32, 1.21, 0.96, 0.54, 0.49, 0.23, 0.20, 0.65]),
 5: array([1.34, 1.22, 0.97, 0.54, 0.49, 0.23, 0.20, 0.66]),
 6: array([1.35, 1.23, 0.98, 0.55, 0.50, 0.23, 0.20, 0.67]),
 7: array([1.36, 1.24, 0.99, 0.55, 0.50, 0.24, 0.20, 0.67]),
 8: array([1.37, 1.26, 1.00, 0.56, 0.51, 0.24, 0.20, 0.68]),
 9: array([1.39, 1.27, 1.01, 0.56, 0.51, 0.24, 0.21, 0.69]),
 10: array([1.40, 1.28, 1.02, 0.57, 0.52, 0.24, 0.21, 0.69]),
 11: array([1.41, 1.29, 1.03, 0.57, 0.52, 0.24, 0.21, 0.70]),
 12: array([1.42, 1.30, 1.04, 0.58, 0.53, 0.25, 0.21, 0.71]),
 13: array([1.44, 1.32, 1.05, 0.58, 0.53, 0.25, 0.21, 0.71]),
 14: array([1.45, 1.33, 1.06, 0.59, 0.54, 0.25, 0.22, 0.72]),
 15: array([1.46, 1.34, 1.07, 0.60, 0.54, 0.25, 0.22, 0.73]),
 16: array([1.48, 1.35, 1.08, 0.60, 0.55, 0.26, 0.22, 0.73]),
 17: array([1.49, 1.37, 1.09, 0.61, 0.55, 0.26, 0.22, 0.74]),
 18: array([1.51, 1.38, 1.10, 0.61, 0.56, 0.26, 0.22, 0.75]),
 19: array([1.52, 1.39, 1.11, 0.62, 0.56, 0.26, 0.23, 0.75])}
```

In this case, note that the first entry of the vector

$$\begin{bmatrix}1 & 1 & \cdots & 1\end{bmatrix} \cdot A^j$$

appears to be an increasing function of j.

Thus, for example we expect that given an initial population $\mathbf{p}^{(0)}$ with $p_0 > 0$, the total population is increasing as a function of j, rather than decaying.

[29]:
```
p = 10*sbv(1,8)
computePops(onePowers(f=fB,s=sB,iter=35,skip=2),p)
```

[29]:
```
{0: 10.0,
 2: 13.35,
 4: 13.210999999999999,
 6: 13.46822875,
 8: 13.72457581875,
 10: 13.982033212249998,
 12: 14.244391579877968,
 14: 14.511705408818932,
 16: 14.784034904839313,
 18: 15.061474708657759,
 20: 15.344121005426935,
 22: 15.632071494972397,
 24: 15.925425714260237,
 26: 16.224285070706642,
 28: 16.528752874710595,
 30: 16.838934375390473,
 32: 17.154936796988775,
 34: 17.47686937593729}
```

[ ]: