SciPy

Lecture 07

Dr. Colin Rundel

What is SciPy

> Fundamental algorithms for scientific computing in Python

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

SciPy vs NumPy

> In an ideal world, NumPy would contain nothing but the array data type and the most basic operations: indexing, sorting, reshaping, basic elementwise functions, etc. All numerical code would reside in SciPy. However, one of NumPy’s important goals is compatibility, so NumPy tries to retain all features supported by either of its predecessors. Thus, NumPy contains some linear algebra functions and Fourier transforms, even though these more properly belong in SciPy. In any case, SciPy contains more fully-featured versions of the linear algebra modules, as well as many other numerical algorithms. If you are doing scientific computing with Python, you should probably install both NumPy and SciPy. Most new features belong in SciPy rather than NumPy.

From scipy.org/faq/

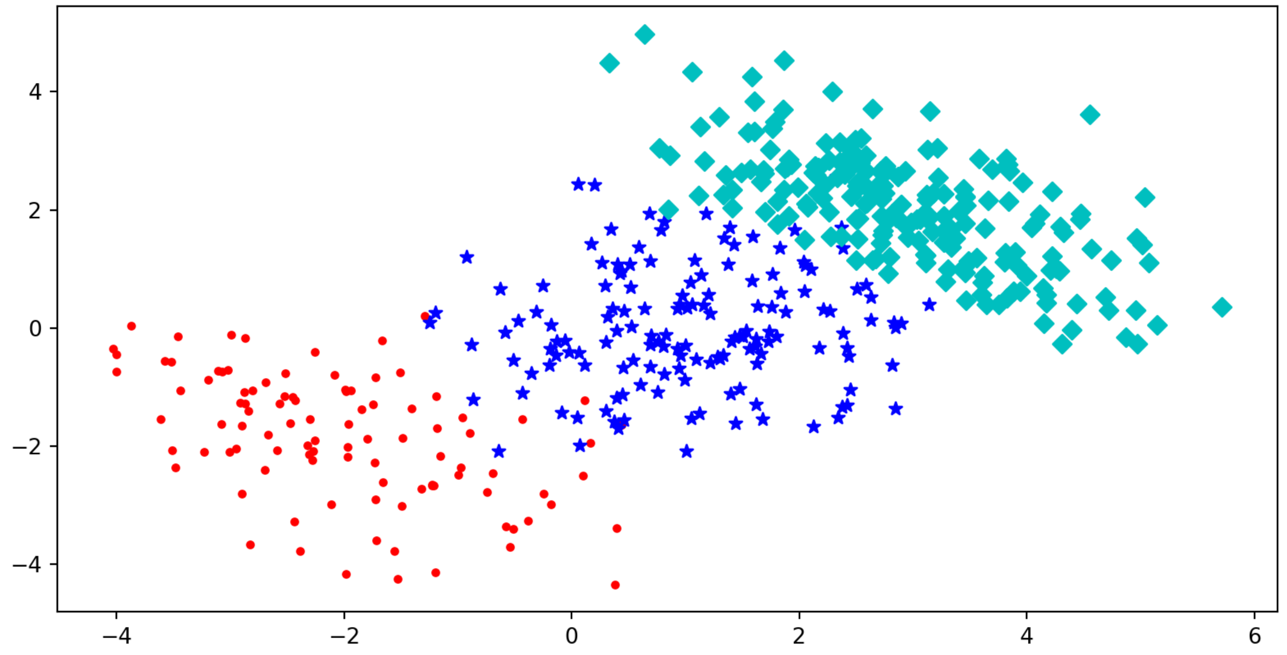
Example 1
k-means clustering

Data

```python
rng = np.random.default_rng(seed = 1234)

cl1 = rng.multivariate_normal([-2,-2], [[1,-0.5],[-0.5,1]], size=100)
cl2 = rng.multivariate_normal([1,0], [[1,0],[0,1]], size=150)
cl3 = rng.multivariate_normal([3,2], [[1,-0.7],[-0.7,1]], size=200)

pts = np.concatenate((cl1,cl2,cl3))
```



k-means clustering

```python
from scipy.cluster.vq import kmeans
ctr, dist = kmeans(pts, 3)
```

```python
ctr
```

```
array([[-2.0396, -1.8566],
       [ 0.9112, -0.1872],
       [ 2.864 ,  1.954 ]])
```

```python
dist
```

```
1.2209235923868729
```

```python
cl1.mean(axis=0)
```

