CME 193: Introduction to Scientific Python

Lecture 5: Numpy, Scipy, Matplotlib

Sven Schmit

`stanford.edu/~schmit/cme193`

Contents

- Second part of course
- Numpy
- Scipy
- Matplotlib
- Exercises

Congrats, we are halfway!

Up to now

- Covered the basics of Python
- Worked on a bunch of tough exercises

From now

- Cover specific topics
- Less exercises
- Time for project

Feedback

Thanks for the great feedback, very useful

Remaining topics

- Numpy, Scipy, Matplotlib (today)
- IPython notebooks, Pandas, Statsmodels, SKLearn
- Exception handling, unit testing, recursion
- Brief look at some more modules
  - Flask
  - Regex
  - ... (suggestions welcome)

Contents

- Second part of course
- Numpy
- Scipy
- Matplotlib
- Exercises

Numpy

- Fundamental package for scientific computing with Python
- N-dimensional array object
- Linear algebra, Fourier transform, random number capabilities
- Building block for other packages (e.g. Scipy)
- Open source

Numpy

- Fundamental package for scientific computing with Python
- N-dimensional array object
- Linear algebra, Fourier transform, random number capabilities
- Building block for other packages (e.g. Scipy)
- Open source

import numpy as np

Basics:

```
import numpy as np

A = np.array([[1, 2, 3], [4, 5, 6]])
print A
# [[1 2 3]
#  [4 5 6]]

Af = np.array([1, 2, 3], float)
```

Slicing as usual.

More basics

```
np.arange(0, 1, 0.2)
# array([ 0. ,  0.2,  0.4,  0.6,  0.8])

np.linspace(0, 2*np.pi, 4)
# array([ 0.0,  2.09,  4.18, 6.28])

A = np.zeros((2,3))
# array([[ 0.,  0.,  0.],
#        [ 0.,  0.,  0.]])
# np.ones, np.diag
A.shape
# (2, 3)
```

More basics

```
np.random.random((2,3))
# array([[ 0.78084261,  0.64328818,  0.55380341],
#        [ 0.24611092,  0.37011213,  0.83313416]])

a = np.random.normal(loc=1.0, scale=2.0, size=(2,2))
# array([[ 2.87799514,  0.6284259 ],
#        [ 3.10683164,  2.05324587]])

np.savetxt("a_out.txt", a)
# save to file
b = np.loadtxt("a_out.txt")
# read from file
```

Arrays are mutable

```
A = np.zeros((2, 2))
# array([[ 0.,  0.],
#        [ 0.,  0.]])
C = A
C[0, 0] = 1

print A
# [[ 1.  0.]
#  [ 0.  0.]]
```

Array attributes

```
a = np.arange(10).reshape((2,5))

a.ndim     # 2 dimension
a.shape    # (2, 5) shape of array
a.size     # 10 # of elements
a.T        # transpose
a.dtype    # data type
```

Basic operations

Arithmetic operators: **elementwise** application

```
a = np.arange(4)
# array([0, 1, 2, 3])

b = np.array([2, 3, 2, 4])

a * b  # array([ 0,  3,  4, 12])
b - a  # array([2, 2, 0, 1])

c = [2, 3, 4, 5]
a * c  # array([ 0,  3,  8, 15])
```

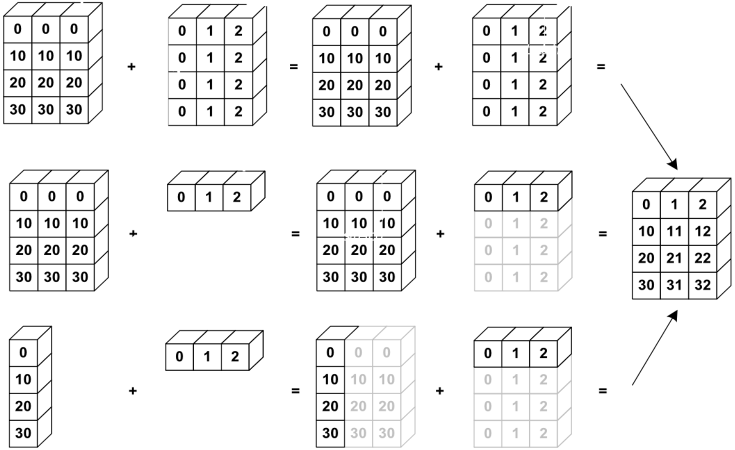
Also, we can use `+=` and `*=`.

Array broadcasting

When operating on two arrays, numpy compares shapes. Two dimensions are compatible when

1. They are of equal size
2. One of them is 1

Array broadcasting



Array broadcasting with scalars

This also allows us to add a constant to a matrix or multiply a matrix by a constant

```
A = np.ones((3,3))

print 3 * A - 1
# [[ 2.  2.  2.]
#  [ 2.  2.  2.]
#  [ 2.  2.  2.]]
```

Vector operations

- inner product
- outer product
- dot product (matrix multiplication)

```
# note: numpy automatically converts lists
u = [1, 2, 3]
v = [1, 1, 1]

np.inner(u, v)
# 6
np.outer(u, v)
# array([[1, 1, 1],
#        [2, 2, 2],
#        [3, 3, 3]])
np.dot(u, v)
# 6
```

Matrix operations

First, define some matrices:

```
A = np.ones((3, 2))
# array([[ 1.,  1.],
#        [ 1.,  1.],
#        [ 1.,  1.]])
A.T
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])

B = np.ones((2, 3))
# array([[ 1.,  1.,  1.],
#        [ 1.,  1.,  1.]])
```

Matrix operations

```
np.dot(A, B)
# array([[ 2.,  2.,  2.],
#        [ 2.,  2.,  2.],
#        [ 2.,  2.,  2.]])

np.dot(B, A)
# array([[ 3.,  3.],
#        [ 3.,  3.]])

np.dot(B.T, A.T)
# array([[ 2.,  2.,  2.],
#        [ 2.,  2.,  2.],
#        [ 2.,  2.,  2.]])

np.dot(A, B.T)
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# ValueError: shapes (3,2) and (3,2) not aligned:  ...
# ... 2 (dim 1) != 3 (dim 0)
```

Operations along axes

```
a = np.random.random((2,3))
# array([[ 0.9190687 ,  0.36497813,  0.75644216],
#        [ 0.91938241,  0.08599547,  0.49544003]])
a.sum()
# 3.5413068994445549
a.sum(axis=0)  # column sum
# array([ 1.83845111,  0.4509736 ,  1.25188219])
a.cumsum()
# array([ 0.9190687 ,  1.28404683,  2.04048899,  2.9598714 ,
#         3.04586687,  3.5413069 ])
a.cumsum(axis=1)  # cumulative row sum
# array([[ 0.9190687 ,  1.28404683,  2.04048899],
#        [ 0.91938241,  1.00537788,  1.50081791]])
a.min()
# 0.0859954690403677
a.max(axis=0)
# array([ 0.91938241,  0.36497813,  0.75644216])
```

Slicing arrays

More advanced slicing

```
a = np.random.random((4,5))

a[2, :]
# third row, all columns
a[1:3]
# 2nd, 3rd row, all columns
a[:, 2:4]
# all rows, columns 3 and 4
```

Iterating over arrays

- Iterating over multidimensional arrays is done with respect to the first axis: `for row in A`
- Looping over all elements: `for element in A.flat`

Reshaping

Reshape using `reshape`. Total size must remain the same.

Resize using `resize`, always works: chopping or appending zeros

First dimension has 'priority', so beware of unexpected results

Try it!

Reshaping

Reshape using `reshape`. Total size must remain the same.

Resize using `resize`, always works: chopping or appending zeros

First dimension has ‘priority’, so beware of unexpected results

Try it!

Reshaping

Reshape using `reshape`. Total size must remain the same.

Resize using `resize`, always works: chopping or appending zeros

First dimension has ‘priority’, so beware of unexpected results

Try it!

Matrix operations

```
import numpy.linalg
```

`eye(3)`	Identity matrix
`trace(A)`	Trace
`column_stack((A,B))`	Stack column wise
`row_stack((A,B,A))`	Stack row wise

Linear algebra

```
import numpy.linalg
```

`qr`	Computes the QR decomposition
`cholesky`	Computes the Cholesky decomposition
`inv(A)`	Inverse
`solve(A,b)`	Solves $Ax = b$ for A full rank
`lstsq(A,b)`	Solves $\arg\min_x \|Ax - b\|_2$
`eig(A)`	Eigenvalue decomposition
`eig(A)`	Eigenvalue decomposition for symmetric or hermitian
`eigvals(A)`	Computes eigenvalues.
`svd(A, full)`	Singular value decomposition
`pinv(A)`	Computes pseudo-inverse of A

