22. Two-Dimensional Arrays

Topics
- Motivation
- The numpy Module
- Subscripting functions and 2d Arrays
- Google Page Rank

Visualizing

<table>
<thead>
<tr>
<th></th>
<th>12</th>
<th>17</th>
<th>49</th>
<th>61</th>
</tr>
</thead>
<tbody>
<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>

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)

Rows and Columns

<table>
<thead>
<tr>
<th>12</th>
<th>17</th>
<th>49</th>
<th>61</th>
</tr>
</thead>
<tbody>
<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

<table>
<thead>
<tr>
<th>12</th>
<th>17</th>
<th>49</th>
<th>61</th>
</tr>
</thead>
<tbody>
<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 column 2.

Entries

<table>
<thead>
<tr>
<th>12</th>
<th>17</th>
<th>49</th>
<th>61</th>
</tr>
</thead>
<tbody>
<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 the (1,2) entry.

Where Do They Come From?

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

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 list of lists.

Accessing Entries

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

```python
>>> 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

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

2D arrays are objects

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

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])
```

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

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

A 3x3 times table

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)
```

Equivalent!

```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 0 is set up when $i = 0$

```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 $i = 1$

```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 $i = 2$

Functions and 2D Arrays

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].

```python
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 $n \times n$ probability array has the property that its entries are nonnegative and that the sum of the entries in each column is 1.

\[
\begin{array}{ccc}
0.2 & 0.6 & 0.2 \\
0.7 & 0.3 & 0.3 \\
0.1 & 0.1 & 0.5 \\
\end{array}
\]

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.

\[
\begin{array}{ccc}
5 & 6 & 1 \\
2 & 0 & 3 \\
4 & 3 & 1 \\
\end{array}
+\begin{array}{ccc}
5/11 & 6/9 & 1/5 \\
2/11 & 0/9 & 3/5 \\
4/11 & 3/9 & 1/5 \\
\end{array}
\]

A Function that Returns a Random Probability Array

```python
def probM(n):
    A = randuM(n,n)
    for j in range(n):
        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

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

Suppose there are 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

At Node 2

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></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

In terms of popularity: Island 1 > Island 0 > Island 2

[1142.85, 1357.14, 500.0] is the "stationary vector"

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:

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

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

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

\[w[0] = 0.2v[0] + 0.6v[1] + 0.2v[2]\]
\[w[1] = 0.7v[0] + 0.3v[1] + 0.3v[2]\]
\[w[2] = 0.1v[0] + 0.1v[1] + 0.5v[2]\]

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

\[ P \] =

\[
\begin{align*}
a &= \frac{1}{3} \\
b &= \frac{1}{2} \\
c &= \frac{1}{4}
\end{align*}
\]

PageRank From the Stationary Vector

The probability vector is:

\[
\begin{align*}
0.5723 & \quad 0.8206 \\
0.7876 & \quad 0.2609 \\
0.2064 & \quad 0.8911 \\
0.2429 & \quad 0.4100 \\
\end{align*}
\]

Webpage 5 has PageRank 0.