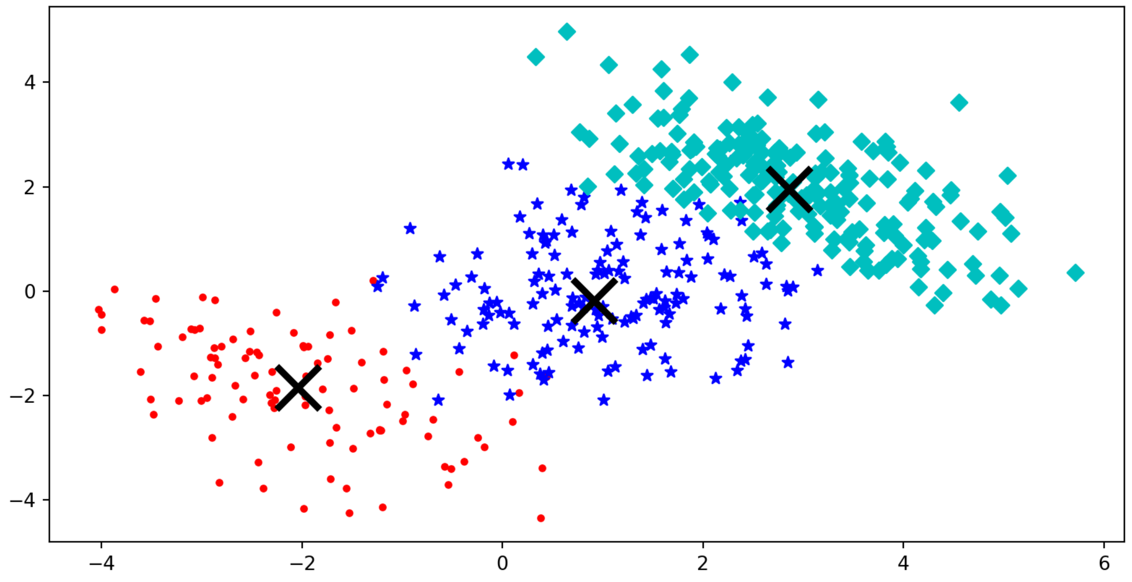
```
array([-2.0047, -1.8728])
```

```python
cl2.mean(axis=0)
```

```
array([1.0385, 0.0142])
```

```python
cl3.mean(axis=0)
```

```
array([2.9464, 2.0251])
```


4
2
0
−2
−4
−4
−2
0
2
4
6

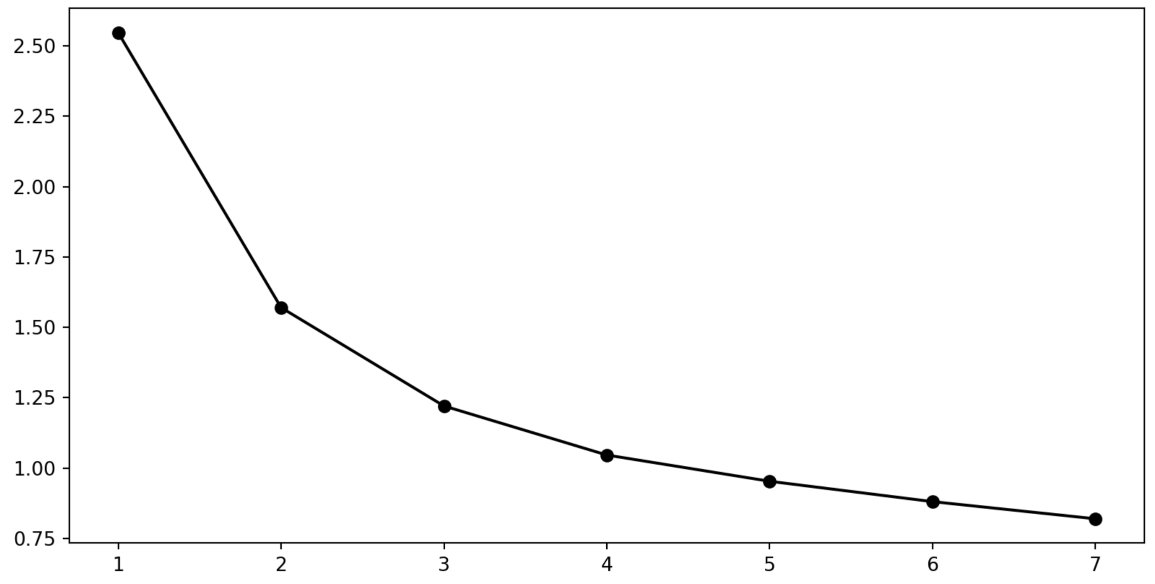
k-means distortion plot

> The mean (non-squared) Euclidean distance between the observations passed and the centroids generated.

```python
ks = range(1,8)
dists = [kmeans(pts, k)[1] for k in ks]
```

```python
np.array(dists).reshape(-1)
```

```
array([2.547 , 1.5701, 1.2204, 1.0461, 0.9528, 0.8804, 0.8195])
```



Example 2
Numerical integration

Basic functions

For general numeric integration in 1D we use `scipy.integrate.quad()`, which takes as arguments the function to be integrated and the lower and upper bounds of the integral.

```
from scipy.integrate import quad
```

```
quad(lambda x: x, 0, 1)
```

```
(0.5, 5.551115123125783e-15)
```

```
quad(np.sin, 0, np.pi)
```

```
(2.0, 2.220446049250313e-14)
```

```
quad(np.sin, 0, 2*np.pi)
```

```
(2.0329956258200796e-16, 4.3998892617845996e-14)
```

```
quad(np.exp, 0, 1)
```

```
(1.7182818284590453, 1.9076760487502457e-14)
```

Normal PDF

The PDF for a normal distribution is given by,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$$

```python
def norm_pdf(x, μ, σ):
  return (1/(σ * np.sqrt(2*np.pi))) * np.exp(-0.5 * ((x - μ)/σ)**2)
```

```python
norm_pdf(0, 0, 1)
```

```
0.3989422804014327
```

```python
norm_pdf(np.Inf, 0, 1)
```

```
0.0
```

```python
norm_pdf(-np.Inf, 0, 1)
```

```
0.0
```

Checking the PDF

We can check that we've implemented a valid pdf by integrating the pdf from $-\inf$ to $\inf$,

```python
quad(norm_pdf, -np.inf, np.inf)
```

```
Error: TypeError: norm_pdf() missing 2 required positional arguments: 'μ' and
'σ'
```

```python
quad(lambda x: norm_pdf(x, 0, 1), -np.inf, np.inf)
```

```
(0.9999999999999997, 1.0178191380347127e-08)
```

```python
quad(lambda x: norm_pdf(x, 17, 12), -np.inf, np.inf)
```

```
(1.0000000000000002, 4.113136862574909e-09)
```

Truncated normal PDF

$$f(x) = \begin{cases} \frac{c}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right), & \text{for } a \le x \le b \\ 0, & \text{otherwise.} \end{cases}$$

```python
def trunc_norm_pdf(x, μ=0, σ=1, a=-np.inf, b=np.inf):
  if (b < a):
      raise ValueError("b must be greater than a")

  x = np.asarray(x).reshape(-1)
  full_pdf = (1/(σ * np.sqrt(2*np.pi))) * np.exp(-0.5 * ((x - μ)/σ)**2)
  full_pdf[(x < a) | (x > b)] = 0
  return full_pdf
```

Testing trunc_norm_pdf

```
trunc_norm_pdf(0, a=-1, b=1)
```

```
array([0.3989])
```

```
trunc_norm_pdf(2, a=-1, b=1)
```

```
array([0.])
```

```
trunc_norm_pdf(-2, a=-1, b=1)
```

```
array([0.])
```

```
trunc_norm_pdf([-2,1,0,1,2], a=-1, b=1)
```

```
array([0.    , 0.242 , 0.3989, 0.242 , 0.    ])
```

```
quad(lambda x: trunc_norm_pdf(x, a=-1, b=1), -np.inf, np.inf)
```

```
(0.682689492137086, 2.0147661317082566e-11)
```

```
quad(lambda x: trunc_norm_pdf(x, a=-3, b=3), -np.inf, np.inf)
```

```
(0.9973002039367396, 7.451935936375609e-09)
```

Fixing trunc_norm_pdf

```python
def trunc_norm_pdf(x, μ=0, σ=1, a=-np.inf, b=np.inf):
  if (b < a):
      raise ValueError("b must be greater than a")
  x = np.asarray(x).reshape(-1)

  nc = 1 / quad(lambda x: norm_pdf(x, μ, σ), a, b)[0]

  full_pdf = nc * (1/(σ * np.sqrt(2*np.pi))) * np.exp(-0.5 * ((x - μ)/σ)**2)
  full_pdf[(x < a) | (x > b)] = 0

  return full_pdf
```

```python
trunc_norm_pdf(0, a=-1, b=1)
```

```
array([0.5844])
```

```python
trunc_norm_pdf(2, a=-1, b=1)
```

```
array([0.])
```

```python
trunc_norm_pdf(-2, a=-1, b=1)
```

```
array([0.])
```

```python
trunc_norm_pdf([-2,1,0,1,2], a=-1, b=1)
```

```
array([0.    , 0.3544, 0.5844, 0.3544, 0.    ])
```

```python
quad(lambda x: trunc_norm_pdf(x, a=-1, b=1), -np
```

```
(1.0, 2.9512170485190836e-11)
```

```python
quad(lambda x: trunc_norm_pdf(x, a=-3, b=3), -np
```

```
(0.9999999999999998, 7.472109098127788e-09)
```

Multivariate normal

$$f(\mathbf{x}) = \det(2\pi\boldsymbol{\Sigma})^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mu)\right)$$

```python
def mv_norm(x, µ, Σ):
  x = np.asarray(x)
  µ = np.asarray(µ)
  Σ = np.asarray(Σ)

  return ( np.linalg.det(2*np.pi*Σ)**(-0.5) *
           np.exp(-0.5 * (x - µ).T @ np.linalg.solve(Σ, (x-µ)) ) )
```

```python
norm_pdf(0,0,1)
```

```
0.3989422804014327
```

```python
mv_norm([0], [0], [[1]])
```

```
0.3989422804014327
```

```python
mv_norm([0,0], [0,0], [[1,0],[0,1]])
```

```
0.15915494309189535
```

```python
mv_norm([0,0,0], [0,0,0],
        [[1,0,0],[0,1,0],[0,0,1]])
```

```
0.06349363593424098
```

2d & 3d numerical integration

are supported by `dblquad()` and `tplquad()` respectively (see `nquad()` for higher dimensions)

```python
from scipy.integrate import dblquad, tplquad
```

```python
dblquad(lambda y, x: mv_norm([x,y], [0,0], np.identity(2)),
        a=-np.inf, b=np.inf,
        gfun=lambda x: -np.inf,   hfun=lambda x: np.inf)
```

```
(1.0000000000000322, 1.3150127836061 5e-08)
```

```python
tplquad(lambda z, y, x: mv_norm([x,y,z], [0,0,0], np.identity(3)),
        a=0, b=np.inf,
        gfun=lambda x:   0, hfun=lambda x:   np.inf,
        qfun=lambda x,y: 0, rfun=lambda x,y: np.inf)
```

```
(0.12500000000036066, 1.4697203688867502e-08)
```

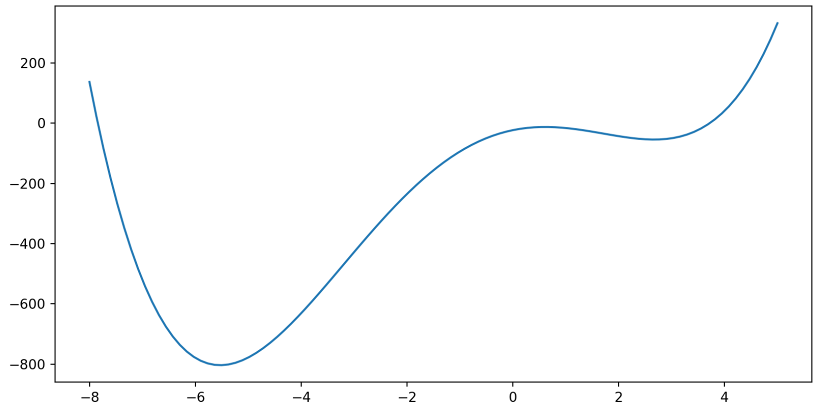
Example 3
(Very) Basic optimization

Scalar function minimization

```python
def f(x):
    return x**4 + 3*(x-2)**3 - 15*(x)**2 + 1
```

```python
from scipy.optimize import minimize_scalar
minimize_scalar(f, method="Brent")
```

```
 message: 
          Optimization terminated successfully;
          The returned value satisfies the
termination criteria
          (using xtol = 1.48e-08 )
 success: True
     fun: -803.3955308825884
       x: -5.528801125219663
     nit: 11
    nfev: 16
```



```python
minimize_scalar(f, method="bounded", bounds=[0,6
```

```
 message: Solution found.
 success: True
  status: 0
     fun: -54.21003937712762
       x: 2.668865104039653
     nit: 12
    nfev: 12
```

```python
minimize_scalar(f, method="bounded", bounds=[-8,
```

```
 message: Solution found.
 success: True
  status: 0
     fun: -803.3955308825871
       x: -5.528801009134004
     nit: 12
    nfev: 12
```

Results

```python
res = minimize_scalar(f)
type(res)
```

```
<class 'scipy.optimize._optimize.OptimizeResult'>
```

```python
dir(res)
```

```
['fun', 'message', 'nfev', 'nit', 'success', 'x']
```

```python
res.success
```

```
True
```

```python
res.x
```

```
-5.528801125219663
```

```python
res.fun
```

```
-803.3955308825884
```

More details

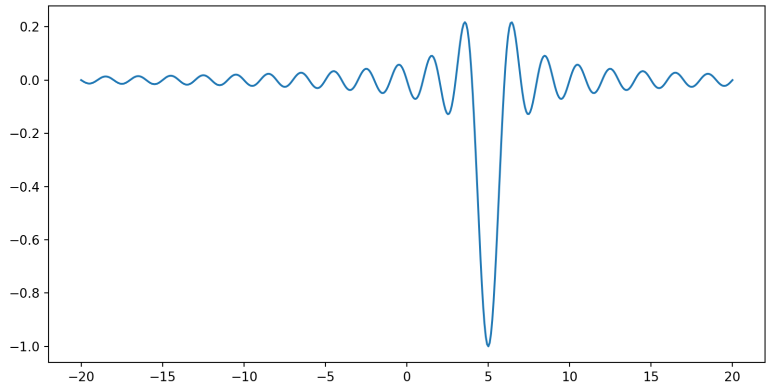
```python
from scipy.optimize import show_options
show_options(solver="minimize_scalar")
```

```
brent
=====

Options
-------
maxiter : int
    Maximum number of iterations to perform.
xtol : float
    Relative error in solution `xopt` acceptable for convergence.
disp: int, optional
    If non-zero, print messages.
        0 : no message printing.
        1 : non-convergence notification messages only.
        2 : print a message on convergence too.
        3 : print iteration results.
```

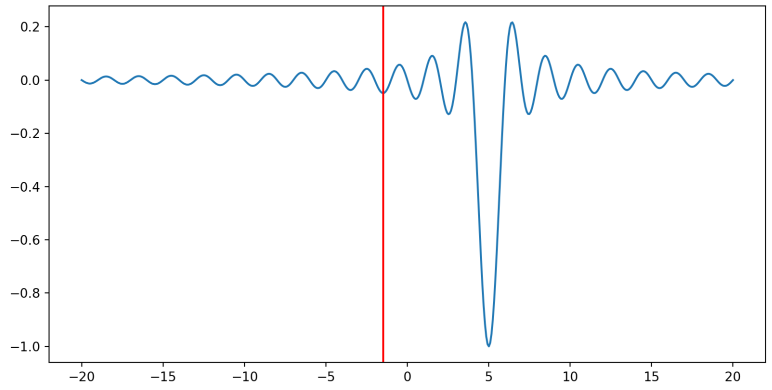
Local minima

```python
def f(x):
  return -np.sinc(x-5)
```



```python
res = minimize_scalar(f); res
```

```
 message: 
          Optimization terminated successfully;
          The returned value satisfies the
termination criteria
          (using xtol = 1.48e-08 )
 success: True
     fun: -0.049029624014074166
       x: -1.4843871263953001
     nit: 10
    nfev: 14
```



Random starts

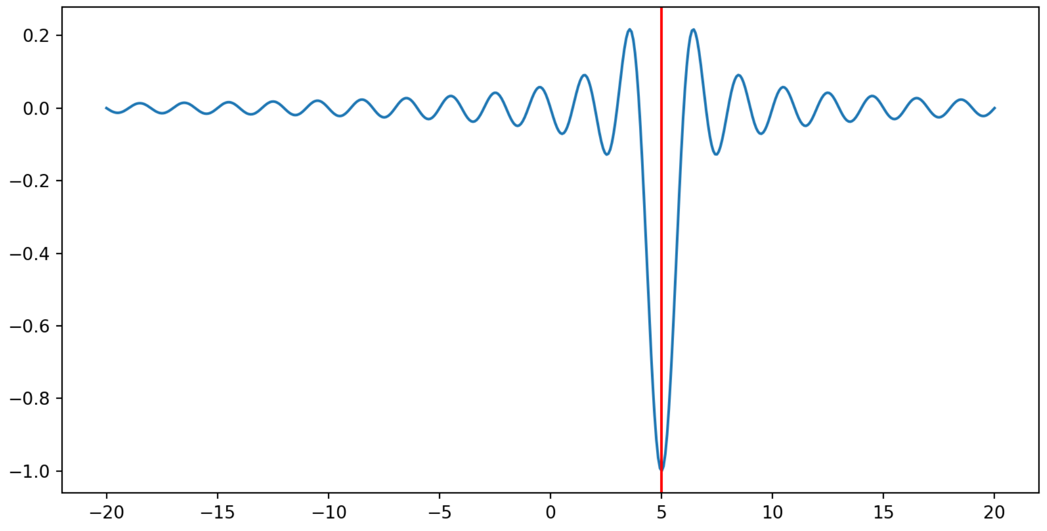
```python
rng = np.random.default_rng(seed=1234)

lower = rng.uniform(-20, 20, 100)
upper = lower + 1

sols = [minimize_scalar(f, bracket=(l,u))
        for l,u in zip(lower, upper)]
funs = [sol.fun for sol in sols]

best = sols[np.argmin(funs)]
best
```

```
 message: 
          Optimization terminated successfully;
          The returned value satisfies the
termination criteria
          (using xtol = 1.48e-08 )
 success: True
     fun: -1.0
       x: 5.000000000618556
     nit: 8
    nfev: 11
```



Back to Rosenbrock’s function

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

```python
def f(x):
  return (1-x[0])**2 + 100*(x[1]-x[0]**2)**2
```

```python
minimize(f, [0,0])
```

```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 2.843987518235081e-11
        x: [ 1.000e+00  1.000e+00]
      nit: 19
      jac: [ 3.987e-06 -2.844e-06]
 hess_inv: [[ 4.948e-01  9.896e-01]
            [ 9.896e-01  1.984e+00]]
     nfev: 72
     njev: 24
```

```python
minimize(f, [-1,-1])
```

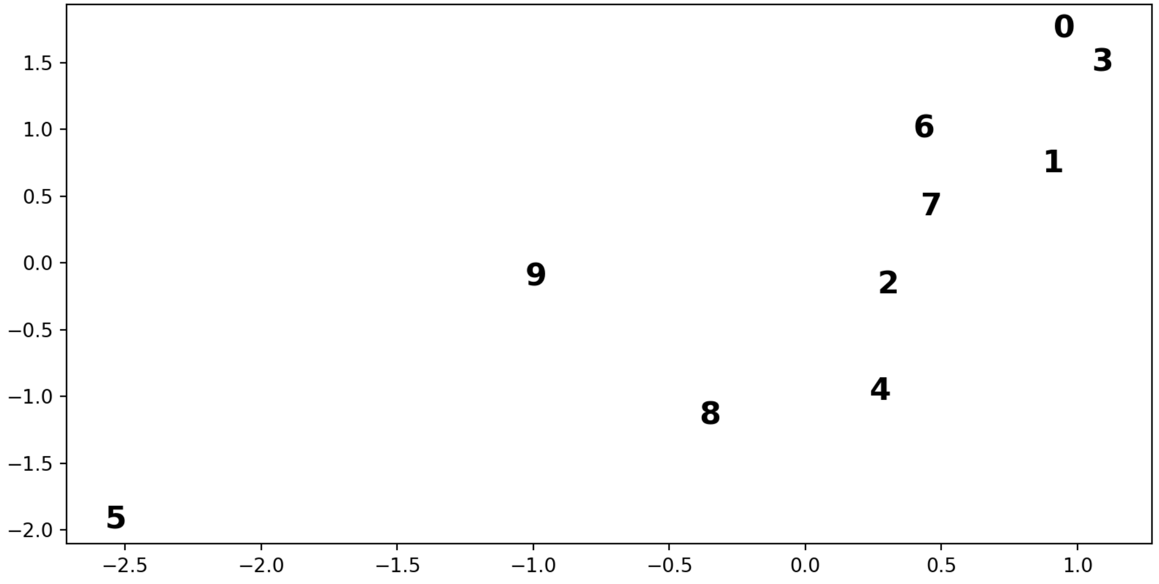
```
  message: Optimization terminated successfully.
  success: True
   status: 0
      fun: 1.9950032694539075e-11
        x: [ 1.000e+00  1.000e+00]
      nit: 31
      jac: [ 2.789e-07 -1.275e-07]
 hess_inv: [[ 5.085e-01  1.016e+00]
            [ 1.016e+00  2.037e+00]]
     nfev: 120
     njev: 40
```