Fourier transform

`import numpy.fft`

- `fft` 1-dimensional DFT
- `fft2` 2-dimensional DFT
- `fftn` N-dimensional DFT
- `ifft` 1-dimensional inverse DFT (etc.)
- `rfft` Real DFT (1-dim)
- `ifft` Imaginary DFT (1-dim)

Random sampling

```
import numpy.random
```

`rand(d0,d1,...,dn)`	Random values in a given shape
`randn(d0, d1, ...,dn)`	Random standard normal
`randint(lo, hi, size)`	Random integers [lo, hi)
`choice(a, size, repl, p)`	Sample from a
`shuffle(a)`	Permutation (in-place)
`permutation(a)`	Permutation (new array)

Distributions in random

`import numpy.random`

The list of distributions to sample from is quite long, and includes

- `beta`
- `binomial`
- `chisquare`
- `exponential`
- `dirichlet`
- `gamma`
- `laplace`
- `lognormal`
- `pareto`
- `poisson`
- `power`

Contents

- Second part of course
- Numpy
- Scipy
- Matplotlib
- Exercises

What is SciPy?

SciPy is a library of algorithms and mathematical tools built to work with NumPy arrays.

- linear algebra - *scipy.linalg*
- statistics - *scipy.stats*
- optimization - *scipy.optimize*
- sparse matrices - *scipy.sparse*
- signal processing - *scipy.signal*
- etc.

Scipy Linear Algebra

Slightly different from numpy.linalg. Always uses BLAS/LAPACK support, so could be faster.

Some more functions.

Functions can be slightly different.

Scipy Optimization

- General purpose minimization: CG, BFGS, least-squares
- Constrainted minimization; non-negative least-squares
- Minimize using simulated annealing
- Scalar function minimization
- Root finding
- Check gradient function
- Line search

Scipy Statistics

- Mean, median, mode, variance, kurtosis
- Pearson correlation coefficient
- Hypothesis tests (ttest, Wilcoxon signed-rank test, Kolmogorov-Smirnov)
- Gaussian kernel density estimation

See also SciKits (or scikit-learn).

Scipy sparse

- Sparse matrix classes: CSC, CSR, etc.
- Functions to build sparse matrices
- sparse.linalg module for sparse linear algebra
- sparse.csgraph for sparse graph routines

Scipy signal

- Convolutions
- B-splines
- Filtering
- Continuous-time linear system
- Wavelets
- Peak finding

Scipy IO

Methods for loading and saving data

- Matlab files
- Matrix Market files (sparse matrices)
- Wav files

Contents

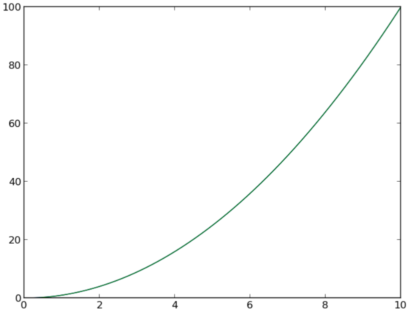
- Second part of course
- Numpy
- Scipy
- Matplotlib
- Exercises

What is Matplotlib?

- Plotting library for Python
- Works well with Numpy
- Syntax similar to Matlab

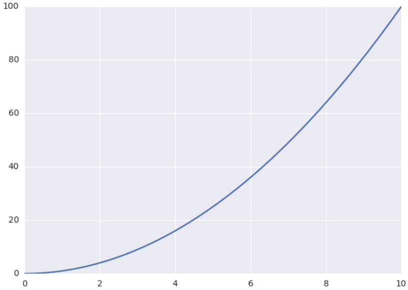
Scatter Plot

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```



Seaborn makes plot pretty

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
plt.plot(x, y)
plt.show()
```

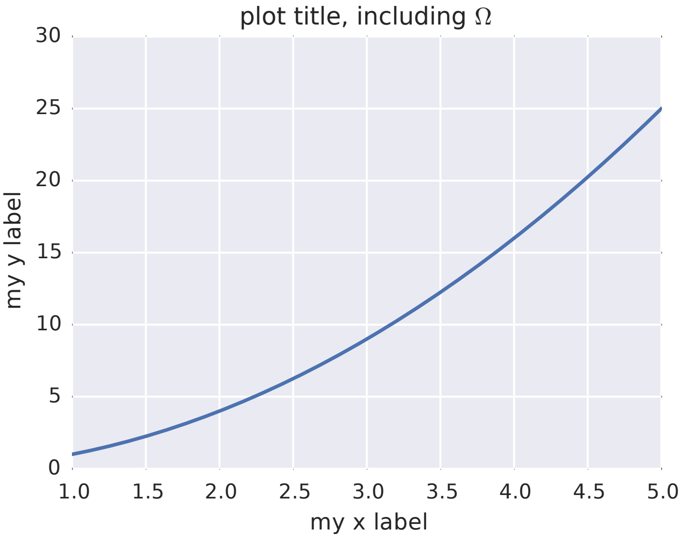


Scatter Plot

Adding titles and labels

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

f, ax = plt.subplots(1, 1, figsize=(5,4))

x = np.linspace(0, 10, 1000)
y = np.power(x, 2)
ax.plot(x, y)
ax.set_xlim((1, 5))
ax.set_ylim((0, 30))
ax.set_xlabel('my x label')
ax.set_ylabel('my y label')
ax.set_title('plot title, including $\Omega$')

plt.tight_layout()
plt.savefig('line_plot_plus.pdf')
```

Scatter Plot

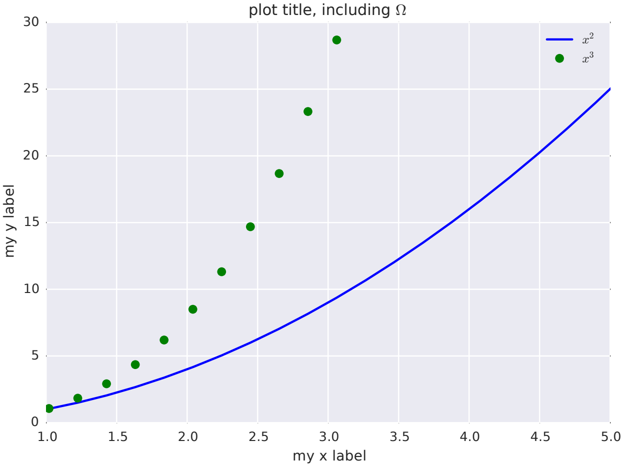


Scatter Plot

Adding multiple lines and a legend

```
x = np.linspace(0, 10, 50)
y1 = np.power(x, 2)
y2 = np.power(x, 3)

plt.plot(x, y1, 'b-', label='$x^2$')
plt.plot(x, y2, 'go', label='$x^3$')
plt.xlim((1, 5))
plt.ylim((0, 30))
plt.xlabel('my x label')
plt.ylabel('my y label')
plt.title('plot title, including $\Omega$')
plt.legend()

plt.savefig('line_plot_plus2.pdf')
```

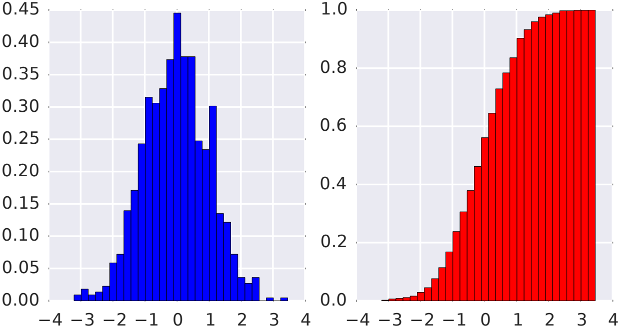
Scatter Plot



Histogram

```
data = np.random.randn(1000)

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(6,3))

# histogram (pdf)
ax1.hist(data, bins=30, normed=True, color='b')

# empirical cdf
ax2.hist(data, bins=30, normed=True, color='r',
         cumulative=True)

plt.savefig('histogram.pdf')
```

Histogram



0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
−4 −3 −2 −1 0 1 2 3 4
1.0
0.8
0.6
0.4
0.2
0.0
−4 −3 −2 −1 0 1 2 3 4

Box Plot

```
samp1 = np.random.normal(loc=0., scale=1., size=100)
samp2 = np.random.normal(loc=1., scale=2., size=100)
samp3 = np.random.normal(loc=0.3, scale=1.2, size=100)

f, ax = plt.subplots(1, 1, figsize=(5,4))

ax.boxplot((samp1, samp2, samp3))
ax.set_xticklabels(['sample 1', 'sample 2', 'sample 3'])
plt.savefig('boxplot.pdf')
```

Box Plot

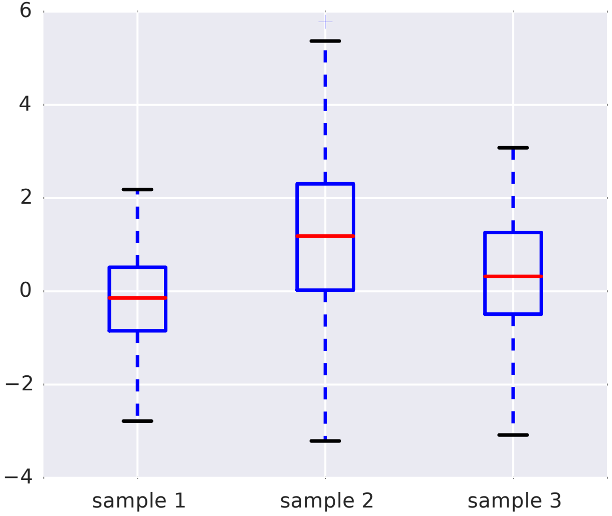
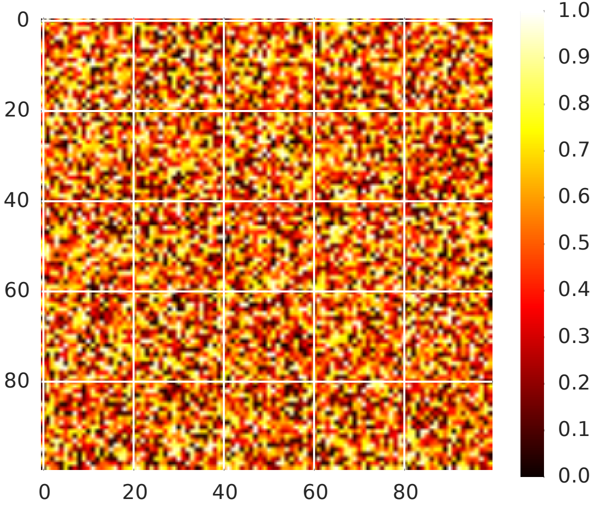


Image Plot

```
A = np.random.random((100, 100))

plt.imshow(A)
plt.hot()
plt.colorbar()

plt.savefig('imageplot.pdf')
```

Image Plot

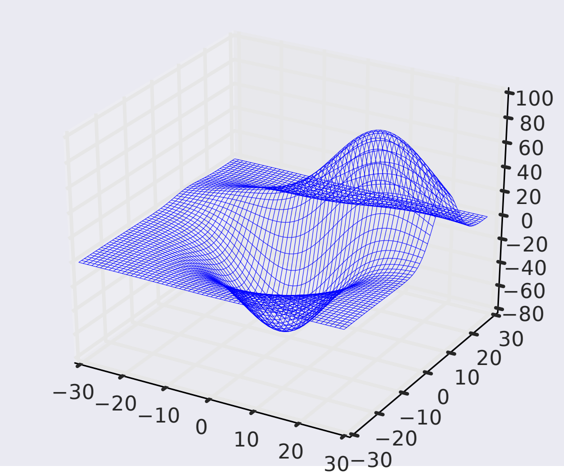


Wire Plot

matplotlib toolkits extend funtionality for other kinds of visualization

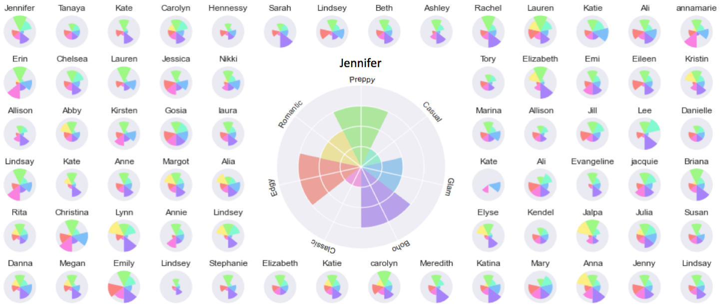
```
from mpl_toolkits.mplot3d import axes3d

ax = plt.subplot(111, projection='3d')
X, Y, Z = axes3d.get_test_data(0.1)
ax.plot_wireframe(X, Y, Z, linewidth=0.1)

plt.savefig('wire.pdf')
```

Wire Plot



Possibilities

A lot is possible, but not always easy to figure out how...



Contents

- Second part of course
- Numpy
- Scipy
- Matplotlib
- Exercises

Exercises

See course website for exercises for this week.

Get to know the person next to you and do them in pairs!

Let me know if you have any question

Class ends at 5:35pm.