Section 6.10.6
Diagonalization + Powers of A

Factorings of A

1. LU
2. QR
3. Diagonalization
4. SVD

$$A\mathbf{x}_1 = \lambda_1 \mathbf{x}_1$$
$$A\mathbf{x}_2 = \lambda_2 \mathbf{x}_2$$
$$\vdots$$
$$A\mathbf{x}_n = \lambda_n \mathbf{x}_n$$

$$AX = A\begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ \lambda_1\mathbf{x}_1 & \lambda_2\mathbf{x}_2 & \cdots & \lambda_n\mathbf{x}_n \\ | & | & & | \end{bmatrix}$$

$$= \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & & 0 \\ & \lambda_2 & & \\ & & \ddots & \\ 0 & & & \lambda_n \end{bmatrix}$$

$$= \begin{bmatrix} x_{1a}\lambda_1 & x_{2a}\lambda_2 & \cdots & \\ x_{1b}\lambda_1 & x_{2b}\lambda_2 & \cdots & \\ \vdots & & & \\ x_{1n}\lambda_n & x_{2n}\lambda_n & \cdots & \end{bmatrix}$$

Λ

Λ = capital Lambda

λ

$$AX = X\Lambda$$

$$\Lambda = X^{-1}AX$$

$$A = X\Lambda X^{-1}$$

Diagonalization factoring

$$A = \begin{bmatrix} a & 1 \\ 0 & a \end{bmatrix} \qquad c = (a-\lambda)(a-\lambda)$$

$$\lambda_1 = a \quad \lambda_2 = a$$

repeating λ

No Go X is Singular $\Rightarrow$ no X^{-1}

Symmetric Matrix

$X \Rightarrow Q$ eigenvectors

$$S = Q\Lambda Q^T$$

$$Q^{-1} = Q^T$$

Powers of A

A^k

$A \cdot A \cdot A \cdots A$ (k times)

$$\Lambda^2 = \begin{bmatrix} \lambda_1^2 & & 0 \\ & \lambda_2^2 & \\ 0 & & \ddots \\ & & & \lambda_n^2 \end{bmatrix}$$

$A \cdot A \sim O(n^3)$

$A^k \sim O((k-1)n^3)$

$$A^2 = X\Lambda X^{-1} X\Lambda X^{-1} = X\Lambda^2 X^{-1}$$

($X^{-1}X = I$)

$$A^3 = X\Lambda^2 X^{-1} X\Lambda X^{-1} = X\Lambda^3 X^{-1}$$

$$A^k = X\Lambda^k X^{-1}$$

6.10.6 Diagonalization and Powers of A

Here we will first establish a factoring of a matrix based on its eigenvalues and eigenvectors. This factoring is called *diagonalization*. Then we will use the factoring to find a computationally efficient way to compute powers of the matrix ($\boldsymbol{A}^k$). Then will extend this to $\boldsymbol{A}^k\,\boldsymbol{v}$, where $\boldsymbol{v}$ is a vector. This final result has direct application to difference equations, which are used to solve several types of problems.

6.10.6.1

Diagonalization

The eigenvalues and eigenvectors of the n-by-n matrix $\boldsymbol{A}$ give us n equations.

$$\begin{aligned}\boldsymbol{A}\,\boldsymbol{x_1} &= \lambda_1\,\boldsymbol{x_1}\\ \boldsymbol{A}\,\boldsymbol{x_2} &= \lambda_2\,\boldsymbol{x_2}\\ &\vdots\\ \boldsymbol{A}\,\boldsymbol{x_n} &= \lambda_n\,\boldsymbol{x_n}\end{aligned}$$

We wish to combine the n equations into one matrix equation.

Consider the product of $\boldsymbol{A}$ and a matrix called $\boldsymbol{X}$ containing the eigenvectors of $\boldsymbol{A}$ as the columns.

$$\begin{aligned}\boldsymbol{A}\,\boldsymbol{X} &= \boldsymbol{A}\begin{bmatrix} | & | & & | \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_n} \\ | & | & & | \end{bmatrix}\\ &= \begin{bmatrix} | & | & & | \\ \lambda_1\,\boldsymbol{x_1} & \lambda_2\,\boldsymbol{x_2} & \cdots & \lambda_n\,\boldsymbol{x_n} \\ | & | & & | \end{bmatrix}\\ &= \begin{bmatrix} | & | & & | \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_n} \\ | & | & & | \end{bmatrix}\begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}\\ &= \boldsymbol{X}\,\boldsymbol{\Lambda}\end{aligned}$$

The matrix $\boldsymbol{\Lambda}$ is a diagonal eigenvalue matrix. Λ = Capital Lambda

If the matrix has *Linearly Independent Eigenvectors*, which is sure to be the case when all eigenvalues are different (no repeating λs), then $\boldsymbol{X}$ is invertible.

$$\begin{aligned}\boldsymbol{A}\,\boldsymbol{X} &= \boldsymbol{X}\,\boldsymbol{\Lambda}\\ \boldsymbol{X}^{-1}\boldsymbol{A}\,\boldsymbol{X} &= \boldsymbol{\Lambda}\end{aligned}$$

$$\boxed{\boldsymbol{A} = \boldsymbol{X}\,\boldsymbol{\Lambda}\,\boldsymbol{X}^{-1}}$$

This factoring of $\boldsymbol{A}$ is called the **diagonalization** factoring.

When does Diagonalization not work?

Notice that for the diagonalization factoring to work we need to find the inverse of the eigenvector matrix. So each eigenvector *must be independent* of the other eigenvectors.

One example of a matrix with repeating eigenvectors is a 2-by-2 matrix with the same values on the forward diagonal and a zero on the backward diagonal. The determinant of the characteristic eigenvalue matrix for

$$\boldsymbol{A} = \begin{bmatrix} a & n \\ 0 & a \end{bmatrix} \text{ is } (a-\lambda)(a-\lambda).$$

Such a matrix has two eigenvalues of a, so it will also have repeating eigenvectors. Thus the eigenvector matrix is singular and may not be inverted.

```
>> A = [5 7; 0 5]
A =
     5     7
     0     5
>> [X, L] = eig(A)
X =
    1.0000   -1.0000
         0    0.0000
L =
     5     0
     0     5
>> rank(X)
ans =
     1
```

Conversely, when a matrix has distinct eigenvalues, it has *Linearly Independent Eigenvectors* and the matrix may be diagonalized.

Diagonalization of a Symmetric Matrix

Symmetric matrices have a simplified diagonalization because the matrix of eigenvectors of a symmetric matrix, $\boldsymbol{S}$, is orthogonal (orthonormal and square) (See *Eigenvalues and Eigenvectors of Symmetric Matrices*). Recall also from *Matrix Transpose Properties* that from the spectral theorem, orthogonal matrices have the property $\boldsymbol{Q}^T = \boldsymbol{Q}^{-1}$. Thus the diagonalization of a symmetric matrix is

$$\boxed{\boldsymbol{S} = \boldsymbol{Q}\,\boldsymbol{\Lambda}\,\boldsymbol{Q}^{\boldsymbol{T}}}.$$

Note: Recall that the columns of orthonormal matrices must be unit vectors (length of 1). That is required to use a transposed matrix instead of a calculated inverse matrix. MATLAB's `eig()`

function returns unit length columns when passed a numberic matrix, but not necessarily if a symbolic math matrix is passed. If you calculate the eigenvectors by hand, be sure to change them to unit vectors.

6.10.6.2

Powers of A

How does one compute matrix $\boldsymbol{A}$ raised to the power k ($\boldsymbol{A}^k$)?

The brute force approach is to simply multiply $\boldsymbol{A}$ by itself $k-1$ times. This could be slow if k and the size of the matrix (n) are large.

If $\boldsymbol{A}$ is a n-by-n square matrix, $\boldsymbol{A}\,\boldsymbol{A}$ requires n^3 multiply and addition operations ($\boldsymbol{O}(n^3)$ – *Big-O notation*). Thus $\boldsymbol{A}^k$ has complexity of $\boldsymbol{O}\left((k-1)n^3\right)$; or if we are clever about the order of the multiplications, it could be reduced to $\boldsymbol{O}\left(\log_2(k)n^3\right)$. That is:

$$\boldsymbol{A}^k = \underbrace{\underbrace{\underbrace{\boldsymbol{A}\,\boldsymbol{A}}_{\boldsymbol{A}^2}\;\boldsymbol{A}^2}_{\boldsymbol{A}^4}\;\boldsymbol{A}^4 \ldots \boldsymbol{A}^{k/2}}_{\boldsymbol{A}^k}$$

Fortunately, diagonalization gives us a faster way to compute $\boldsymbol{A}^k$ as a stand-alone matrix. The complexity of this method is $\boldsymbol{O}(n^3)$

$$\begin{aligned}\boldsymbol{A}^2 &= \boldsymbol{X}\,\boldsymbol{\Lambda}\,\boldsymbol{X}^{-1}\,\boldsymbol{X}\,\boldsymbol{\Lambda}\,\boldsymbol{X}^{-1} &= \boldsymbol{X}\,\boldsymbol{\Lambda}^2\,\boldsymbol{X}^{-1}\\ \boldsymbol{A}^3 &= \boldsymbol{X}\,\boldsymbol{\Lambda}^2\,\boldsymbol{X}^{-1}\,\boldsymbol{X}\,\boldsymbol{\Lambda}\,\boldsymbol{X}^{-1} &= \boldsymbol{X}\,\boldsymbol{\Lambda}^3\,\boldsymbol{X}^{-1}\end{aligned}$$

$$\boxed{\boldsymbol{A}^k = \boldsymbol{X}\,\boldsymbol{\Lambda}^k\,\boldsymbol{X}^{-1}}$$

Because the $\boldsymbol{\Lambda}$ matrix is diagonal, only the individual λ values need be raised to the k power (element-wise exponent).

$$\boldsymbol{\Lambda}^k = \begin{bmatrix}\lambda_1^k & 0 & \cdots & 0\\ 0 & \lambda_2^k & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \lambda_n^k\end{bmatrix}$$

Example Power of A

In this example, the matrix is symmetric, so the inverse simplifies to a transpose.

```
>> S = gallery('moler', 4)
S =
     1    -1    -1    -1
    -1     2     0     0
    -1     0     3     1
    -1     0     1     4
>> [Q, L] = eig(S)
Q =
    -0.8550    0.1910    0.3406   -0.3412
    -0.4348   -0.6421   -0.6219    0.1093
    -0.2356    0.6838   -0.4490    0.5248
    -0.1562   -0.2894    0.5437    0.7722
L =
    0.0334         0         0         0
         0    2.2974         0         0
         0         0    2.5477         0
         0         0         0    5.1215

>> Q * L.^5 * Q'
ans =
         425        -162        -639        -912
        -162         110         204         273
        -639         204        1022        1389
        -912         273        1389        2138
>> S^5
ans =
         425        -162        -639        -912
        -162         110         204         273
        -639         204        1022        1389
        -912         273        1389        2138
```

Orthogonal matrix

A few observations ...

- Considerable computation is saved by using the element-wise exponent operator on $\mathbf{\Lambda}$ (L in the MATLAB code). This gives the same result as a matrix exponent because $\mathbf{\Lambda}$ is a diagonal matrix.
- The eigenvalues and eigenvectors of a matrix are often complex. In the example above, I used a symmetric matrix because such will always have only real eigenvalues and eigenvectors. When a matrix has complex eigenvalues, the method still works. After multiplying the eigenvalues by the eigenvectors, the imaginary values may not be exactly zero due to round off errors, but they will be very small. Use the `real()` function to get only the real componet of the complex numbers.

6.10.6.3

Change of Basis

Application: difference equation

When we wish to multiply $\boldsymbol{A}^k$ by a vector ($\boldsymbol{A}^k\boldsymbol{v}$), we could diagonalize the matrix to find $\boldsymbol{A}^k$ and then multiply by the vector. But there is another approach that also lets us take advantage of the eigenvalue/eigenvector properties to accelerate the computation, and it also makes apparent the *steady state* (when k goes to infinity) of the system.

First we need to change the vector into a linear combination of the eigenvectors of $\boldsymbol{A}$.

$$\boldsymbol{v} = c_1\,\boldsymbol{x_1} + c_2\,\boldsymbol{x_2} + \cdots + c_n\,\boldsymbol{x_n},$$

This is called a *change of bases*, because we are using the eigenvectors as a new coordinate frame to describe the vector instead of the normal Cartesian bases vectors.

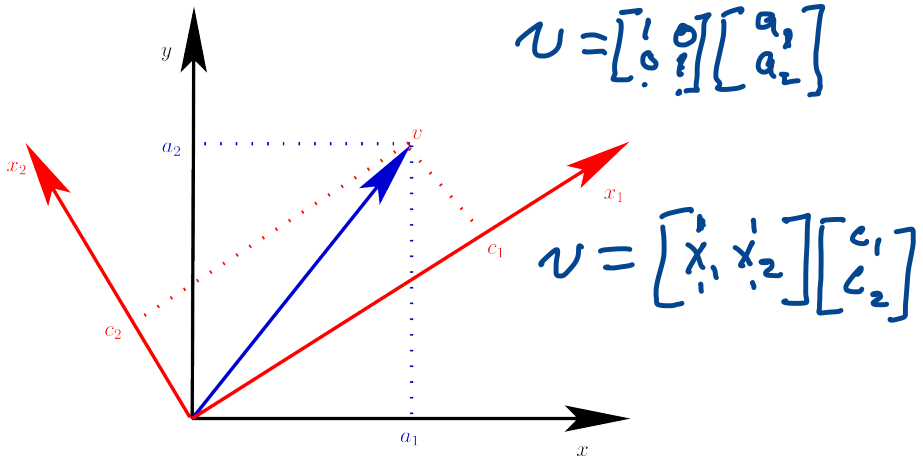


Fig. 6.10: Vector $\boldsymbol{v}$ is changed from coordinates (a_1, a_2) using the standard bases vectors to coordinates (c_1, c_2) using the eigenvectors of $\boldsymbol{A}$ as the bases vectors.

In matrix form, the vector $\boldsymbol{v}$ is described as $c = X^{-1}v$

$$\boldsymbol{v} = \boldsymbol{X}\,\boldsymbol{c} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_n} \\ | & | & & | \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}.$$

So, finding the linear coefficients is just a matter of solving a system of linear equations: $\boldsymbol{c} = \boldsymbol{X}^{-1}\,\boldsymbol{v}$, or in MATLAB: `c = X\v`.

Now, to multiply by powers of $\boldsymbol{A}$,

$$
\begin{aligned}
\boldsymbol{Av} &= \boldsymbol{AXc} \\
&= \boldsymbol{X\Lambda c} \\
&= \lambda_1 c_1 \boldsymbol{x_1} + \lambda_2 c_2 \boldsymbol{x_2} + \cdots + \lambda_n c_n \boldsymbol{x_n} \\
\boldsymbol{A^2 v} &= \boldsymbol{AX\Lambda c} \\
&= \boldsymbol{X\Lambda^2 c} \\
&= \lambda_1^2 c_1 \boldsymbol{x_1} + \lambda_2^2 c_2 \boldsymbol{x_2} + \cdots + \lambda_n^2 c_n \boldsymbol{x_n} \\
\boldsymbol{A^k v} &= \boldsymbol{AX\Lambda^{k-1} c} \\
&= \boldsymbol{X\Lambda^k c} \\
&= \lambda_1^k c_1 \boldsymbol{x_1} + \lambda_2^k c_2 \boldsymbol{x_2} + \cdots + \lambda_n^k c_n \boldsymbol{x_n}
\end{aligned}
$$

The matrix $\boldsymbol{\Lambda}$ is a diagonal eigenvalue matrix.

Let's illustrate this with an example. Consider the following matrix and its eigenvectors and eigenvalues.

```
>> A = [1 2; 5 4]
A =
     1     2
     5     4
>> [X, L] = eig(A)
X =
   -0.7071   -0.3714
    0.7071   -0.9285
L =
    -1     0
     0     6
```

Let us find $\boldsymbol{A}^k\boldsymbol{v}$ where $\boldsymbol{v}$ is a linear combination of the eigenvectors.

```
>> v = [3; 11]
v =
     3
    11
>> c = X\v
c =
     1
     2

>> A^2 * v
ans =
   143
   361
% now check with eigenvalues and eigenvectors
>> l1^2 * x1 + l2^2 * 2 * x2
ans =
```

```
143
361
```

So to generalize, we can say that when

$$\boldsymbol{v} = c_1\,\boldsymbol{x_1} + c_2\,\boldsymbol{x_2} + \cdots + c_n\,\boldsymbol{x_n},$$

then

$$\boldsymbol{A}^k\boldsymbol{v} = \lambda_1^k c_1\,\boldsymbol{x_1} + \lambda_2^k c_2\,\boldsymbol{x_2} + \cdots + \lambda_n^k c_n\,\boldsymbol{x_n}.$$

Eigenvalues and eigenvectors are often complex, but when the terms are added together, the result will be real (zero imaginary part).

Note also, that if $0 \leq \lambda_i < 1.0$, that terms goes to zero when k goes to infinity. Likewise, if any $\lambda_i > 1.0$, the result goes to infinity. To have a steady-state, non-zero result one eigenvalue must be 1.0 and $0 \leq \lambda_i \leq 1.0$ for the other terms.

Difference Equations

One of the main applications where you might want to compute the power of a matrix multiplied by a vector is for a difference equation, which are the discrete form of a differential equation.

We have a difference equation when the state of system at observation k is its state in the previous observation multiplied by a matrix. Note that the *state* of a system can be a set of any desired conditions of the system put into a vector, e.g., position, velocity, acceleration, or anything else that describes a property of the system at each observation.

$$\boldsymbol{u_k} = \boldsymbol{A}\,\boldsymbol{u_{k-1}}.$$

Let's consider the state of the system at the first observation as given by its initial condition.

$$\boldsymbol{u_1} = \boldsymbol{A}\,\boldsymbol{u_0}.$$

After the second observation ...

$$\boldsymbol{u_2} = \boldsymbol{A}\,\boldsymbol{u_1} = \boldsymbol{A}\,\boldsymbol{A}\,\boldsymbol{u_0} = \boldsymbol{A}^2\,\boldsymbol{u_0}.$$

After k observations ...

$$\boxed{\boldsymbol{u_k} = \boldsymbol{A}^k\,\boldsymbol{u_0}.}$$

A simple application of a difference equation is the Fibonacci sequence of numbers. This is the sequence of numbers starting with 0 and 1 and continuing where each number is the sum of the previous two numbers: {0, 1, 1, 2, 3, 5, 8, 13, 21, ... }: Fibonacci(k) = Fibonacci(k-1) + Fibonacci(k-2). This sequence of numbers occurs naturally in nature and is fodder for programming problems

using recursion and dynamic programming, but it can also be described as a difference equation, from which we can find a closed form solution using a change of basis.

The state of the system here is a vector with Fibonacci(k) and Fibonacci(k - 1).

$$Fibonacci(k) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} Fibonacci(k-1) \\ Fibonacci(k-2) \end{bmatrix}$$

$$Fibonacci(k) = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^k \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

As a class activity, let's use the change of basis strategy to compute Fibonacci(k) for any value of k.

Note: Using the quadratic equation, one can quickly see that the two eigenvalues are:

$$\frac{1+\sqrt{5}}{2} \text{ and } \frac{1-\sqrt{5}}{2}.$$

The algebra of the finding the eigenvectors and the change of basis coefficients gets a little messy, but you can find it here.

We will just let MATLAB compute it.

6.10.6.4

Application: Tracking Populations

We can use matrix – vector multiplication to track changes to groups of populations.

We use a vector $\boldsymbol{p}$ to represent the population of each group, and matrix $\boldsymbol{A}$ defines the changes to the group populations between sample times. If we sample once a year, then after one year, the population is:

$$\boldsymbol{p_1} = \boldsymbol{A}\,\boldsymbol{p_0}.$$

After the second year ...

$$\boldsymbol{p_2} = \boldsymbol{A}\,\boldsymbol{A}\,\boldsymbol{p_0} = \boldsymbol{A}^2\,\boldsymbol{p_0}.$$

After k years ...

$$\boldsymbol{p_k} = \boldsymbol{A}^k\,\boldsymbol{p_0}.$$

In this example, we look at wildlife populations. We divide animals into two groups. Animals such as foxes, coyotes, lions, and owls are predators – they eat other animals. Other animals such as rabbits, squirrels, and mice are vegetarian – they eat vegetation and are sometimes eaten by predators.

The model that I used for the populations is:

$$\boldsymbol{population_{k+1}} = \begin{bmatrix} 0.6 & p/3 \\ -3p & 1.2 \end{bmatrix} \begin{bmatrix} predators_k \\ vegetarians_k \end{bmatrix}$$

The predators are dependent on the vegetarian population to survive; whereas, the vegetarians would grow uncontrolled at 20 % per year without the predators keeping their populations in check. The variable p relates to the rate at which predators eat vegetarians. There is a 9-to-1 relationship between vegetarians being eaten and predators being sustained from hunting.

To find a value for p that allows the system to converge to constant non-zero values, I solved for finding an eigenvalue of 1.0 and found that, $p = \sqrt{0.08}$. Fortunately, the other eigenvalue was 0.8 (less than 1), so we have a stable system

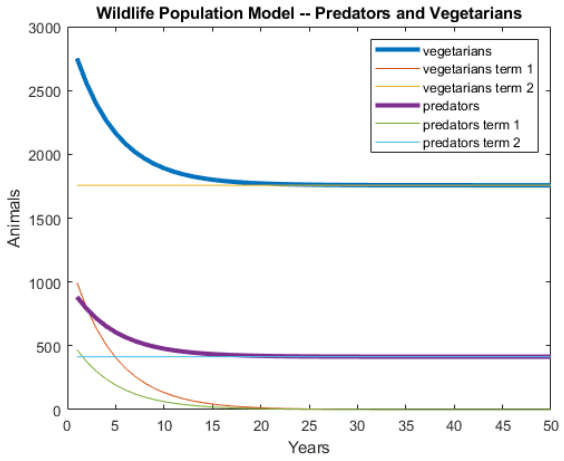
```
% Script to evaluate the population of predator and vegitarian wildlife
% over many years.

start = [1000; 3000];     % starting population [predators; vegetarians]

p = sqrt(0.08);            % found such that l(1) = 1.0
A = [ 0.6 p/3; -3*p 1.2];
[X, L] = eig(A);
l = diag(L);
c = X\start;  % coefficients for change of basis

population = zeros(2,50);
eigOne = zeros(2,50);
eigTwo = zeros(2,50);
for k = 1:50
    eigOne(:,k) = l(1)^k * c(1) * X(:,1);
    eigTwo(:,k) = l(2)^k * c(2) * X(:,2);
    population(:,k) = eigOne(:,k) + eigTwo(:,k);
end
pred1 = eigOne(1,:);
pred2 = eigTwo(1,:);
veg1 = eigOne(2,:);
veg2 = eigTwo(2,:);
predators = population(1,:);
vegetarians = population(2,:);
figure, plot(vegetarians, 'LineWidth', 3)
hold on
plot(veg1)
plot(veg2)
plot(predators, 'LineWidth', 3)
plot(pred1)
plot(pred2)
legend({'vegetarians', 'vegetarians term 1', 'vegetarians term 2', ...
    'predators',  'predators term 1', 'predators term 2'})
```

```
title('Wildlife Population Model -- Predators and Vegetarians');
xlabel('Years');
ylabel('Animals');
hold off
```

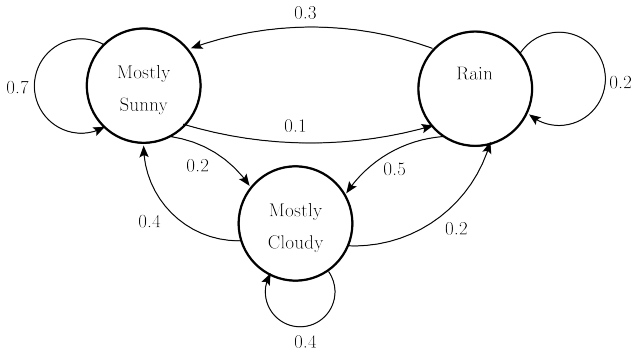


6.10.6.5

Application: Markov Chains

A Markov matrix is a stochastic (statistical) matrix showing probabilities of something transitioning from one state to another in one observation period. There are many applications where things can be thought of have one of a finite number of states. We often visualize state transition probabilities with a finite state machine (FSM) diagram. Then from a FSM, we can build a Markov matrix and make predictions about the future.

