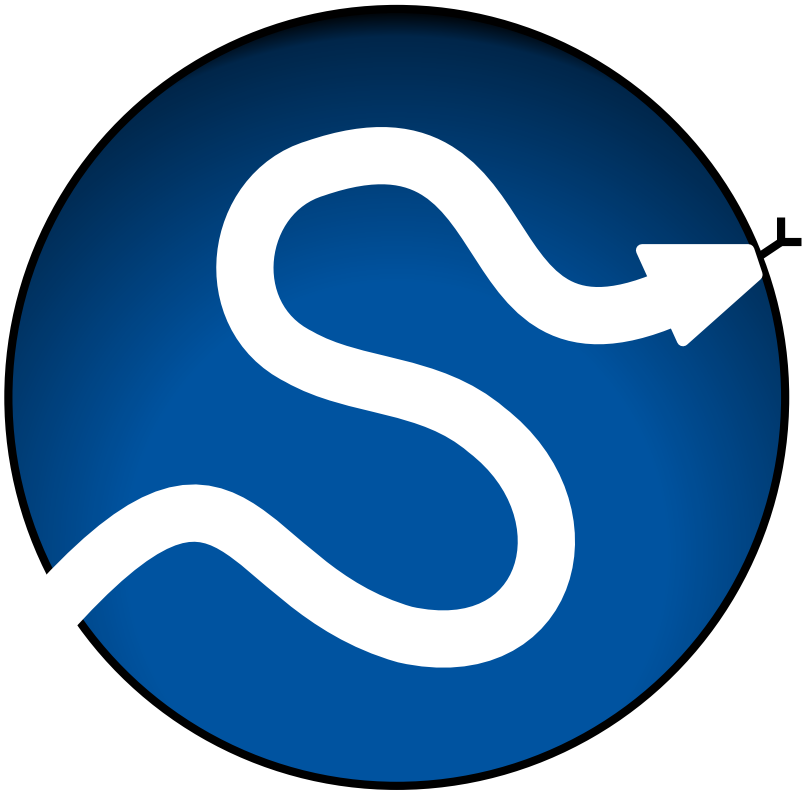


SciPy

scipy.org/scipylib/

*Scipy = **Sci**entific **Py**thon*

scipy.constants	Физические и математические константы
scipy.fft	Преобразование Фурье
scipy.integrate	Интегрирование
scipy.interpolate	Интерполяция
scipy.io	Ввод/вывод
scipy.linalg	Линейная алгебра
scipy.ndimage	Обработка изображений
scipy.optimize	Оптимизации
scipy.signal	Обработка данных
scipy.sparse	Разреженные матрицы
scipy.spatial	Пространственные структуры (триангуляция и т.п.)
scipy.special	Математические специальные функции
scipy.stats	Статистика

scipy.io

- MATLAB
- IDL
- Matrix Market
- Fortran
- Netcdf
- Harwell-Boeing
- Wav
- Arff

```
>>> from scipy import io as spio
>>> a = np.ones((3, 3))
>>> spio.savemat('file.mat', {'a': a}) # savemat expects a dictionary
>>> data = spio.loadmat('file.mat', struct_as_record=True)
>>> data['a']
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

scipy.io: Поддержка NetCDF

```
>>> from scipy.io import netcdf
>>> f = netcdf.netcdf_file('/mnt/data/topo/gebco/GEBCO_2014_1D.nc', 'r')
>>> f.variables

OrderedDict([('x_range', <scipy.io.netcdf.netcdf_variable at 0x7f5f50159048>),
             ('y_range', <scipy.io.netcdf.netcdf_variable at 0x7f5f50159630>),
             ('z_range', <scipy.io.netcdf.netcdf_variable at 0x7f5f501595f8>),
             ('spacing', <scipy.io.netcdf.netcdf_variable at 0x7f5f501597f0>),
             ('dimension',
              <scipy.io.netcdf.netcdf_variable at 0x7f5f50159860>),
             ('z', <scipy.io.netcdf.netcdf_variable at 0x7f5f50159898>)])

>>> f.variables['x_range'].data
array([-180.,  180.])
```

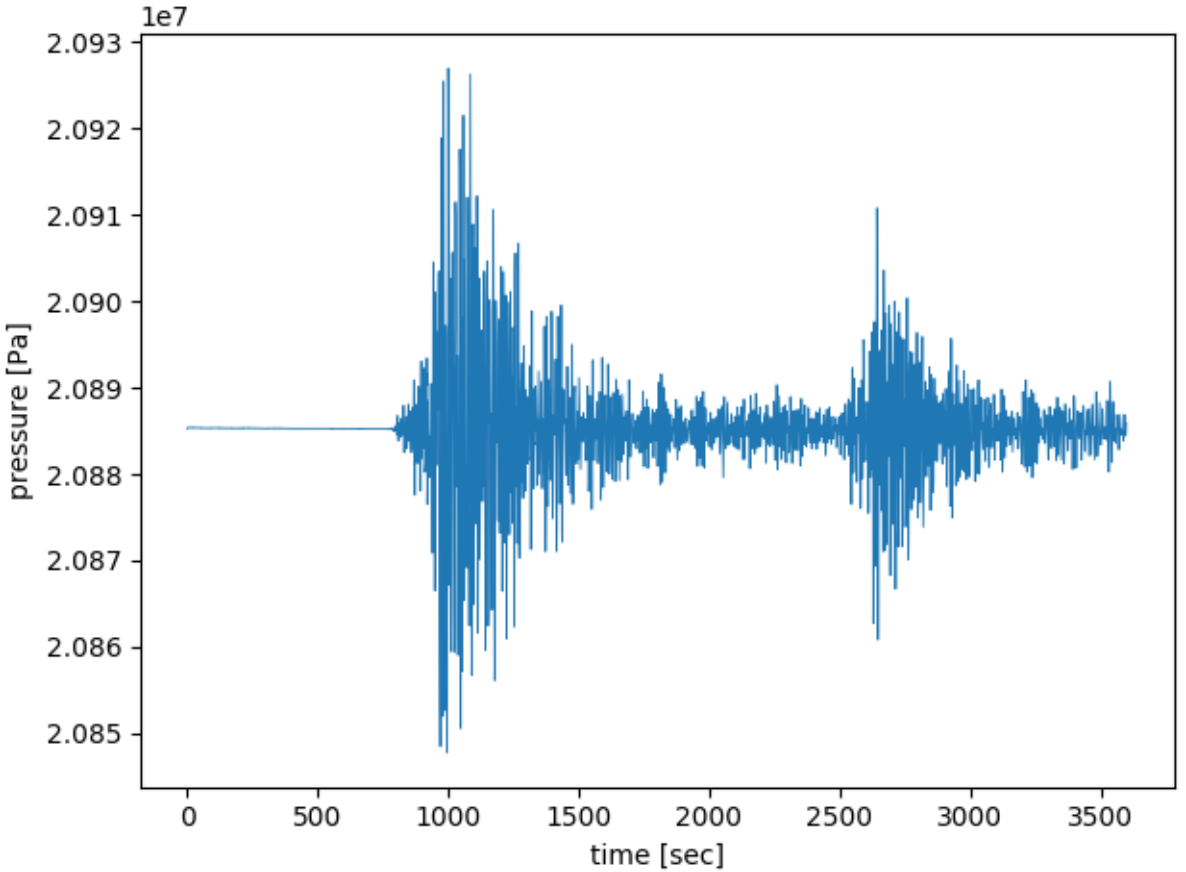
scipy.constants: физические и математические константы

CODATA Recommended Values of the Fundamental Physical Constants (2018):
https://physics.nist.gov/cuu/Constants/

