CME 193: Introduction to Scientific Python
Lecture 6: More on matrices

Austin Benson

Institute for Computational and Mathematical Engineering (ICME)

January 28, 2014

vector-vector dot product

$$\begin{pmatrix}1 & 2 & 3\end{pmatrix}\begin{pmatrix}4\\5\\7\end{pmatrix} = 1\cdot 4 + 2\cdot 5 + 3\cdot 7 = 35$$

```
import numpy as np

a = [1, 2, 3]
b = [4, 5, 7]
print np.dot(a, b)
```

Matrix-vector multiplication

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \\ 7 \end{pmatrix} = \begin{pmatrix} 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 7 \\ 3 \cdot 4 + 2 \cdot 5 + 1 \cdot 7 \end{pmatrix} = \begin{pmatrix} 35 \\ 29 \end{pmatrix}$$

```
import numpy as np

a = [[1, 2, 3], [3, 2, 1]]
b = [4, 5, 7]
print np.dot(a, b)
```

Matrix-matrix multiplication

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 4 & 1 \\ 5 & 1 \\ 7 & 0 \end{pmatrix} = \begin{pmatrix} 35 & 3 \\ 29 & 5 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 4 \\ 5 \\ 7 \end{pmatrix} = \begin{pmatrix} 35 \\ 29 \end{pmatrix} \quad \text{(last slide)}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot 1 + 2 \cdot 1 + 3 \cdot 0 \\ 3 \cdot 1 + 2 \cdot 1 + 1 \cdot 0 \end{pmatrix} = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

```
import numpy as np

a = [[1, 2, 3], [3, 2, 1]]
b = [[4, 1], [5, 1], [7, 0]]
print np.dot(a, b)
```

Stacking

- `vstack`: "vertical stack"
- `np.vstack((A, B))` $\rightarrow \begin{pmatrix} A \\ B \end{pmatrix}$
- `hstack`: "horizontal stack"
- `np.hstack((C, D))` $\rightarrow \begin{pmatrix} C & D \end{pmatrix}$

Stacking

```python
import numpy as np

v = [1, 2, 3]
w = [3, 2, 1]
A = np.vstack((v, w))

x = [4, 5, 7]
y = [1, 1, 0]
B = np.hstack((x, y))
print np.dot(A, B)
```

Stacking

Careful!!

```
 Traceback (most recent call last):
File "stack1.py", line 10, in <module>
print np.dot(A, B)
ValueError:  objects are not aligned
```

```
import numpy as np

x = [4, 5, 7]
y = [1, 1, 0]
B = np.hstack((x, y))
print B
```

$$B = \begin{pmatrix} 4 & 5 & 7 & 1 & 1 & 0 \end{pmatrix}$$

Stacking: transpose problems

```python
import numpy as np

v = [1, 2, 3]
w = [3, 2, 1]
A = np.vstack((v, w))

x = np.transpose(np.array([4, 5, 7]))
y = np.transpose(np.array([1, 1, 0]))
B = np.hstack((x, y))
print np.dot(A, B)
```

Vectors

The transpose of numpy vectors are vectors! (1-dimensional)

```
import numpy as np

x = np.array([4, 5, 7])
print x.shape # (3,)
print x.T.shape # (3,)
print np.transpose(x).shape # (3,)
```

Vectors → matrices

Need to convert to matrices

```
import numpy as np

v = [1, 2, 3]
w = [3, 2, 1]
A = np.vstack((v, w))

x = np.transpose([[4, 5, 7]])
y = np.transpose([[1, 1, 0]])
B = np.hstack((x, y))
print np.dot(A, B)
```

Working transposes

```
import numpy as np

x = np.array([[4, 5, 7]])
print x.shape # (1, 3)
print x.T.shape # (3, 1)
print np.transpose(x).shape # (3, 1)
```

Other subtleties

```python
import numpy as np

x = np.array([2, 2])
y = [2, 2]
A = np.array([[1, 2], [1, 2]])
print np.dot(A, x), np.dot(A, y)
print np.dot(x, A), np.dot(y, A)
```

Other subtleties

```
import numpy as np

x = np.array([2, 2])
y = [2, 2]
A = np.array([[1, 2], [1, 2]])
print np.dot(A, x), np.dot(A, y)
print np.dot(x, A), np.dot(y, A)
```

```
[6 6] [6 6]
[4 8] [4 8]
```

Other subtleties

```
import numpy as np

A = np.array([[1, 1], [1, 1]])
print A / 2
```

Other subtleties

```
import numpy as np

A = np.array([[1, 1], [1, 1]])
print A / 2
```

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

Other subtleties

```
import numpy as np

A = np.array([[1, 1], [1, 1]], dtype='float')
print A / 2
```

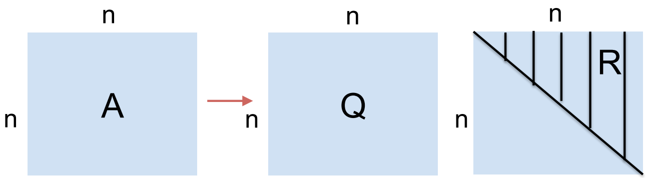
Other subtleties

```
import numpy as np

A = np.array([[1, 1], [1, 1]], dtype='float')
print A / 2
```

```
[[ 0.5 0.5]
 [ 0.5 0.5]]
```

