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

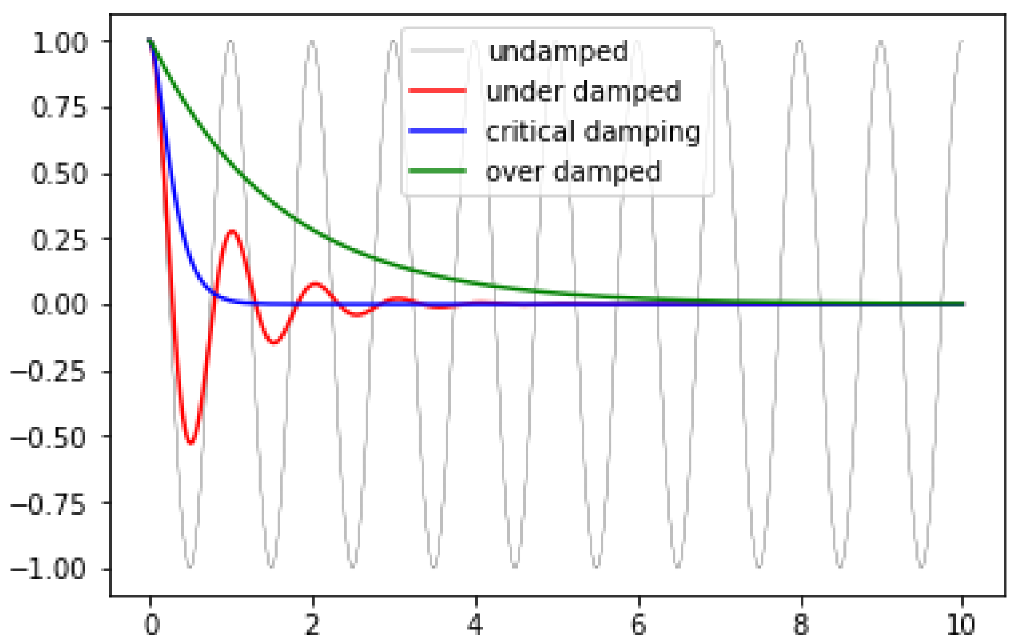
1.1 Task1

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

Answer: The t test mainly compares two mean values and lets us know the significant difference between them.

```
In [3]: # needed to display the graphs
        %matplotlib inline
        from pylab import *

In [4]: from numpy import *
        from scipy.integrate import quad, dblquad, tplquad

In [5]: # integration
        val, abserr = quad(lambda x: exp(-x ** 2),  Inf, Inf)
        val, abserr

Out[5]: (0.0, 0.0)

In [6]: from scipy.integrate import odeint, ode

In [7]: # 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
```

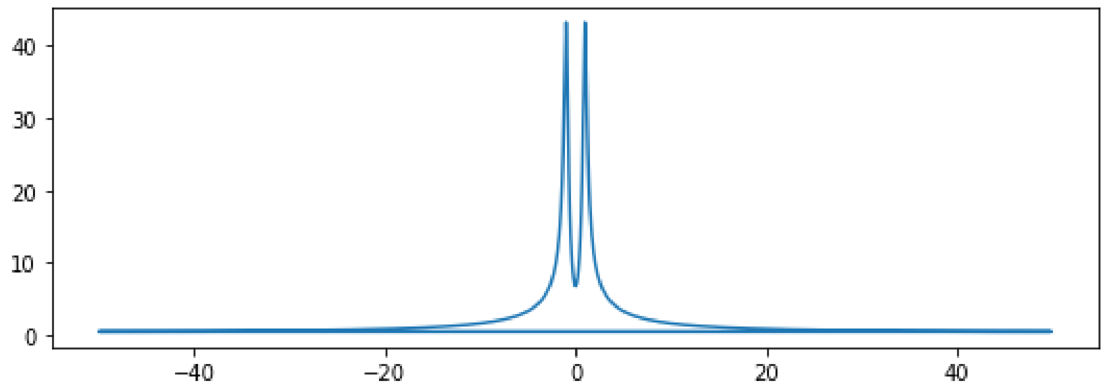
```
y0 = [1.0, 0.0]

# time coodinate 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 [8]: from scipy.fftpack import *
```

```
In [9]: # 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 [10]: A = array([[1,2,3], [4,5,6], [7,8,9]])
         b = array([1,2,3])

In [11]: # solve a system of linear equations
         x = solve(A, b)
         x

Out[11]: array([-0.23333333,  0.46666667,  0.1       ])

In [12]: # eigenvalues and eigenvectors
         A = rand(3,3)
         B = rand(3,3)

         evals, evecs = eig(A)

         evals

Out[12]: array([1.55375813, 0.12530276, 0.60946086])

In [13]: evecs

Out[13]: array([[ 0.78407807,  0.7877097 ,  0.7416807 ],
                [ 0.48056174,  0.13277072, -0.64156642],
                [ 0.39278748, -0.60156909,  0.19570965]])
```

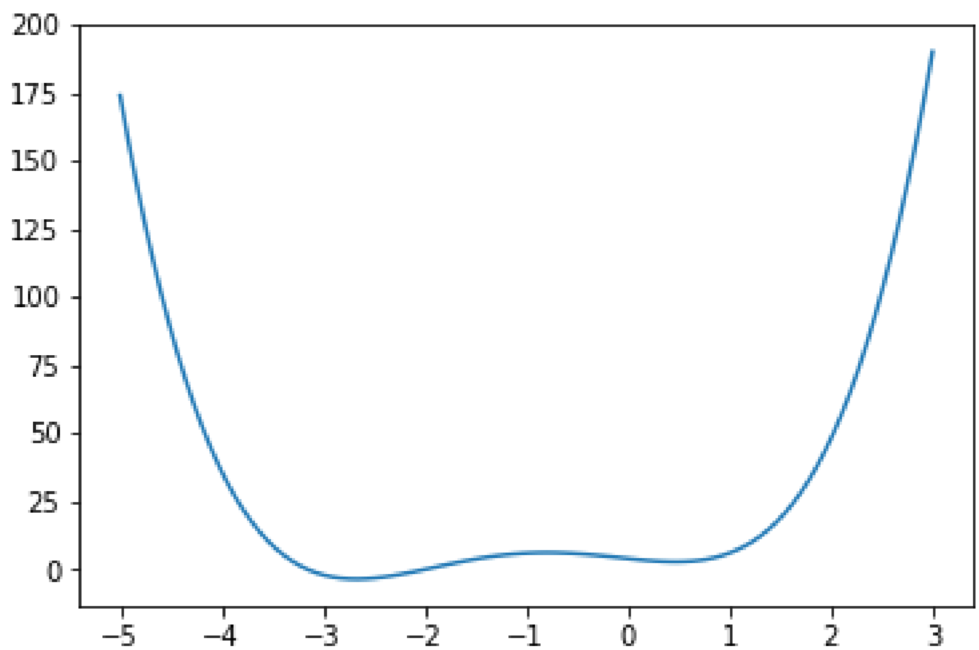
```
In [14]: svd(A)
```

```
Out[14]: (array([[-0.74797216,  0.51449424, -0.41932484],
                 [-0.51900446, -0.84717876, -0.11367729],
                 [-0.4137294 ,  0.13260401,  0.90069093]]),
          array([1.71910097, 0.62000701, 0.11132479]),
          array([[-0.46015198, -0.57183803, -0.6791623 ],
                 [ 0.46731389, -0.8064175 ,  0.36236521],
                 [-0.75490257, -0.1506389 ,  0.63830246]]))
```

1.1.2 Optimization

```
In [15]: from scipy import optimize
```

```
In [16]: 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 [17]: x_min = optimize.fmin_bfgs(f, -0.5)
         x_min
```

```
Optimization terminated successfully.
         Current function value: 2.804988
```

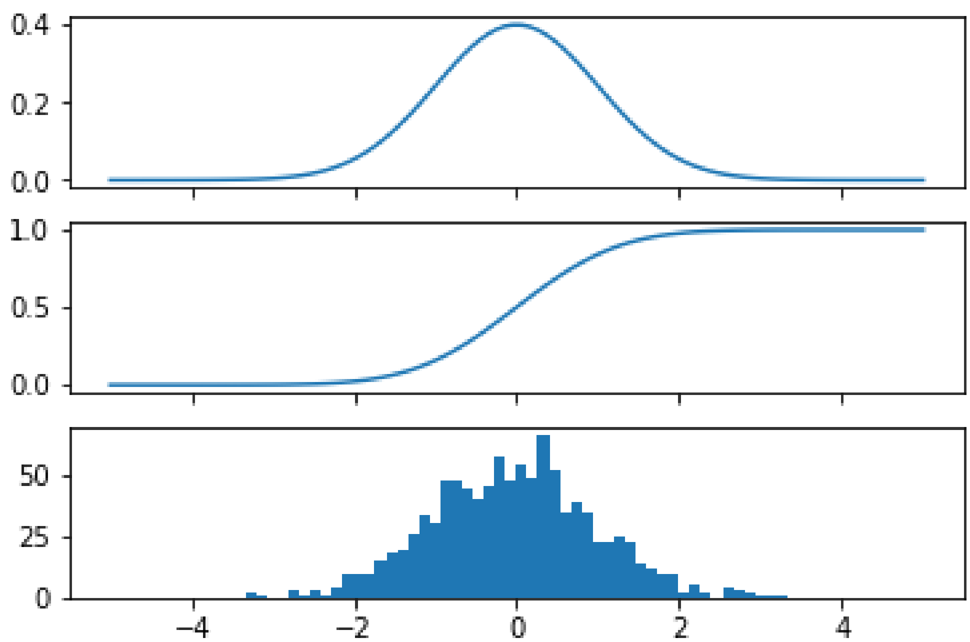
```
Iterations: 4
Function evaluations: 18
Gradient evaluations: 6
```

Out[17]: array([0.46961743])

1.1.3 Statistics

In [18]:
```
from scipy import stats
```

In [19]:
```
# 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 [20]: Y.mean(), Y.std(), Y.var()
```

```
Out[20]: (0.0, 1.0, 1.0)
```

```
In [21]: # t-test example
         t_statistic, p_value = stats.ttest_ind(Y.rvs(size=1000), Y.rvs(size=1000))
         t_statistic, p_value
```

```
Out[21]: (-0.6081300125911772, 0.5431703678985558)
```