```
>>> import scipy.constants as sc
>>> sc.c
299792458.0

>>> sc.g
9.80665

>>> sc.G
6.67408e-11

>>> sc.value('standard atmosphere')
101325.0

>>> sc.value('speed of light in vacuum')
299792458.0
```

scipy.fft

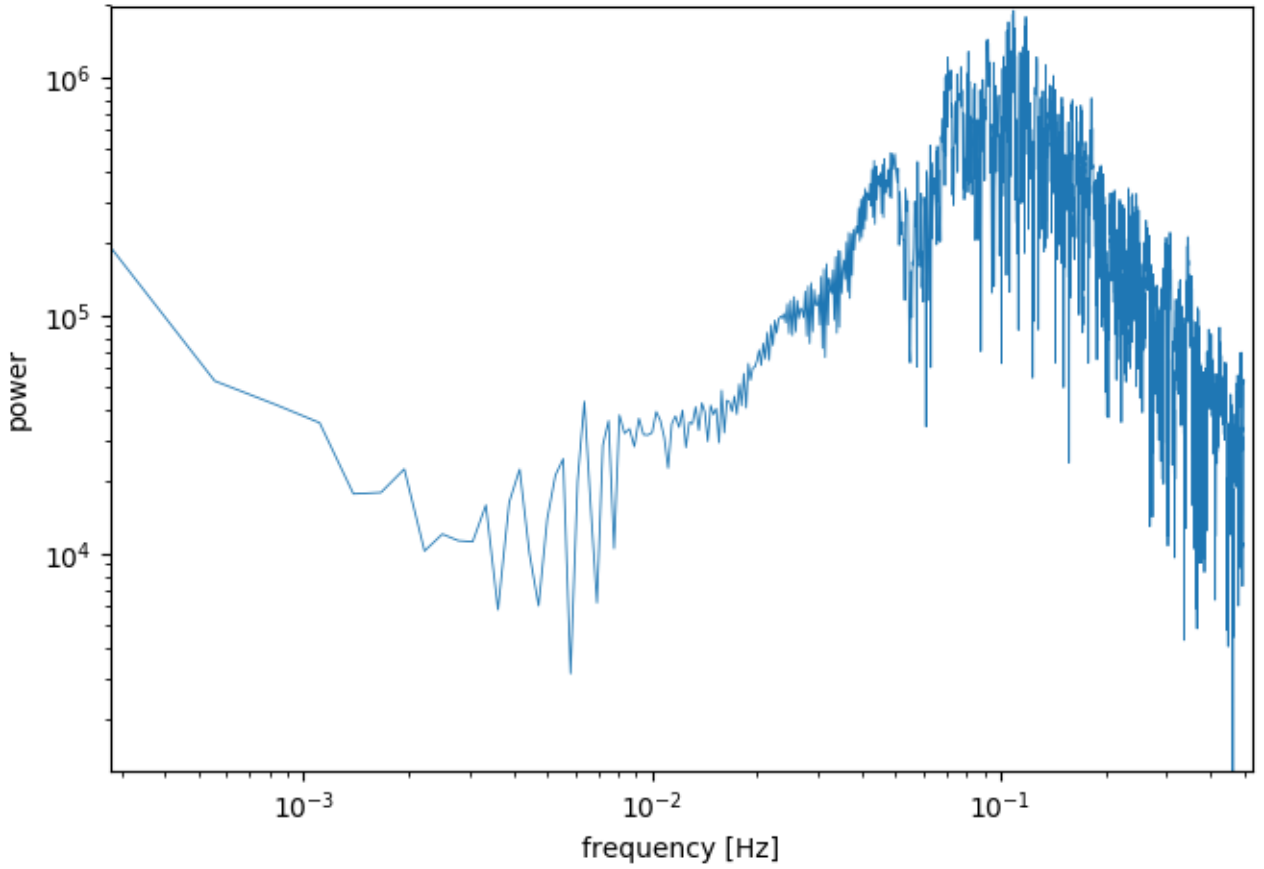
```
>>> import scipy.fft as fp
>>> import scipy
>>> import numpy as np