Here is a simple FSM to give odds for the weather on the next day given the current weather conditions.



We interpret the FSM diagram to be that if it is sunny today, there is a 70% chance that it will also be sunny tomorrow, 10% chance of rain tomorrow, and 20% percent chance that it will be cloudy tomorrow. Similarly, we can assign probabilities for tomorrow's weather if it is cloudy or raining today.

We can put the transition probabilities into a Markov matrix – named after the Russian mathematician Andrei Andreevich Markov (1856 - 1922). Since the values in the matrix are probabilities, each value is between zero and one. The columns represents the transition probabilities for starting in a given state. Thus the sum of each column must be one. The rows represent the probabilities for being in a state on the next observation.

The columns and rows are here ordered as sunny, cloudy, and rain.

$$\boldsymbol{M} = \begin{bmatrix} 0.7 & 0.4 & 0.3 \\ 0.2 & 0.4 & 0.5 \\ 0.1 & 0.2 & 0.2 \end{bmatrix}$$

Matrix multiplication tells us the weather forecast for tomorrow if it is cloudy today.

```
>> M = [0.7, 0.4, 0.3; 0.2, 0.4, 0.5; 0.1, 0.2, 0.2]
M =
    0.7000    0.4000    0.3000
    0.2000    0.4000    0.5000
    0.1000    0.2000    0.2000
>> M * [0; 1; 0]
ans =
    0.4000
    0.4000
    0.2000
```

Similar to what we saw in *Application: Tracking Populations*, we need to raise $\boldsymbol{M}$ to the k power to predict the weather in k days.

$$\boldsymbol{p_k} = \boldsymbol{M}^k \, \boldsymbol{p_0}.$$

So if it is cloudy today, we can expect the following on the day after tomorrow.

```
>> M^2 * [0; 1; 0]
ans =
    0.5000
    0.3400
    0.1600
```

We will always see something interesting when we look at the eigenvalues of a Markov matrix. One eigenvalue will always be one and the others will be less than one. As we saw before, this means that the forecast for the distant future converges and becomes independent to the current state of the system.

```
>> [X, L] = eig(M)
X =
    0.8529    0.7961    0.1813
    0.4713   -0.5551   -0.7801
    0.2245   -0.2410    0.5988
```

```
L =
    1.0000         0         0
         0    0.3303         0
         0         0   -0.0303
```

The steady state of the system only depends on the eigenvector corresponding to the eigenvalue of value one. The other terms will go to zero in future observations.

```
>> c_sun = X\[1; 0; 0];
>> c_clouds = X\[0; 1; 0];
>> c_rain = X\[0; 0; 1];
[redacted]
>> steady_state = X(:,1)*c_sun(1)
steady_state =
    0.5507
    0.3043
    0.1449
>> steady_state = X(:,1)*c_clouds(1)
steady_state =
    0.5507
    0.3043
    0.1449
>> steady_state = X(:,1)*c_rain(1)
steady_state =
    0.5507
    0.3043
    0.1449
```

