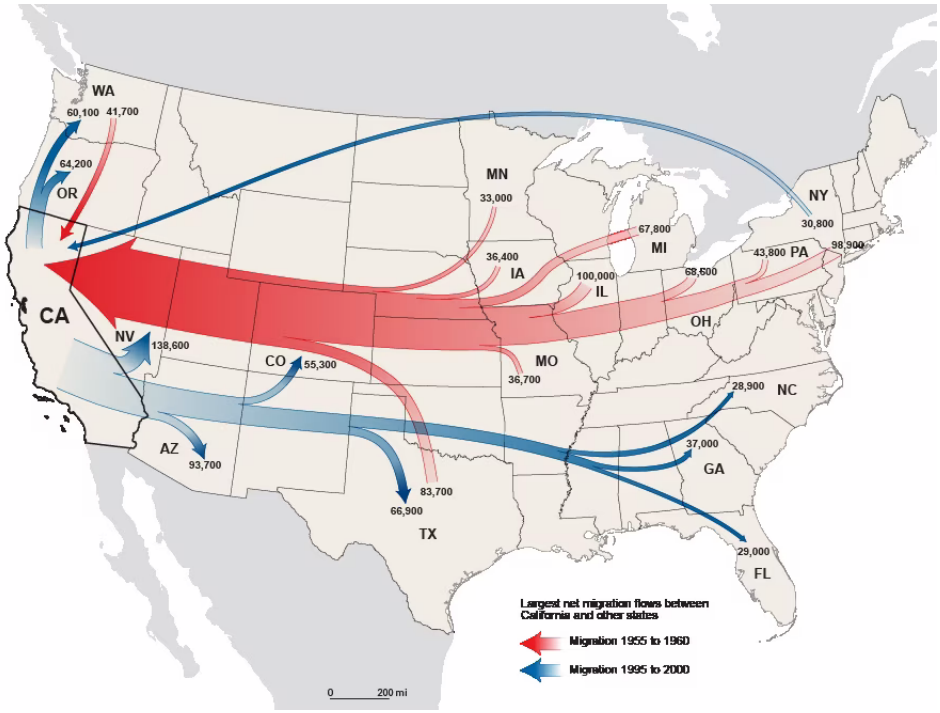
California: A Linear Model for Migration
Mathematical Programming with Python

MATH 2604: Advanced Scientific Computing 4
Spring 2025
Monday/Wednesday/Friday, 1:00-1:50pm

https://people.sc.fsu.edu/~jburkardt/classes/python_2025/california/california.pdf

---



Create and analyze a simple model of migration to California.

Linear Algebra

- *In a linear algebra problem, data is given as vectors such as x;*
- *A linear transformation of data is represented as a matrix, such as A;*
- *The transformation step is by multiplication: $y = A * x$;*
- *Repeated transformations involve $A^2, A^3, ..., A^k$;*
- *The structure of A can predict what will happen to x;*
- *Analysis of A and x may tell us whether x is likely to be constant, or grow, or decay;*
- `np.matmul()` *or* `np.dot()` *computes $A * x$;*
- `np.linalg.linsys()` *allows us to go backwards.*
- `np.linalg.eig()` *gives us the growth/decay patterns;*

1 California, Here We Come...or Go!

We consider a very simple problem that can lead to a linear system we can understand. Sometime around 1960, it was said that

> Every year, 30% of the population of California leaves the state, and 10% of the population of the other states moves to California.

This very odd statement seems like it can't be true. Does it mean that California is going to empty out? That the whole country will simply buzz with people moving back and forth? Let's see if we can get some information out of this claim.

We will assume that in 1960, there were 16 million people in California, and 180 million people in the US (and hence, 164 million US citizens who were not in California.)

So according to the statement, in 1961:

- 30% of 16 million people left California
- 10% of 164 milliion peple moved into California

which results in the California population jumping to 27,500,000 and the remaining non-California population dropping to 152,400,000.

Then in 1962:

- 30% of 27,500,000 left California
- 10% of 152,400,000 moveds into California

resulting in California's population jumping further to 34,560,000.

These jumps seem too huge and too fast, but let's not worry ao much about that. Let's see what happens if we assume this behavior continues for the next 20 years, and, of course, let's let Python do the work of computing, tabulating, and plotting this data!

2 A scalar system

Let `CA(y)` and `US(y)` represent the population of California, and the non-Californian US population, in the current year `y`. The statement suggests that, to estimate the populations for the next year, we can write

```
CA[y+1] = 0.70 CA[y] + 0.10 US[y]
US[y+1] = 0.30 CA[y] + 0.90 US[y]
```

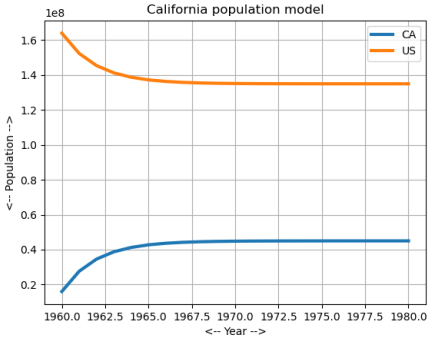
Now if we initialize these values, and put them in a loop, we can print a table of the evolving populations:

```
import numpy as np

n = 21
CA = np.zeros ( n )
US = np.zeros ( n )
for i in range ( 0, n ):
  if ( i == 0 ):
    CA[i] =  16000000
    US[i] = 164000000
  else:
    CA[i] = 0.70 CA[i-1] + 0.10 US[i-1]
    US[i] = 0.30 CA[i-1] + 0.90 US[i-1]
  print ( 'Year ', i, ' CA = ', CA[i], ' US = ', US[i] )
```

3 A Fixed Point Problem

A display of our results using `matplotlib()` may be surprising:



Despite the constant churning suggested by the statement, it turns out that, at least for this case, the populations seem to settle down to steady values.

Actually, now that we see this, we can even predict the exact values that the populations are tending to, by solving the following system, which represents a fixed point problem:

$$CA = 0.70\,CA + 0.10\,US$$
$$US = 0.30\,CA + 0.90\,US$$

and simplifying the first equation (or the second one!)

$$0.30\,CA = 0.10\,US$$
$$CA = \frac{1}{3}\,US$$

which means that, in the limit, California must contain 1/4 of the total US population:

$$CA = 180,000,000/4 = 45,000,000$$

with the other 135,000,000 citizens living elsewhere.

4 A Matrix version

We can rewrite the migration problem in matrix terms. Let `pop` be a vector containing the two values `CA` and `US`. Let `A` be the following matrix:

$$A = \begin{bmatrix} 0.70 & 0.10 \\ 0.30 & 0.90 \end{bmatrix}$$

Then we can write our yearly population update as

```
pop[i] = A * pop[i-1]
```

We get the same results, but the important thing is that, by thinking of this as a linear algebra problem, we suddenly acquire some tools that can let us ask and answer some interesting questions.

For instance, we can easily back up the calculation, that is, take the population data for year y, and compute from that the population data for the previous year. We do this by calling `np.linalg.solve()` to solve the linear system:

```
pop[i-1] = np.linalg.solve ( A, pop[i] )
```

so, for example, working from the 1961 data:

```
np.linalg.solve ( A, np.array ( [ 27,500,000, 152,400,000 ] ) )
```

returns the 1960 data [16,000,000, 180,000,000]. We will soon see some other advantages of the matrix approach.

The matrix version of our population code would be

```
import numpy as np

A = np.array ( [ \
  [ 0.70, 0.10 ], \
  [ 0.30, 0.90 ] ] )
n = 21
pop = np.zeros ( [ n, 2 ] )
for i in range ( 0, n ):
  if ( i == 0 ):
    pop[i,0] =  16000000
    pop[i,1] = 164000000
  else:
    pop[i,:] = np.matmul ( A, pop[i-1,:]
  print ( 'Year ', i, ' CA = ', pop[i,0], ' US = ', pop[i,1] )
```

5 Eigenvalues

When we repeatedly predict the next populations, we are writing

$$\text{pop[i]} = A * \text{pop[i-1]}$$

but using recursion we can rewrite this in terms of the initial vector `pop[0]`:

$$\text{pop[i]} = A^i * \text{pop[0]}$$

This means that it would be very helpful to understand what happens when we take high powers of the matrix A.