>>> sig = scipy.loadtxt('OYP_KMA03_2011-03-11T053500_1.0Hz.dat')
```



scipy.fft

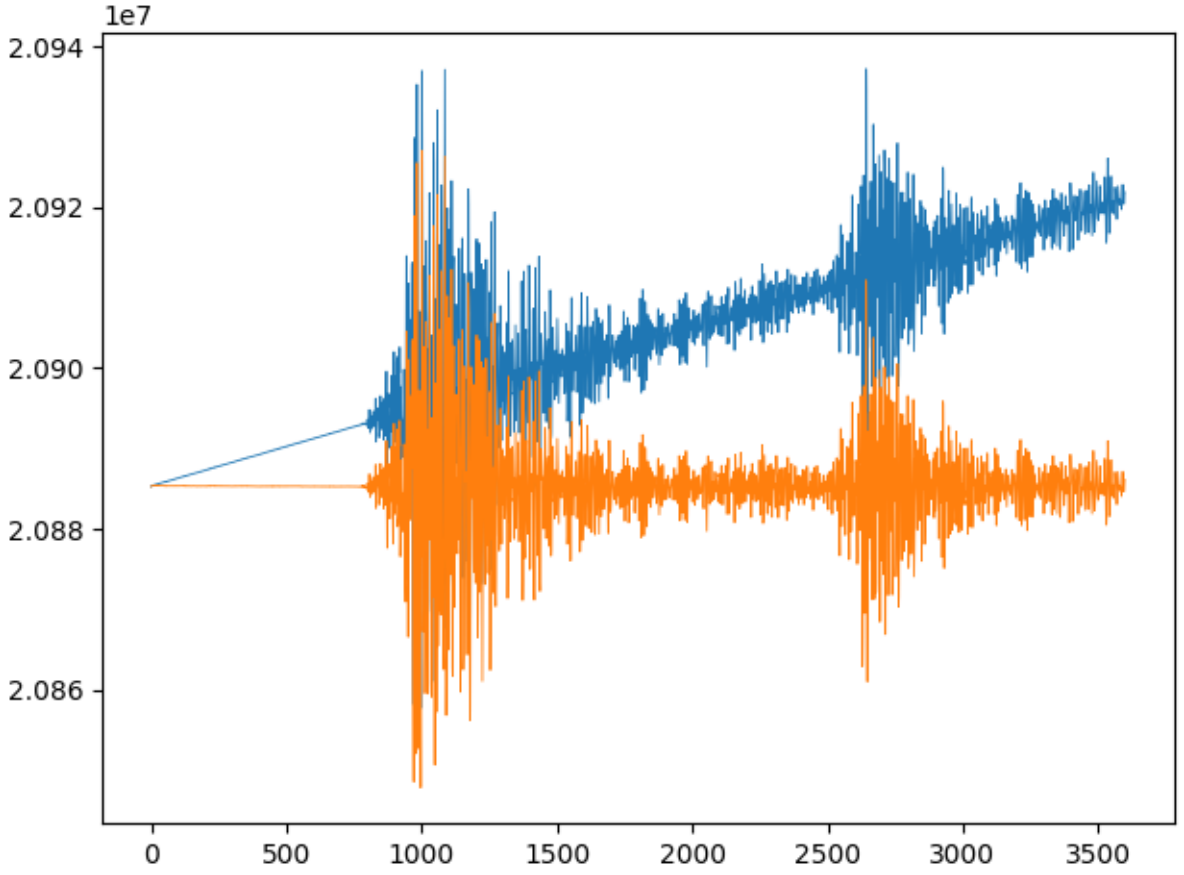
```
>>> sample_freq = fp.fftfreq(sig.size)
>>> sample_freq
array([ 0.        ,  0.00027778,  0.00055556, ..., -0.00083333,
       -0.00055556, -0.00027778])
