Intro. Comp. for Data Science (FMI08)

Dr. Nono Saha

May 17, 2023

Max Planck Institute for Mathematics in the Sciences
University of Leipzig/ScaDS.AI

Spring 2023

Course plan

1. Introduction to SciPy
2. Example 1 - k-means clustering
3. Example 2 - Numerical integration
4. (Very) Basic optimization
5. Example 4 - Spatial Tools
6. Example 5 - stats

Introduction to SciPy

What is `SciPy`

Fundamental algorithms for scientific computing in Python.

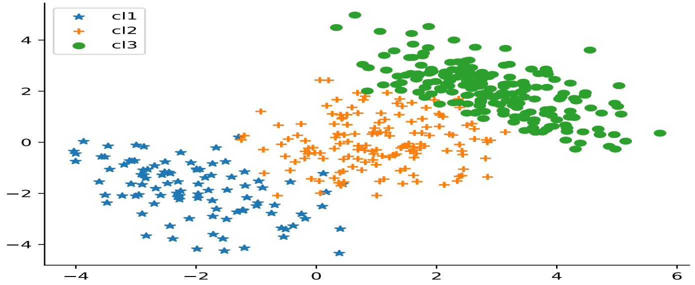
Subpackage	Description
`cluster`	Clustering algorithms
`odr`	Orthogonal distance regression
`constants`	Physical and mathematical constants
`optimize`	Optimization and root-finding routines
`fftpack`	Fast Fourier Transform routines
`signal`	Signal processing
`integrate`	Integration and ordinary differential equation solvers
`sparse`	Sparse matrices and associated routines
`interpolate`	Interpolation and smoothing spline
`spatial`	Spatial data structures and algorithms

You can also find alternative subpackage to `NumPy` such as: `io`, `linalg`.

Example 1 - k-means clustering

Example 1 - k-means clustering: data

```
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))

```



Example 1 - k-means clustering: example

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

## array([[ 2.85409537,  1.94511779],
##        [ 0.89789235, -0.20527898],
##        [-2.03956666, -1.85662027]])

dist
## 1.2206927437557962


cl1.mean(axis=0)
## array([-2.00474615, -1.87275596])

cl2.mean(axis=0)
## array([1.03849018, 0.01417119])

cl3.mean(axis=0)
## array([2.94641907, 2.02514165])

```

Example 1 - k-means clustering: algorithm

k-means clustering is a method for finding clusters and cluster centres in a set of unlabeled data. Given an initial set of k centers, the k-means algorithm alternates the two steps:

1. For each centre, we identify the subset of training points (its cluster) that is closer to it than any other centre.

$$S_i^{(t)} = \{x_p : ||x_p - m_i^{(t)}||^2 \le ||x_p - m_j^{(t)}||^2 \forall j, 1 \le j \le k\}$$

   where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

2. Recalculate the means (or centroids):

$$m_i^{t+1} = \frac{1}{|S^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

Homework 6: Implement your version of k-means and comment on the complexity.

Example 2 - Numerical integration

Example 2 - Numerical integrations: 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 integration.

Simple examples:

```
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)

```

Example 2 - Numerical integrations: Normal PDF

The PDF for a normal distribution is given by,

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

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

norm_pdf(0,0,1)
## 0.3989422804014327

norm_pdf(np.Inf, 0, 1)
## 0.0

norm_pdf(-np.Inf, 0, 1)
## 0.0
```

Example 2 - Numerical integrations: checking our PDF

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

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


```

Question: Will this work? Why?

Example 2 - Numerical integrations: checking our PDF

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

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

Question: Will this work? Why?

Simple debugging: add default parameters

```
quad(lambda x: norm_pdf(x, 0, 1), -np.inf, np.inf)
## (0.9999999999999997, 1.0178191380347127e-08)

quad(lambda x: norm_pdf(x, 17, 12), -np.inf, np.inf)
## (1.0000000000000002, 4.113136862574909e-09)
```

Example 2 - Numerical integrations: truncated normals

The PDF for a normal distribution is given by,

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

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

```

Example 2 - Numerical integrations: testing our pdf function

```
trunc_norm_pdf(0, a=-1, b=1)
## array([0.39894228])

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.        , 0.24197072, 0.39894228, 0.24197072, 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)
```

Example 2 - Numerical integrations: fixing our function

What are the changes to perform?

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

trunc_norm_pdf(0, a=-1, b=1)
## array([0.58436857])

trunc_norm_pdf(2, a=-1, b=1)
## array([0.])

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

Example 2 - Numerical integrations: multivariate normal

$$f(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp(-\frac{1}{2}(\mathsf{x}-\mu)^T\Sigma^{-1}(\mathsf{x}-\mu))$$

```python
def mv_norm(x, mu, sigma):
  x = np.asarray(x)
  mu = np.asarray(mu)
  sigma = np.asarray(sigma)
  return np.linalg.det(2*np.pi*sigma)**(-0.5) * np.exp(-0.5 * (x
     - mu).T @ np.linalg.solve(sigma, (x-mu)) )

```

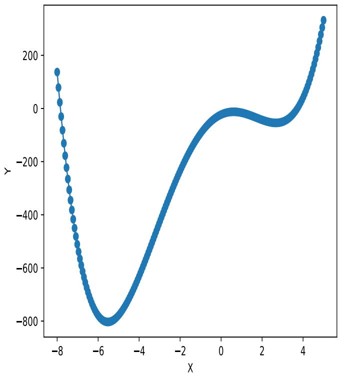
```python
norm_pdf(0,0,1)
## 0.3989422804014327

mv_norm([0], [0], [[1]])
## 0.3989422804014327

mv_norm([0,0], [0,0],
    [[1,0],[0,1]])
## 0.15915494309189535

```

```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,
    1.3150127836618008e-08)
```

Example 3 - (Very) Basic optimization

Example 3 - (Very) Basic optimization: scalar function minimization

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




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

##      fun: -803.3955308825884
##     nfev: 17
##      nit: 11
##  success: True
##        x: -5.528801125219663

minimize_scalar(f, method="bounded", bounds=[0,6])
##      fun: -54.21003937712762
##  message: 'Solution found.'
##     nfev: 12
##   status: 0
##  success: True
##        x: 2.668865104039653
```

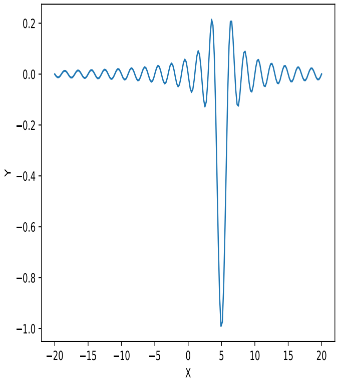
Example 3 - (Very) Basic optimization: results

```python
res = minimize_scalar(f)
type(res)
## <class 'scipy.optimize.optimize.OptimizeResult'>

dir(res)
## ['fun', 'nfev', 'nit', 'success', 'x']

res.success
## True

res.x
## -5.528801125219663

```

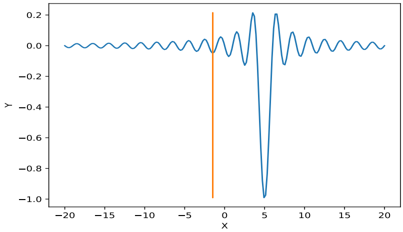
```python
from scipy.optimize import show_options
show_options(solver="minimize_scalar")
```

Example 3 - (Very) Basic optimization: local minima

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

```

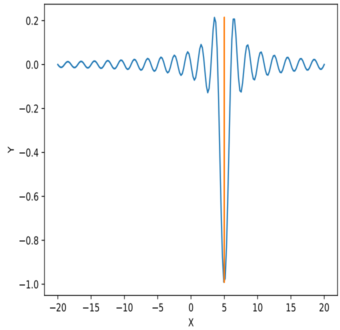



```
res = minimize_scalar(f)
res
##      fun: -0.049029624014074166
##     nfev: 15
##      nit: 10
##  success: True
##        x: -1.4843871263953001

```



Example 3 - (Very) Basic optimization: random starts

```
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

##      fun: -1.0
##     nfev: 12
##      nit: 8
##  success: True
##        x: 5.000000000618556
```



