PHY407
Lab Assignment #3: Gaussian Quadrature and Numerical Differentiation

Q1 by Hayley Agler, Q2,3 by Nicholas Pavanel

September 2020

Q.1

a) i.

The Dawson function

$$D(x) = e^{-x^2} \int_0^x e^{t^2} \tag{1}$$

was calculated at x=4 using three different methods of integration: the Trapezoidal rule Simpson's rule, and Gaussian quadrature. The accuracy of such methods increases with larger N values. For the Trapezoidal rule, machine precision (the maximum accuracy with which the computer can represent the result) for $N \approx 10^8$, whereas the accuracy of Simpon's rule reaches this point at $N \approx 10^4$ [1]. Gaussian quadrature is able to return exact answers for a small number of sample points. This can be seen in the table below as the "true" value of the Dawson function at x=4 (taken as the value given by scipy.special.dawsn) is, to 6 significant digits, 0.129348, and Gaussian quadrature returns this result with only N=32. The Trapezoidal rule does not give successfully return the same result even at N=2048, and Simpson's rule returns this result for N=512.

N (slices/sample)	Trapezoidal Rule	Simpson's Rule	Gaussian Quadrature
8	0.2622478	0.18269096	0.1290106
32	0.13958009	0.130011181	0.129348
128	0.129998302	0.129350941	0.129348
512	0.129388688	0.12934801	0.129348
2048	0.12935054	0.129348001	0.129348

Table 1: Dawson function for x=4 evaluated at various values of N using the Trapezoidal rule, Simpson's rule, and Gaussian quadrature.

a) ii.

The relative error compared to the true value of the Dawson function at x=4 was calculated by $\frac{gaussianvalue - truevalue}{truevalue}$ where *gaussianvalue* is the value found using Gaussian quadrature, and similarly for the Trapezoidal and Simpson's rule error. The Gaussian error was also calculated using equation 1: $\epsilon = I_{2N} - I_N$. Figure 1 shows a plot of the magnitude of the relative errors. From Figure 1, it can be seen that the relative Trapezoidal error is the largest, then the Simpson's rule error, and finally the Gaussian quadrature error. The Simpson's and Trapezoidal error get smaller with increasing N, however the Gaussian quadrature error fluctuates with different N. The error estimate from equation (1) gives similar values as the relative error for the Gaussian quadrature.

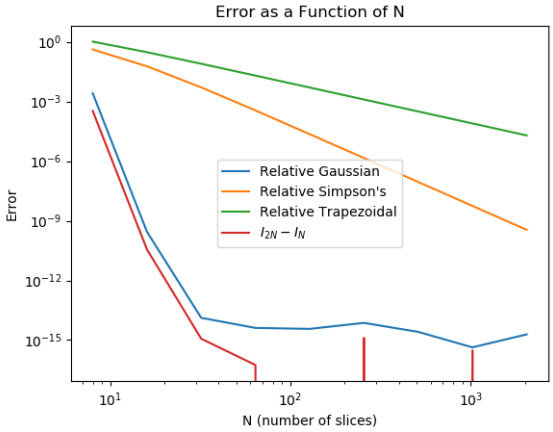

Figure 1

b)

A Gaussian quadrature with N=100 was used to calculate the probability of blowing snow from the following equation:

$$P\left(u_{10}, T_a, t_h\right) = \frac{1}{\sqrt{2\pi}\delta} \int_0^{u_{10}} \exp\left[-\frac{(\bar{u}-u)^2}{2\delta^2}\right] \mathrm{d}u \tag{2}$$

for values of $u_{10} = (6, 8, 10)$, and $t_h = (24, 48, 72)$ hours. The probability P is plotted as a function of Ta in Figure 2. The probability of blowing snow is greatest near -20 °C, but for higher windspeeds the probability is higher for lower temperatures, and for lower windspeeds the probabiity is higher for warmer temperatures. For u_{10}=10, the probability is greatest near -26 °C, but for u_{10}=6, the probability is greatest near -18 °C. The higher the wind strength, the the greater the probability of blowing snow, which makes sense as stronger winds will cause the blowing snow. The probability of blowing snow decreases with the age of the snow, which makes sense since snow compacts under its weight and becomes more dense with time.

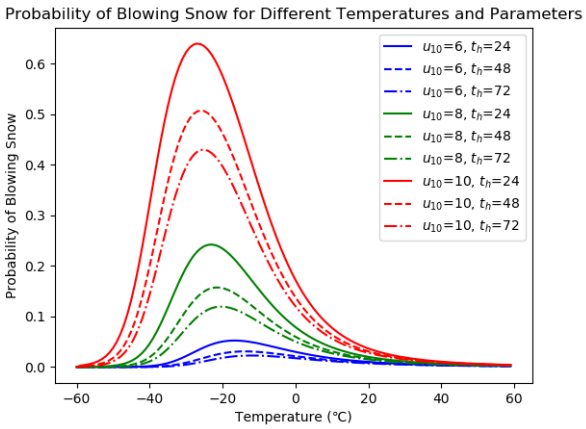

Figure 2: Probability of blowing snow for varying temperatures (T_a), wind strength (u_{10}), and age of snow (t_h).

Q.2

a)

See Figure 3. Note that, unlike the classical harmonic oscillator (CHO), the ground state energy wavefunction, $n = 0$, is not zero. This agrees with predictions that even in the ground state there are quantum fluctuations. Further difference from the CHO can be recognized when we picture the probability density of the quantum harmonic oscillator (QHO) at low energy levels such as the plotted energy levels of Figure 3. The probability density, $\|\psi_n(x)\|^2$, tells us the probability of finding the QHO at position x. The squares of the plots of Figure 3 imply that the QHO is likely to be found close to the origin for low energy levels, whereas the CHO would be likely to be found at its turning points.

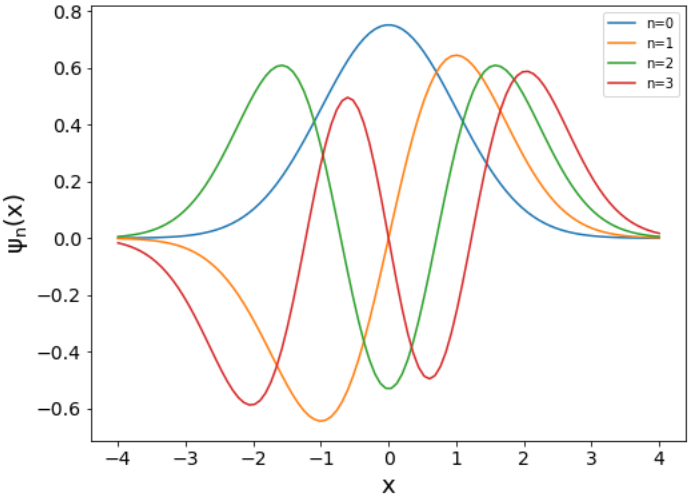

Figure 3: This plot shows a quantum harmonic oscillator's wavefunctions over a domain of $-4 \leq x \leq 4$ *for energy levels* $n = 0, 1, 2, 3$.

b)

See Figure 4.

c)

See Table 2. Note that the quantum uncertainty of position and momentum, as well as the mean square of the position and momentum, are equal for the QHO for all energy levels. We know that the Uncertainty Principle applies to position and momentum, and that if one measures either the position or momentum to a specific certainty the uncertainty in the other will be larger. This exercise shows that when we are not measuring either position or momentum, or when we are measuring them with equal certainty, the uncertainty in both are equal as well. We may also note that the energy of the QHO is quantized, and increases from its ground state energy of 0.5 by 1 for each singular increase in energy level.

