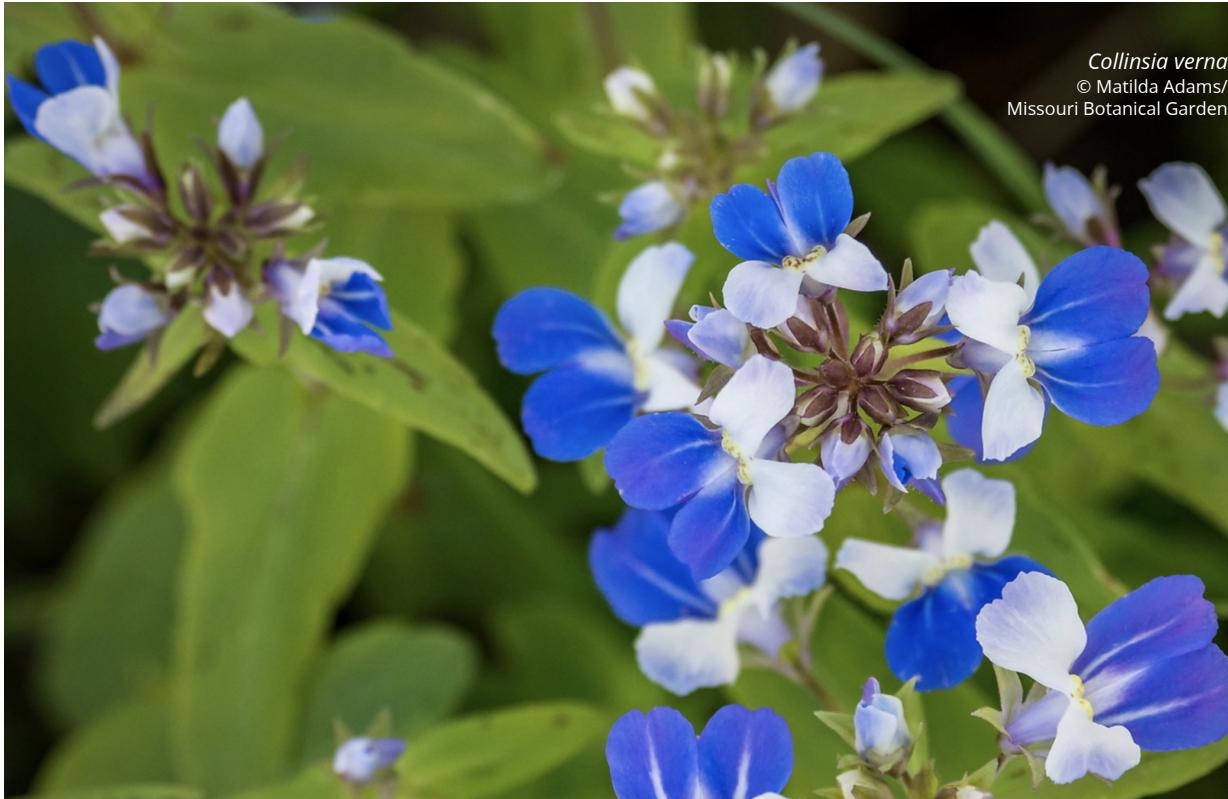


Mon, Dec 07

Lecture 13A:

Python: SciPy

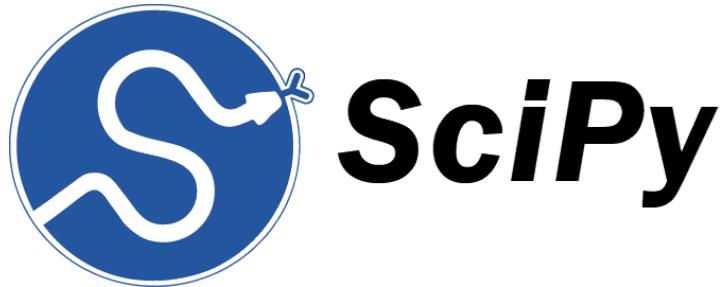


Practical Bioinformatics (Biol 4220)
Instructor: Michael Landis
Email: michael.landis@wustl.edu



Lecture 13A outline

1. Overview of SciPy
2. Survey of SciPy modules
3. Lab 10B overview



[SciPy](#) is an open source ecosystem that extends Python's functionality for math, science, and engineering

This ecosystem includes its open source software, its community, and its conferences

The SciPy library itself provides methods for signal processing, optimization, integration, statistics, and more

SciPy library organization

→	cluster	Clustering algorithms
→	constants	Physical and mathematical constants
	fftpack	Fast Fourier Transform routines
→	integrate	Integration and ordinary differential equation solvers
	interpolate	Interpolation and smoothing splines
	io	Input and Output
	linalg	Linear algebra
	ndimage	N-dimensional image processing
	odr	Orthogonal distance regression
→	optimize	Optimization and root-finding routines
	signal	Signal processing
	sparse	Sparse matrices and associated routines
	spatial	Spatial data structures and algorithms
	special	Special functions
→	stats	Statistical distributions and functions

scipy.constants

```
>>> from scipy import constants
>>> constants.pi                                # pi
3.141592653589793
>>> constants.golden                            # golden ratio, (1+5^.5)/2
1.618033988749895
>>> constants.Avogadro                          # Avogadro's number
6.022140857e+23
>>> constants.speed_of_light                   # speed of light in vacuum
299792458.0
>>> constants.electron_mass                    # mass of electron
9.10938356e-31
>>> constants.proton_mass                     # mass of proton
1.672621898e-27
>>> constants.neutron_mass                    # mass of neutron
1.674927471e-27
>>> scipy.constants.physical_constants.keys()  # returns (values, units, precision)
>>> scipy.constants.physical_constants        # ... contains 100's of items
{'Wien displacement law constant': (0.0028977685, 'm K', 5.1e-09),
 'atomic unit of 1st hyperpolarizability': (3.20636151e-53, 'C^3 m^3 J^-2', 2.8e-60),
 'atomic unit of 2nd hyperpolarizability': (6.2353808e-65, 'C^4 m^4 J^-3', 1.1e-71),
 'atomic unit of electric dipole moment': (8.47835309e-30, 'C m', 7.3e-37),
 'atomic unit of electric polarizability': (1.648777274e-41, 'C^2 m^2 J^-1', 1.6e-49),
 'atomic unit of electric quadrupole moment': (4.48655124e-40, 'C m^2', 3.9e-47),
 ...
}
```

*Mathematical and (mostly) physical
constants, units, and precisions*

scipy.stats

Provides functions for a large number of probability distributions, along with many other statistical operations

- probability distributions
- statistical tests
- frequency statistics
- summary statistics
- statistical distances
- contingency tables
- kernel density estimators

distributions in *scipy.stats*

Probability distributions assign relative frequencies to possible outcomes for a random experiment -- e.g. *a coin flip*

All *scipy.stats* distributions provide the same set of functions:

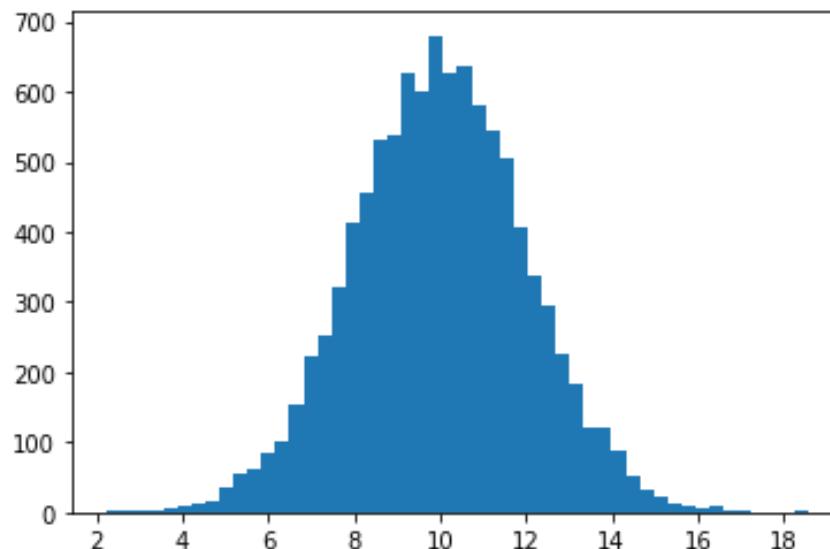
- simulate random variates
- compute probabilities for data
- compute moments (mean, variance, ...)
- fit parameters to data

distributions in *scipy.stats*

```
>>> from scipy import stats
>>> # distribution object
>>> stats.norm
<scipy.stats._continuous_distns.norm_gen object at 0x7fdf98b67d68>
>>> # generate 4 normal RVs
>>> x = stats.norm.rvs(loc=10,scale=2,size=4)
array([10.11179033, 10.10902411, 10.65111753, 10.32368948])
>>> stats.norm.pdf(x=x, loc=10, scale=2 )
array([0.19915978, 0.19917499, 0.18917553, 0.19687573])
>>> # return mean, variance, skewness, kurtosis (mvsk)
>>> stats.norm.stats(loc=10, scale=2, \
                      moments='mvsk')
(array(10.), array(4.), array(0.), array(0.))
>>> # generate 1000 normal RVs with mean=10, scale=2
>>> y = stats.norm.rvs(loc=10,scale=2,size=1000)
>>> # estimate mean and scale from simulated data
>>> stats.norm.fit( data=y, loc=50, scale=9
(9.939835173664239, 2.004585580838588)
```

distributions in *scipy.stats*

```
>>> from scipy import stats
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> # simulate 10000 RVs, mean=10, scale=2
>>> y = stats.norm.rvs(loc=10,scale=2,size=10000)
>>> y
array([13.23832222, 13.05731999,  7.30259037, ...,  9.8595785,
       9.46715211,  9.98579946])
>>> fig,ax = plt.subplots()
>>> p = ax.hist(y,bins=50)
>>> fig.show()
```



distributions in *scipy.stats*

Roughly 100 distributions available,
each with different properties

Type	Name	Class	Use
Continuous	uniform	<i>stats.uniform</i>	"flat" over interval
	normal	<i>stats.norm</i>	random sums
	exponential	<i>stats.expon</i>	events with rates
	beta	<i>stats.beta</i>	flexible on [0,1]
Discrete	bernoulli	<i>stats.bernoulli</i>	single coin-flip
	binomial	<i>stats.binom</i>	many coin-flips
	Poisson	<i>stats.poisson</i>	# events w/ rates

summary statistics in *scipy.stats*

Use ***summary statistics*** to describe simple properties of a data sample, e.g. *the sample mean*

```
>>> from scipy import stats
>>> import numpy as np
>>> x = stats.norm.rvs(loc=10, scale=2, size=1000)
>>> np.mean(x)          # sample mean
9.907750191507521
>>> stats.hmean(x)      # harmonic mean
9.4599529015089185
>>> stats.gmean(x)      # geometric mean
9.6924828402578722
>>> stats.sem(x)        # standard error of sample mean
0.06353084579162438
>>> stats.skew(x)       # sample skew (asymmetry)
0.011612124738411802
>>> stats.kurtosis(x)   # sample kurtosis (fat-tailedness)
-0.0055425614273967305
>>> stats.describe(x)   # give summary of sample data
DescribeResult(nobs=1000,
               minmax=(3.7313631618255805, 16.221776607589973),
               mean=9.907750191507521,
               variance=4.0361683669991582,
               skewness=0.011612124738411802,
               kurtosis=-0.0055425614273967305)
```

statistical tests in *scipy.stats*

Many statistical tests that are used to analyze biological data are available, including:

- t-tests: *ttest_1samp*, *ttest_ind*, *ttest_rel*
- Chi-square: *chisquare*
- Kolmogorov-Smirnov: *kstest*, *ks_1samp*, *ks_2samp*
- Mann-Whitney: *mannwhitneyu*
- Cressie-Read power divergence: *power_divergence*
- Wilcoxon signed-rank test: *wilcoxon*
- Kruskal-Wallis H-test: *kruskal*
- ...and dozens more

