Matrix Programming

Linear Algebra

NumPy arrays make operations on rectangular data easy

But they are not quite mathematical matrices

```
>>> a = array([[1, 2], [3, 4]])
>>> a * a
array([[ 1,  4],
       [ 9, 16]])
```

Operators act *elementwise*

So this does what you think

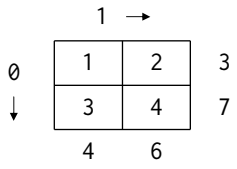
```
>>> a + a
array([[ 2,  4],
       [ 6,  8]])
```

And NumPy is sensible about *scalar* values

```
>>> a + 1
array([[ 2,  3],
       [ 4,  5]])
```

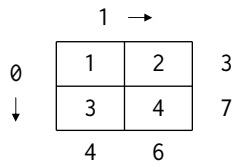

Lots of useful utilities

```
>>> sum(a)
10
>>> sum(a, 0)
array([4, 6])
>>> sum(a, 1)
array([3, 7])
```



Lots of useful utilities

```
>>> sum(a)
10
>>> sum(a, 0)
array([4, 6])
>>> sum(a, 1)
array([3, 7])
```



What does sum(a, 2) do?

Example: disease statistics

- One row per patient
- Columns are hourly responsive T cell counts

```
>>> data[:, 0]   # t0 count for all patients
array([1, 0, 0, 2, 1])
>>> data[0, :]   # all samples for patient 0
array([1, 3, 3, 5, 12, 10, 9])
```

Example: disease statistics

- One row per patient
- Columns are hourly responsive T cell counts

```
>>> data[:, 0]   # t0 count for all patients
array([1, 0, 0, 2, 1])
>>> data[0, :]   # all samples for patient 0
array([1, 3, 3, 5, 12, 10, 9])
```

Why are these 1D rather than 2D?


```
>>> mean(data)
6.8857
```

Intriguing, but not particularly meaningful

```
>>> mean(data, 0)   # over time
array([ 0.8, 2.6, 4.4, 6.4, 10.8, 11., 12.2])

>>> mean(data, 1)   # per patient
array([ 6.14, 4.28, 16.57, 2.14, 5.29])
```

Select the data for people who started with a responsive T cell count of 0

```
>>> data[:, 0]
array([1., 0., 0., 2., 1.])
>>> data[:, 0] == 0.
array([False, True, True, False, False],
      dtype=bool)
>>> data[ data[:, 0] == 0 ]
array([[ 0.,  1.,  2.,  4.,  8.,  7.,  8.],
       [ 0.,  4., 11., 15., 21., 28., 37.]])
```


Find the mean T cell count over time for people who started with a count of 0

```
>>>              data[:, 0]
```

Column 0

Find the mean T cell count over time for people who started with a count of 0

```
>>> data[:, 0] == 0
```

Column 0 is 0

Find the mean T cell count over time for people who started with a count of 0

```
>>> data[ data[:, 0] == 0 ]
```

Rows where column 0 is 0

Find the mean T cell count over time for people who started with a count of 0

```
>>> mean(data[ data[:, 0] == 0 ], 0)
```

Mean along axis 0 of
rows where column 0 is 0

Find the mean T cell count over time for people who started with a count of 0

```
>>> mean(data[ data[:, 0] == 0 ], 0)
array([ 0.,  2.5,  6.5,  9.5,  14.5,  17.5,  22.5])
```

Find the mean T cell count over time for people who started with a count of 0

```
>>> mean(data[ data[:, 0] == 0 ], 0)
array([ 0.,  2.5,  6.5,  9.5, 14.5, 17.5, 22.5])
```

Key to good array programming: no loops!
Just as true for MATLAB or R as for NumPy

What about "real" matrix multiplication?

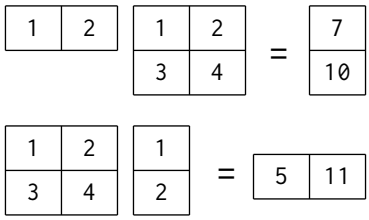
```
>>> a = array([[1, 2], [3, 4]])
>>> dot(a, a)
array([[ 7, 10],
       [15, 22]])
>>> v = arange(3)    # [0, 1, 2]
>>> dot(v, v)        # 0*0 + 1*1 + 2*2
5
```

Dot product only works for sensible shapes

```
>>> dot(ones((2, 3)), ones((2, 3)))
ValueError: objects are not aligned
```

NumPy does not distinguish row/column vectors

```
>>> v = array([1, 2])
>>> a = array([[1, 2], [3, 4]])
>>> dot(v, a)
array([ 7, 10])
>>> dot(a, v)
array([ 5, 11])
```



Can also use the `matrix` subclass of `array`

```
>>> m = matrix([[1, 2], [3, 4]])
>>> m
matrix([[ 1, 2],
        [ 3, 4]])
>>> m*m
matrix([[ 7, 10],
        [15, 22]])
```

Use `matrix(a)` or `array(m)` to convert

Which should you use?

If your problem is linear algebra, `matrix` will probably be more convenient

- Treats vectors as N×1 matrices

Otherwise, use `array`

- Especially if you're representing grids, rather than mathematical matrices

Always look at

`http://www.scipy.org/Numpy_Example_List_With_Doc`

before writing any functions of your own

```
conjugate          histogram
   convolve            lstsq
      correlate           npv
          diagonal            roots
              fft                 solve
                  gradient            svd
```

Fast...

...and someone else has debugged them

created by

Richard T. Guy

November 2010