Example 4
Spatial Tools

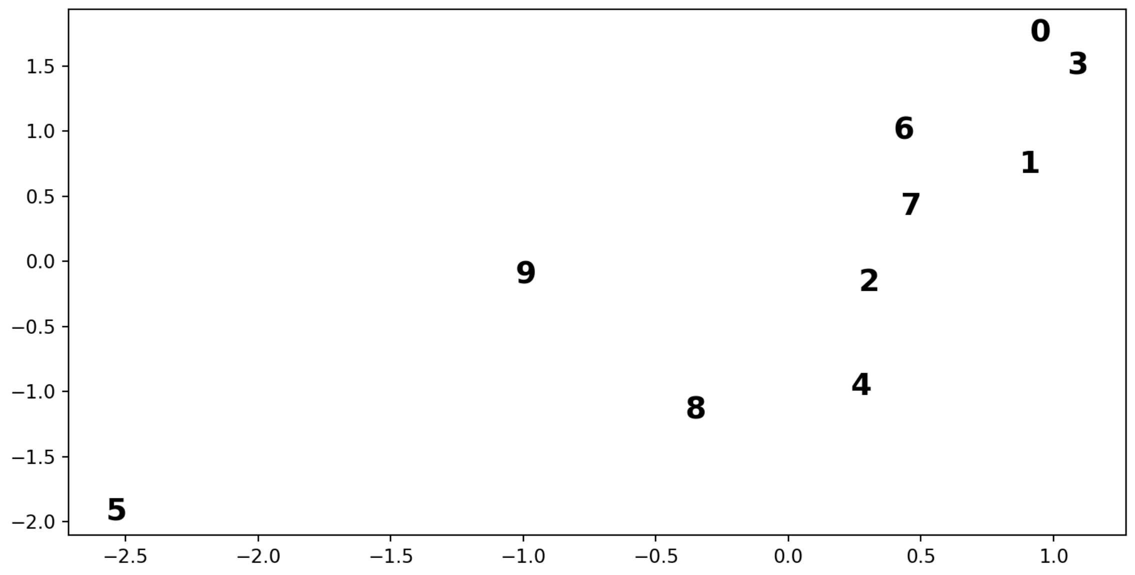
Nearest Neighbors

```python
rng = np.random.default_rng(seed=12345)
pts = rng.multivariate_normal(
  [0,0], [[1,.8],[.8,1]],
  size=10
)
pts
```

```
array([[ 0.9511,  1.7504],
       [ 0.9079,  0.744 ],
       [ 0.3058, -0.1628],
       [ 1.0924,  1.5028],
       [ 0.275 , -0.9601],
       [-2.5332, -1.9207],
       [ 0.4351,  1.0057],
       [ 0.4622,  0.4238],
       [-0.351 , -1.1458],
       [-0.9887, -0.1039]])
```



KD Trees



```python
from scipy.spatial import KDTree
kd = KDTree(pts)
```

```python
dist, i = kd.query(pts[6,:], k=3)
i
```

```
array([6, 1, 7])
```

```python
dist
```

```
array([0.    , 0.5404, 0.5825])
```

```python
dist, i = kd.query(pts[2,:], k=5)
i
```

```
array([2, 7, 4, 1, 6])
```

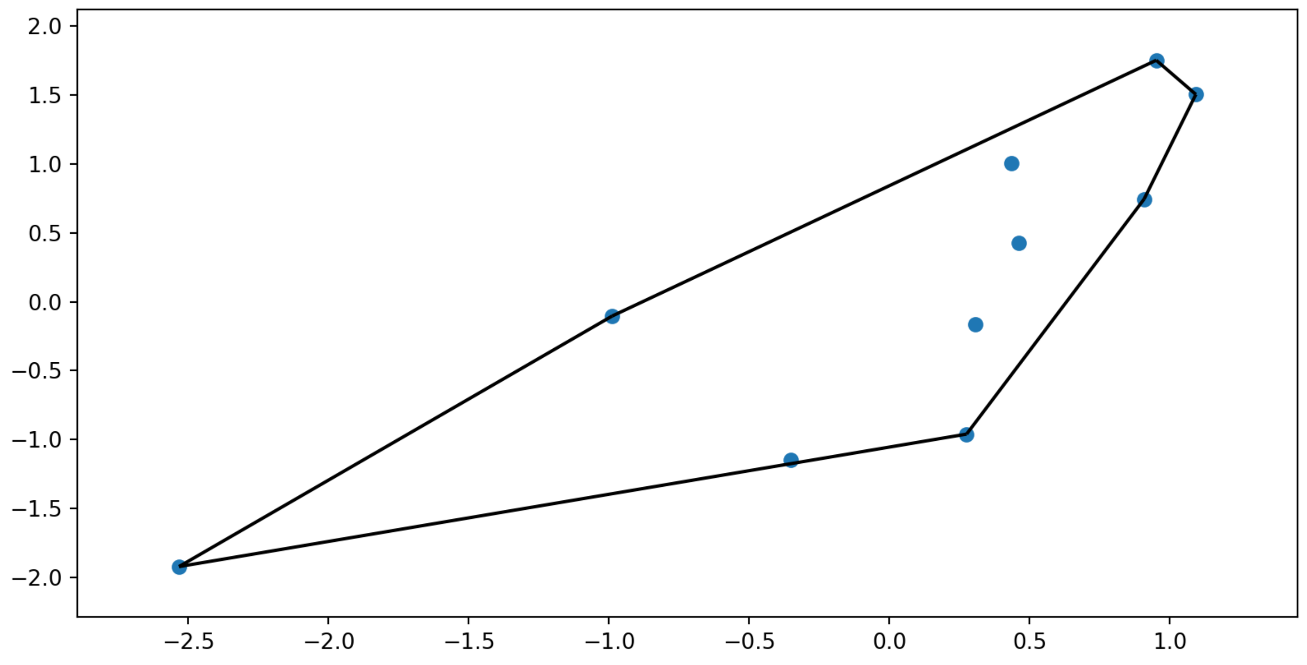
Convex hulls

```python
from scipy.spatial import ConvexHull
hull = ConvexHull(pts)
```

```python
hull.vertices
```

```
array([3, 0, 9, 5, 4, 1], dtype=int32)
```

```python
scipy.spatial.convex_hull_plot_2d(hull)
```



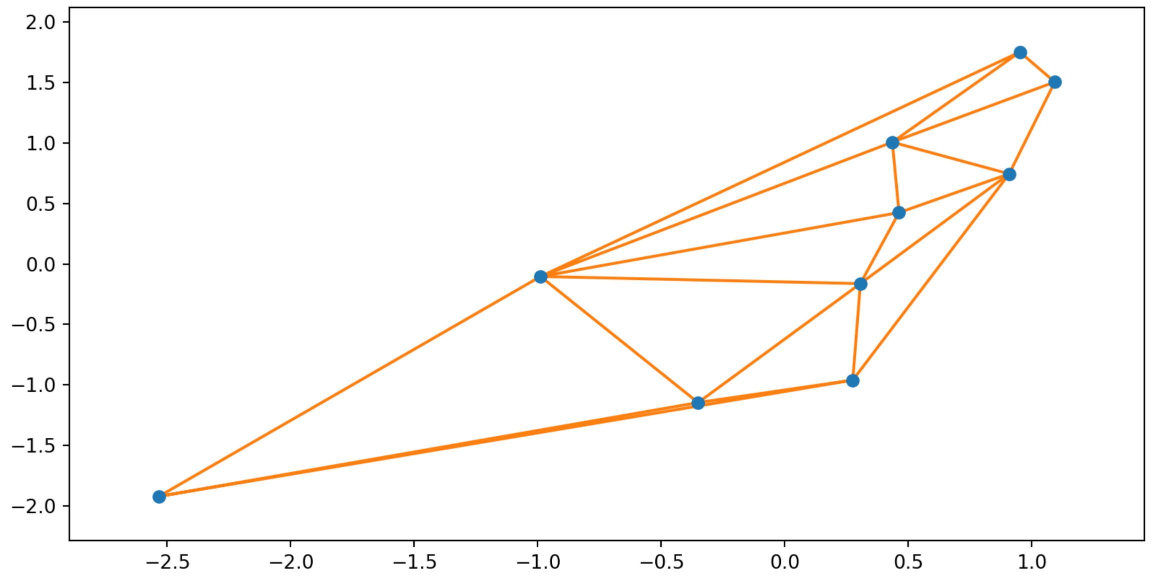
Delaunay triangulations

```python
from scipy.spatial import Delaunay
tri = Delaunay(pts)
```

```python
tri.simplices.T
```

```
array([[8, 4, 9, 8, 4, 6, 0, 6, 7, 7, 1, 7],
       [9, 8, 8, 4, 1, 1, 6, 0, 9, 6, 7, 1],
       [5, 5, 2, 2, 2, 3, 3, 9, 2, 9, 2, 6]], dtype=int32)
```

```python
scipy.spatial.delaunay_plot_2d(tri)
```



