22. Two-Dimensional Arrays

Topics

Motivation
The numpy Module
Subscripting functions and 2d Arrays
GooglePage Rank
Visualizing

A 2D array has rows and columns.

This one has 3 rows and 4 columns.

We say it is a “3-by-4” array (a.k.a matrix)

Can have a 2d array of strings or objects.

But we will just deal with 2d arrays of numbers.
### Rows and Columns

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>17</td>
<td>49</td>
<td>61</td>
</tr>
<tr>
<td>38</td>
<td>18</td>
<td>82</td>
<td>77</td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td>12</td>
<td>10</td>
</tr>
</tbody>
</table>

This is row 1.
## Rows and Columns

This is column 2.
Entries

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>17</td>
<td>49</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>18</td>
<td>82</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>53</td>
<td>12</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

This is the (1,2) entry.
Where Do They Come From?

Entry \((i,j)\) is the distance from city \(i\) to city \(j\)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Amsterdam</td>
<td>Berlin</td>
<td>Bordeaux</td>
<td>Brussels</td>
<td>Copenhagen</td>
<td>Dublin</td>
<td>Lisbon</td>
<td>London</td>
<td>Madrid</td>
<td>Milan</td>
<td>Munich</td>
<td>Paris</td>
<td>Rome</td>
<td>Zurich</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>650.594</td>
<td>1084.367</td>
<td>204.7</td>
<td>766.456</td>
<td>945.404</td>
<td>2254.519</td>
<td>476.014</td>
<td>1763.664</td>
<td>1071.746</td>
<td>820.138</td>
<td>503.852</td>
<td>1657.65</td>
<td>818.734</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>651.304</td>
<td>0</td>
<td>1634.132</td>
<td>764.797</td>
<td>379.35</td>
<td>1506.491</td>
<td>2904.264</td>
<td>1036.101</td>
<td>2333.429</td>
<td>1033.506</td>
<td>502.556</td>
<td>1053.617</td>
<td>1513.741</td>
<td>644.044</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1064.547</td>
<td>1630.51</td>
<td>0</td>
<td>890.135</td>
<td>1785.177</td>
<td>1444.887</td>
<td>1174.082</td>
<td>975.717</td>
<td>703.237</td>
<td>1018.437</td>
<td>1284.774</td>
<td>562.938</td>
<td>1508.036</td>
<td>1021.895</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>207.37</td>
<td>767.381</td>
<td>891.026</td>
<td>0</td>
<td>908.03</td>
<td>775.414</td>
<td>2061.177</td>
<td>306.244</td>
<td>1590.322</td>
<td>881.246</td>
<td>784.539</td>
<td>310.61</td>
<td>1467.06</td>
<td>628.274</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>768.376</td>
<td>381.155</td>
<td>1785.684</td>
<td>906.197</td>
<td>0</td>
<td>1645.681</td>
<td>2956.016</td>
<td>1177.511</td>
<td>2465.161</td>
<td>1414.722</td>
<td>1080.551</td>
<td>1205.349</td>
<td>2111.726</td>
<td>1186.559</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>539.79</td>
<td>1499.75</td>
<td>1439.475</td>
<td>769.049</td>
<td>1640.41</td>
<td>0</td>
<td>2509.627</td>
<td>453.606</td>
<td>2139.772</td>
<td>1541.326</td>
<td>1554.039</td>
<td>683.562</td>
<td>2227.14</td>
<td>1300.364</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2261.111</td>
<td>2757.07</td>
<td>1171.614</td>
<td>2066.689</td>
<td>2951.741</td>
<td>2611.451</td>
<td>0</td>
<td>2142.281</td>
<td>626.066</td>
<td>2150.158</td>
<td>2448.668</td>
<td>1748.502</td>
<td>2535.253</td>
<td>2185.753</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>478.973</td>
<td>1038.94</td>
<td>978.688</td>
<td>308.242</td>
<td>1179.603</td>
<td>465.078</td>
<td>2148.821</td>
<td>0</td>
<td>1677.966</td>
<td>1180.519</td>
<td>1094.131</td>
<td>402.746</td>
<td>1766.323</td>
<td>927.559</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1762.485</td>
<td>2326.44</td>
<td>702.088</td>
<td>1508.073</td>
<td>2403.115</td>
<td>2144.045</td>
<td>525.152</td>
<td>1673.555</td>
<td>0</td>
<td>1591.506</td>
<td>1978.157</td>
<td>1200.076</td>
<td>1366.603</td>
<td>1669.123</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1074.277</td>
<td>1035.63</td>
<td>1019.438</td>
<td>905.351</td>
<td>1415.052</td>
<td>1672.432</td>
<td>2152.653</td>
<td>1202.042</td>
<td>1580.336</td>
<td>0</td>
<td>492.726</td>
<td>847.819</td>
<td>594.634</td>
<td>279.253</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>622.285</td>
<td>562.946</td>
<td>1262.385</td>
<td>783.436</td>
<td>1078.905</td>
<td>1564.472</td>
<td>2450.067</td>
<td>1091.302</td>
<td>1576.362</td>
<td>490.958</td>
<td>0</td>
<td>626.256</td>
<td>929.666</td>
<td>314.143</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>602.799</td>
<td>1048.47</td>
<td>583.226</td>
<td>308.387</td>
<td>1203.429</td>
<td>669.622</td>
<td>1573.377</td>
<td>400.452</td>
<td>1282.622</td>
<td>949.469</td>
<td>830.414</td>
<td>0</td>
<td>1413.096</td>
<td>663.608</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1600.357</td>
<td>1514.24</td>
<td>1509.625</td>
<td>1492.011</td>
<td>1976.329</td>
<td>2257.272</td>
<td>2540.624</td>
<td>1708.102</td>
<td>1936.207</td>
<td>596.54</td>
<td>930.032</td>
<td>1431.299</td>
<td>0</td>
<td>865.323</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>621.854</td>
<td>845.704</td>
<td>1021.629</td>
<td>653.216</td>
<td>1166.023</td>
<td>1419.699</td>
<td>2189.521</td>
<td>949.309</td>
<td>1669.309</td>
<td>279.652</td>
<td>315.154</td>
<td>653.299</td>
<td>865.456</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Where Do they Come From?