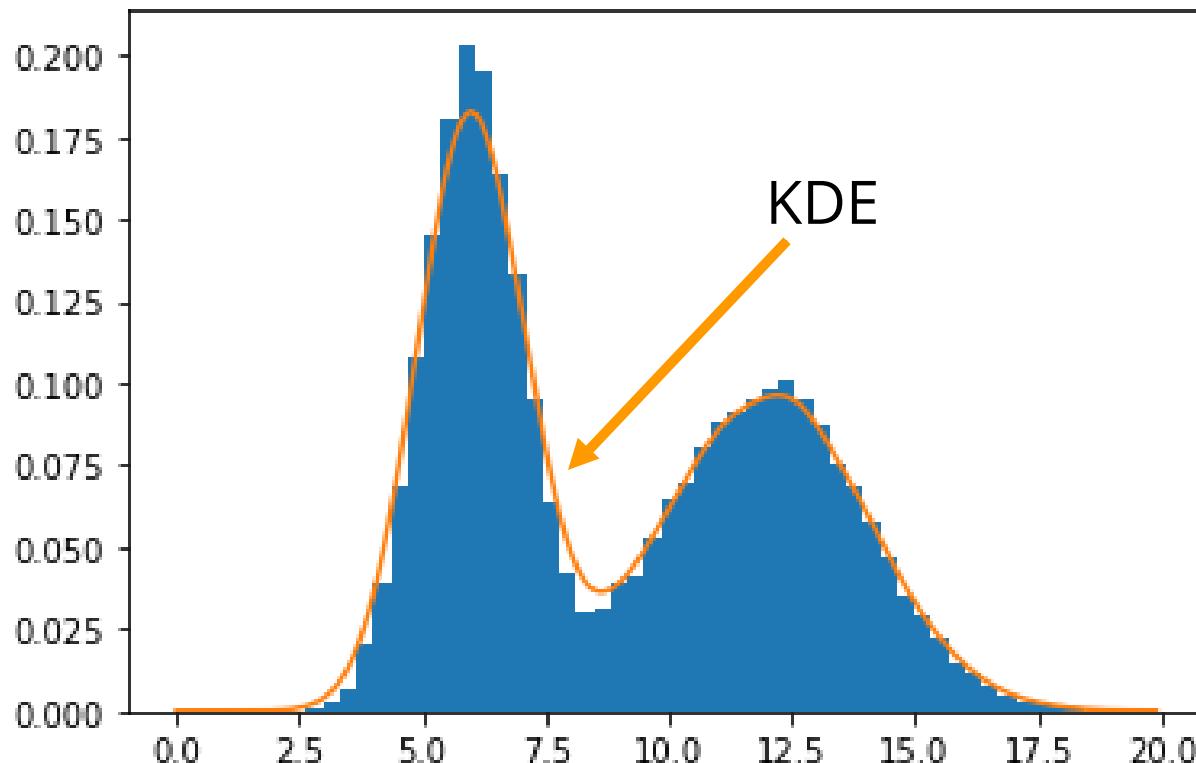
KDEs in *scipy.stats*

Kernel density estimators (KDEs) approximate a data sample as a continuous probability density

```
>>> from scipy import stats
>>> import matplotlib
>>> import matplotlib.pyplot as plt
>>> a = stats.norm.rvs(loc=12,scale=2,size=10000) # create samples, a
>>> b = stats.norm.rvs(loc=6,scale=1,size=10000) # create samples, b
>>> z = np.append(a, b)                         # mix samples a and b
>>> f = kernel = stats.gaussian_kde(z)          # fit KDE object to data, z
>>> x = np.arange(0,20,0.1)                      # range of input values for f()
>>> f(x)                                         # density for each value, f(pos)
array([1.33471717e-11, 4.45526152e-11, 1.42288247e-10, 4.34870069e-10,
       1.27215449e-09, 3.56303164e-09, 9.55708453e-09, 2.45585700e-08,
       6.04813171e-08, 1.42815299e-07, 3.23510616e-07, 7.03432155e-07,
       ...
>>> fig,ax = plt.subplots()                        # create empty plot
>>> h = ax.hist(z,bins=50,density=True)          # plot histogram (normalized)
>>> ax.plot(x, f(x))                           # plot KDE curve
>>> plt.show()                                # show plot
```

KDEs in *scipy.stats*

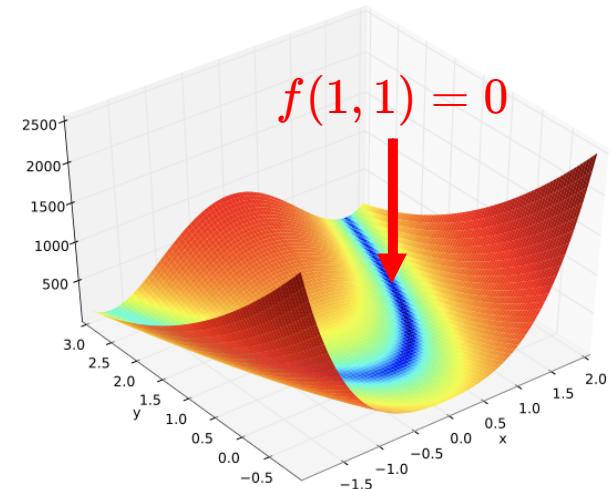
Kernel density estimators (KDEs) approximate a data sample as a continuous probability density