We can plot it to see the convergence. The markers show the starting and ending of the forecast period.

```
% MarkovWeather.m

M = [0.7, 0.4, 0.3; 0.2, 0.4, 0.5; 0.1, 0.2, 0.2];
[X, L] = eig(M);
l = diag(L);
c_sun = X\[1; 0; 0];
c_clouds = X\[0; 1; 0];
c_rain = X\[0; 0; 1];
N = 20;
F_sun = zeros(3, N);
F_clouds = zeros(3, N);
F_rain = zeros(3, N);

for k = 1:N
    F_sun(:,k) = c_sun(1)*X(:,1) + c_sun(2)*l(2)^k*X(:,2) ...
                 + c_sun(3)*l(3)^k*X(:,3);
```

```
    F_clouds(:,k) = c_clouds(1)*X(:,1) + c_clouds(2)*l(2)^k*X(:,2) ...
                + c_clouds(3)*l(3)^k*X(:,3);
    F_rain(:,k) = c_rain(1)*X(:,1) + c_rain(2)*l(2)^k*X(:,2) ...
                + c_rain(3)*l(3)^k*X(:,3);
end

figure, plot3(F_sun(1,:), F_sun(2,:), F_sun(3,:))
hold on
plot3(F_clouds(1,:), F_clouds(2,:), F_clouds(3,:))
plot3(F_rain(1,:), F_rain(2,:), F_rain(3,:))
plot3(F_sun(1,1), F_sun(2,1), F_sun(3,1), 'r', 'Marker', 'o')
plot3(F_clouds(1,1), F_clouds(2,1), F_clouds(3,1), 'b', 'Marker', 'o')
plot3(F_rain(1,1), F_rain(2,1), F_rain(3,1), 'm', 'Marker', 'o')
plot3(F_sun(1,N), F_sun(2,N), F_sun(3,N), 'k', 'Marker', '*')
grid on
xlabel('sunny')
ylabel('cloudy')
zlabel('rain')
legend({'start sunny', 'start cloudy', 'start raining'})
hold off
```

Why Eigenvalue of One?

Markov matrices have an eigenvalue of one because the columns all add to one. Recall that we find the eigenvalues by solving for $det(\boldsymbol{A} - \boldsymbol{I}\lambda) = 0$, meaning that $\boldsymbol{A} - \boldsymbol{I}\lambda$ must be a singular matrix, which it is when $\lambda = 1$. Each row is -1 of the sum of the other rows.

```
>> M = [0.7, 0.4, 0.3; 0.2, 0.4, 0.5; 0.1, 0.2, 0.2];
>> rank(M)
ans =
     3
>> d = M - eye(3)
d =
   -0.3000    0.4000    0.3000
    0.2000   -0.6000    0.5000
    0.1000    0.2000   -0.8000
>> rank(d)
ans =
     2
```

Why All Eigenvalues Less Than or Equal to One?

First of all, it is impossible to have eigenvalues greater than one. From *Change of Basis*, we know that if any eigenvalue is > 1 then $\boldsymbol{M}^k p_0 \to \infty$ when $k \to \infty$, which implies that one or

more value in $\boldsymbol{M}$ is > 1. That is not possible for a stochastic matrix.

A more rigorous proof is easy to show for a 2-by-2 matrix, and is intuitive but difficult to formally verify for larger matrices.

Recall two eigenvalue properties from *Finding Eigenvectors and Eigenvalues*:

1. The trace of a matrix (sum of the diagonal values) is the sum of the eigenvalues.
2. The determinant of a matrix is the product of the eigenvalues.

Since all entries of a Markov matrix are probalilites, every entry must be ≤ 1, the identity matrix has the highest trace of any valid Markov matrix. For a n-by-n identity matrix, the trace is n, thus all of the eigenvalues of an identity matrix are one. All other valid Markov matrices have a trace less than n; therefore, the sum of the eigenvalues is $\leq n$. It is clear to see from a 2-by-2 matrix that all eigenvalues are ≤ 1 since one eigenvalue is equal to one.

$$trace\left(\begin{bmatrix} a & 1-b \\ 1-a & b \end{bmatrix}\right) = a + b \leq 2$$

$$1 + \lambda_2 \leq 2$$

$$\lambda_2 \leq 1$$

When Hadamard's inequality is applied to Markov matrices, we see that the maximum determinant of a Markov matrix is one, which is the case for the identity matrix. All other Markov matrices have a determinant less than one. Therefore, since the product of the eigenvalues is ≤ 1.

$$\begin{vmatrix} \text{a} & \text{1-b} \\ \text{1-a} & \text{b} \end{vmatrix} = a + b - 1 \leq 1$$

The only Markov matrix with determinant of one is the identity matrix. Since the product of the eigenvalues is equal to the determinant, $\lambda_1 \lambda_2 \leq 1$, and since $\lambda_1 = 1$, $\lambda_2 \leq 1$.