UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Essential Data Skills for Business Analytics

Lecture 12 (04/17, 04/19): Optimization and Statistics: SciPy

Decision, Operations & Information Technologies
Robert H. Smith School of Business
Spring, 2017

Scientific Applications

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- There are a few third-party packages available for scientific computing that extend Python's basic math module:
  - NumPy/SciPy – numerical and scientific function libraries.
  - Numba – Python compiler that support JIT compilation.
  - ALGLIB – numerical analysis library.
  - PyGSL – Python interface for GNU Scientific Library.
  - ScientificPython – collection of scientific computing modules.

SciPy

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- By far, the most commonly used packages are those in the SciPy stack. These packages include:
  - NumPy – fundamental package for scientific computing.
  - SciPy – efficient numerical routines.
  - Matplotlib – plotting library.
  - IPython – interactive computing.
  - SymPy – symbolic computation library.
  - Pandas – data analysis library.
  - …

Install SciPy Stack

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- https://www.scipy.org/install.html
- Mac and Linux users can install pre-built binary packages for the SciPy stack using pip.
- Pip can install pre-built binary packages in the wheel package format.
- Pip does not work well for Windows because the standard pip package index site, PyPI, does not yet have Windows wheels for some packages, such as SciPy.

Install SciPy Stack

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- To install via pip on Mac or Linux, first upgrade pip to the latest version:

  python -m pip install --upgrade pip

- Then install the SciPy stack packages with pip by using the --user flag. This installs packages for your local user, and does not need extra permissions to write to the system directories:

  pip install --user numpy scipy matplotlib ipython sympy pandas

NumPy

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- The fundamental package for scientific computing with Python. It contains:
  - A powerful N-dimensional array (ndarray) object.
  - Sophisticated (broadcasting/universal) functions.
  - Tools for integrating C/C++ and Fortran code.
  - Useful linear algebra, Fourier transform, and random number capabilities.
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

NumPy & Class ndarray

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> import numpy
>>> import numpy as np
>>> from numpy import *
```

docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.html

```
>>> dir(numpy)
>>> dir(numpy.ndarray)
>>> help(dtype)
>>> help(ndarray)
>>> help(ndarray.dtype)
```

NumPy Data Types

- NumPy supports a much greater variety of numerical types than Python does:
  - bool_
  - int_, intc, intp, u/int8, u/int16, u/int32, u/int64
  - float_, float16, float32, float64
  - complex_, complex64, complex128
- NumPy numerical types are instances of dtype (data-type) objects:
  - numpy.dtype(object, align, copy)

NumPy Data Types

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
array([0, 1, 2], dtype=uint8)
>>> z.dtype
dtype('uint8')
```

NymPy Arrays

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- The main feature of NumPy is an array object:
  - Arrays can be N-dimensional.
  - Array elements have to be the same type.
  - Array elements can be accessed, sliced, and manipulated in the same way as the lists.
  - The number of elements in the array is fixed.
  - Shape of the array can be changed.
- Built-in NumPy array creation:
  - array(), arange(), ones(), zeros(), …

NumPy Arrays

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> np.array([2,3,1,0])
array([2, 3, 1, 0])
>>> np.array([[1,2.0],[0,0],(1+1j,3.)])
array([[ 1.+0.j, 2.+0.j],
       [ 0.+0.j, 0.+0.j],
       [ 1.+1.j, 3.+0.j]])

>>> np.zeros((2,3))
array([[ 0., 0., 0.],
       [ 0., 0., 0.]])
```

NumPy Arrays

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2,10,dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8.,
9.])
>>> np.arange(2,3,0.1)
array([ 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
2.6, 2.7, 2.8, 2.9])
```

NumPy Arrays

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> a = np.arange(3)
>>> print(a)
[0, 1, 2]
>>> a
array([0, 1, 2])
>>> np.arange(9).reshape(3,3)
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]]
>>> np.arange(8).reshape(2,2,2)
array([[[0, 1],
        [2, 3]],

       [[4, 5],
        [6, 7]]])
```

NumPy Arrays

- linspace(start, stop[, num, endpoint, retstep, dtype]) – creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values.

- random.random([size]) – creates arrays with random floats over the interval [0.,1.).
- random.randint(low[, high, size, dtype]) – creates arrays with random integers from low (inclusive) to high (exclusive).

NumPy Arrays

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> np.linspace(1.,4.,6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])


>>> np.random.random((2,3))
array([[ 0.96826, 0.30919, 0.58381],
       [ 0.56865, 0.33730, 0.41241]])
>>> np.random.randint(1,7,(2,3))
array([[2, 4, 5, 2, 3, 6],
       [6, 3, 1, 4, 1, 5]]
```

NumPy Linear Algebra

- All linear algebra routines expect an object that can be converted into a 2-dimensional array.
- The output is also a two-dimensional array.
  - dot(a, b[, out]) – dot product of two arrays.
  - trace(a[, offset, axis1, axis2, dtype, out]) – returns the sum along diagonals of the array.
  - inv(a) – computes the inverse of a matrix.
  - eig(a) – eigenvalues and right eigenvectors of a square array.
  - solve(a, b) – solves a linear matrix equation, or system of linear scalar equations.

NumPy LinAlg

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> from numpy import *
>>> from numpy.linalg import *
>>> a = array([[1.0,2.0],[3.0,4.0]])
>>> print(a)
array([[ 1., 2.],
       [ 3., 4.]])
>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])
>>> inv(a) # inverse
array([[-2. , 1. ],
       [ 1.5, -0.5]])
```

NumPy LinAlg

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> u = eye(2) # unit 2x2 matrix; "eye" ~ "I"
>>> u
array([[ 1., 0.],
       [ 0., 1.]])
>>> trace(u)
2.0
>>> j = array([[0.0,-1.0],[1.0,0.0]])
>>> dot(j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
>>> eig(j) # get eigenvalues & eigenvectors
(array([ 0.+1.j, 0.-1.j]),
 array([[ 0.70710+0.j,      0.70710+0.j],
        [ 0.00000-0.70710j, 0.00000+0.70710j]]))
```

NumPy LinAlg

- There are two alcohol solutions: 50% & 90%.
- How many gallons of each solution to be mixed to get 10 gallons of 74% alcohol solution?

$x_1+x_2=10$, $0.5x_1+0.9x_2=0.74*10=7.4 \rightarrow AX=Y$

```
>>> A = array([[1.0,1.0],[0.5,0.9]])
>>> Y = array([[10.0],[7.4]])
>>> solve(A, Y) # solve linear equations
array([[ 4.],
       [ 6.]])
```

NumPy LinAlg

- A drone flying with the wind could cover 60 miles in 2 hours.
- The return trip against the wind took 2.5 hours.
- How fast was the drone?
- What was the air speed?

Trip	Rate	Time	Distance
With wind	d + w	2	60
Against wind	d - w	2.5	60

```
>>> A = array([[2.0,2.0],[2.5,-2.5]])
>>> Y = array([[60.0],[60.0]])
>>> solve(A, Y) # solve linear equations
array([[ 27.],
       [  3.]])
```

NymPy Matrix Versus Array

- NumPy matrices are strictly 2-dimensional, while NumPy arrays (ndarrays) are N-dimensional.
- Matrix objects are a subclass of ndarray, so they inherit all the attributes and methods of ndarrays.
- The main advantage of NumPy matrices is that they provide a convenient notation for matrix multiplication:
  - e.g. If A and B are matrices, then A*B is their matrix product.

NymPy Matrices

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> A = matrix('1.0 2.0; 3.0 4.0')
>>> A
matrix([[ 1., 2.],
        [ 3., 4.]])
>>> type(A)
<class 'numpy.matrixlib.defmatrix.matrix'>
>>> Y = matrix('5.0; 7.0')
>>> print(A.I) # inverse
[[-2.   1. ]
 [ 1.5 -0.5]]
>>> print(A.I*Y) # multiplication
[[-3.],
 [ 4.]])
>>> solve(A, Y) # solving linear equations
matrix([[-3.],
        [ 4.]])
```

SciPy

- A collection of mathematical algorithms and convenience functions built on the NumPy extension of Python.
- An interactive Python session for manipulating and visualizing data.
- A data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.

```
>>> from numpy import *
>>> from matplotlib import *
>>> from matplotlib.pyplot import *
```

SciPy Sub-Modules

- **cluster** – clustering algorithms
- **integrate** – integration and ordinary differential equation solvers
- **interpolate** – interpolation and smoothing splines
- **io** – input and output
- **linalg** – linear algebra
- **optimize** – optimization and root-finding routines
- **stats** – statistical distributions and functions

```
>>> from scipy import linalg, optimize
>>> from scipy import *
```

SciPy Clustering

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

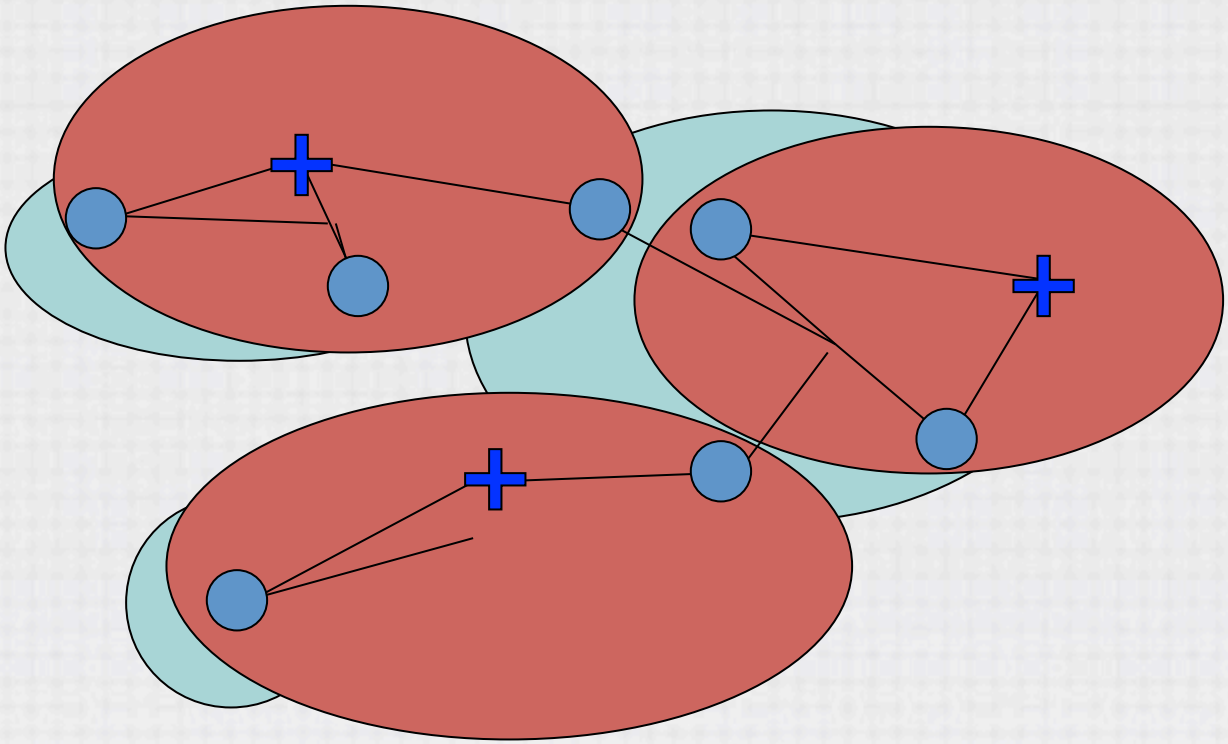
```
>>> from scipy.cluster.vq import *
```

- Clustering – finds clusters and cluster centers in a set of unlabeled data.
- Intuitively, a cluster comprises a group of data points whose inter-point distances are small compared to the distances to points outside of the cluster.

SciPy *k*-means Clustering

- scipy.cluster.vq
  - **kmeans(obs, k_or_guess[, iter, thresh, ...])** – perform k-means on a set of observation vectors forming k clusters
  - **kmeans2(data, k[, iter, thresh, minit, ...])** – classify a set of observations into k clusters using the k-means algorithm
- Given an initial set of *k* centers, the *k*-means algorithm alternates the two steps:
  - For each center, we identify the subset of training points (its cluster) that is closer to it than any other center.
  - The means of each feature for the data points in each cluster are computed, and this mean vector becomes the new center for that cluster.

k-means Clustering (*k*=3)



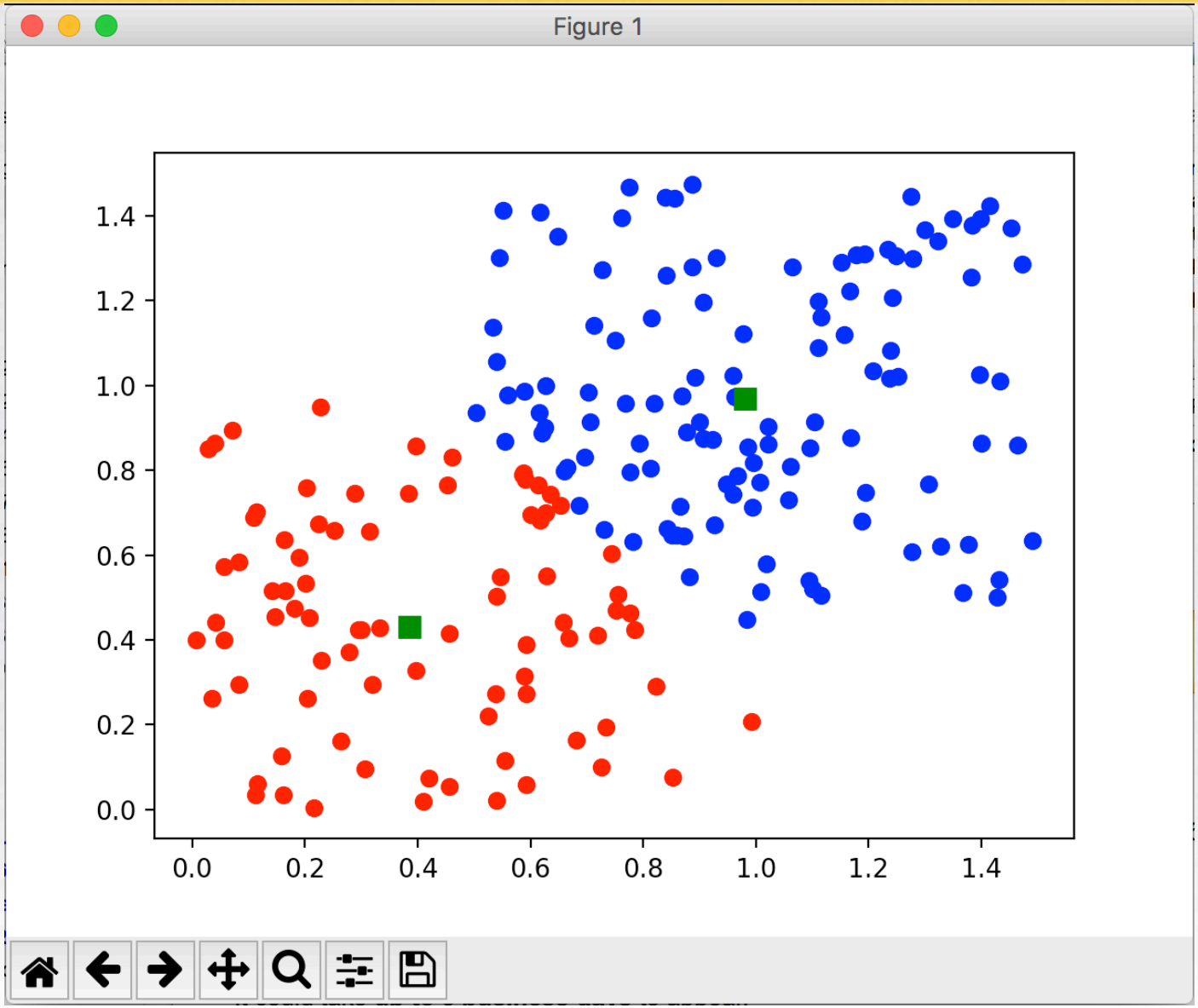
SciPy 2-means Clustering

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
from pylab import *
from numpy import *
from numpy.random import *
from scipy.cluster.vq import *

# data generation
data = vstack((rand(100,2)+array([.5,.5]),rand(100,2)))
# computing k-means with k = 2 (2 clusters)
centroids,_ = kmeans(data,2)
# assign each sample to a cluster
index,_ = vq(data,centroids)
# some plotting using numpy's logical indexing
plot(data[index==0,0],data[index==0,1],'or',
     data[index==1,0],data[index==1,1],'ob')
plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
show()
```

SciPy 2-means Clustering



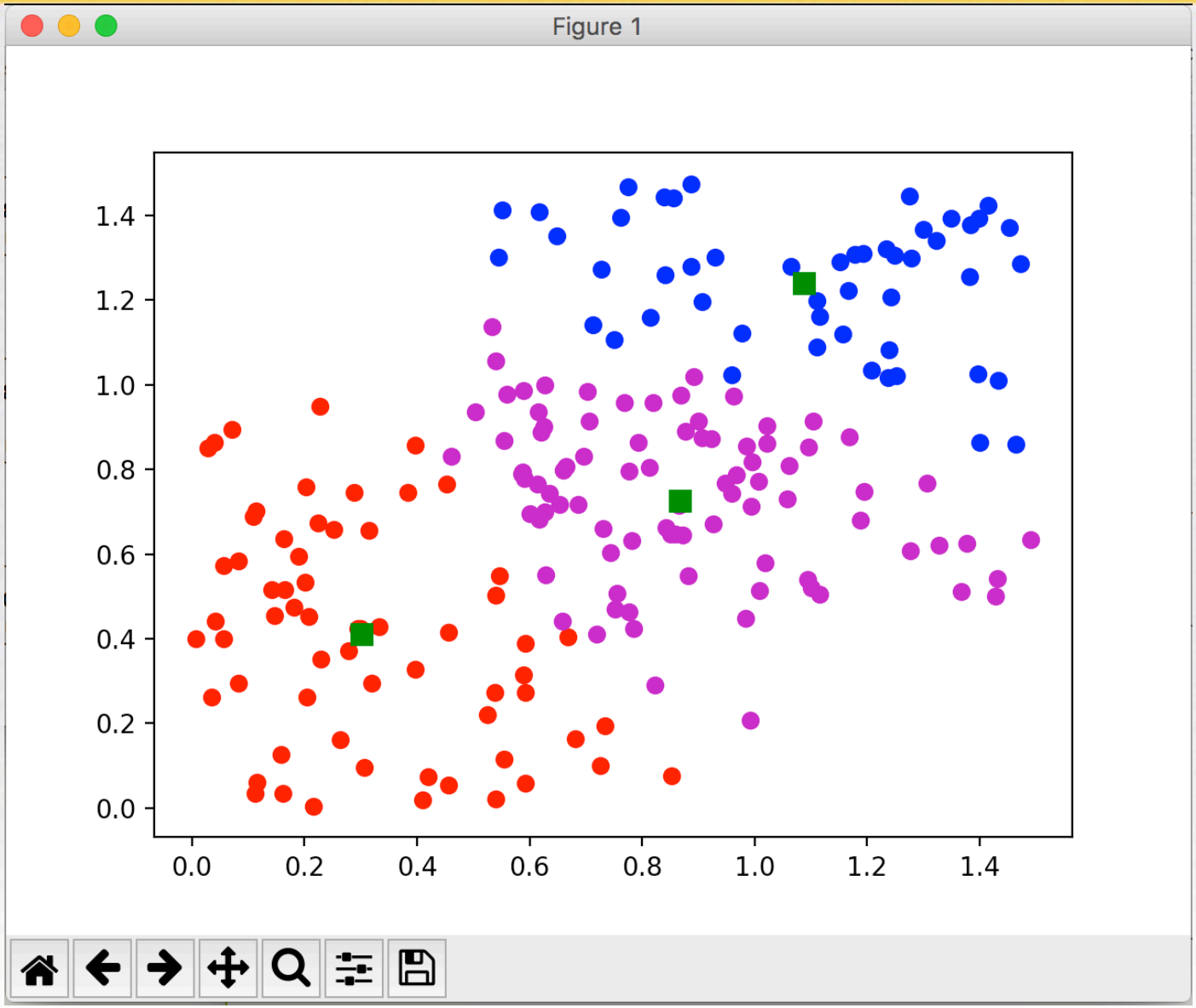
SciPy *3*-means Clustering

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
...
# data generation
data = vstack((rand(100,2)+array([.5,.5]),rand(100,2)))
# computing k-means with k = 3 (3 clusters)
centroids,_ = kmeans(data,3)
# assign each sample to a cluster
index,_ = vq(data,centroids)
# some plotting using numpy's logical indexing
plot(data[index==0,0],data[index==0,1],'or',
     data[index==1,0],data[index==1,1],'ob',
     data[index==2,0],data[index==2,1],'om')
plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
show()
```

SciPy *3*-means Clustering

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS



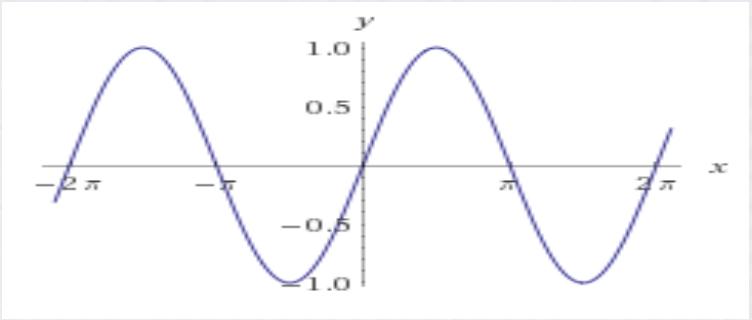
SciPy Integration

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

- Methods for Integrating Functions given a function object:
  - **quad** – general purpose integration
  - **dblquad** – general purpose double integration
  - **tplquad** – general purpose triple integration
  - **fixed_quad** – integrate f(x) using Gaussian quadrature
  - **quadrature** – Integrate with tolerance using Gaussian quadrature
  - **romberg** – integrate f(x) using Romberg integration
- Methods for I.F. given a fixed set of samples:
  - **trapz** – use trapezoidal rule to compute integral
  - **cumtrapz** – use trapezoidal rule to cumulatively compute integral
  - **simps** – use Simpson's rule to compute integral
  - **romb** – use Romberg Integration to compute integral

SciPy Integration

- **np.sin** defines the sine function
- Integral *x=0* to *x=π* using **quad**



$$\int \sin(x)\, dx$$

```
>>> result = scipy.integrate.quad(np.sin,
0,np.pi)
>>> print(result)
(2.0, 2.2204460492503130e-14)
# 2 with a very small error margin!
>>> result = scipy.integrate.quad(np.sin,-
np.inf,+np.inf)
>>> print(result)
(0.0, 0.0) # Integral does not converge
```

SciPy Optimization

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
>>> from scipy.optimize import *
>>> help(scipy.optimize)
```