scipy.optimize

```
>>> import numpy as np
>>> from scipy.optimize import minimize
>>> def rosen(x):
...     # Rosenbrock function, for a=1 and b=100
...     # f(x,y) = (1-x)^2 + 100*(y-x^2)^2
...     return (1.0-x[0])**2 + 100.0*(x[1]-x[0]**2.0)**2.0
>>>
>>> rosen( [1.0, 1.0] )          # optimal value, f(1,1) = 0.0
0.0
>>> rosen( [1.01, 1.01] )        # worse value
0.01030099999999994
>>> rosen( [1.05, 1.05] )        # worse value
0.2781249999999999
>>> x0 = np.array([1.3, 0.7])    # initial guess
>>> # find the optimal value for x
>>> res = minimize(rosen, x0, method='nelder-mead',
                  options={'xtol': 1e-8, 'disp': True})
Optimization terminated successfully.
    Current function value: 0.000000
    Iterations: 79
    Function evaluations: 150
>>> print(res.x)                # estimate for x
[1. 1.]
>>> print(res.fun)              # minimum, approx 0.0
3.3736077629532093e-18
```

Methods that find arguments that maximize a user-defined function



$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$
$$a = 1, b = 100$$

scipy.integrate

Various methods to integrate functions,
fixed samples, and ODEs

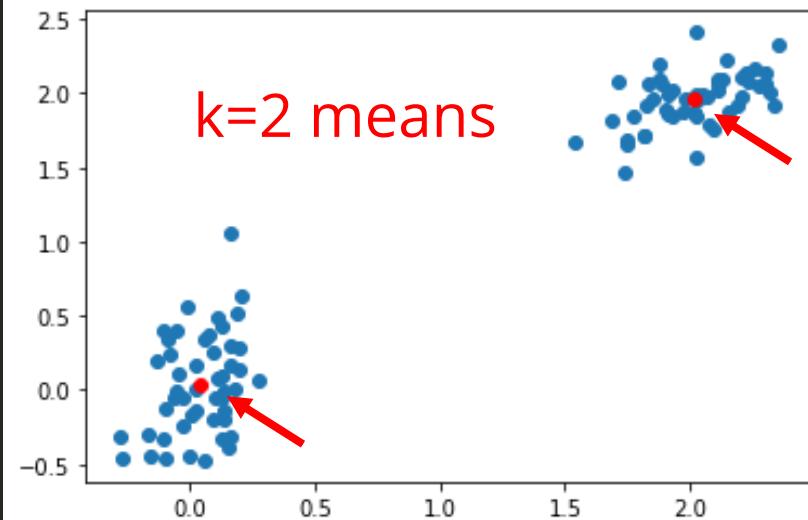
```
>>> from scipy import integrate
>>> # define function to integrate
>>> def x2(x):
...     return x**2
...
>>> # numerical integration using quadrature
>>> y, err = integrate.quad(x2, 0, 4)
>>> y
21.3333333333336      # numerical integrand
>>> err
2.368475785867001e-13  # numerical error
>>>
>>> print(4**3 / 3.)    # analytical result
21.3333333333
```

$$\begin{aligned} I &= \int_0^4 x^2 dx \\ &= F(4) - F(0) \\ &= \frac{4^3}{3} - \frac{0^3}{3} = 21\frac{1}{3} \end{aligned}$$

scipy.clustering

Various functions for inferring latent structures in data, e.g.
k-means, vector-quantization, and hierarchical clustering

```
>>> import numpy as np
>>> from scipy.cluster.vq import vq, kmeans, whiten
>>> import matplotlib.pyplot as plt
>>> # 50 pts at (0,0)
>>> a = np.random.multivariate_normal([ 0,  0],
...                                     [[ 4,  1], [1,  4]],
...                                     size=50)
...
>>> # 50 pts at (30,10)
>>> b = np.random.multivariate_normal([30, 10],
...                                     [[10,  2], [2,  1]],
...                                     size=50)
...
>>> features = np.concatenate((a, b))      # x = [ a, b ]
>>> x = whiten(features)                 # normalize data
>>> y, e = kmeans(whitened, 2)           # infer k=2 means
>>> plt.scatter(x[:, 0], x[:, 1])        # plot data, x
>>> plt.scatter(y[:, 0], y[:, 1], c='r')  # plot means, y
>>> plt.show()
```



Lab 13A

github.com/WUSTL-Biol4220/home/labs/lab_13A.md