

Computational Physics

Topic 03 : Computational Problems involving Markov Chains

Lecture 01 : Review of Markov Chains

Dr Kieran Murphy

Department of Science, WIT.
(kmurphy@wit.ie)

Autumn Semester, 2020

Outline

- Some simple models
- Terminology and definitions

Example (Land of Oz)

Description

The Land of Oz is blessed by many things, but not by good weather. They never have two nice days in a row. If they have a nice day, they are just as likely to have snow as rain the next day. If they have snow or rain, they have an even chance of having the same the next day. If there is change from snow or rain, only half of the time is this a change to a nice day.

Example (Land of Oz)

Description

The Land of Oz is blessed by many things, but not by good weather. They never have two nice days in a row. If they have a nice day, they are just as likely to have snow as rain the next day. If they have snow or rain, they have an even chance of having the same the next day. If there is change from snow or rain, only half of the time is this a change to a nice day.

Formalisation ... as a transition matrix

$$\begin{array}{c} \vdots \\ \text{from } \vdots \end{array} \begin{array}{c} \text{Rain} \\ \text{Nice} \\ \text{Snow} \end{array} \begin{array}{c} \text{... to ...} \\ \begin{array}{ccc} \text{Rain} & \text{Nice} & \text{Snow} \end{array} \\ \left(\begin{array}{ccc} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{array} \right) \end{array}$$

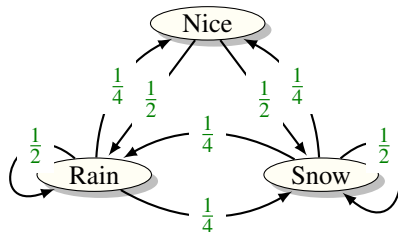
Example (Land of Oz)

Description

The Land of Oz is blessed by many things, but not by good weather. They never have two nice days in a row. If they have a nice day, they are just as likely to have snow as rain the next day. If they have snow or rain, they have an even chance of having the same the next day. If there is change from snow or rain, only half of the time is this a change to a nice day.

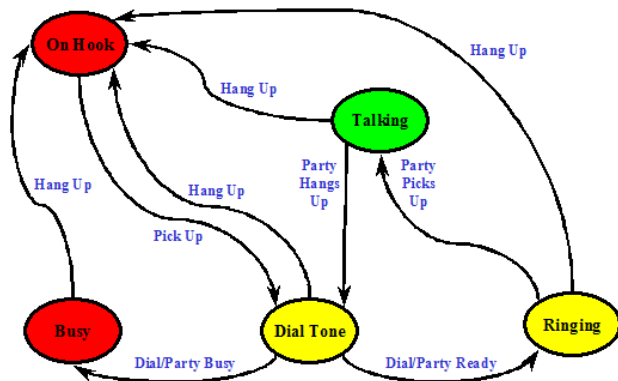
Formalisation ... as a transition matrix ... and finite state machine

		... to ...		
		Rain	Nice	Snow
from ...	Rain	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{4}$
	Nice	$\frac{1}{2}$	0	$\frac{1}{2}$
	Snow	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{2}$



Example (Model Based Testing)

Model-Based Testing* is the automatic generation of efficient test procedure sequences — using finite state machines or Markov chains – to represent a system's requirements and specified functionality.



*Google “Model Based Testing” or read Harry Robinson’s (Microsoft) paper
[http://www.ecs.csun.edu/~rlingard/COMP595VAV/GraphTheory Techniques In Model-Based Testing.pdf](http://www.ecs.csun.edu/~rlingard/COMP595VAV/GraphTheory%20Techniques%20In%20Model-Based%20Testing.pdf)

Example (Worker Employment Model)

I

Example 1

Consider a worker who, at any given month, t , is either unemployed (state 0) or employed (state 1). Suppose that, over a one month period:

- An unemployed worker finds a job with probability $\alpha \in (0, 1)$.
- An employed worker loses their job and becomes unemployed with probability $\beta \in (0, 1)$.

Formulate model as a Markov chain[†]. Then:

- 1 What is the average duration of unemployment?
- 2 Over the long-run, what fraction of time does a worker find themselves unemployed?
- 3 Conditional on employment, what is the probability of becoming unemployed at least once over the next 12 months?

[†]Since this model has only two states we could model this using the geometric distribution.

Specifying a Markov Chain

Any Markov process/chain can be described as follows:

- We have a set of **states**, $S = \{s_1, s_2, \dots, s_r\}$.

For the *Land of Oz* example the states were

$$S = \{\underbrace{\text{Rain}}_{s_1}, \underbrace{\text{Nice}}_{s_2}, \underbrace{\text{Snow}}_{s_3}\}$$

- The process starts (at **step/stage** 0) in one of these states and moves successively from one state to another, one **step/stage** at a time.
- Each move is called a **step**, and after n steps the process is at **stage** n , generating a **run**.

For the *Land of Oz* example some possible runs are:

- Rain, Rain, Rain, Nice, Snow, Rain, Snow, Snow, Nice, ...
- Rain, Rain, Rain, Rain, Rain, Rain, Rain, Rain, ...

while the following sequence is not possible (Why?)

- Snow, Snow Snow, Nice, Nice, Snow, Snow, Snow, ...

Specifying a Markov Chain

- If the chain is currently in state s_i , then it moves to state s_j at the next step/stage with a **transition probability** denoted by p_{ij} .

The probability does not depend upon which states the chain was in before the current state \implies The next state only depends on the current state and not the past states (Markov memory less property).

For the *Land of Oz* example the matrix of transition probabilities is

$$\begin{array}{c} \vdots \\ \text{from } \vdots \end{array} \begin{array}{c} \text{Rain} \\ \text{Nice} \\ \text{Snow} \end{array} \begin{array}{c} \dots \text{ to } \dots \\ \text{Rain} \quad \text{Nice} \quad \text{Snow} \end{array} \left(\begin{array}{ccc} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{array} \right) = P$$

- The Markov chain can remain in the state it is in, and this occurs with probability p_{ii} .

Specifying a Markov Chain

- The random variable X_n represents the state of the Markov chain at stage n . It assumes values $S = \{s_1, s_2, \dots, s_r\}$ with probability distribution

$$\mathbf{u}_n = (u_{n1}, u_{n2}, \dots, u_{nr})$$

- The Markov memoryless property can be expressed in terms of conditional probability as follows

$$Pr(X_{n+1} = s | X_n, X_{n-1}, \dots, X_0) = Pr(X_{n+1} = s | X_n)$$

(No extra uncertainty is removed by knowing what has happened in the past.)

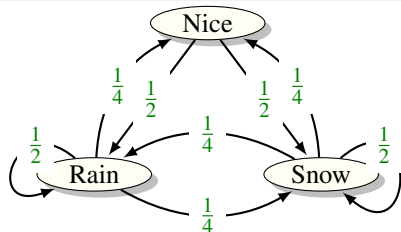
Of interest are:

- Given the initial state (i.e. given \mathbf{u}_0) what is the probability distribution for the state at some later stage n ?
- What is the long term behaviour of the chain?

Transition Matrix

$$P = \begin{matrix} & \begin{matrix} \text{Rain} & \text{Nice} & \text{Snow} \end{matrix} \\ \begin{matrix} \vdots \\ \text{from } \vdots \\ \vdots \end{matrix} & \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} \end{matrix}$$

... to ...



- From the total law of probability the **sum of the entries along every row equals one**. (Sum of probabilities of outgoing arcs on each node equals one).[‡]
- If given the probability distribution for some stage n , then the probability distribution for some later stage $n + m$, is

$$\mathbf{u}_{n+m} = \mathbf{u}_n P^m$$

In other words, to move the distribution forward m units of time, we post-multiply by P^m .

[‡]The sum down the columns need not be one (sum of probabilities of incoming arcs), but if the column totals are also one then this special type of process is called a **doubly stochastic** (more later).

Example (Land of Oz, cont)

Question

Today it is raining in the Land of Oz. What is the probability distribution for tomorrow?

Solution

Since $s_1 = \text{Rain}$, we have $\mathbf{u}_0 = (1, 0, 0)$ and we want \mathbf{u}_1 .

Using $\mathbf{u}_1 = \mathbf{u}_0 P$ we have

$$(1, 0, 0) \begin{pmatrix} 1/2 & 1/4 & 1/4 \\ 1/2 & 0 & 1/2 \\ 1/4 & 1/4 & 1/2 \end{pmatrix} = (1/2, 1/4, 1/4)$$

So the probability of rain tomorrow is 1/2, nice tomorrow is 1/4, and snow tomorrow is 1/4.

Example (Land of Oz, cont)

Question

Today it is nice in the Land of Oz. What is the probability of it raining in two days time?

Solution

Since $s_2 = \text{Nice}$, we have $\mathbf{u}_0 = (0, 1, 0)$ and we want the first component of \mathbf{u}_2 .

Using $\mathbf{u}_2 = \mathbf{u}_0 P^2$ we have

$$\begin{aligned} (0, 1, 0) \begin{pmatrix} 0.5 & 0.25 & 0.25 \\ 0.5 & 0 & 0.5 \\ 0.25 & 0.25 & 0.5 \end{pmatrix}^2 &= (0, 1, 0) \begin{pmatrix} 0.438 & 0.188 & 0.375 \\ 0.375 & 0.250 & 0.375 \\ 0.375 & 0.188 & 0.438 \end{pmatrix} \\ &= (0.375, 0.250, 0.375) \end{aligned}$$

So the probability of rain in two days time is 0.375, probability of being nice is 0.250 and probability of snow is 0.375.

Python Implementation

Python setup

setup.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from numpy.random import default_rng
4 rng = default_rng(12)
```

Define states

states.py

```
1 # label states for output
2 labels = np.array(["Rain", "Nice", "Show"])
3 states = np.arange(0, len(labels))
4 print("Labels:", labels)
5 print("States:", states)
```

```
Labels: [ 'Rain' 'Nice' 'Show' ]
States: [0 1 2]
```

Python Implementation

Define transition matrix, P

Transition probabilities:
 $\begin{bmatrix} 0.5 & 0.25 & 0.25 \\ 0.5 & 0. & 0.5 \\ 0.25 & 0.25 & 0.5 \end{bmatrix}$

```
1 # transition probability matrix
2 P = np.array([[1/2, 1/4, 1/4], [1/2, 0, 1/2], [1/4, 1/4, 1/2] ])
3 print("Transition probabilities:\n", P)
```

Compute distributions

prob_distribution .py

```
1 u0 = np.array([1,0,0])
2 print("Stage 0:", u0)
3 print("Stage 1:", np.dot(u0,P))
4
5 u0 = np.array([0,1,0])
6 u2 = np.dot( np.dot(u0,P), P)
7 print("\nStage 2:", u2)
8 u2 = np.dot(u0, np.linalg.matrix_power(P,2))
9 print("or directly by P squared:", u2)
```

Stage 0: [1 0 0]
 Stage 1: [0.5 0.25 0.25]

Stage 2: [0.375 0.25 0.375]
 or directly by P squared: [0.375 0.25 0.375]

Python Implementation

III

Rather than computing the probability distribution of each state we can simulate individual runs. To simplify things we write a generic function:

simulate_run.py

```

1 def simulate_run(t_n, P, s_0, rng):
2
3     # preallocate space for history
4     history = np.zeros(t_n+1, dtype=int)
5
6     # initial state
7     history[0] = s_0
8
9     for k in range(t_n):
10         p = P[history[k]]
11         history[k+1] = rng.choice(states, p=p)
12
13     return history
14
15 history = simulate_run(5, P, 0, default_rng(42) )
16 print(history)
17 print([labels[k] for k in history])

```

```

[0 2 1 2 2 0]
['Rain', 'Show', 'Nice', 'Show', 'Show', 'Rain']

```

Python Implementation

IV

Next we write a second generic function to generate a table of plots — each plot represent a single run of the Markov chain.

plot_runs.py

```
1 def plot_runs(t_n, P, s_0, labels, rng, filename=None, rows=3, cols=4):
2
3     fig, axs = plt.subplots(rows, cols, figsize=(10,4), sharey=True, sharex=True)
4
5
6     for ax in fig.axes:
7         history = simulate_run(t_n, P, s_0, rng)
8         ax.plot(history, "o-", linewidth=2)
9
10        plt.yticks(states, labels)
11    plt.suptitle(f"Sample runs ({t_n} stages) using initial state={labels[history[0]]}")
12
13    if filename is not None: plt.savefig(filename, bbox_inches="tight")
14
15    plt.show()
16
17 plot_runs(15, P, 0, labels, default_rng(667), filename="oz_runs.pdf")
```

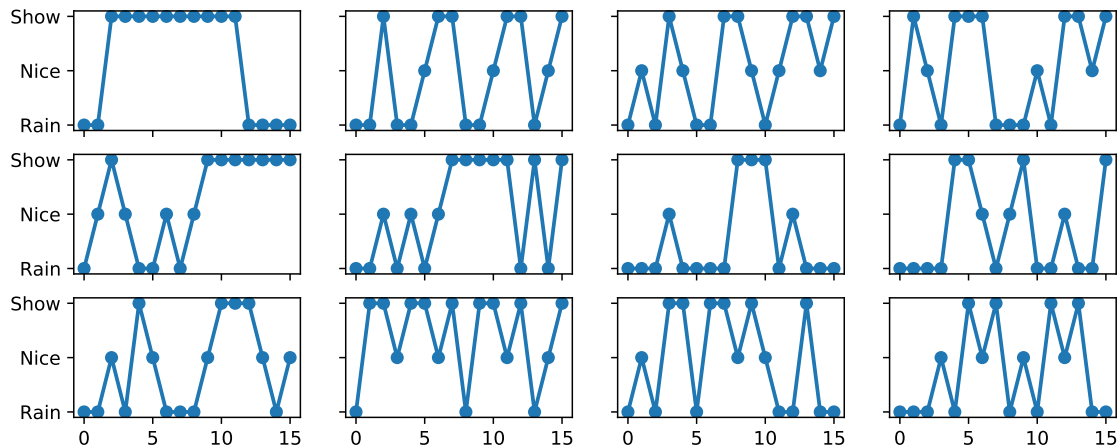

Now we can use our function `simulate_run` to generate a few sample runs — each run is called a **trajectory**.

runs.py

```
1 fig , axs = plt.subplots(3, 4, figsize=(10,4), sharey=True , sharex=True)
2
3 t_n = 15
4 rng = default_rng(667)
5
6 for ax in fig.axes:
7     history = simulate_run(t_n, P, 0, rng)
8     ax.plot(history , "o-", linewidth=2)
9
10 plt.yticks(states , labels)
11 plt.suptitle(f"Sample runs ({t_n} stages) using initial state={labels[history[0]]}")
12 plt.savefig("oz_runs.pdf", bbox_inches="tight")
13 plt.show()
```

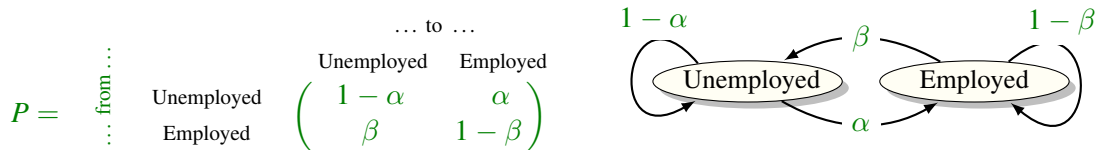
Land of Oz, Sample Runs

Sample runs (15 stages) using initial state=Rain



Worker Employment Model, Representation

Formulating as a Markov chain we have:



And in python (we have to pick a value for α and β , and typically we want to see the effect of different choices).

worker_model.py

```

1 labels = np.array(["Unemployed", "Employed"])
2 states = np.arange(0, len(labels))
3
4 alpha, beta = 0.1, 0.1
5 P = np.array([[1 - alpha, alpha], [beta, 1 - beta]])
  
```

Worker Employment Model, Implementation

Define model

`worker_model_simulate.py`

```
1 labels = np.array(["Unemployed", "Employed"])
2 states = np.arange(0, len(labels))
3
4 alpha, beta = 0.1, 0.01
5 P = np.array([[1-alpha, alpha], [beta, 1-beta]])
```

(I have picked the parameter values semi-randomly. They correspond to a 1/10 chance of getting a job in a month when unemployed and a 1/100 chance of losing their job in a month if employed.)

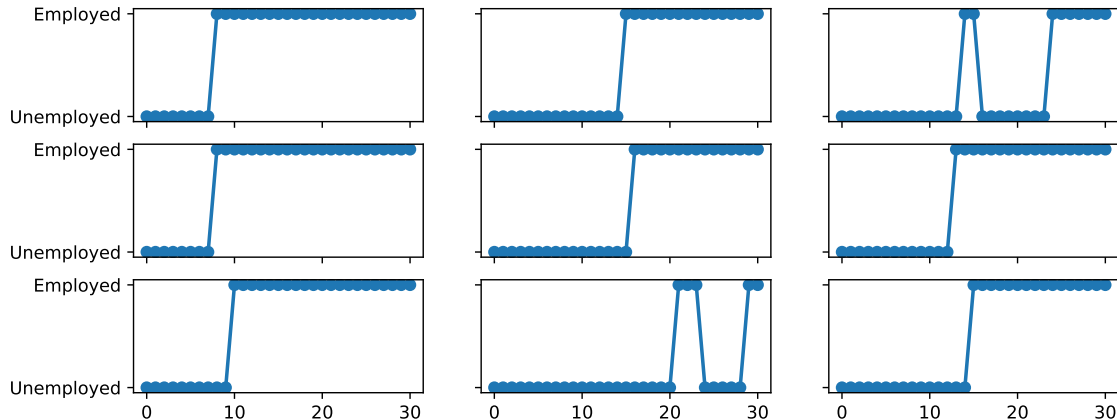
Generate plots of runs

`worker_runs.py`

```
1 plot_runs(30, P, 0, labels, default_rng(667), filename="worker_runs.pdf", cols=3)
```

Worker Employment Model, Sample Runs ($\alpha = 0.1$, $\beta = 0.01$)

Sample runs (30 stages) using initial state=Unemployed



I

worker_history.py

```
1 t_n = 100_000
2 rng = default_rng(667)
3 history = simulate_run(t_n, P, 0, rng)
4
5 print(history[:100])
```

```
[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1  
 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

We want to know how a unemployed worker remains unemployed, or an employed worker remains employed. In terms of the model output this corresponds to computing run lengths of zero and ones in the history. General procedure is to get the difference of successive values using the `np.diff` function. Then

- 1 corresponds to switching start from 'Employed' to 'Unemployed'.
- -1 corresponds to switching start from 'Unemployed' to 'Employed'.
- 0 corresponds to no change.

```
1 t_n = 100_000
2 rng = default_rng(667)
3 history = simulate_run(t_n, P, 0, rng)
4
5 print(history[:100])
```

[illegible]

```
1 diff = np.diff(history)
2 print(diff[:100])
```

[illegible]

Computing Run Lengths

`worker_history_run_starts.py`

```
1 run_starts, = np.where(diff == -1)
2 run_starts = np.hstack( ([1], run_starts))
3 print(run_starts[:10])
```

```
[ 1  75 233 276 530 670 871 1113 1287 1357]
```

`worker_history_run_ends.py`

```
1 run_ends, = np.where(diff == 1)
2 print(run_ends[:10])
```

```
[ 7  83 238 278 537 682 881 1115 1304 1358]
```

`worker_history_run_lens.py`

```
1 n = min(len(run_ends), len(run_starts))
2 run_lens = run_ends[:n] - run_starts[:n]
3 print(run_lens[:10])
```

```
[ 6  8  5  2  7 12 10  2 17  1]
```


Computing Run Lengths

IV

So to answer the questions

- What is the average duration of unemployment?

```
1 print(run_lens.mean())
```

worker_history_run_average_len.py

10.145922746781116

- Over the long-run, what fraction of time does a worker find themselves unemployed?

```
1 print((history==0).mean())
```

worker_history_pr_unemployed.py

0.0945790542094579

- Conditional on employment, what is the probability of becoming unemployed at least once over the next 12 months?

Left as exercise

Sensitivity Analysis for parameter α

I

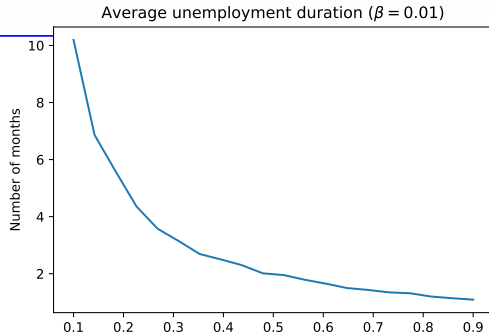
worker_pr_unemployed_wrt_alpha.py

```
1 t_max = 100_000
2 beta = 0.01
3 alphaValues = np.linspace(0.1, 0.9, 20)
4 y = []
5 for alpha in alphaValues:
6
7     P = np.array([[1-alpha, alpha], [beta, 1-beta] ])
8
9     history = simulate_run(t_n, P, 0, rng)
10
11     diff = np.diff(history)
12     run_starts, = np.where(diff == -1)
13     run_starts = np.hstack([1], run_starts))
14     run_ends, = np.where(diff == 1)
15     n = min(len(run_ends), len(run_starts))
16     run_lens = run_ends[:n] - run_starts[:n]
17     y.append(run_lens.mean())
```

Sensitivity Analysis for parameter α

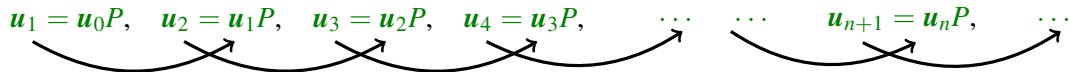
`worker_pr_unemployed_wrt_alpha_plot.py`

```
1 plt.plot(alphaValues , y)
2 plt.title("")
3 plt.title("Average unemployment duration ( $\beta=0.01$ )")
4 plt.ylabel("Number of months")
5 plt.xlabel("Probability of unemployed worker finding a job ,  $\alpha$ ")
6 plt.savefig("worker_pr_unemployed_wrt_alpha.pdf")
7 plt.show()
```



Equilibrium (Steady State) Distribution

Consider the situation where we are given some starting probability distribution for the state and iteratively calculate the probability distribution for each successive stage. i.e.

$$u_1 = u_0P, \quad u_2 = u_1P, \quad u_3 = u_2P, \quad u_4 = u_3P, \quad \dots \quad \dots \quad u_{n+1} = u_nP, \quad \dots$$


Q: What happens to the sequence of generated probability distributions?

A: Under conditions:

- all states of the Markov chain communicate with each other (i.e., it is possible to go from each state, possibly in more than one step, to every other state),
- the Markov chain is not periodic (a periodic Markov chain is a chain in which, e.g., you can only return to a state in an even number of steps),

it is possible to show that the sequence of generated probability distributions converge to a finite probability distribution called the **equilibrium distribution** or **steady state distribution**.

Equilibrium (Steady State) Distribution

Rather than going through the process of actually generating the sequence of probability distributions the **steady state distribution** can be calculated from the **balance equation**

$$\mu = \mu P \quad \text{subject to} \quad \sum_{i=1}^r \mu_i = 1 \quad (1)$$

- The equation is called a balance equation because it balances the probability of leaving and entering at each of the states. (This is analogous to the balance equations in the Network flow problems).
- The condition $\sum_{i=1}^r \mu_i = 1$ is required to ensure that the solution represents a probability distribution.

Example (Equilibrium distribution)

Question

What is the steady state probability distribution for the weather in the Land of Oz?

Solution

Solving $\mu = \mu P$ subject to $\sum \mu_i = 1$ we have on setting $\mu = (\mu_1, \mu_2, \mu_3)$.

$$\begin{array}{rcl}
 (\mu_1, \mu_2, \mu_3) = (\mu_1, \mu_2, \mu_3) \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{4} & \frac{1}{2} \end{pmatrix} & \implies & \begin{array}{l} 4\mu_1 = 2\mu_1 + 2\mu_2 + \mu_3 \\ 4\mu_2 = \mu_1 + \mu_3 \\ 4\mu_3 = \mu_1 + 2\mu_2 + 2\mu_3 \end{array} \\
 \sum \mu_i = 1 & \implies & 1 = \mu_1 + \mu_2 + \mu_3
 \end{array}$$

We now appear to have four equations in three unknowns, but one of the equations is redundant — This is called an **over-determined** linear system.

Example (Equilibrium distribution)

Simplifying we get

$$\begin{aligned}
 -2\mu_1 + 2\mu_2 + \mu_3 &= 0 \\
 \mu_1 - 4\mu_2 + \mu_3 &= 0 \\
 \mu_1 + 2\mu_2 - 2\mu_3 &= 0 \\
 \mu_1 + \mu_2 + \mu_3 &= 1
 \end{aligned}
 \Rightarrow
 \overbrace{\begin{pmatrix} -2 & 2 & 1 \\ 1 & -4 & 1 \\ 1 & 2 & -2 \\ 1 & 1 & 1 \end{pmatrix}}^A
 \overbrace{\begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix}}^{\mu}
 = \overbrace{\begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}}^b$$

$$\Rightarrow \mu_1 = \frac{2}{5}, \quad \mu_2 = \frac{1}{5}, \quad \mu_3 = \frac{2}{5}$$

Hence in the, long term, one-fifth of the days are nice and two-fifths are rain and two-fifths are snow.

For larger systems reproducing these step manually is painful. Instead we can use python, we can

- Construct the over-determined linear system $A\mu = b$ and solve for vector μ .
- Start with any initial distribution and iterate $\mathbf{u}_{n+1} = \mathbf{u}_n P$ until $\|\mathbf{u}_{n+1} - \mathbf{u}_n\|$ is small enough.

Python Implementation, solving linear system

equ_system.py

```

1 # define Markov chain
2 P = np.array( [ [1/2,1/4,1/4], [1/2,0,1/2], [1/4,1/4,1/2] ])
3 n = P.shape[0]
4
5 # build matrix (note the transpose)
6 A = np.vstack( (P.T - np.eye(n), np.ones(n)) )
7 print("A =", A)
8
9 # build RHS vector
10 b = np.hstack( (np.zeros(n), [1]) )
11 print("\nb = ", b)
12
13 # solve the over-determined system
14 x = np.linalg.lstsq(A, b, rcond=None)
15 print("\nx = ", x[0])
16
17 # verify that we have a distribution: sum mu = 1
18 print("\n", np.isclose(mu.sum(),1))
19 # verify that we have a steady state distribution: mu = mu P
20 print(np.isclose( np.dot(mu, P), mu))

