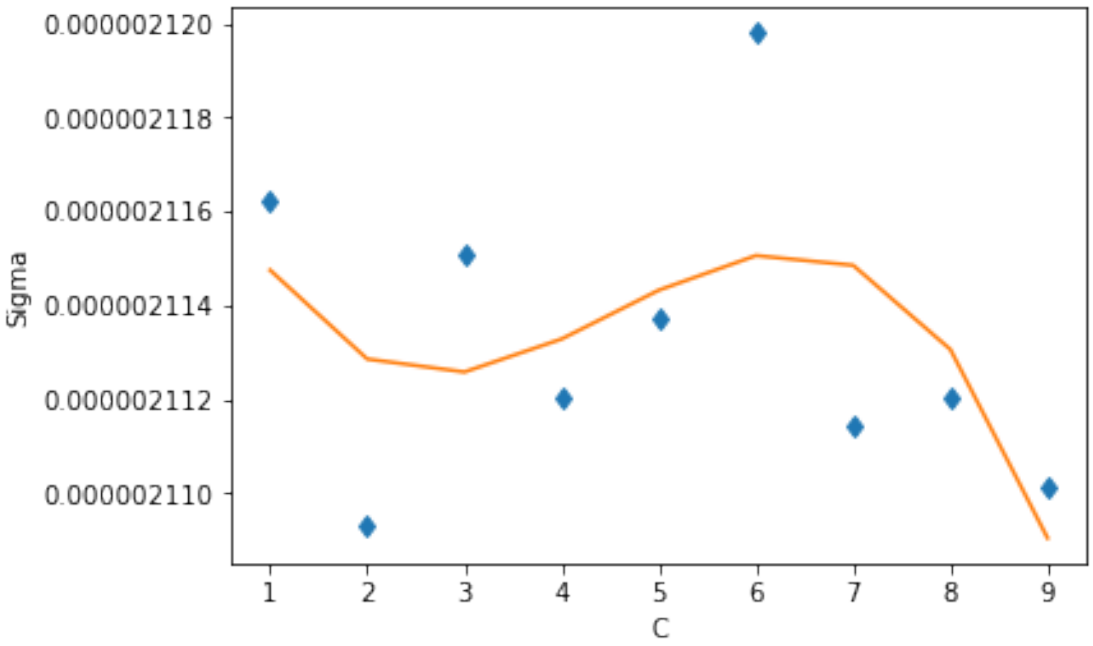
PS5

April 20, 2017

In [1]:
```
import numpy as np
from numpy.polynomial import Polynomial as P
#import plotly
#import plotly.plotly as py
#import plotly.figure_factory as ff
import matplotlib.pyplot as plt
#Integrand function
def f(x,H):
    return (x-5)*np.exp(-(x/2-3))+H

#Calculates the coefficients of linear weight function.
def findw(f,H,lower,upper,normalize):
    #Find the linear function.
    slope=(f(upper,H)-f(lower,H))/(upper-lower)
    a=slope
    b=-slope*upper+f(upper,H)
    #Normalization.
    A=(a/2)*(upper**2)+b*upper-(a/2)*(lower**2)-b*lower
    if normalize:
        a/=A
        b/=A
    return [a,b]
#Performs integration.
def integrate(f,lower,upper,N,C):
    H=C
    w=findw(f,H,lower,upper,True)
    #Generate uniform random inputs.
    inputs=np.random.rand(N)
    a=w[0]/2
    b=w[1]
    c=-(a*lower**2+b*lower)

    SUM=0
    SUM2=0

    inverse_inputs=[]
    for i in inputs:
        p=[(-b-np.sqrt(b**2-4*a*(c-i)))/(2*a),(-b+np.sqrt(b**2-4*a*(c-i)))/
```

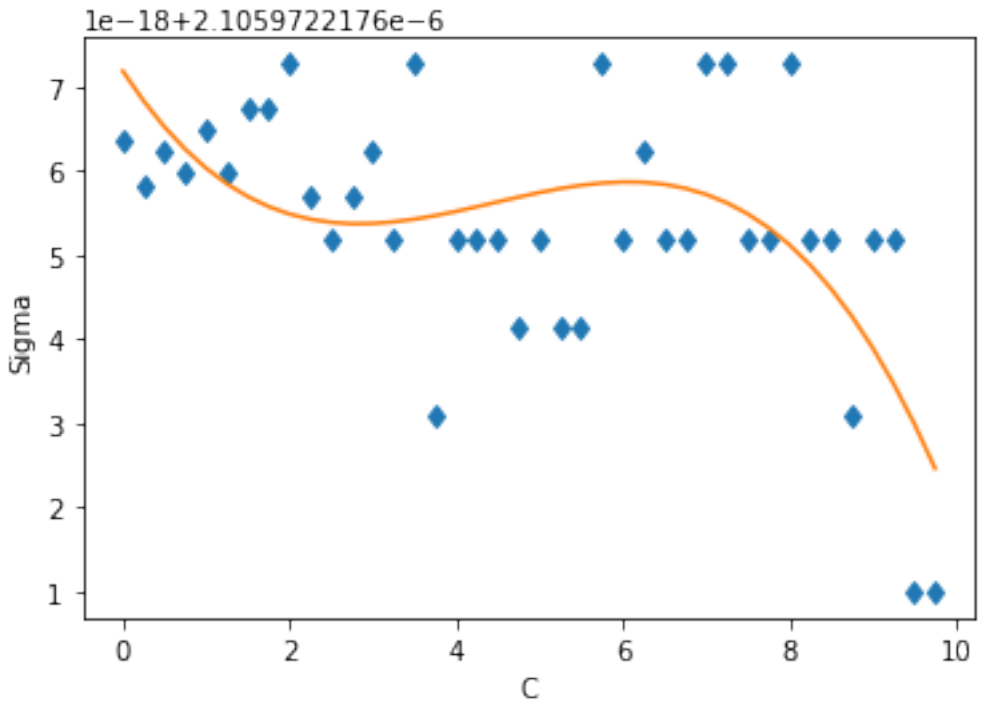
```
            if p[0]>=lower and p[0]<=upper:
                inverse_inputs.append(p[0])
            else :
                inverse_inputs.append(p[1])

        inverse_inputs=np.array(inverse_inputs)
        #Calculate f(inverse(x))/w(inverse(x)).
        outputsF=f(inverse_inputs,H)
        outputsW=w[0]*(inverse_inputs)+w[1]
        outputs=outputsF/outputsW
        SUM=outputs.sum()
        SUM2=(outputs*outputs).sum()
        var=SUM2/N-(SUM/N)**2
        var=var/N
        #Store generated points for variance calculation.
        Vsum=outputs.sum()
        return Vsum/N-H*(upper-lower),(upper-lower)**2*var

sigmas=[]
sigma=0
I=0
l=[9.5,10]
C=np.arange(1,10,1)
for c in C:
    I=0
    sigma=0
    temp_sigmas=[]
    temp_results=[]
    for i in range (0,len(l)-1):
        for p in range (0,100):
            temp,temp2=integrate(f,l[i],l[i+1],1000,c)
            temp_sigmas.append(temp2)
            temp_results.append(temp)
        sigma+=np.mean(temp_sigmas)
        I+=np.mean(temp_results)
    sigmas.append(np.sqrt(sigma))

plt.plot(C,sigmas,'d')
plt.ylabel('Sigma')
plt.xlabel('C')
z = np.polyfit(C,sigmas,3)
p = np.poly1d(z)
plt.plot(C,p(C))
plt.show()
print(np.argmax(sigmas))
```



5

In [2]:
```
def theoretical_sigma(f,lower,upper,N,C):

    w=findw(f,C,lower,upper,True)

    flower=f(lower,C)
    fmiddle=f((lower+upper)/2,C)
    fupper=f(upper,C)

    wlower=w[0]*lower+w[1]
    wmiddle=w[0]*(lower+upper)/2+w[1]
    wupper=w[0]*upper+w[1]

    gupper=fupper/wupper
    gmiddle=fmiddle/wmiddle
    glower=flower/wlower

    b=(gupper-glower)/(upper-lower)
    c=2*(gupper-2*gmiddle+glower)/(upper-lower)**2
    meanu=0
    meanu2=2/3*((upper-lower)/2)**3
    meanu2/=(upper-lower)
    meanu3=0
```

```
    meanu4=2/5*((upper-lower)/2)**5
    meanu4/=(upper-lower)

    #var=b**2*(meanu2-meanu**2)+2*b*c*(meanu3-meanu*meanu2)+c**2*(meanu4-me
    var=c**2*(meanu4-meanu2**2)
    var/=N

    return (upper-lower)*np.sqrt(var)




C=np.arange(0,10,0.25)
sigmas=[]
for i in C:
    sigmas.append(theoretical_sigma(f,9.5,10,1000,i))


plt.plot(C,sigmas,'d')
plt.ylabel('Sigma')
plt.xlabel('C')
z = np.polyfit(C,sigmas,3)
p = np.poly1d(z)
plt.plot(C,p(C))
plt.show()

print(np.argmax(sigmas))
```



8

In [ ]: