Data Processing and Analysis in Python
Lecture 17
Scientific Computing and SciPy

UNIVERSITY OF MARYLAND

ROBERT H. SMITH
SCHOOL OF BUSINESS

DR. ADAM LEE

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
- https://www.scipy.org/

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

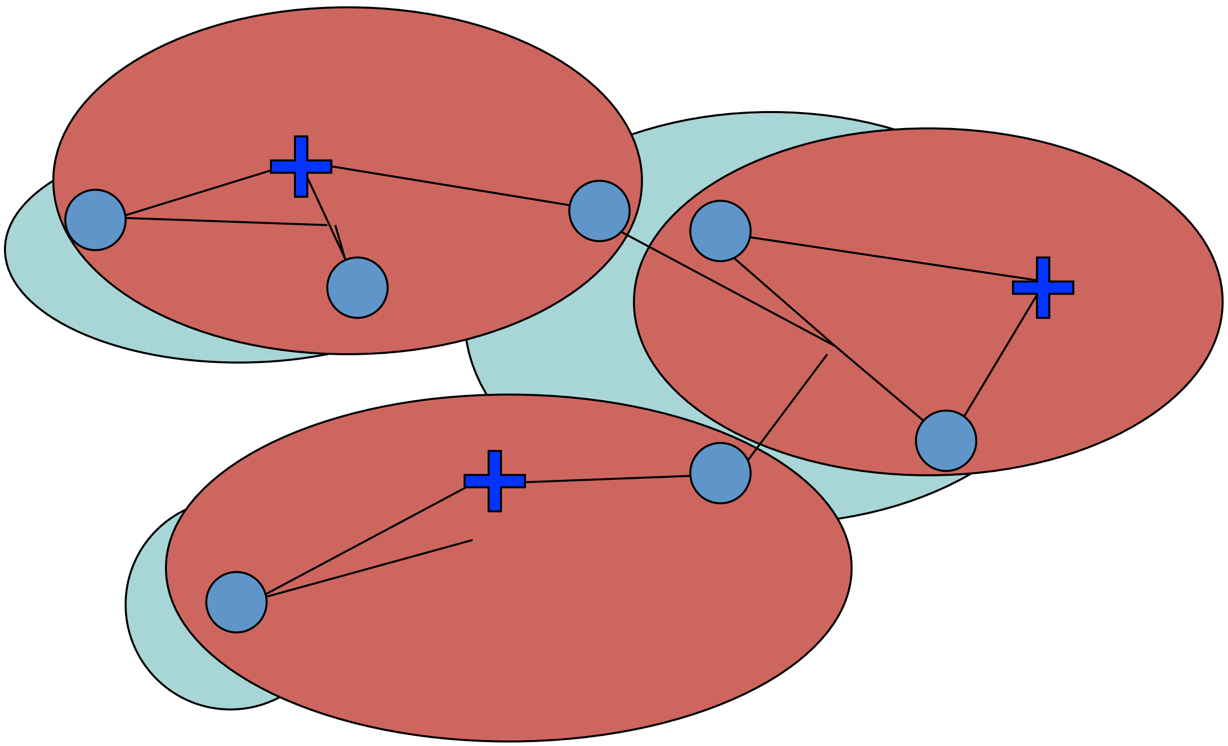
```
>>> from scipy.cluster.vq import *
```

- **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

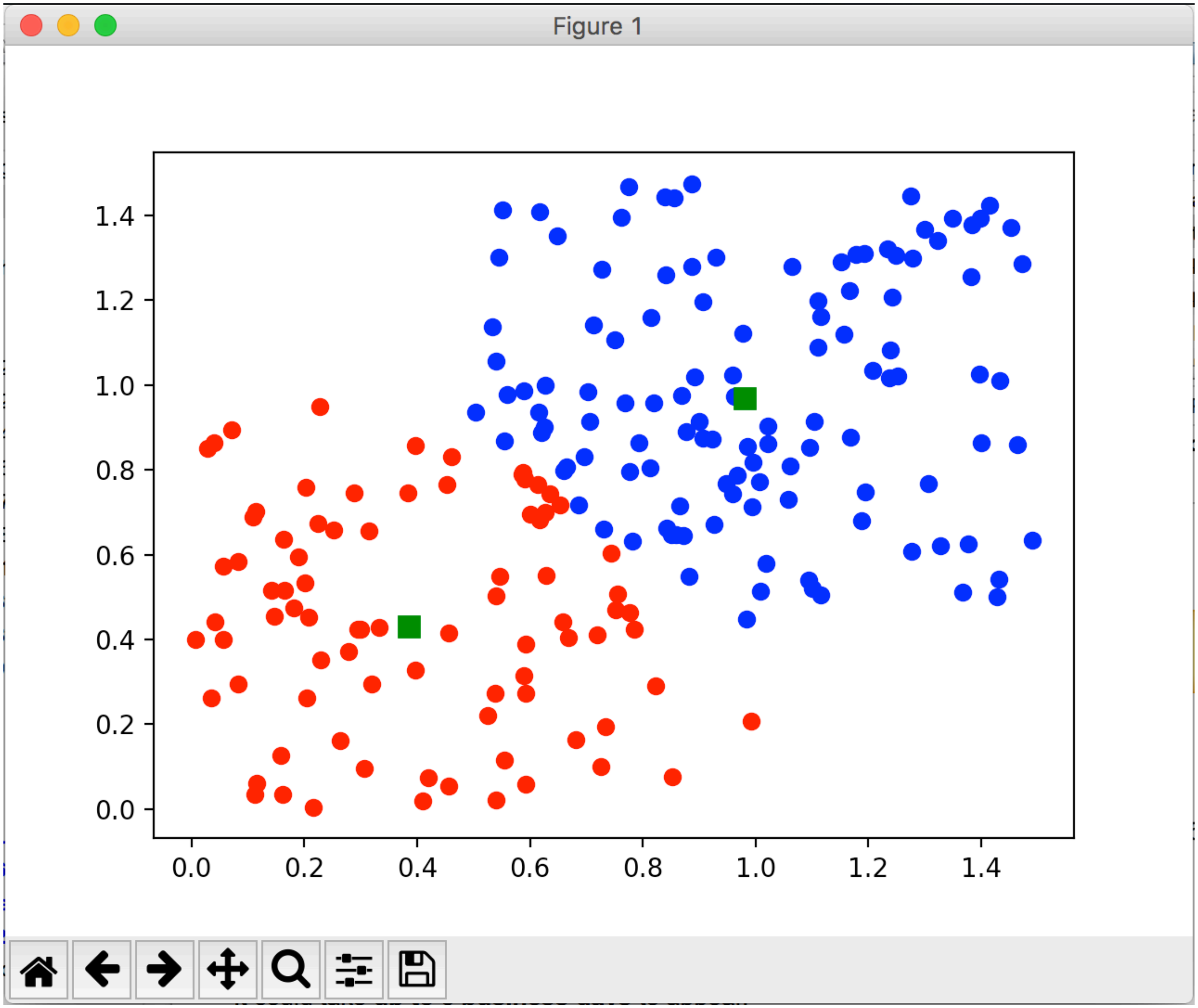
```
from numpy.random import *
from scipy.cluster.vq import *
from pylab import *

# 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)
# plot different color for each cluster by its index
plot(data[index==0,0],data[index==0,1],'or')
plot(data[index==1,0],data[index==1,1],'ob')
plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
show()
```

SciPy 2-means Clustering

```
from numpy import array, vstack
from numpy.random import rand
# data generation
data = vstack((rand(100,2)+array([.5,.5]),rand(100,2)))
from scipy.cluster.vq import kmeans, vq
# compute k-means with k = 2 (2 clusters)
centroids,_ = kmeans(data,2)
# assign each sample to a cluster
index,_ = vq(data,centroids)
from matplotlib.pyplot import plot, show
# plot different color for each cluster by its index
plot(data[index==0,0],data[index==0,1],'or')
plot(data[index==1,0],data[index==1,1],'ob')
plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
show()
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

SciPy 2-means Clustering

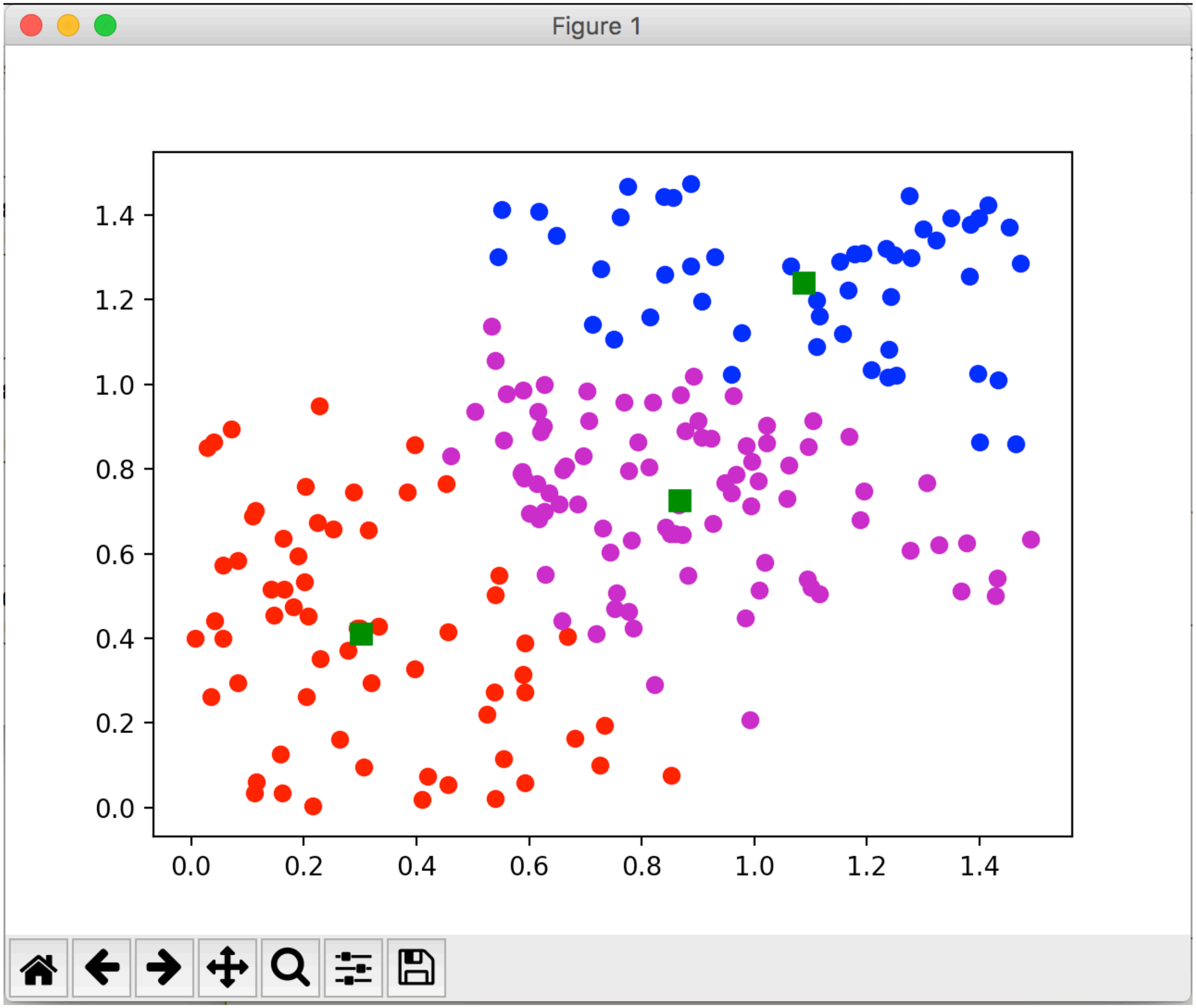


SciPy 3-means Clustering

```
from numpy import array, vstack
from numpy.random import rand
# data generation
data = vstack((rand(100,2)+array([.5,.5]),rand(100,2)))
from scipy.cluster.vq import kmeans, vq
# compute k-means with k = 3 (3 clusters)
centroids,_ = kmeans(data,3)
# assign each sample to a cluster
index,_ = vq(data,centroids)
from matplotlib.pyplot import plot, show
# plot different color for each cluster by its index
plot(data[index==0,0],data[index==0,1],'or')
plot(data[index==1,0],data[index==1,1],'ob')
plot(data[index==2,0],data[index==2,1],'om')
plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
show()
```


UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

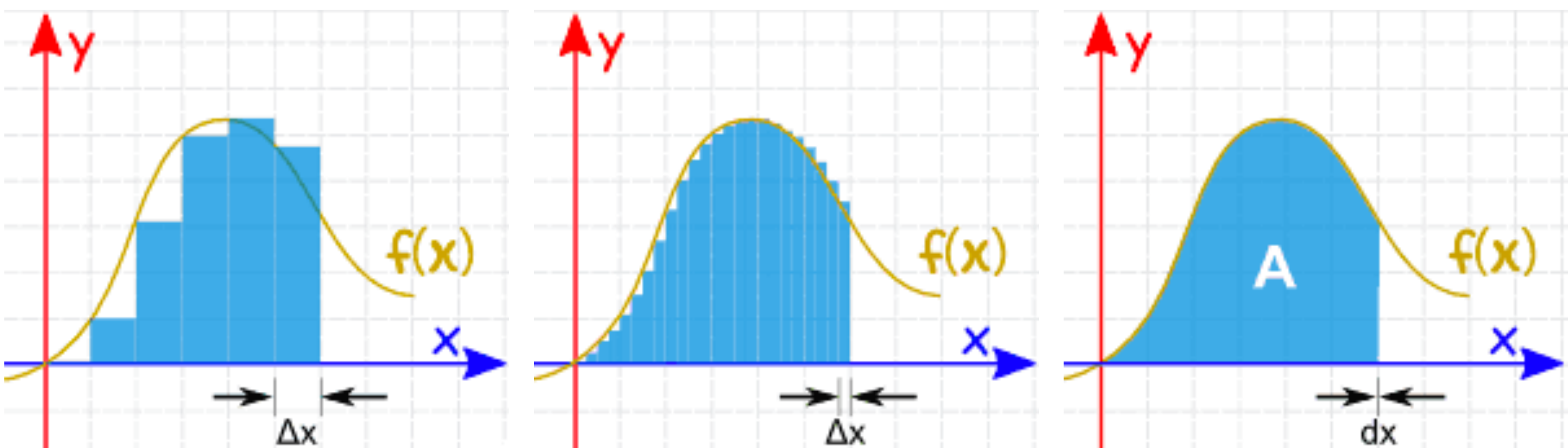
SciPy 3-means Clustering



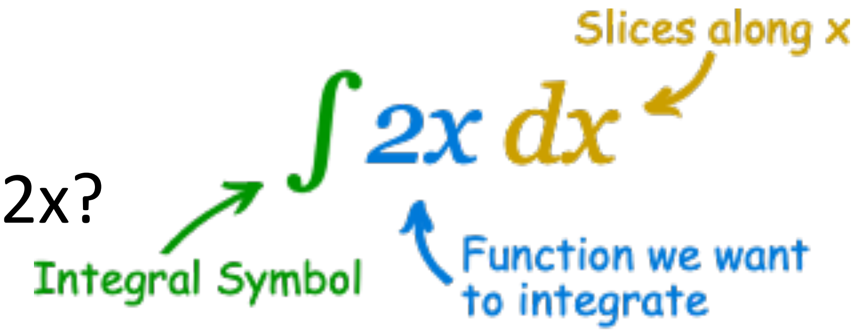
UNIVERSITY OF MARYLAND
ROBERT H. SMITH SCHOOL OF BUSINESS

Integration

- Can be used to find areas, volumes, central points, etc.
- A way of adding slices to find the whole
- Example – find the area under the curve of a function:
  - Calculate the function at a few points and add up slices of width Δx
  - We can make Δx a lot smaller and add up many small slices
  - As Δx approaches zero as dx, the area approaches the true answer



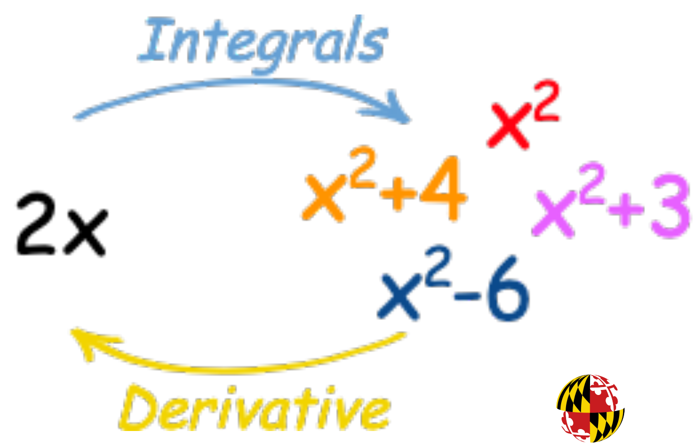
Integration



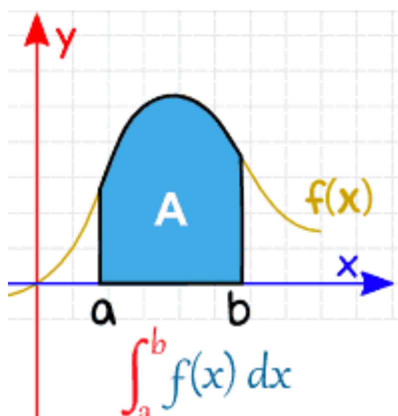
- Example: What is an integral of 2x?
- Notation:
  - The symbol is a stylish "S" for "Sum"
  - Follows by the "integrand" function

$$\int 2x\,dx = x^2 + C$$

  - Finish with "dx" to mean width in the x direction approaches zero
  - "C" is the "Constant of Integration", because many functions whose derivative is 2x



- Definite integrals:

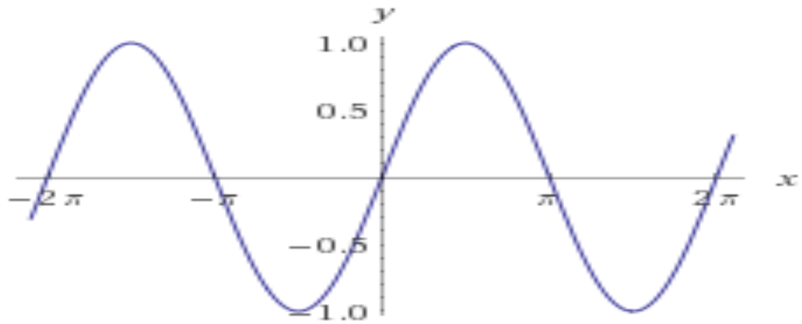


SciPy Integration

- 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**



$$\int_0^\pi \sin(x)\, dx = 2$$

```
>>> result = scipy.integrate.quad(np.sin,
0,np.pi)
>>> print(result)
(2.0, 2.220446049250313e-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
```

Optimization

- **Optimization** for minimizing or maximizing objective function
  - Possibly subject to constraints
  - To solver linear programing, curve fitting, root finding, etc.
- Example − minimum or maximum value of a function
  - To find the minimum value of the objective function $x^2 + 1$, when choosing x from real numbers
  - The minimum value is 1, occurring at x = 0

$$\min_{x \in \mathbb{R}} \; (x^2 + 1)$$

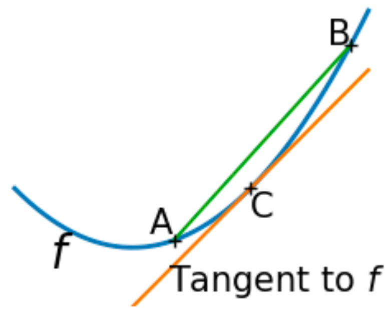
- Example − optimal input arguments
  - To find argument x in the interval (−∞,−1] that minimizes the objective function $x^2 + 1$
  - The answer is x = −1, since x = 0 is infeasible

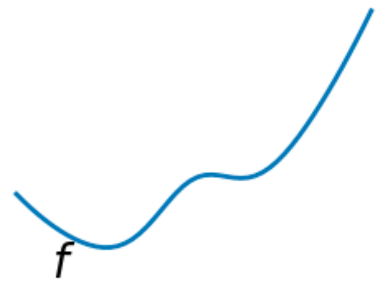
$$\arg\min_{x \in (-\infty, -1]} \; x^2 + 1$$

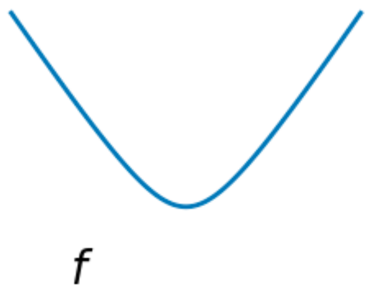
SciPy Optimization

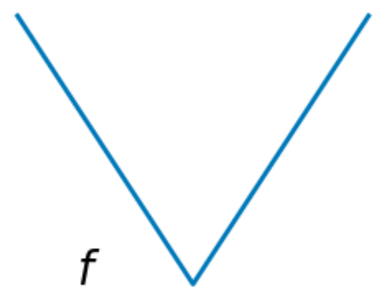
```
>>> from scipy.optimize import *
>>> help(scipy.optimize)
```

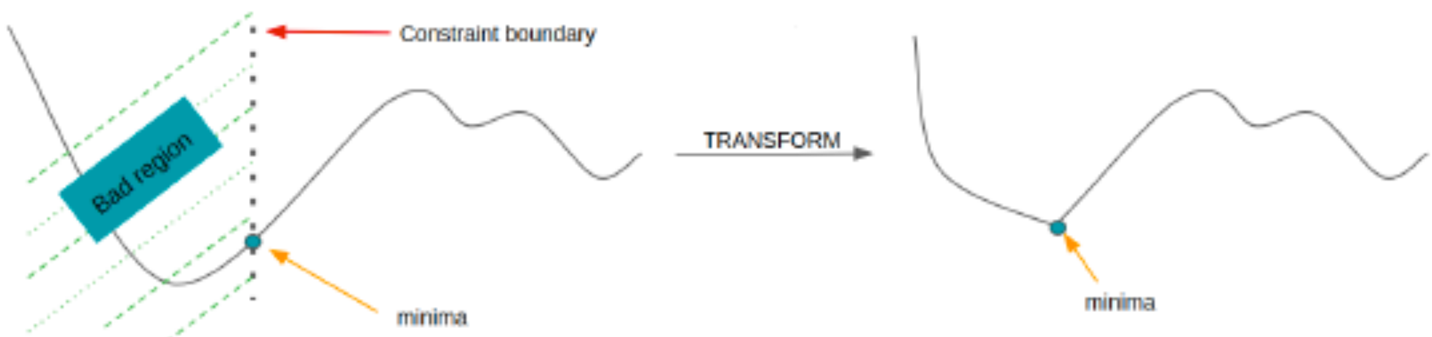
- Objective function:
  - Convex versus non-convex
  - Smooth versus non-smooth
  - Noisy versus exact-cost
  - Constrained versus unconstrained











SciPy Optimization

- **minimize_scalar** − scalar univariate function optimization
- **minimize** – local multivariate function optimization:
  - **LinearConstraint**, **NonlinearConstraint** − constraints