Example 3 - (Very) Basic optimization: Rosenbrock's function

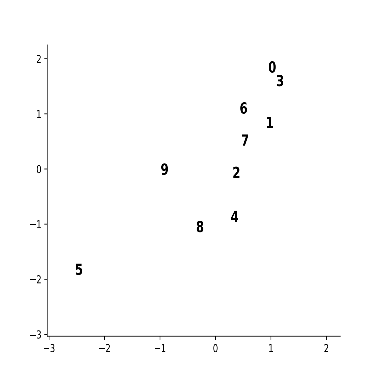
$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

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

minimize(f, [0,0])
##       fun: 2.844030241790906e-11
##  hess_inv: array([[0.49482454, 0.98957635],
##        [0.98957635, 1.98394216]])
##       jac: array([ 3.98673382e-06, -2.84416264e-06])
##   message: 'Optimization terminated successfully.'
##      nfev: 72
##       nit: 19
##      njev: 24
##    status: 0
##   success: True
##         x: array([0.99999467, 0.99998932])

minimize(f, [-1,-1]).x
## array([0.99999553, 0.99999106])
```

Example 4 - Spatial Tools

NumPy - Example 4 - Spatial tools: KD trees

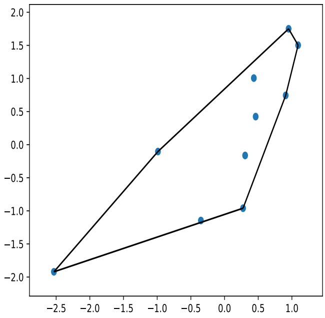
```
from scipy.spatial import KDTree
kd = KDTree(pts)
kd
## <scipy.spatial.kdtree.KDTree
    object at 0x1026b44c0>

dir(kd)

dist, i = kd.query(pts[6,:], k=3)
dist
## array([0.        , 0.54041133,
    0.58254815])

i
## array([6, 1, 7])

dist, i = kd.query(pts[2,:], k=5)
i
## array([2, 7, 4, 1, 6])
```



NumPy - Example 4 - Spatial tools: convex hulls

```python
from scipy.spatial import ConvexHull
hull = ConvexHull(pts)
hull
## <scipy.spatial.qhull.ConvexHull object at 0x14778d700>

dir(hull)

hull.simplices
## array([[0, 3],
##        [4, 5],
##        [9, 5],
##        [9, 0],
##        [1, 3],
##        [1, 4]], dtype=int32)

```

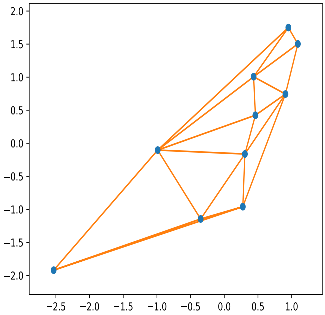
```python
scipy.spatial.convex_hull_plot_2d(hull)

```



NumPy - Example 4 - Spatial Tools: delaunay triangulations

```python
from scipy.spatial import Delaunay
tri = Delaunay(pts)
tri
## <scipy.spatial.qhull.Delaunay
    object at 0x1477a0fd0>

dir(tri)

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

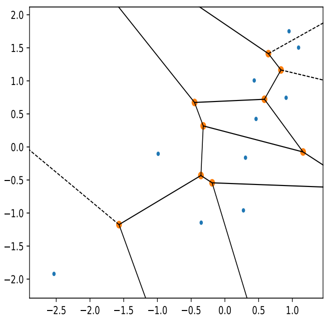
```python
scipy.spatial.
    delaunay_plot_2d(tri)
```



NumPy - Example 4 - Spatial Tools: voronoi diagrams

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

## <scipy.spatial.qhull.Voronoi object at 0x1477c25b0>

dir(vor)

vor.vertices

## array([[ -1.56917821, -1.17533646],
## [  7.94738786, -27.97463108],[ -0.3550644 ,  -0.43215628],
## [ -0.18923926,  -0.54294902],[ 1.98860973,  -0.62693469],
## [  0.83175084, 1.16435674],....

```

```
scipy.spatial.voronoi_plot_2d(vor)

```



Example 5 - stats

Example 5 - stats: 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