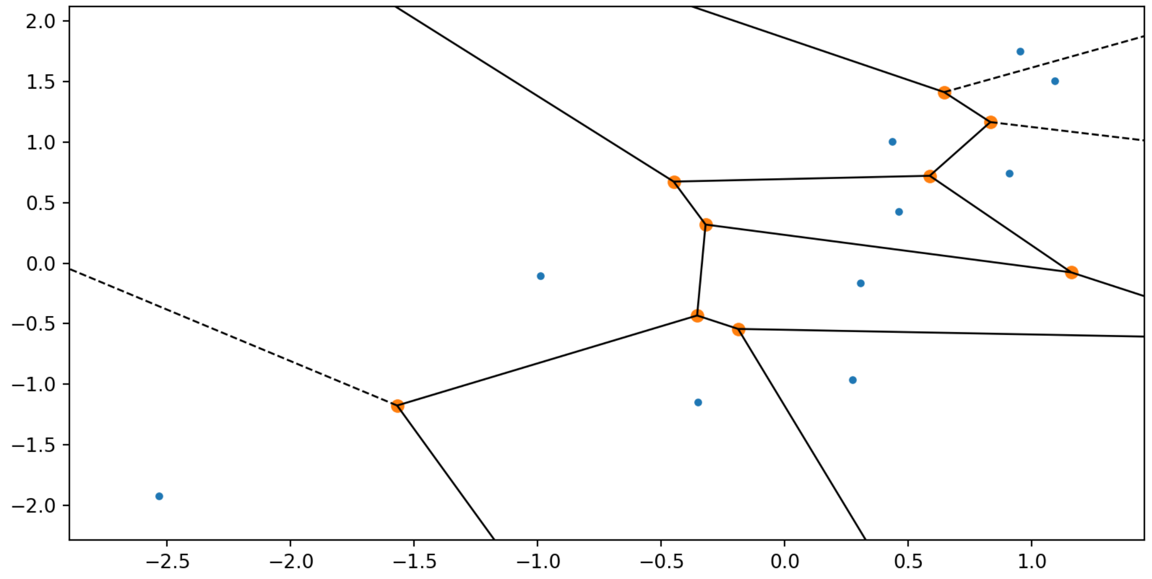
Voronoi diagrams

```
from scipy.spatial import Voronoi
vor = Voronoi(pts)
```

```
vor.vertices.T
```

```
array([[ -1.5692,   7.9474,  -0.3551,  -0.1892,   1.9886,   0.8318,   0.6448,  -2.9865,  -0.3209,
-0.4499,   1.1593,   0.5865],
       [ -1.1753, -27.9746,  -0.4322,  -0.5429,  -0.6269,   1.1644,   1.4115,   3.9278,   0.3184,   0.673
,  -0.0762,   0.7212]])
```

```
scipy.spatial.voronoi_plot_2d(vor)
```



Example 5
statistics

Distributions

Implements classes for 104 continuous and 19 discrete distributions,

- `rvs` - Random Variates
- `pdf` - Probability Density Function
- `cdf` - Cumulative Distribution Function
- `sf` - Survival Function (1-CDF)
- `ppf` - Percent Point Function (Inverse of CDF)
- `isf` - Inverse Survival Function (Inverse of SF)
- `stats` - Return mean, variance, (Fisher's) skew, or (Fisher's) kurtosis
- `moment` - non-central moments of the distribution

Basic usage

```python
from scipy.stats import norm, gamma, binom, uniform
```

```python
norm().rvs(size=5)
```

```
array([-0.3006, -0.172 ,  0.5723,  0.9794, -1.9397])
```

```python
uniform.pdf([0,0.5,1,2])
```

```
array([1., 1., 1., 0.])
```

```python
binom.mean(n=10, p=0.25)
```

```
2.5
```

```python
binom.median(n=10, p=0.25)
```

```
2.0
```

```python
gamma(a=1,scale=1).stats()
```

```
(1.0, 1.0)
```

```python
norm().stats(moments="mvsk")
```

Freezing

Model parameters can be passed to any of the methods directory, or a distribution can be constructed using a specific set of parameters, which is known as freezing.

```python
norm_rv = norm(loc=-1, scale=3)
norm_rv.median()
```

```
-1.0
```

```python
unif_rv = uniform(loc=-1, scale=2)
unif_rv.cdf([-2,-1,0,1,2])
```

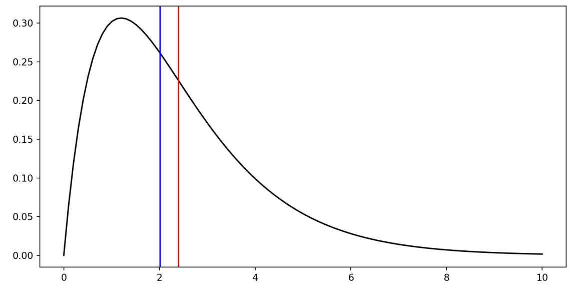
```
array([0. , 0. , 0.5, 1. , 1. ])
```

```python
unif_rv.rvs(5)
```

```
array([ 0.1149,  0.5345,  0.4967, -0.913 ,
-0.5714])
```

```python
g = gamma(a=2, loc=0, scale=1.2)

x = np.linspace(0, 10, 100)
plt.plot(x, g.pdf(x), "k-")
plt.axvline(x=g.mean(), c="r")
plt.axvline(x=g.median(), c="b")
```



MLE

Maximum likelihood estimation is possible via the `fit()` method,

```python
x = norm.rvs(loc=2.5, scale=2, size=1000, random_state=1234)
norm.fit(x)
```

```
(2.5314811643075235, 1.946132398754459)
```

```python
norm.fit(x, loc=2.5) # provide a guess for the parameter
```

```
(2.5314811643075235, 1.946132398754459)
```

```python
x = gamma.rvs(a=2.5, size=1000)
gamma.fit(x) # shape, loc, scale
```

```
(2.3525886566168865, 0.019729456209898893, 1.0868982157388611)
```

```python
y = gamma.rvs(a=2.5, loc=-1, scale=2, size=1000)
gamma.fit(y) # shape, loc, scale
```

```
(2.9173472217905676, -1.1499921149325845, 1.7505550912408578)
```