```

```

A = [[-0.5   0.5   0.25]
      [ 0.25 -1.    0.25]
      [ 0.25  0.5  -0.5 ]
      [ 1.    1.    1.   ]]

```

```
b = [0. 0. 0. 1.]
```

```
x = [0.4 0.2 0.4]
```

```

True
[ True  True  True]

```


Python Implementation, iterate system

equ_iterate .py

```
1 tol = 1E-5
2
3 # start with uniform distribution
4 u = np.ones(n) / n
5
6 # iterate till convergence
7 for k in range(100):
8     u_old = u
9     u = np.dot(u_old, P)
10    if np.linalg.norm(u - u_old) < tol: break
11 else:
12     print("Poor/no convergence. Exceeded max iterations")
13
14 # output result
15 print(u)
```

[0.40000025 0.19999949 0.40000025]

Class Structure

The state space of a Markov chain can be partitioned into a set of non-overlapping **communicating classes**.

- States i and j are in the same communicating class if there is some way of getting from state i to state j , AND there is some way of getting from state j to state i .
- It needn't be possible to get between i and j in a single step, but it must be possible over some number of steps to travel between them in both ways.
- We write $i \leftrightarrow j$ if states i and j are in same communicating class.
- Every state is a member of exactly one communication class.

Definition 2 (Communicating Class)

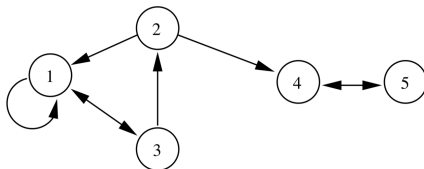
Consider a Markov chain with state space S and transition matrix P , and consider states $i, j \in S$. Then state i communicates with state j if:

- there exists some iteration, $t = u$, such that $(P^u)_{ij} > 0$, AND
- there exists some iteration, $t = v$, u such that $(P^v)_{ji} > 0$.

Example

The communicating class associated with the following transition diagram are:

$\{1, 2, 3\}, \quad \{4, 5\}$



State 2 leads to state 4, but state 4 does not lead back to state 2, so they are in different communicating classes.

Definition 3

A communicating class of states is **closed** if it is not possible to leave that class. That is, the communicating class C is closed if $p_{ij} = 0$ whenever $i \in C$ and $j \notin C$.

In the transition diagram above

- Class $\{1, 2, 3\}$ is not closed: it is possible to escape to class $\{4, 5\}$.
- Class $\{4, 5\}$ is closed: it is not possible to escape.

Absorbing States and Irreducible Models

A closed class consisting of exactly one element is often of interest:

Definition 4 (Absorbing State)

A state, i , is said to be **absorbing** if the set $\{i\}$ is a closed class.

Markov chains whose states are in single communicating class are easier to analyse — in particular they have a unique equilibrium distribution.

Definition 5

A Markov chain, with transition matrix, P , is said to be **irreducible** if $i \leftrightarrow j$ for all $i, j \in S$. That is, the chain is irreducible if the state space S is a single communicating class.

Applications/Exercises

Example 6

A process moves on the integers 1, 2, 3, 4, and 5. It starts at 1 and, on each successive step, moves to an integer greater than its present position, moving with equal probability to each of the remaining larger integers. State five is an absorbing state. Find the expected number of steps to reach state five.

Example 7

Smith is in jail and has 3 dollars; he can get out on bail if he has 8 dollars. A guard agrees to make a series of bets with him. If Smith bets A dollars, he wins A dollars with probability 0.4 and loses A dollars with probability 0.6. Find the probability that he wins 8 dollars before losing all of his money if

- he bets 1 dollar each time (timid strategy).
- he bets, each time, as much as possible but not more than necessary to bring his fortune up to 8 dollars (bold strategy).

Which strategy gives Smith the better chance of getting out of jail?