  - **Bounds** − simple bound constraints
- **basinhopping**, **bruce**, **differential_evolution** − global optimization

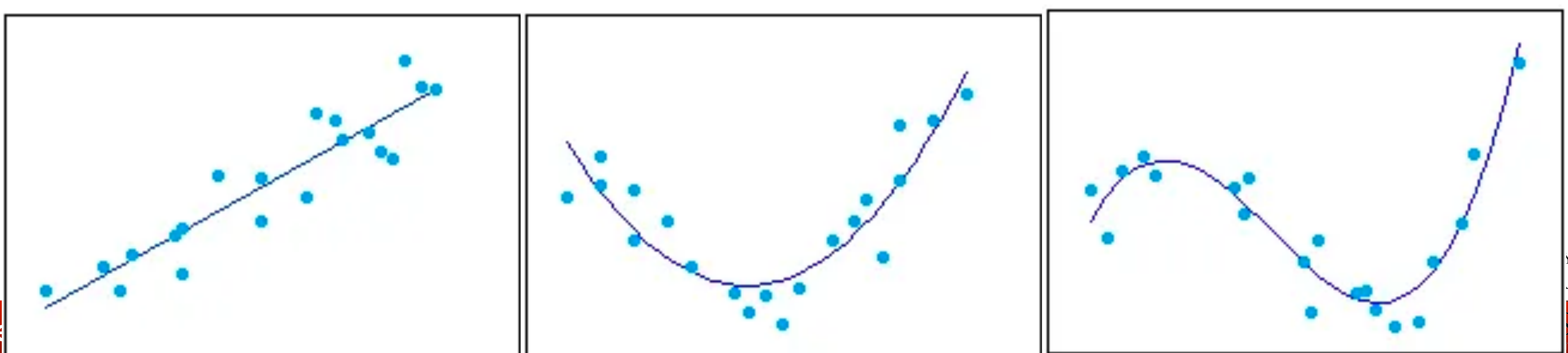
- **least_squares** − nonlinear least-squares
- **lsq_linear** − linear least-squares
- **curve_fit** − curve fitting algorithms

- **root_scalar**, **bisect**, **newton** − root finding for scalar functions
- **root** − multivariate equation system root finding

- **linprog** − linear programming

Curve Fitting – Regression Analysis

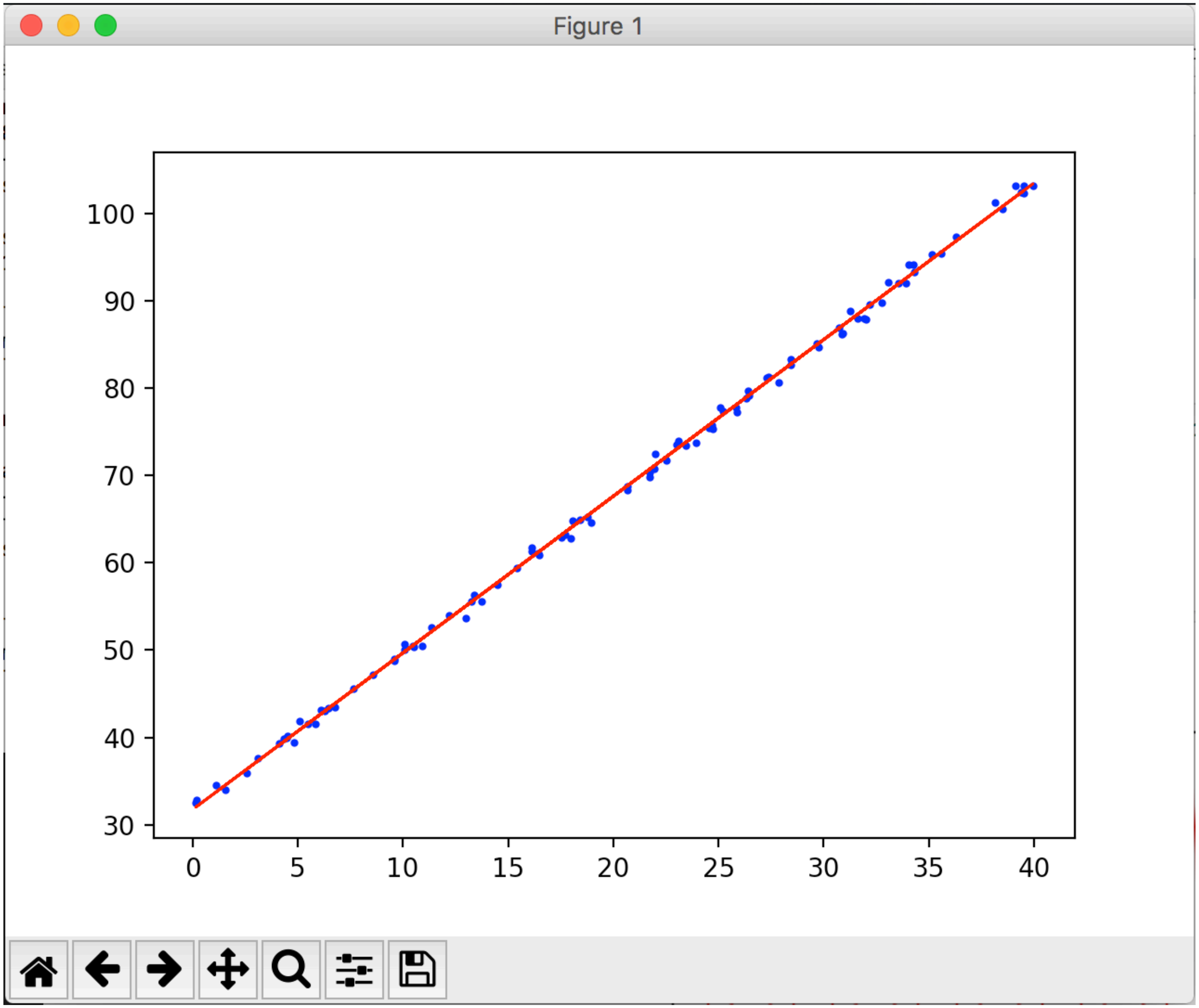
- Model the best fit to the specific curves in dataset:
  - With one independent variable, the curvature uses a fitted line
  - With multiple regression, curved relationships are not so apparent
- To determine the univariate polynomial term to include, simply count the number of bends in the line:
  - The most common method is the linear model.
  - Quadratic terms model one bend
  - Cubic terms model two bends



SciPy Curve Fitting

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

SciPy Curve Fitting



SciPy Statistics

```
>>> from scipy.stats import *
```

- Random variable class meant for subclassing:
  - **rv_continuous** – generic continuous random variable class
  - **rv_discrete** – generic discrete random variable class
- Probability distributions:
  - **norm** – a normal continuous random variable
  - **pareto** – a Pareto continuous random variable
  - **uniform** – a uniform continuous random variable
  - **binom** – a binomial discrete random variable
  - **geom** – a geometric discrete random variable
  - **poisson** – a Poisson discrete random variable
- Statistical functions:
  - **linregress** – linear least-squares regression for 2 measurements

SciPy Linear Regression

- p, residuals, rank, singular_values, rcond, v = **numpy.polyfit**(x, y, deg, rcond=, full=, w=, cov=)
- x, residuals, rank, s = **numpy.linalg.lstsq**(a, b, rcond=)
- x, residuals, rank, s = **scipy.linalg.lstsq**(a, b, cond=, overwrite_a=, ...)
- slope, intercept, r_value, p_value, std_err = **scipy.stats.linregress**(x, y=)
- popt, pcov = **scipy.optimize.curve_fit**(f, xdata, ydata, p0=, sigma=, ...)

SciPy Linear Regression

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

SciPy Linear Regression