>>> sig_fft = fp.fft(sig)
>>> pidxs = np.where(sample_freq > 0)
>>> freqs = sample_freq[pidxs]
>>> power = np.abs(sig_fft)[pidxs]
>>> freqs[power.argmax()]
0.10888888888888888
```

scipy.fft



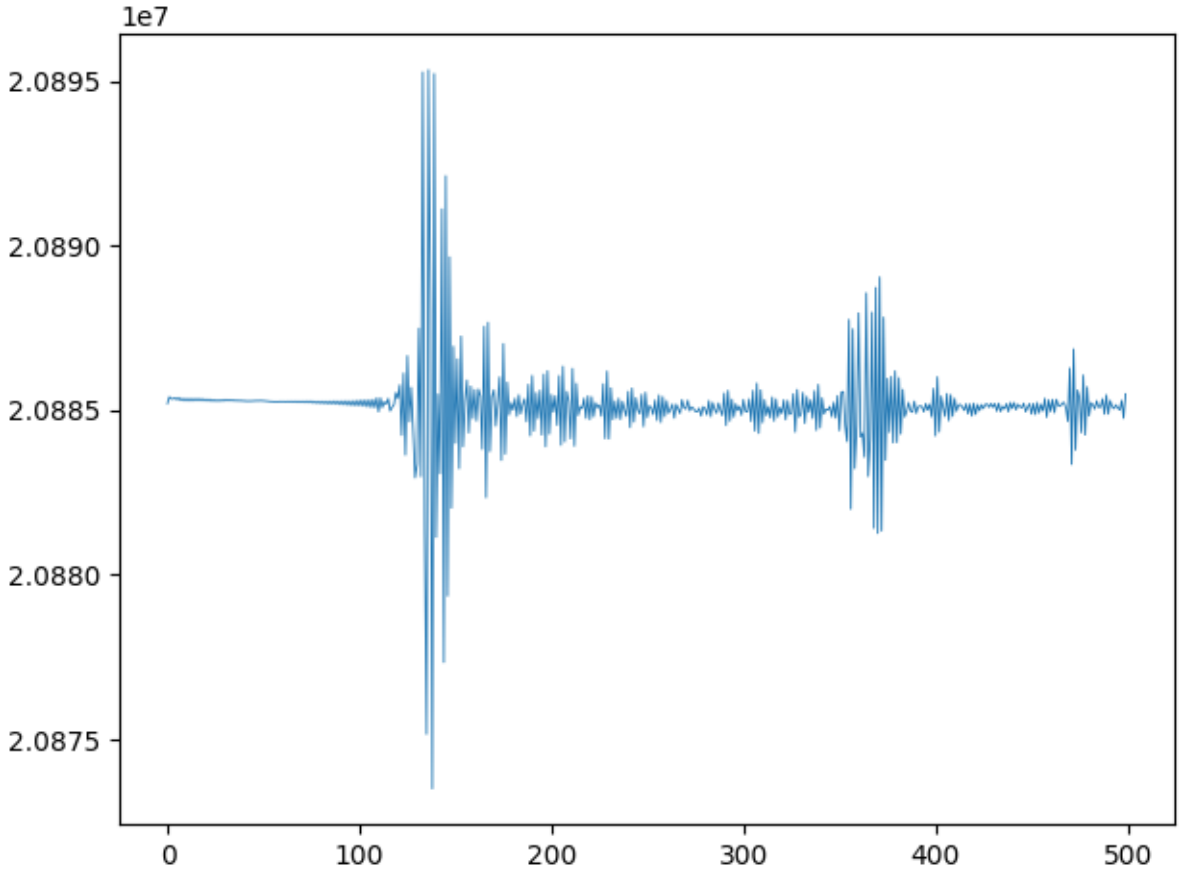
scipy.signal: обработка данных