Q.3

a)

```
#Pseudocode
```

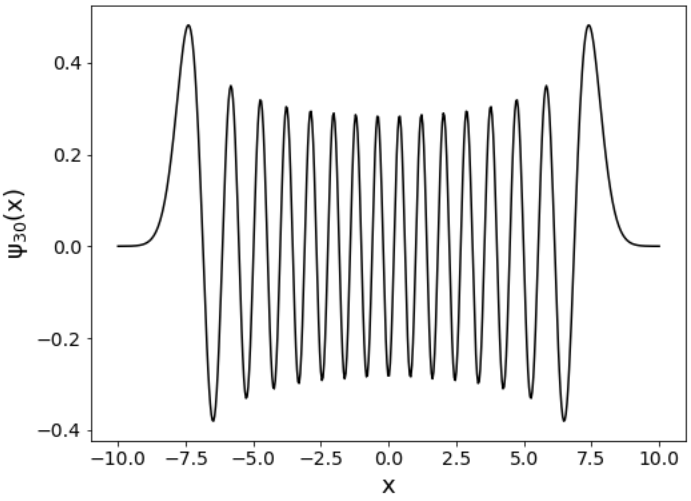

Figure 4: This plot shows a quantum harmonic oscillator's wavefunction over a domain of $-4 \leq x \leq 4$ for the 30^{th} energy level ($n = 30$). Note that if we now consider $\|\psi_n(x)\|^2$, the QHO begins to resemble the CHO in that the oscillator is most likely to be found close to its turning points. This behaviour continues for higher energy levels in accordance with the Correspondence Principle.

```
#creating w(x,y)
#open file
#initialize an 2d array of zeros size 1201x1201
#create a double for loop over the length of the tile: 1201
#for each iteration of the loop:
    #read a line of the file
    #unpacked the line
    #set the correct location in the zeros array to the unpacked line value
        #this will be arcoss the first row of the array, then the second, and so on
        #the double for loop travels across the top row, then the second, and so on

#calculating the gradient of w
#create a double for loop that will run over the rows of w(x,y)
#initialize an list that will hold the derivatives at each point
#for each iteration (point in w(x,y)), append to the initialized list a tuple: (dw/dx,dw/dy)
#at each iteration of the outside loop, a new list is initalized to mimic w(x,y)
#for the the first point, use a forward dif scheme, middle points central, end backwards

#create plots of w and I
#imshow the array for w
#compute I from the eqn at the bottom of page 212 in the text (assume light shining horizontally w unit
#imshow I
```

b)

Figures 5, 6, and 7 show that elevation maps are better at showing large ranges of elevations, whereas intensity maps are better at showing detail at small ranges of elevations. In the case of Lake Geneva, the elevation map only shows a rough approximation where the lake is; it shows the general depression that the lake resides in. But it also shows a mountainous region in the lower right hand corner. The intensity map clearly shows the full coast of the lake. However, to achieve this level of detail large intensity values were

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13
$\langle X^2 \rangle$	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5	10.5	11.5	12.51	13.49
$\langle P^2 \rangle$	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5	10.5	11.5	12.5	13.48
$\langle \sqrt{X^2} \rangle$	0.71	1.22	1.58	1.87	2.12	2.35	2.55	2.74	2.92	3.08	3.24	3.39	3.54	3.67
$\langle \sqrt{P^2} \rangle$	0.71	1.22	1.58	1.87	2.12	2.35	2.55	2.74	2.92	3.08	3.24	3.39	3.54	3.67
E	0.5	1.5	2.5	3.5	4.5	5.5	6.5	7.5	8.5	9.5	10.5	11.5	12.5	13.48

n	14	15
$\langle X^2 \rangle$	14.43	15.49
$\langle P^2 \rangle$	14.5	15.63
$\langle \sqrt{X^2} \rangle$	3.8	3.94
$\langle \sqrt{P^2} \rangle$	3.8	3.94
E	14.47	15.56

Table 2: This table shows the output of a program that calculates $\langle X^2 \rangle = \int_{-\infty}^{\infty} x^2 |\psi_n(x)|^2 dx$, $\langle P^2 \rangle = \int_{-\infty}^{\infty} |\frac{d\psi_n(x)}{dx}|^2 dx$, $\sqrt{\langle X^2 \rangle}$, $\sqrt{\langle P^2 \rangle}$, *and* $E = \frac{1}{2}(\langle X^2 \rangle + \langle P^2 \rangle)$ *for energy levels* ($n = 0, 1, ..., 15$) *of a quantum harmonic oscillator.*

mapped to zero as described in the caption of Figure 5. As a result, the lake is clearly shown in the intensity map, but the mountainous region in the bottom right hand corner is non-existent. In the case of the island of O'ahu, Hawaii, as shown in Figures 6 and 7, the elevation map clearly shows two mountain ranges, as well as the tallest peaks in each mountain range, but it struggles to provide the outline of the island's coasts. The intensity map picks up the coasts of the island better, and possibly even picks out the small island of Moku Manu at (633,168), but the peaks of the mountain ranges cannot be seen as easily. Thus, we may conclude that elevation maps are better at showing large ranges of elevations, whereas intensity maps are better at showing detail at small ranges of elevations.

Code

Q1

a)

```
#import necessary modules
import numpy as np
from scipy import special
from gaussxw import gaussxw
import matplotlib.pyplot as plt

#taking from my solution in lab2
#define function we want to integrate
def f(x):
    return np.exp(x**2)

#define empty arrays and N1,N2 arrays
a=0
b=4

N1=np.array([2**3,2**4,2**5,2**6,2**7,2**8,2**9,2**10,2**11])
h1=np.zeros(len(N1))
h2=np.zeros(len(N1))
N2=np.array([2**3*2,2**4*2,2**5*2,2**6*2,2**7*2,2**8*2,2**9*2,2**10*2,2**11*2])
```