In fact, for our problem, we found that the populations tend towards a limiting value, so that

$$\text{pop[i]} = A * \text{pop[i-1]} \approx \text{pop[i-1]}$$

so that we have approximated a solution to an eigenvalue system corresponding to an eigenvalue $\lambda = 1$.

$$A * \text{pop[i-1]} \approx 1 \cdot \text{pop[i-1]}$$

We could have found out that A has an eigenvalue of 1, and hence, that there is a limiting, unchanging solution to the problem, by asking for the eigenvalues and eigenvectors. In Python, we can do this as follows:

```
import numpy as np

A = np.array ( [ \
  [ 0.70, 0.10], \
  [ 0.30, 0.90 ] ] )

eval,evec=np.linalg.eig(A)

print (eval)
[0.6 1. ]       # Eigenvalues are 0.6 and 1.0

print (evec)
[[-0.70710678 -0.31622777]   # Eigenvectors are columns
 [ 0.70710678 -0.9486833 ]]
```

Let's try to understant what this is saying. The first eigenvalue is 0.6, and the first eigenvector is the first two columns of `evec`, so when we multiply the eigenvector by `A`, here's what we expect:

```
A * [ -0.70710678,   = 0.6 * [ -0.70710678,   = [ -0.424264068,
       0.70710678 ]             0.70710678 ]       0.424264068 ]
```

while for the second eigensystem, we have

```
A * [ -0.31622777,   = 1.0 * [ -0.31622777,   = [ -0.31622777,
      -0.94868330]             -0.94868330 ]      -0.94868330 ]
```

Now the second eigenvector has the property that the first component is 1/3 the second. And any vector with this relationship will be preserved by multiplying by `A`.

For our population problem, it turns out that we can rewrite our starting point in terms of the two eigenvectors:

```
pop[0] = [  16,000,000   = [ -29,000,000   + [  45,000,000
           164,000,000 ]      29,000,000 ]     135,000,000 ]
```

and now it should be clear that repeatedly multiplying `pop[0]` by `A` is going to make the first component dwindle away, while the second component is unchanged, which is exactly the behavior we observed.

We are carrying out a version of what is known as the *power method*, in which a starting vector is multiplied again and again by a given matrix. In many cases, this procedure will drive the starting vector closer and closer to an eigenvector associated with an eigenvalue that is larger in magnitude than all the other eigenvalues of the matrix.

6 Stochastic Matrices

We see that both columns 1 and 2 of the matrix sum to 1.0. This means that for any vector x, the components of the result vector $Ax = A*x$ will be formed simply by rearranging the components of x in given proportions. So multiplying by A simply mixes up the entries of x. But there must be more, because we don't simply mix up the entries. If we repeatedly multiply by A, the sequence of results is tending towards a limiting value.

These observations all follow from the facts that A is a stochastic matrix (its columns each sum to 1) and all entries of A are strictly positive. These facts guarantee the "mixing" property of A, and the existence of a unique real eigenvalue of value 1, strictly larger in magnitude than all other eigenvalues. These facts, in turn, guarantee that the sequence $x, Ax, A^2 * x, ..., A^n * x...$ will converge to a multiple of this eigenvector, unless we have picked a starting vector that is completely orthogonal to the dominant eigenvector. (However, such a bad starting point would have to have at least one negative entry!).

See the Perron-Frobenius theorem for details.

7 Probability and a "Population" of 1

Let's go back to our migration problem, but now let's start with 1 person in California, and 0 elsewhere. After one time step, our population vector becomes:

$$\begin{bmatrix} 0.70 & 0.10 \\ 0.30 & 0.90 \end{bmatrix} \begin{bmatrix} 1.00 \\ 0.00 \end{bmatrix} = \begin{bmatrix} 0.70 \\ 0.30 \end{bmatrix}$$

Rather than worrying about splitting a single person into pieces, we can interpret this result probabilistically. After 1 time unit, there is a 70% chance that the person remains in California, and an 30% change of moving

out. If we repeat the multiplication process enough times, our result will tend towards the limit

$$p^* = \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix}$$

which we can now interpret to mean that, after a reasonable amount of time, the single person has a 25% chance of being in California, and a 75% chance of being elsewhere in the US.

We could use this approach to model the random migration of several people over time. More interestingly, a variation on the power method, with a probabilistic interpretation, is used to estimate the importance of all the web pages on the Internet, for the Google Page Rank algorithm. We will consider this idea in a later discussion.