```
>>> from scipy import signal
>>> t = np.arange(0,3600,1)
>>> sig_t = sig + t*10

>>> sig_d  = signal.detrend(sig_t) + sig[0]
```



scipy.signal: обработка данных

```
signal.resample(sig,500)
```



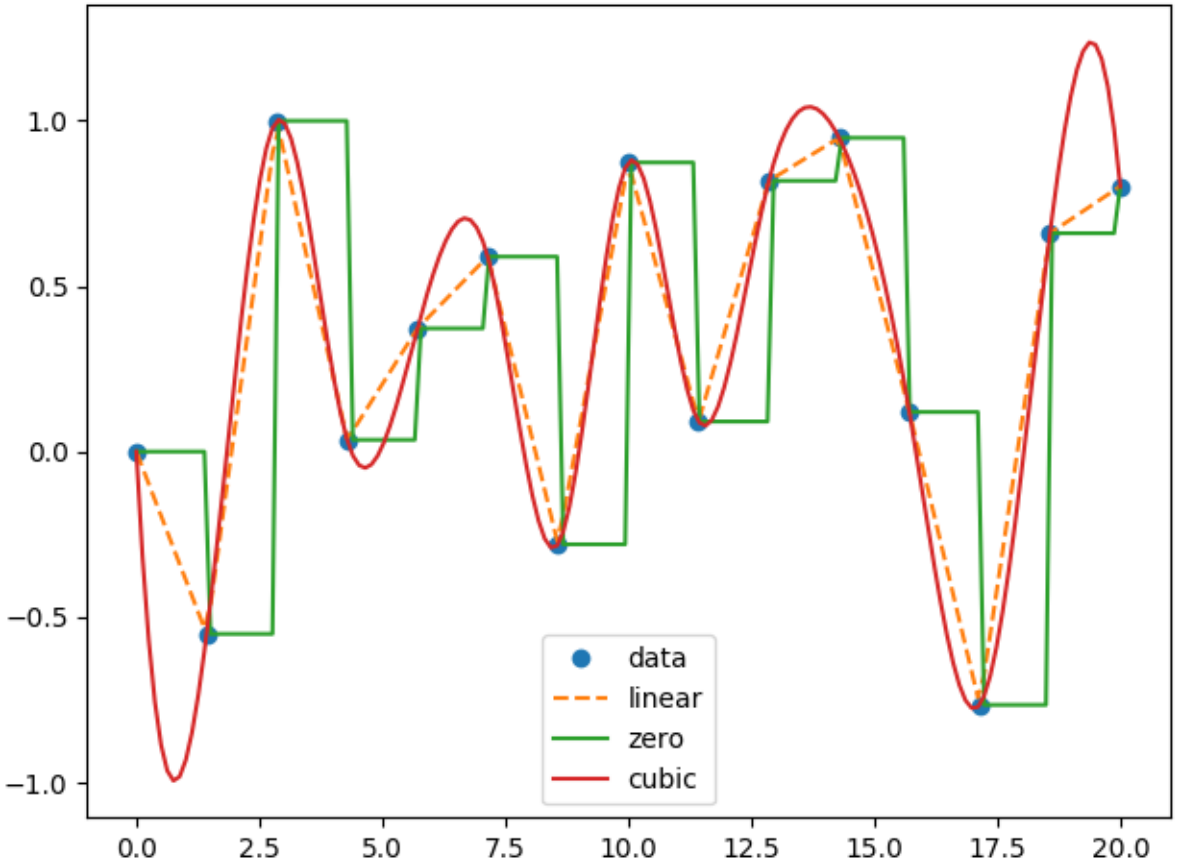
scipy.interpolate: интерполяция

- Одномерная
- Многомерная
- 1-D сплайны
- 2-D сплайны

scipy.interpolate

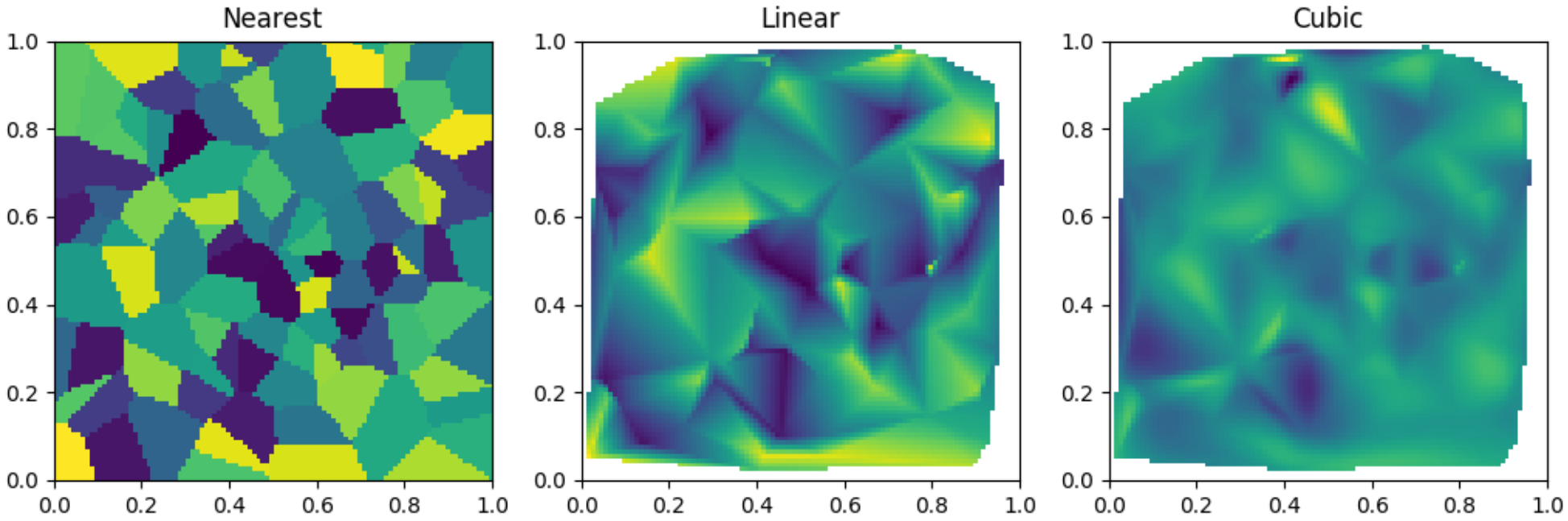
```
>>>  x = np.linspace(0, 20,num=15)
...: y = np.sin(-x**3/5.0)
...:
...: f = si.interp1d(x,y)
...: f_z = si.interp1d(x,y,kind='zero')
...: f_c = si.interp1d(x,y,kind='cubic')
```

scipy.interpolate



scipy.interpolate

```
>>>  grid_x, grid_y = np.mgrid[0:1:100j, 0:1:100j]
...: points = np.random.rand(100, 2)
...: values = np.random.randint(100, size=100)

...: grid_z0 = si.griddata(points, values, (grid_x, grid_y), method='nearest')
...: grid_z1 = si.griddata(points, values, (grid_x, grid_y), method='linear')
...: grid_z2 = si.griddata(points, values, (grid_x, grid_y), method='cubic'
```



scipy.optimize: минимизация

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

