Hochschule Bonn-Rhein-Sieg

Learning and Adaptivity, SS18

Assignment 01 (15-April-2018)

Sathiya Ramesh, Pradheep Krishna Muthukrishnan Padmanabhan, Naresh Kumar Gurulingan

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 (http://www.scipy.org/scipylib/index.html)

Task1

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

If we observe two independent samples from the same or different population, the test measures whether the average value differs significantly across samples. If we observe a large p-value, for example larger than 0.05 or 0.1, then we cannot reject the null hypothesis(ie there is no relationship between two measured phenomena). If the p-value is smaller than the threshold, e.g. 1%, 5% or 10%, then we reject the null hypothesis of equal averages. Reference : https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html)

In [21]:
```
from scipy import stats
np.random.seed(0)
# t test for similar distribution
rvs1 = stats.norm.rvs(loc=5,scale=10,size=500)
rvs2 = stats.norm.rvs(loc=5,scale=10,size=500)
print(stats.ttest_ind(rvs1,rvs2))
```

```
Ttest_indResult(statistic=0.63712464735944219, pvalue=0.52418977648171061)
```

In [22]:
```
#t test for  different distribution
rvs1 = stats.norm.rvs(loc=5,scale=10,size=500)
rvs2 = stats.norm.rvs(loc=15,scale=10,size=500)
stats.ttest_ind(rvs1,rvs2)
```

Out[22]:
```
Ttest_indResult(statistic=-15.808708304536363, pvalue=2.0870006162930632e-50)
```

In [4]:
```
# needed to display the graphs
%matplotlib inline
from pylab import *
```

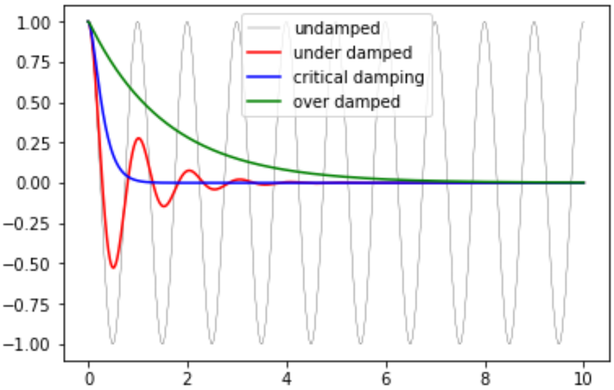
In [5]:
```
from numpy import *
from scipy.integrate import quad, dblquad, tplquad
```

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

Out[3]:
```
(0.0, 0.0)
```

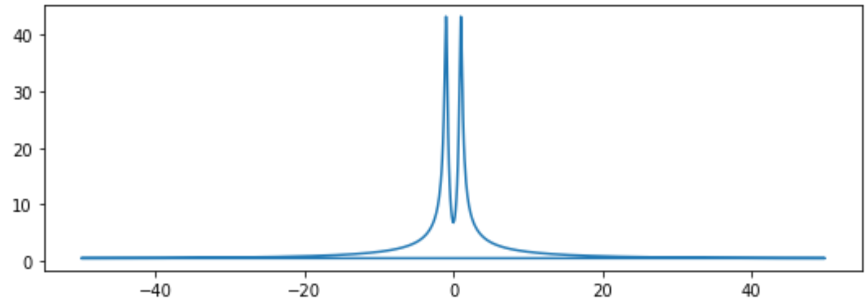
In [4]:
```
from scipy.integrate import odeint, ode
```

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 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 [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));
```



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([-0.23333333,  0.46666667,  0.1       ])
```

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

evals, evecs = eig(A)

evals
```

Out[10]:
```
array([ 1.53141394+0.        j, -0.30073306+0.01693434j,
       -0.30073306-0.01693434j])
```

In [11]:
```
evecs
```

Out[11]:
```
array([[ 0.62642903+0.        j,  0.2369939 +0.02930752j,
         0.2369939 -0.02930752j],
       [ 0.39083337+0.        j,  0.27368498-0.04534046j,
         0.27368498+0.04534046j],
       [ 0.67441527+0.        j, -0.93059967+0.        j,
        -0.93059967-0.        j]])
```

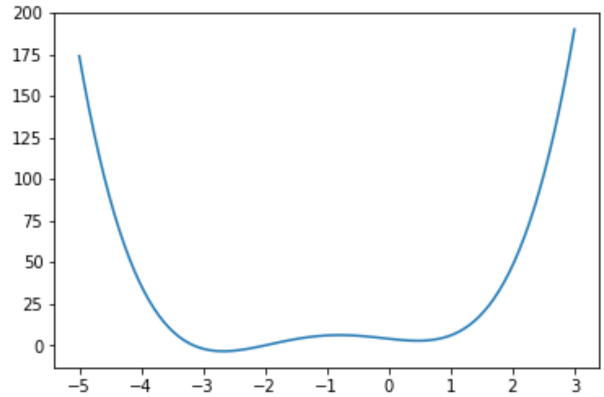
In [12]:
```
svd(A)
```

Out[12]:
```
(array([[-0.58499244, -0.12700336, -0.80103308],
        [-0.34253315, -0.85656668,  0.38595928],
        [-0.73515637,  0.50016365,  0.45758217]]),
 array([1.70639751, 0.46280173, 0.17593598]),
 array([[-0.75462467, -0.59293227, -0.28102123],
        [-0.47124877,  0.78776387, -0.39667704],
        [ 0.45658099, -0.16691138, -0.87388466]]))
```

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
```

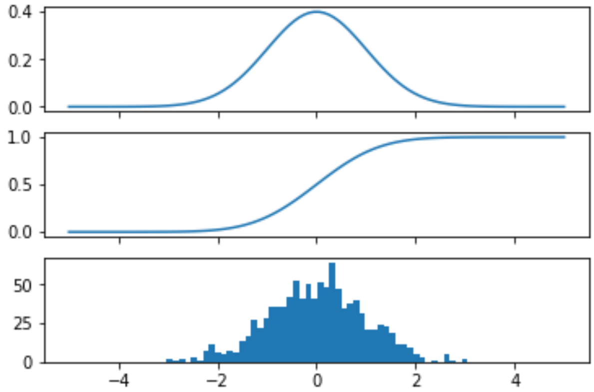
```
Optimization terminated successfully.
         Current function value: 2.804988
         Iterations: 4
         Function evaluations: 18
         Gradient evaluations: 6
```

Out[15]:
```
array([0.46961743])
```

Statistics

In [1]:
```
from scipy import stats
```

In [7]:
```
# 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]: (0.23912536205282356, 0.8110329013998367)