Entry \((i,j)\) is 1 if node \(i\) is connected to node \(j\) and is 0 otherwise.

Captures the connectivity in a network.

Nodes 4 and 6 are connected.
Where Do They Come From

An m-by-n array of pixels.

Each pixel encodes 3 numbers: a red value, a green value, a blue value.

So all the information can be encoded in three 2D arrays.
2d Arrays in Python

A = 
\[
\begin{bmatrix}
12 & 17 & 49 & 61 \\
38 & 18 & 82 & 77 \\
83 & 53 & 12 & 10 \\
\end{bmatrix}
\]

A list of lists.
Accessing Entries

\[
A = \begin{bmatrix}
12 & 17 & 49 & 61 \\
38 & 18 & 82 & 77 \\
83 & 53 & 12 & 10 \\
\end{bmatrix}
\]

\[A[1][2]\]

\[
A = \begin{bmatrix}
[12, 17, 49, 61], [38, 18, 82, 77], [83, 53, 12, 10] \\
\end{bmatrix}
\]
**Accessing Entries**

\[
A = \begin{bmatrix}
12 & 17 & 49 & 61 \\
38 & 18 & 82 & 77 \\
83 & 53 & 12 & 10 \\
\end{bmatrix}
\]

A[2][1] = 38
Setting Up 2D Arrays

Here is a function that returns a reference to an m-by-n array of zeros:

```python
def zeros(m,n):
    v = []
    for k in range(n):
        v.append(0.0)
    A = []
    for k in range(m):
        A.append(v)
    return A
```
Setting Up 2D Arrays

Here is a function that returns a reference to an m-by-n array of zeros:

```python
def zeros(m, n):
    v = [0 for k in range(n)]
    A = [v for k in range(m)]
    return A
```

This implementation uses list “comprehensions”.
Python is Awkward

Turns out that base Python is not very handy for 2D array manipulations.

The numpy module makes up for this.

We will learn just enough numpy so that we can do elementary plotting, image processing and other things.
Introduction to numpy

A few essentials illustrated by examples.
Setting up a 2D Array of 0’s

```python
>>> from numpy import *
>>> m = 3
>>> n = 4
>>> A = zeros((m,n))
>>> A
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

Note how the row and column dimensions are passed to zeros
Accessing an Entry

```python
>>> A = zeros((3,2))
>>> A[2,1] = 10
>>> A
array([[  0.,   0.],
       [  0.,   0.],
       [  0.,  10.]])
```

Accessing an Entry

>>> A = array([[1,2,3],[4,5,6]])

>>> A
array([[1, 2, 3],
       [4, 5, 6]])

Using the array constructor to build a 3-by-2 array. Note all the square brackets.
Use Copy to Avoid Aliasing

```
>>> A = array([[1,2],[3,4]])
>>> B = A
>>> A[1,1] = 10
>>> B
array([[ 1,  2],
       [ 3, 10]])
```

```
>>> A = array([[1,2],[3,4]])
>>> B = copy(A)
>>> A[1,1] = 10
>>> B
array([[ 1,  2],
       [ 3,  4]])
```

2D arrays are objects
You Can Add and Subtract Arrays

```python
>>> x = array([10,20,30])
>>> y = array([1,2,3])
>>> z = x - y
>>> z
array([9,18,27])
```

\[
\begin{align*}
[10,20,30] - [1,2,3] &= [9,18,27]
\end{align*}
\]
You Can Apply Various Functions to Arrays

```python
>>> x = array([10,20,30])
>>> y = array([1,2,3])
>>> z = abs(y-x)
>>> z
array([9,18,27])
```
Iteration and 2D Arrays

Lots of Nested Loops
Nested Loops and 2D Arrays

A = array((3,3))
for i in range(3):
    for j in range(3):
        A[i,j] = (i+1)*(j+1)

A 3x3 times table
Nested Loops and 2D Arrays

\[ A = \text{array}((3,3)) \]

Allocates memory, but doesn’t put any values in the boxes. Much more efficient than the Repeated append framework.
Understanding 2D Array Set-Up

```python
for i in range(3):
    for j in range(3):
        A[i,j] = (i+1)*(j+1)
```

```python
for i in range(3):
    A[i,0] = (i+1)*(0+1)
    A[i,1] = (i+1)*(1+1)
    A[i,2] = (i+1)*(2+1)
```

Equivalent!
Understanding 2D Array Set-Up

for i in range(3):
    A[i,0] = (i+1)*(0+1)
    A[i,1] = (i+1)*(1+1)
    A[i,2] = (i+1)*(2+1)

Row 0 is set up when i = 0
Understanding 2D Array Set-Up

```python
for i in range(3):
    A[i, 0] = (i+1)*(0+1)
    A[i, 1] = (i+1)*(1+1)
    A[i, 2] = (i+1)*(2+1)
```

Row 1 is set up when $i = 1$
Understanding 2D Array Set-Up

```python
for i in range(3):
    A[i,0] = (i+1)*(0+1)
    A[i,1] = (i+1)*(1+1)
    A[i,2] = (i+1)*(2+1)
```

Row 2 is set up when $i = 2$
Assume

```python
from random import uniform as randu
from numpy import *
```

Let's write a function `randuM(m,n)` that returns an m-by-n array of random numbers, each chosen from the uniform distribution on [0,1].
A Function that Returns an 
$n$-by-$n$ Array of Random Numbers

def randuM(m,n):
    A = zeros((m,n))
    for i in range(m):
        for j in range(n):
            A[i,j] = randu(0,1)
    return A
Probability Arrays

A \( nxn \) probability array has the property that its entries are nonnegative and that the sum of the entries in each column is 1.

<table>
<thead>
<tr>
<th></th>
<th>.2</th>
<th>.6</th>
<th>.2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>.7</td>
<td>.3</td>
<td>.3</td>
</tr>
<tr>
<td></td>
<td>.1</td>
<td>.1</td>
<td>.5</td>
</tr>
</tbody>
</table>
Probability Arrays

To generate a random probability array, generate a random matrix with nonnegative entries and then divide the numbers in each column by the sum of the numbers in that column.

\[
\begin{array}{ccc}
5 & 6 & 1 \\
2 & 0 & 3 \\
4 & 3 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
\frac{5}{11} & \frac{6}{9} & \frac{1}{5} \\
\frac{2}{11} & \frac{0}{9} & \frac{3}{5} \\
\frac{4}{11} & \frac{3}{9} & \frac{1}{5} \\
\end{array}
\]
A Function that Returns a Random Probability Array

def probM(n):
    A = randuM(n,n)
    for j in range(n):
        # Normalize column j
        s = 0;
        for i in range(n):
            s += A[i,j]
        for i in range(n):
            A[i,j] = A[i,j]/s
    return A
Here is a Network
Think of a node as an island

Think of a node as a Web page

A node

A Transition
Probability

.1
.1
.3
.6
.7
.2
.2
.3
.1
.5
A node

With prob 0.1, a person on island 1 will hop to island 2.
A Random Process

Suppose there are a 1000 people on each node.

At the sound of a whistle they hop to another node in accordance with the “outbound” probabilities.
At Node 0
At Node 1

Graph with nodes 0, 1, and 2 connected by arrows with weights 0.2, 0.7, 0.6, 0.3, 0.1, 0.2, 0.1, 0.1, and 0.5. Numbers 300 and 600 are placed on the edges.
At Node 2

Diagram showing nodes and connections with numerical values.
# The Population Distribution

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>Node 1</td>
<td>1000</td>
<td>1300</td>
</tr>
<tr>
<td>Node 2</td>
<td>1000</td>
<td>700</td>
</tr>
</tbody>
</table>
# Repeat

<table>
<thead>
<tr>
<th>Node</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>1000</td>
<td>1120</td>
</tr>
<tr>
<td>Node 1</td>
<td>1300</td>
<td>1300</td>
</tr>
<tr>
<td>Node 2</td>
<td>700</td>
<td>580</td>
</tr>
</tbody>
</table>
# After 100 Iterations

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>1142.85</td>
<td>1142.85</td>
</tr>
<tr>
<td>Node 1</td>
<td>1357.14</td>
<td>1357.14</td>
</tr>
<tr>
<td>Node 2</td>
<td>500.00</td>
<td>500.00</td>
</tr>
</tbody>
</table>

Appears to reach a Steady State
### After 100 Iterations

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>1142.85</td>
<td>1142.85</td>
</tr>
<tr>
<td>Node 1</td>
<td>1357.14</td>
<td>1357.14</td>
</tr>
<tr>
<td>Node 2</td>
<td>500.00</td>
<td>500.00</td>
</tr>
</tbody>
</table>

In terms of popularity: Island 1 > Island 0 > Island 2
# After 100 Iterations

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 0</td>
<td>1142.85</td>
<td>1142.85</td>
</tr>
<tr>
<td>Node 1</td>
<td>1357.14</td>
<td>1357.14</td>
</tr>
<tr>
<td>Node 2</td>
<td>500.00</td>
<td>500.00</td>
</tr>
</tbody>
</table>

[1142.85, 1357.14, 500.0] is the “stationary vector”
Computing the Stationary Vector Involves a Probability Array

0.2 0.6 0.2
0.7 0.3 0.3
0.1 0.1 0.5

0 -> 1: 0.7
1 -> 2: 0.1
2 -> 0: 0.5

Stationary Vector: (0.2, 0.6, 0.2, 0.7, 0.3, 0.3, 0.1, 0.1, 0.5)
Computing the Stationary Vector Involves a Probability Array

The (0,1) entry is the Prob of hopping from island 1 to island 0
Transition Probability Array

\[ P[i,j] \text{ is the probability of hopping from node } j \text{ to node } i \]
Formula for Updating the Distribution Vector

\[
P = \begin{bmatrix}
0.2 & 0.6 & 0.2 \\
0.7 & 0.3 & 0.3 \\
0.1 & 0.1 & 0.5 \\
\end{bmatrix}
\]

\[
w[0] = 0.2 \cdot v[0] + 0.6 \cdot v[1] + 0.2 \cdot v[2]
\]

\[
w[1] = 0.7 \cdot v[0] + 0.3 \cdot v[1] + 0.3 \cdot v[2]
\]

\[
w[2] = 0.1 \cdot v[0] + 0.1 \cdot v[1] + 0.5 \cdot v[2]
\]

V is the old distribution vector, w is the updated distribution vector.
Formula for Updating the Distribution Vector

\[
P = \begin{pmatrix}
0.2 & 0.6 & 0.2 \\
0.7 & 0.3 & 0.3 \\
0.1 & 0.1 & 0.5 \\
\end{pmatrix}
\]

\[
w[0] = P[0,0]*v[0] + P[0,1]*v[1] + P[0,2]*v[2]
\]
\[
w[1] = P[1,0]*v[0] + P[1,1]*v[1] + P[1,2]*v[2]
\]
\[
\]

V is the old distribution vector, w is the updated distribution vector
A Function that Computes the Update

```python
def Update(P,v):
    n = len(x)
    w = zeros((n,1))
    for i in range(n):
        for j in range(n):
            w[i] += P[i,j]*v[j]
    return w
```
Back to PageRank
Background

Index all the pages on the Web from 0 to N-1. (N is around 50 billion.)

The PageRank algorithm orders these pages from “most important” to “least important”.

It does this by analyzing links, not content.
Key Ideas

The Transition Probability Array

A Very Special Random Walk

The Connectivity Array
A Random Walk on the Web

Repeat:
You are on a webpage.
There are m outlinks.
Choose one at random.
Click on the link.
The Connectivity Array

$G[i,j]$ is 1 if there is a link on page $j$ to page $i$.
The Probability Array

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>0</th>
<th>0</th>
<th>b</th>
<th>0</th>
<th>c</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>c</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>b</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>b</td>
<td>0</td>
<td>0</td>
<td>c</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>0</td>
<td>0</td>
<td>a</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>0</td>
<td>b</td>
<td>0</td>
<td>b</td>
<td>0</td>
<td>0</td>
<td>c</td>
</tr>
<tr>
<td>c</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>c</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

a = 1/3
b = 1/2
c = 1/4
PageRank From the Stationary Vector

<table>
<thead>
<tr>
<th>Webpage</th>
<th>PageRank</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.4100</td>
</tr>
<tr>
<td>6</td>
<td>0.2429</td>
</tr>
<tr>
<td>7</td>
<td>0.8911</td>
</tr>
<tr>
<td>5</td>
<td>0.2609</td>
</tr>
<tr>
<td>2</td>
<td>0.7876</td>
</tr>
<tr>
<td>1</td>
<td>0.8206</td>
</tr>
<tr>
<td>3</td>
<td>0.5723</td>
</tr>
</tbody>
</table>

Webpage 5 has PageRank 0.