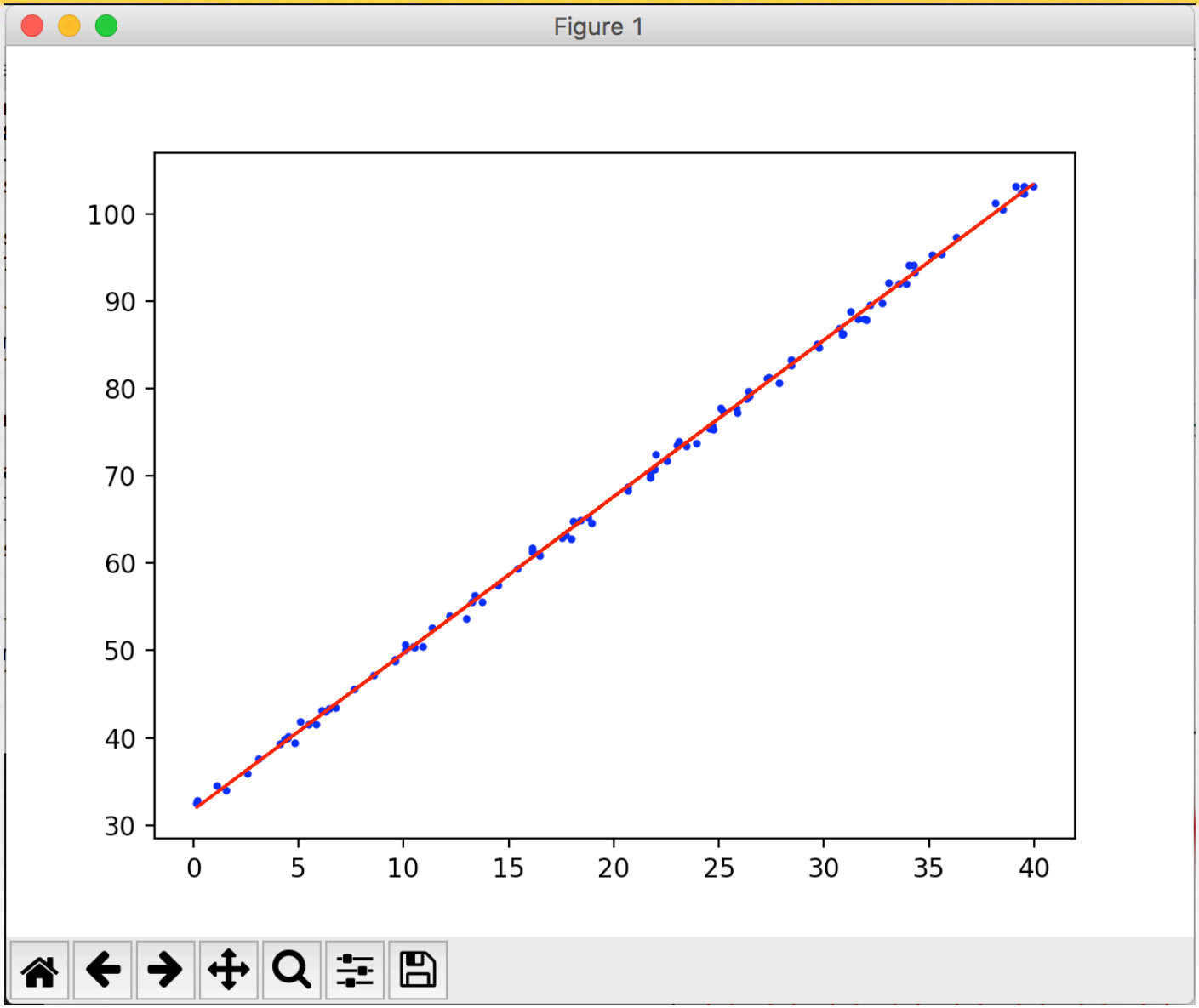
- Provides several commonly used optimization algorithms:
  - Unconstrained and constrained minimization of multivariate scalar functions (minimize) using BFGS, Nelder-Mead Simplex, Newton Conjugate Gradient, COBYLA, SLSQP, …
  - Global (brute-force) optimization routines (e.g. basinhopping, differential_evolution)
  - Least-squares minimization (least_squares) and curve fitting (curve_fit) algorithms
  - Scalar univariate functions minimizers (minimize_scalar) and root finders (newton)
  - Multivariate equation system solvers (root) using hybrid Powell, Levenberg-Marquardt, large-scale Newton-Krylov, …

SciPy Curve Fitting

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
from pylab import *
from numpy import *
from numpy.random import *
from scipy.optimize import *
# lenear regression
def linreg(x,a,b):
    return a*x+b
# data generation
input = randint(0,40,100)
x = input+rand(100)
y = (input*1.8+32)+rand(100)
# curve fitting
attributes,variances = curve_fit(linreg,x,y)
# estimated y
y_modeled = x*attributes[0]+attributes[1]
# plot true and modeled results
plot(x,y,'ob',markersize=2)
plot(x,y_modeled,'-r',linewidth=1)
show()
```

SciPy Curve Fitting



SciPy Linear Regression

UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

```
from pylab import *
from numpy import *
from numpy.random import *
from scipy.stats import *
# data generation
input = randint(0,40,100)
x = input+rand(100)
y = (input*1.8+32)+rand(100)
# linear regression
slope,intercept,r_value,p_value,slope_std_error = stats.linregress(x,y)
# estimated y
y_modeled = x*slope+intercept
# plot true and modeled results
plot(x,y,'ob',markersize=2)
plot(x,y_modeled,'-r',linewidth=1)
show()
```

SciPy Linear Regression