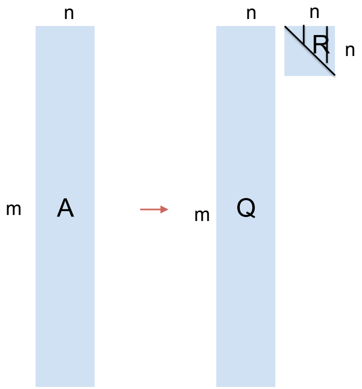
$n \times n$ QR

- A is $n \times n$
- $A = QR$, Q, R $n \times n$
- $Q^T Q = I$
- $R_{ij} = 0$ if $i > j$: "upper triangular"



$m \times n$ QR

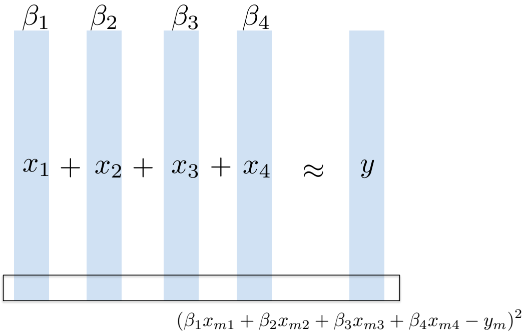
- A is $m \times n$, $m > n$
- $A = QR$, Q $m \times n$, R $n \times n$
- $Q^T Q = I$
- $R_{ij} = 0$ if $i > j$: "upper triangular"



Least squares

- Have several input vectors $x_1, x_2, \ldots, x_n$, each of length m.
- Have a one output vector y of length m
- Want find vector β to minimize

$$\sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij}\beta_j - y_j)^2$$



β_1
β_2
β_3
β_4
$x_1 + x_2 + x_3 + x_4 \approx y$
$(\beta_1 x_{m1} + \beta_2 x_{m2} + \beta_3 x_{m3} + \beta_4 x_{m4} - y_m)^2$

Matrix form of least squares

Matrix form:

$$\min_{\beta} (X\beta - y)^T (X\beta - y)$$

- $X = \begin{pmatrix} x_{11} & \dots & x_{1n} \\ & \vdots & \\ x_{m1} & \dots & x_{1n} \end{pmatrix} = \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix}$
- $\beta = \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_n \end{pmatrix}, y = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix}$

Least squares

$$\hat{\beta} = \min_{\beta} (X\beta - y)^T (X\beta - y)$$

- $X = QR$
- $z = Q^T y$
- $R\hat{\beta} = z$

Moral of the story: QR has a purpose

NumPy least squares

1. $X = QR$
2. $z = Q^T y$
3. $R\hat{\beta} = z$

```
import numpy as np

X = np.array([[1, 2], [3, 4], [7, 8]])
y = [9, 12, 11]
Q, R = np.linalg.qr(X) # 1
z = np.dot(Q.T, y) # 2
betahat = np.linalg.solve(R, z) # 3
print betahat
```

NumPy + SciPy QR

1. $X = QR$
2. $z = Q^T y$
3. $R\hat{\beta} = z$

```python
import numpy as np
import scipy.linalg

X = np.array([[1, 2], [3, 4], [7, 8]])
y = [9, 12, 11]
Q, R = np.linalg.qr(X) # 1
z = np.dot(Q.T, y) # 2
betahat = scipy.linalg.solve_triangular(R, z) # 3
print betahat
```

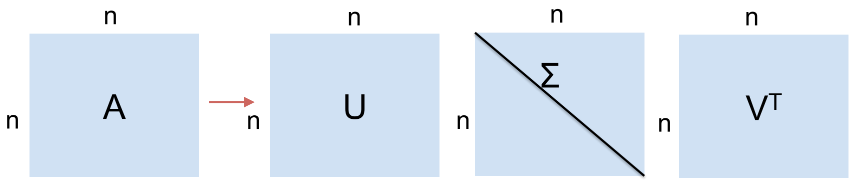
NumPy least squares

```
import numpy as np

A = np.array([[1, 2], [3, 4], [7, 8]])
y = [9, 12, 11]

'''
Q, R = np.linalg.qr(A)
z = np.dot(Q.T, y)
betahat = np.linalg.solve(R, z)
'''

betahat = np.linalg.lstsq(A, y)[0]
print betahat
```

SVD

The singular value decomposition (SVD) of a matrix A:

- A is $n \times n$
- $A = U\Sigma V^T$, all $n \times n$
- $U^T U = V^T V = I$
- Σ is diagonal with $\Sigma_{ii} \geq \Sigma_{jj} \geq 0$ for $i < j$.



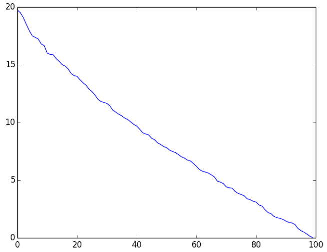
SVD: applications

SVD appears all over the place, for example:

- Principal component analysis
- Low-rank approximations in computational physics
- Signals processing

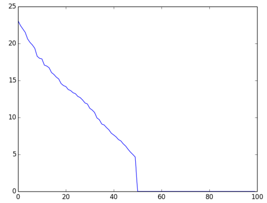
SVD example

```
import numpy as np
import matplotlib.pyplot as plt
A = np.random.randn(100, 100)
U, S, Vt = np.linalg.svd(A)
plt.plot(S)
plt.show()
```



SVD example

```
import numpy as np
import matplotlib.pyplot as plt
A = np.random.randn(100, 50)
B = np.hstack((A, A))
U, S, Vt = np.linalg.svd(B)
plt.plot(S)
plt.show()
```



25
20
15
10
5
0
0
20
40
60
80
100