Day 19

We worked in excel solving differential equations.

Below is some scripts about using Gaussian quadrature that I decided to abandon for now.

In [1]:

```
######################################################################
#
# Functions to calculate integration points and weights for Gaussian
# quadrature
#
# x,w = gaussxw(N) returns integration points x and integration
#           weights w such that sum_i w[i]*f(x[i]) is the Nth-order
#           Gaussian approximation to the integral int_{-1}^1 f(x) dx
# x,w = gaussxwab(N,a,b) returns integration points and weights
#           mapped to the interval [a,b], so that sum_i w[i]*f(x[i])
#           is the Nth-order Gaussian approximation to the integral
#           int_a^b f(x) dx
#
# This code finds the zeros of the nth Legendre polynomial using
# Newton's method, starting from the approximation given in Abramowitz
# and Stegun 22.16.6.  The Legendre polynomial itself is evaluated
# using the recurrence relation given in Abramowitz and Stegun
# 22.7.10.  The function has been checked against other sources for
# values of N up to 1000.  It is compatible with version 2 and version
# 3 of Python.
#
# Written by Mark Newman <mejn@umich.edu>, June 4, 2011
# You may use, share, or modify this file freely
#
######################################################################

from numpy import ones,copy,cos,tan,pi,linspace

def gaussxw(N):

    # Initial approximation to roots of the Legendre polynomial
    a = linspace(3,4*N-1,N)/(4*N+2)
    x = cos(pi*a+1/(8*N*N*tan(a)))

    # Find roots using Newton's method
    epsilon = 1e-15
    delta = 1.0
    while delta>epsilon:
        p0 = ones(N,float)
        p1 = copy(x)
        for k in range(1,N):
            p0,p1 = p1,((2*k+1)*x*p1-k*p0)/(k+1)
        dp = (N+1)*(p0-x*p1)/(1-x*x)
        dx = p1/dp
        x -= dx
        delta = max(abs(dx))

    # Calculate the weights
    w = 2*(N+1)*(N+1)/(N*N*(1-x*x)*dp*dp)

    return x,w

def gaussxwab(N,a,b):
    x,w = gaussxw(N)
    return 0.5*(b-a)*x+0.5*(b+a),0.5*(b-a)*w
```

In [2]:

```
gaussxw(8)
```

Out[2]:

```
(array([ 0.96028986,  0.79666648,  0.52553241,  0.18343464, -0.18343464,
        -0.52553241, -0.79666648, -0.96028986]),
 array([0.10122854, 0.22238103, 0.31370665, 0.36268378, 0.36268378,
        0.31370665, 0.22238103, 0.10122854]))
```

In [3]:

```
def GaussLagQuad8(function):
    x=np.asarray([1.7027963230510100e-1, 9.0370177679937991e-1,\
       2.2510866298661307,    4.2667001702876588,\
       7.0459054023934657,    1.0758516010180995e+1,\
       1.5740678641278005e+1, 2.2863131736889264e+1])
    w=np.asarray([3.6918858934163753e-1, 4.1878678081434296e-1,\
       1.7579498663717181e-1, 3.3343492261215652e-2,\
       2.7945362352256725e-3, 9.0765087733582131e-5,\
       8.4857467162725315e-7, 1.0480011748715104e-9])
    integral = np.sum(w*function(x))
    return(integral)

def gaussHermQuad8(function):
    x = np.asarray([-0.38118699,-1.157193712,-1.981656757,-2.93063742,0.38118699,1.157193712,1.981656757,2.930637
```

```python
    w = np.asarray([0.661147013,0.207802326,0.017077983,0.000199604,0.661147013,0.207802326,0.017077983,.00019960
    integral = np.sum(w*function(x))
    return(integral)

def gaussLegQuad8(function):
    x = np.asarray([-0.183434643,-0.52553241,-0.796666477,-0.960289857,0.183434643,0.52553241,0.796666477,0.96028
    w = np.asarray([0.362683783,0.313706646,0.222381035,0.101228536,0.362683783,0.313706646,0.222381035,0.1012285
    integral = np.sum(w*function(x))
```

In [ ]: