Exercise4 SciPy

April 15, 2018

1 SciPy

The SciPy library is one of the core packages that make up the SciPy stack. It provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization.

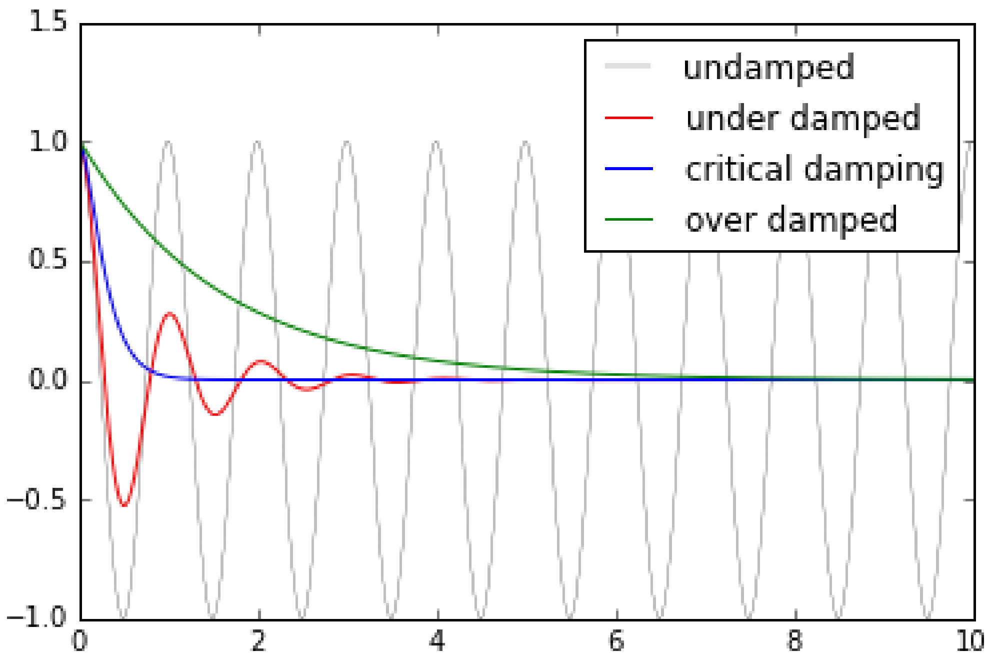
Library documentation: http://www.scipy.org/scipylib/index.html

1.1 Task1

What is t-test? See example in the end of the document

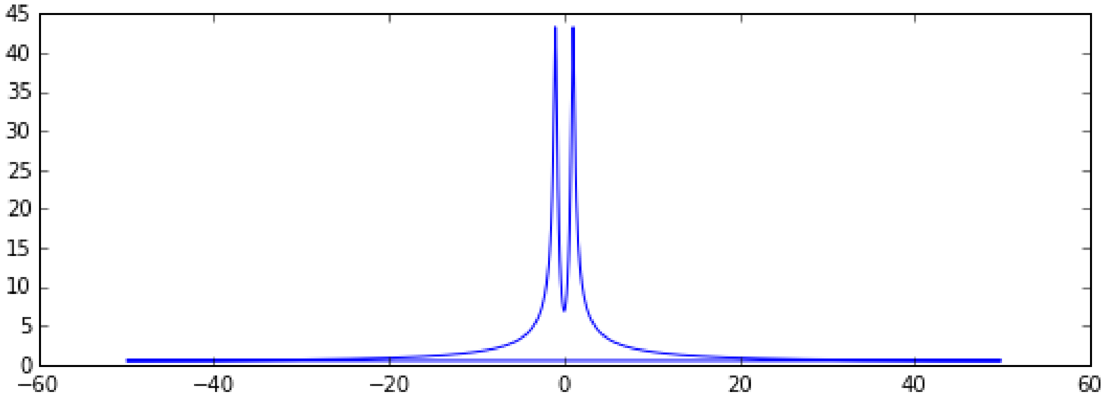
```
In [1]: # needed to display the graphs
        %matplotlib inline
        from pylab import *

In [2]: from numpy import *
        from scipy.integrate import quad, dblquad, tplquad

In [3]: # integration
        val, abserr = quad(lambda x: exp(-x ** 2),  Inf, Inf)
        val, abserr

Out[3]: (0.0, 0.0)

In [3]: from scipy.integrate import odeint, ode

In [6]: odeint?

In [5]: # differential equation
        def dy(y, t, zeta, w0):
            x, p = y[0], y[1]

            dx = p
            dp = -2 * zeta * w0 * p - w0**2 * x

            return [dx, dp]

        # initial state
```

```
y0 = [1.0, 0.0]

# time coordinate to solve the ODE for
t = linspace(0, 10, 1000)
w0 = 2*pi*1.0

# solve the ODE problem for three different values of the damping ratio
y1 = odeint(dy, y0, t, args=(0.0, w0)) # undamped
y2 = odeint(dy, y0, t, args=(0.2, w0)) # under damped
y3 = odeint(dy, y0, t, args=(1.0, w0)) # critial damping
y4 = odeint(dy, y0, t, args=(5.0, w0)) # over damped

fig, ax = subplots()
ax.plot(t, y1[:,0], 'k', label="undamped", linewidth=0.25)
ax.plot(t, y2[:,0], 'r', label="under damped")
ax.plot(t, y3[:,0], 'b', label=r"critical damping")
ax.plot(t, y4[:,0], 'g', label="over damped")
ax.legend();
```



```
In [6]: from scipy.fftpack import *

In [7]: # fourier transform
        N = len(t)
        dt = t[1]-t[0]
```

```
# calculate the fast fourier transform
# y2 is the solution to the under-damped oscillator from the previous section
F = fft(y2[:,0])

# calculate the frequencies for the components in F
w = fftfreq(N, dt)

fig, ax = subplots(figsize=(9,3))
ax.plot(w, abs(F));
```



1.1.1 Linear Algebra

```
In [8]: A = array([[1,2,3], [4,5,6], [7,8,9]])
        b = array([1,2,3])

In [9]: # solve a system of linear equations
        x = solve(A, b)
        x

Out[9]: array([-0.33333333,  0.66666667,  0.        ])

In [10]: # eigenvalues and eigenvectors
         A = rand(3,3)
         B = rand(3,3)

         evals, evecs = eig(A)

         evals

Out[10]: array([ 1.34211023,  0.22314378, -0.11368553])

In [11]: evecs
```

```
Out[11]: array([[-0.24560175, -0.93054452,  0.46758953],
                [-0.84521949, -0.00332278, -0.56599975],
                [-0.4746407 ,  0.36616371,  0.67897299]])
```

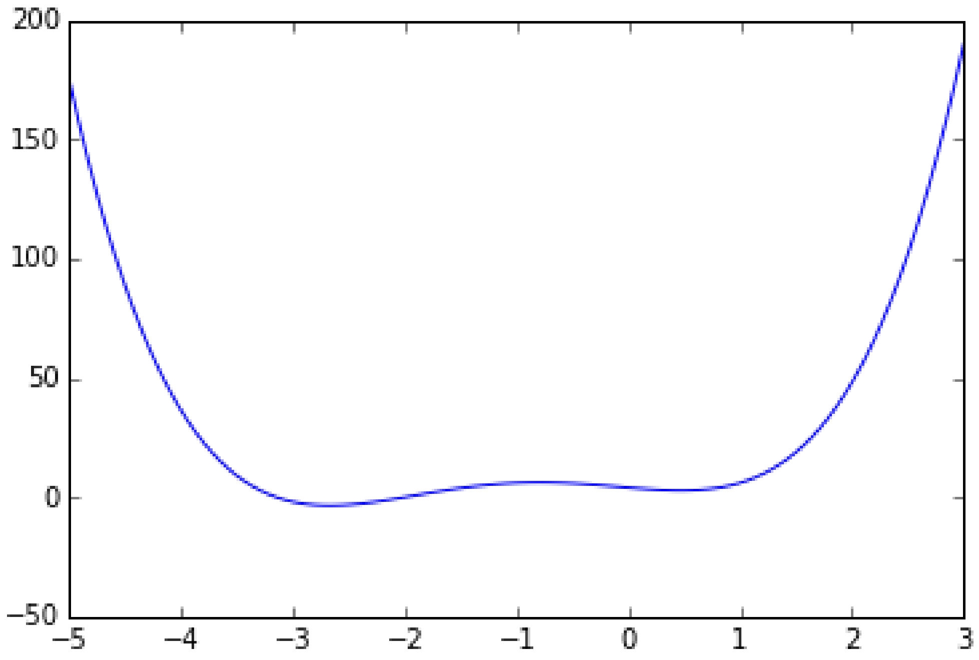
```
In [12]: svd(A)
```

```
Out[12]: (array([[-0.23411483,  0.96016594,  0.15255034],
                 [-0.84815445, -0.12501447, -0.51478677],
                 [-0.47520972, -0.24990547,  0.84363676]]),
          array([ 1.3457633 ,  0.23442523,  0.1079208 ]),
          array([[-0.21657397, -0.81553512, -0.53665463],
                 [ 0.7658064 ,  0.19902251, -0.61149865],
                 [-0.60550497,  0.54340824, -0.58143892]]))
```

1.1.2 Optimization

```
In [13]: from scipy import optimize
```

```
In [14]: def f(x):
             return 4*x**3 + (x-2)**2 + x**4

         fig, ax  = subplots()
         x = linspace(-5, 3, 100)
         ax.plot(x, f(x));
```



```
In [15]: x_min = optimize.fmin_bfgs(f, -0.5)
         x_min
```

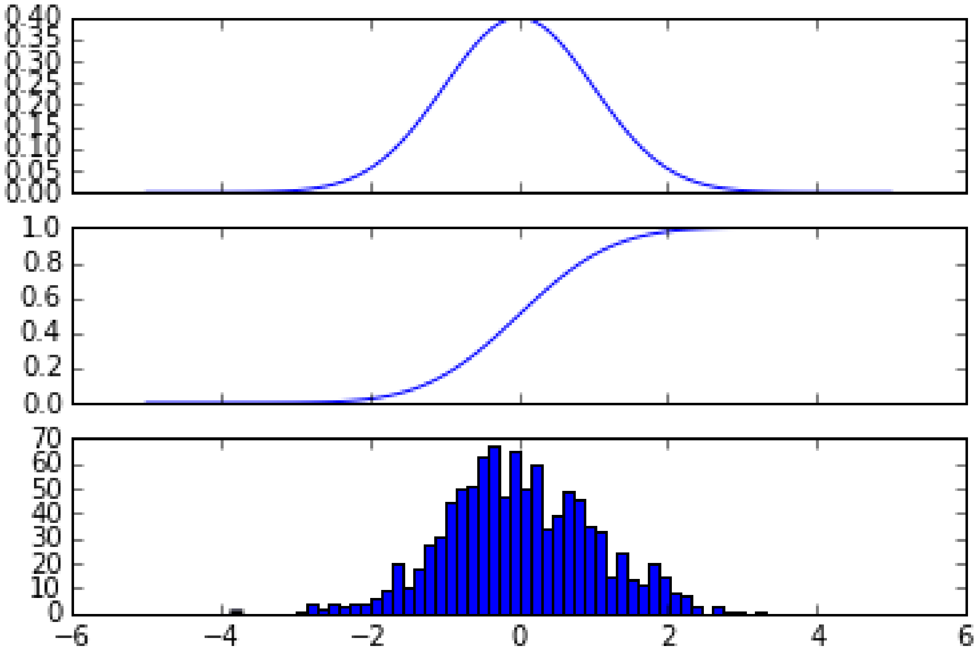
```
Optimization terminated successfully.
         Current function value: 2.804988
         Iterations: 4
         Function evaluations: 24
         Gradient evaluations: 8
```

```
Out[15]: array([ 0.46961745])
```

1.1.3 Statistics

```
In [16]: from scipy import stats
```

```
In [17]: # create a (continous) random variable with normal distribution
         Y = stats.norm()

         x = linspace(-5,5,100)

         fig, axes = subplots(3,1, sharex=True)

         # plot the probability distribution function (PDF)
         axes[0].plot(x, Y.pdf(x))

         # plot the commulative distributin function (CDF)
         axes[1].plot(x, Y.cdf(x));

         # plot histogram of 1000 random realizations of the stochastic variable Y
         axes[2].hist(Y.rvs(size=1000), bins=50);
```



```
In [18]: Y.mean(), Y.std(), Y.var()

Out[18]: (0.0, 1.0, 1.0)

In [19]: # t-test example
         t_statistic, p_value = stats.ttest_ind(Y.rvs(size=1000), Y.rvs(size=1000))
         t_statistic, p_value

Out[19]: (-1.193733722708372, 0.23272385793970249)
```