```
>>> import scipy.optimize as opt

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

>>>opt.minimize(f, (0,0), method='Nelder-Mead')
final_simplex: (array([[ 1.00000439,  1.00001064],
       [ 0.99996163,  0.99992454],
       [ 1.00002803,  1.00005254]]), array([  3.68617692e-10,
  1.63627702e-09,   2.02249112e-09]))
          fun: 3.6861769151759075e-10
      message: 'Optimization terminated successfully.'
         nfev: 146
          nit: 79
       status: 0
      success: True
            x: array([ 1.00000439,  1.00001064])
```

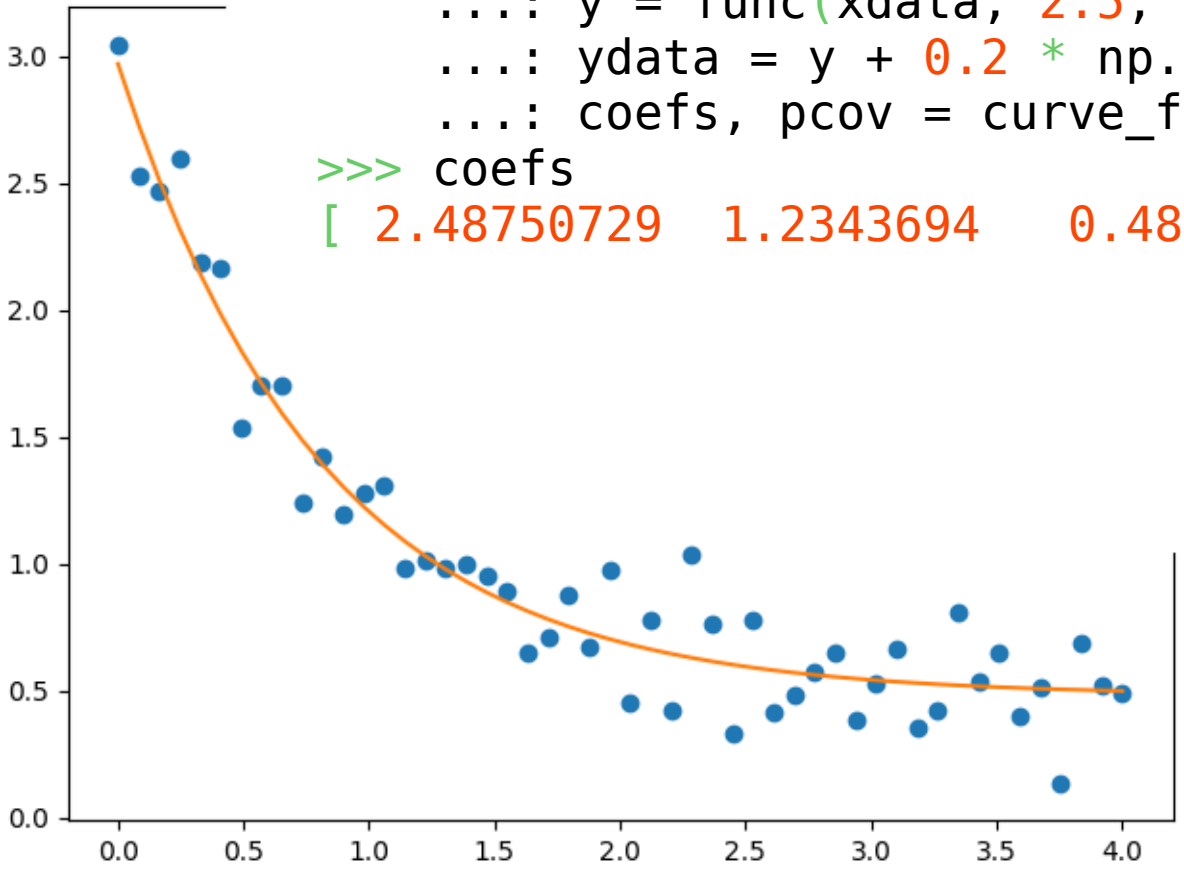
scipy.optimize: минимизация

Алгоритм Broyden-Fletcher-Goldfarb-Shanno (квази-метод Ньютона)

```
>>> opt.minimize(f, (0,0), method='bfgs',tol=1e-9)
      fun: 2.0055911509295948e-11
 hess_inv: array([[ 0.49982839,  0.99975297],
       [ 0.99975297,  2.00463718]])
      jac: array([  7.50728268e-10,  -3.72302189e-10])
  message: 'Optimization terminated successfully.'
     nfev: 104
      nit: 21
     njev: 26
   status: 0
  success: True
        x: array([ 0.99999552,  0.99999104])
```

scipy.optimize: подгонка

```
>>> from scipy.optimize import curve_fit
...:
...: def func(x, a, b, c):
...:     return a * np.exp(-b * x) + c
...:
...: xdata = np.linspace(0, 4, 50)
...: y = func(xdata, 2.5, 1.3, 0.5)
...: ydata = y + 0.2 * np.random.normal(size=len(xdata))
...: coefs, pcov = curve_fit(func, xdata, ydata)
>>> coefs
[ 2.48750729  1.2343694   0.48012604]
```



scipy.optimize: нахождение корней

$$a(x) = \cos(x) - \frac{x}{5}$$

```
>>> def a(x):
    ...:     return np.cos(x)-x/5

>>> opt.brentq(a, -np.pi-1,np.pi+1)
 -3.837467106499057

>>> opt.fsolve(a, 0)
/usr/lib/python3.6/site-packages/scipy/optimize/minpack.py:161:
RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last ten iterations.
  warnings.warn(msg, RuntimeWarning)
array([ 6.08177514])

>>> opt.fsolve(a, np.arange(10))
/usr/lib/python3.6/site-packages/scipy/optimize/minpack.py:161:
 RuntimeWarning: The iteration is not making good progress, as measured by the
  improvement from the last five Jacobian evaluations.
  warnings.warn(msg, RuntimeWarning)

array([  1.32933005,   1.30600582,   1.30724085,  -1.97136594,
         6.0750796 ,   6.04175474, -15.50945335,   6.14222244,
         6.3528904 ,   5.80149314])
```

scipy.optimize: нахождение корней

$$x^2 \cos(y) = 4$$
$$x\,y - y = \pi$$

```
>>> def f(x):
...     r=np.array([0,0],float)
...     r[0]=x[0]**2*np.cos(x[1])-4
...     r[1]=x[0]*x[1]-x[1]-np.pi
...     return r
...

>>> opt.fsolve(f,[1,1])
array([ 3.52667934,  1.24336817])
```

scipy.integrate – функции

```
>>> import scipy.integrate as igr

>>> igr.quad(np.sin,0,np.pi)
(2.0, 2.220446049250313e-14)

>>> igr.quad(np.sin,-np.Inf,np.Inf)
(0.0, 0.0)
```

quad	“technique from the Fortran library QUADPACK”
dblquad	Двойной интерал
tplquad	Тройной интеграл
fixed_quad	Метод Гаусса (по n точкам)
quadrature	Адаптивный метод Гаусса (с заданной точностью)
romberg	Метод Ромберга

scipy.integrate – набор данных

```
>>> import scipy.integrate as igr

>>> x=np.linspace(0,np.pi,10)
>>> igr.simps(np.sin(x))
 5.7282851768386545

>>> np.trapz(np.sin(x))
 5.6712818196177093
```

trapz	Метод трапеций
cumtrapz	Кумулятивный метод трапеций
romb	Метод Ромберга
simps	Метод Симпсона

scipy.integrate: системы дифф. уравнений

$$\frac{dy}{dx} = f(x, y)$$ Задача Коши – initial value problem

$$\frac{dy}{dt} = -2y$$
$$y(0) = 1, 5, 3$$

```
>>> from scipy.integrate import solve_ivp

>>> def der(y, t):
       return -2*y

>>> t=np.linspace(0,10,15)
>>> solve_ivp(der, [0,10], [1,5,3], t_eval=t).y

array([[  1.        ,   0.48979592,  -1.04081633,  -3.59183673,
         -7.16326531, -11.75510204, -17.36734694, -24.        ,
        -31.65306122, -40.32653061, -50.02040816, -60.73469388,
        -72.46938776, -85.2244898 , -99.        ],
       [  5.        ,   4.48979592,   2.95918367,   0.40816327,
         -3.16326531,  -7.75510204, -13.36734694, -20.        ,
        -27.65306122, -36.32653061, -46.02040816, -56.73469388,
        -68.46938776, -81.2244898 , -95.        ],
       [  3.        ,   2.48979592,   0.95918367,  -1.59183673,
         -5.16326531,  -9.75510204, -15.36734694, -22.        ,
        -29.65306122, -38.32653061, -48.02040816, -58.73469388,
        -70.46938776, -83.2244898 , -97.        ]])
```

scipy-lectures.org