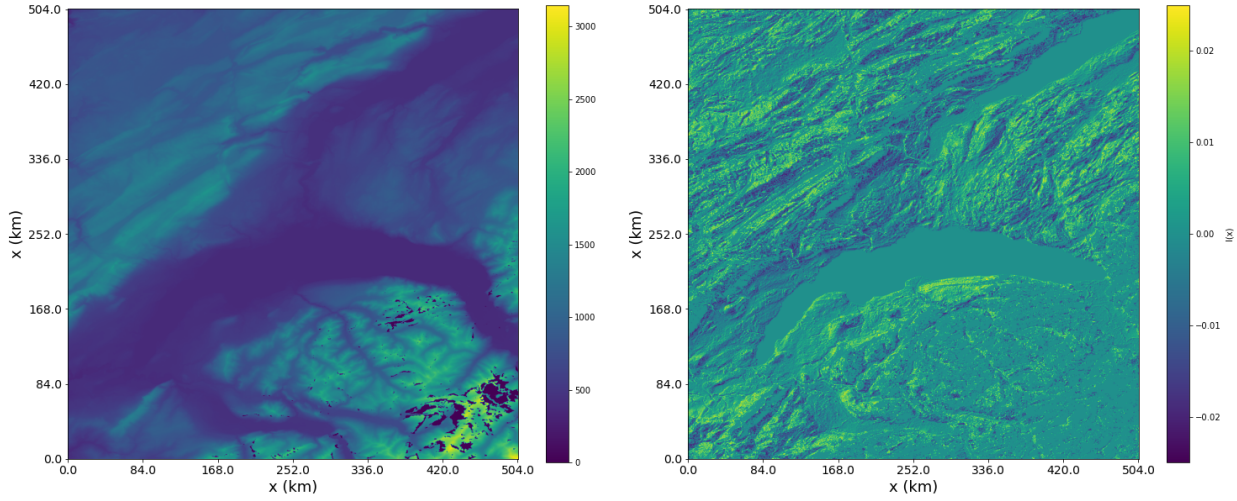

Figure 5: On left is an elevation map of Lake Geneva. Elevation values larger than 25,000 have been set to 0 to preserve the contrast of the map. These values largely occur in the bottom right area of the map. This change does not effect Lake Geneva which can be seen as the dark area in the central part of the map. On the right is an intensity map of Lake Geneva. Intensity values larger than 0.025 have been set to 0 to increase the contrast in which Lake Geneva can be seen. These values also largely occur in the bottom right area of the map. This change helps to clearly display Lake Geneva, which can be see as the area of smooth intensity in the central location of the map. Note that the axes are not to physical scale, but rather to the scale of a distance of 420m between values.

```
s_trap=np.zeros(len(N1))
I1_trap=np.zeros(len(N1))
s_simp=np.zeros(len(N1))
I1_simp=np.zeros(len(N1))

#trapezoidal and simpsons method copied from my lab2 code
#compute integral with trapezoidal method
for i in range(len(N1)):
    h1[i]=(b-a)/N1[i]

    s_trap[i] = 0.5*f(a) + 0.5*f(b)

    for k in range(1,np.int(N1[i])):
        s_trap[i] += f(a+k*h1[i])

    I1_trap[i]=s_trap[i]*h1[i]*np.exp(-4**2)



#redo the integral using simpsons method
for i in range(len(N1)):

    s_simp[i]=f(a)+f(b)
    #odd terms
    for k in range(1,np.int(N1[i]),2):
```

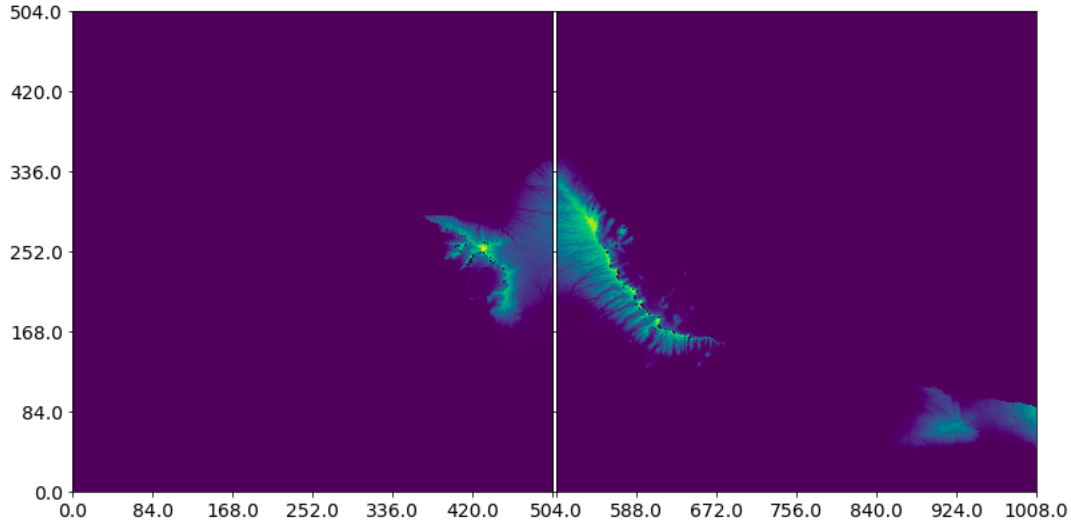

Figure 6: This plot shows an elevation map of O'ahu, Hawaii. Elevation values larger than 25,000 have been set to 0 to preserve the contrast of the map.Note that the axes are not to physical scale, but rather to the scale of a distance of 420m between values.

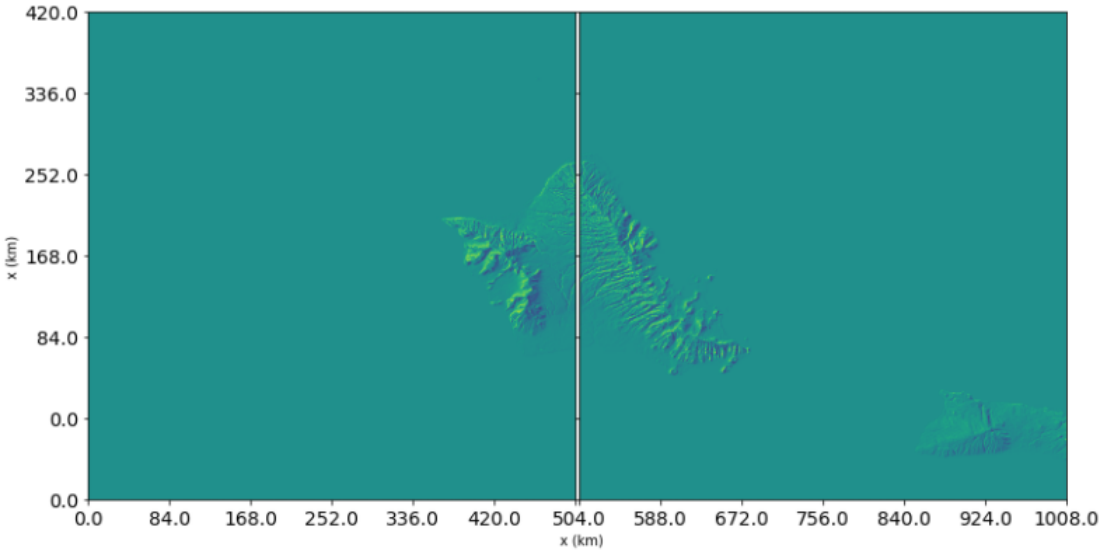

Figure 7: This plot shows an intensity map of O'ahu, Hawaii. Intensity values larger than 0.1 have been set to 0 to increase the contrast in which Lake Geneva can be seen. Note that the axes are not to physical scale, but rather to the scale of a distance of 420m between values.

```
        s_simp[i] += 4*f(a+k*h1[i])
        #even terms
    for k in range(2,np.int(N1[i]),2):
        s_simp[i] += 2*f(a+k*h1[i])

    #multiply by other part of function
    I1_simp[i]=s_simp[i]*(1/3)*h1[i]*np.exp(-4**2)



#gaussian quad
#define empty arrays to store info in for each of the different N values
x2=[0,0,0,0,0,0,0,0,0,0]
w2=[0,0,0,0,0,0,0,0,0,0]
xp2=[0,0,0,0,0,0,0,0,0,0]
wp2=[0,0,0,0,0,0,0,0,0,0]
s2_gaus=np.zeros(len(N1))
I2_gaus=np.zeros(len(N1))
x=[0,0,0,0,0,0,0,0,0,0]
w=[0,0,0,0,0,0,0,0,0,0]
xp=[0,0,0,0,0,0,0,0,0,0]
wp=[0,0,0,0,0,0,0,0,0,0]
s_gaus=np.zeros(len(N1))
I1_gaus=np.zeros(len(N1))
error_gaus=np.zeros(len(N1))

#for loop to calculate integral for different N1, N2 values
for i in range(len(N1)):
    x[i],w[i] = gaussxw(N1[i])
    xp[i] =0.5*(b-a)*x[i] + 0.5*(b+a)
    wp[i] = 0.5*(b-a)*w[i]

    # Perform the integration
    s_gaus[i] = 0.0
    for k in range(N1[i]):
        s_gaus[i] += wp[i][k]*f(xp[i][k])

    I1_gaus[i]=s_gaus[i]*np.exp(-4**2)


    x2[i],w2[i] = gaussxw(N2[i])

    xp2[i] =0.5*(b-a)*x2[i] + 0.5*(b+a)
    wp2[i] = 0.5*(b-a)*w2[i]

    # Perform the integration for N2 values
    s2_gaus[i] = 0.0
    for k in range(N2[i]):
        s2_gaus[i] += wp2[i][k]*f(xp2[i][k])

    I2_gaus[i]=s2_gaus[i]*np.exp(-4**2)

#equation 1 computation of error
    error_gaus[i]=I2_gaus[i]-I1_gaus[i]
```

```
#%% plotting relative error

#calculate relative error
rel_gaus=np.zeros(len(N1))
for i in range(len(N1)):
    rel_gaus[i]=(np.abs(I1_gaus[i]-special.dawsn(4)))/special.dawsn(4)

rel_simp=np.zeros(len(N1))
for i in range(len(N1)):
    rel_simp[i]=np.abs(I1_simp[i]-special.dawsn(4))/special.dawsn(4)

rel_trap=np.zeros(len(N1))
for i in range(len(N1)):
    rel_trap[i]=np.abs(I1_trap[i]-special.dawsn(4))/special.dawsn(4)

#plot the errors
plt.figure(1)
plt.title("Error as a Function of N")
plt.xscale("log")
plt.yscale("log")
plt.xlabel("N (number of slices)")
plt.ylabel("Error")
plt.plot(N1, rel_gaus, label="Relative Gaussian")
plt.plot(N1, rel_simp, label="Relative Simpson's")
plt.plot(N1, rel_trap, label="Relative Trapezoidal")
plt.plot(N1, error_gaus, label="$I_{2N}-I_N$")
plt.legend()
```

b)

```
#import necessary modules
import numpy as np
from gaussxw import gaussxw
import matplotlib.pyplot as plt


#Ta avg hourly temperature in C
Ta=np.arange(-60,60,1)
th= (24, 48, 72) #hours
u10=(6, 8, 10) #m/s

#define function we want to integrate
def P(u, u_mean1,stdev):
    return np.exp(-(u_mean1-u)**2/(2*stdev**2))

stdev=np.zeros(len(Ta))
u_mean1=np.zeros((len(th),len(Ta)))

N=100
a=0
b=u10
```

```
for i in range(len(Ta)):
    for j in range(len(th)):
        stdev[i]=4.3+0.145*Ta[i] + 0.00196*Ta[i]**2
        u_mean1[j,i]=11.2+0.365*Ta[i] +0.00706*Ta[i]**2 +0.9*np.log(th[j])

#gaussian quad calculation

x,w = gaussxw(N)
xp1=[0,0,0]
wp1=[0,0,0]
for j in range(len(th)):
    xp1[j] =0.5*(b[j]-a)*x + 0.5*(b[j]+a)
    wp1[j] = 0.5*(b[j]-a)*w

s_gaus1=np.zeros((len(th),len(u10),len(Ta)))
# Perform the integration
for i in range(len(Ta)):
    for k in range(N):
        for j in range(len(th)): #row,col
            for l in range(len(th)):
                s_gaus1[j,l,0] = 0.0
                s_gaus1[j,l,i] += wp1[j][k]*P(xp1[j][k], u_mean1[l,i], stdev[i])


I_gaus1=np.zeros((len(th),len(u10),len(Ta)))
for i in range(len(Ta)):
    for j in range(len(th)): #row,col
        for l in range(len(th)):
            I_gaus1[j,l,i]=s_gaus1[j,l,i]*1/(np.sqrt(2*np.pi)*stdev[i])



plt.xlabel("Temperature (\u2103)")
plt.ylabel("Probability of Blowing Snow")
plt.title("Probability of Blowing Snow for Different Temperatures and Parameters")
plt.plot(Ta, I_gaus1[0,0,:], label="$u_{10}$=6, $t_{h}$=24", color="blue")
plt.plot(Ta, I_gaus1[0,1,:], label="$u_{10}$=6, $t_{h}$=48", linestyle='dashed', color="blue")
plt.plot(Ta, I_gaus1[0,2,:], label="$u_{10}$=6, $t_{h}$=72",linestyle='dashdot', color="blue")
plt.plot(Ta, I_gaus1[1,0,:], label="$u_{10}$=8, $t_{h}$=24", color="green")
plt.plot(Ta, I_gaus1[1,1,:],linestyle='dashed', label="$u_{10}$=8, $t_{h}$=48", color="green")
plt.plot(Ta, I_gaus1[1,2,:], label="$u_{10}$=8, $t_{h}$=72",linestyle="dashdot", color="green")
plt.plot(Ta, I_gaus1[2,0,:], label="$u_{10}$=10, $t_{h}$=24", color="red")
plt.plot(Ta, I_gaus1[2,1,:],linestyle='dashed', label="$u_{10}$=10, $t_{h}$=48", color="red")
plt.plot(Ta, I_gaus1[2,2,:], label="$u_{10}$=10, $t_{h}$=72",linestyle='dashdot', color="red")
plt.legend()
```

Q2

```
#import needed modules
import numpy as np
import matplotlib.pyplot as plt
```

```
#to check function is working properly, define function for first 5 Hermite polynomials
#h_0 = 1
def h_1(x):
    return 2 * x
def h_2(x):
    return 4 * x**2 - 2
def h_3(x):
    return 8 * x**3 - 12 * x
def h_4(x):
    return 16 * x**4 * - 48 * x**2 + 12
def h_5(x):
    return 32 * x**5 - 160 * x**3 + 120 * x

#2a)

#define a recursive function to compute any Hermite polynomial at x
#for understanding: substitute n=n-1 into eqn 6 from lab handout
def h_poly(n,x):
    if n < 0:
        return -1
    elif n == 0:
        return 1
    elif n == 1:
        return 2 * x
    else:
        return 2 * x * h_poly(n-1,x) - 2 * (n-1) * h_poly(n-2,x)

#check that h_poly(n,x) works
print('1st Hermite Polynomial @ x=2 -','h_poly(1,2):',h_poly(1,2),'Answer:',h_1(2))
print('2nd Hermite Polynomial @ x=3 -','h_poly(2,3):',h_poly(2,3),'Answer:',h_2(3))
print('3rd Hermite Polynomial @ x=5 -','h_poly(3,5):',h_poly(3,5),'Answer:',h_3(5))
print('5th Hermite Polynomial @ x=15 -','h_poly(5,15):',h_poly(5,15),'Answer:',h_5(15))

#define a function to compute the harmonic oscillator wavefunction psi
def psi(n,x):
    return (1/np.sqrt(2**n * np.math.factorial(n) * np.sqrt(np.pi))) * np.exp((-x**2)/(2))
    * h_poly(n,x)

#plot the harmonic oscillator wavefunction for a range of ns on -4<=x<=4
x_a=np.linspace(-4,4,100)
plt.figure(figsize=[8,6])
plt.plot(x_a,psi(0,x_a),label='n=0')
plt.plot(x_a,psi(1,x_a),label='n=1')
plt.plot(x_a,psi(2,x_a),label='n=2')
plt.plot(x_a,psi(3,x_a),label='n=3')
plt.legend()
plt.xlabel(r'$\rm x $',fontsize=18)
plt.ylabel(r'$\rm \psi_{n}(x) $',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('1_a')
plt.show()

#2b)
```

```
#plot the harmonic oscillator wavefunction for a range of ns on -4<=x<=4
x_b=np.linspace(-10,10,500)
plt.figure(figsize=[8,6])
plt.plot(x_b,psi(30,x_b),color='black')
plt.xlabel(r'$\rm x $',fontsize=18)
plt.ylabel(r'$\rm \psi_{30}(x) $',fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.savefig('1_b')
plt.show()

#2c)

#from lecture notes, define gaussxw (for -1->1)
def gaussxw(N):
    # Initial approximation to roots of the Legendre polynomial
    a = np.linspace(3,4*N-1,N)/(4*N+2)
    x = np.cos(np.pi*a+1/(8*N*N*np.tan(a)))
    # Find roots using Newton's method
    epsilon = 1e-15
    delta = 1.0
    while delta>epsilon:
        p0 = np.ones(N,float)
        p1 = np.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

#define the integrand for 1c_x
def integrand_1c_x(n,x,dx):
    return x**2 * np.abs(psi(n,x))**2 * dx


# for both integrals i use change of variables 5.73 from the text

#compute <x^2>
#set integrand bounds
a=-1
b=1
#calculate the sample points and weights
N=100
x,w = gaussxw(N)
ns=[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
#compute the integral using change of variables from the text 5.73
sums_1c_x=[]
for n in ns:
    sum_1c_x=0
```

```
    for i in range(N):
        sum_1c_x += w[i] * integrand_1c_x(n, x[i]/(1 - x[i]**2), (1 + x[i]**2)/(1 - x[i]**2)**2)
    sums_1c_x.append(sum_1c_x)
#check the integration has been successful by printing n=5
print(' Sqrt of n=5 should equal approx. 2.35: n =',str(ns[5]),'=',str(np.sqrt(sums_1c_x[5])))

#compute <p^2>

#define a function to compute the harmonic oscillator wavefunction psi
def d_psi(n,x):
    return (1/np.sqrt(2**n * np.math.factorial(n) * np.sqrt(np.pi))) * np.exp((-x**2)/(2))
    * (-x * h_poly(n,x) + 2 * n * h_poly(n-1,x))

#define the integrand for 1c_p
def integrand_1c_p(n,x,dx):
    return np.abs(d_psi(n,x))**2 * dx

#compute the integral using change of variables from text 5.73
sums_1c_p=[]
for n in ns:
    sum_1c_p=0
    for i in range(N):
        sum_1c_p += w[i] * integrand_1c_p(n, x[i]/(1 - x[i]**2), (1 + x[i]**2)/(1 - x[i]**2)**2)
    sums_1c_p.append(sum_1c_p)

#compute E = 1/2 (<x^2> + <p^2>)
E = 0.5 * (np.array(sums_1c_x) + np.array(sums_1c_p))

#plot position, momentum, energy
plt.plot(ns,np.sqrt(sums_1c_x),label='X')
plt.plot(ns,np.sqrt(sums_1c_p),label='P')
#plt.plot(ns,np.array(sums_1c_x)-np.array(sums_1c_p),label='X-P')
plt.plot(ns,E,label='E')
plt.legend()
plt.show()
```

Q3

Please note that this code includes plots for O'ahu, Hawaii and thus it requires the relevant data to finish running. The code submitted as .py files has the Hawaii data commented out.

```
#b)

#import needed modules
import struct
import numpy as np
import matplotlib.pyplot as plt
from time import time

#initialize w to store the elevation data
w = np.zeros((1201,1201))

#double loop through the import data
f = open('N46E006.hgt','rb')
for i in range(1201):
```

```
    for j in range(1201):
        buf = f.read(2)
        val = struct.unpack('>h',buf)[0]
        if val > 25000 or val < - 25000:
            w[i][j] = 0
        else:
            w[i][j] = val #set w[0][0] northwesternmost point, w[-1][-1] southeasternmost point

#set x labels
x = np.array([0,200,400,600,800,1000,1200])
x = x*420/1000
#plot elevation data w
plt.figure(figsize=[12,10])
plt.xticks([0,200,400,600,800,1000,1200],x,fontsize=14)
plt.yticks([1200,1000,800,600,400,200,0],x,fontsize=14)
plt.ylabel('x (km)',fontsize=18)
plt.xlabel('x (km)',fontsize=18)
plt.imshow(w)
plt.colorbar(label='y (m)')
#plt.savefig('Q3_b_1')

#define derivative methods
def fwd_dif(fx_h,fx,h):
    result=0
    result += (fx_h - fx)/(h)
    return result

def bwd_dif(fx_h,fx,h):
    result=0
    result += (fx - fx_h)/(h)
    return result

#calculate the derivatives of w
#set conditions
h = 420
phi = (-5 * np.pi) / 6
#initialize the list that will hold the derivatives
dw = []
for i in range(1201):
    hold=[]
    for j in range(1201):
        if i == 0 and j == 0:
            dwdx = fwd_dif(w[i][j+1], w[i][j], h)
            dwdy = fwd_dif(w[i+1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif i == 0 and j == 1200:
            dwdx = bwd_dif(w[i][j-1], w[i][j], h)
            dwdy = fwd_dif(w[i+1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif i == 1200 and j == 0:
            dwdx = fwd_dif(w[i][j+1], w[i][j], h)
            dwdy = bwd_dif(w[i-1][j], w[i][j], h)
```

```
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif i == 1200 and j == 1200:
            dwdx = bwd_dif(w[i][j-1], w[i][j], h)
            dwdy = bwd_dif(w[i-1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif i == 0:
            dwdx = fwd_dif(w[i][j+1], w[i][j], h)
            dwdy = fwd_dif(w[i+1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif j == 0:
            dwdx = fwd_dif(w[i][j+1], w[i][j], h)
            dwdy = fwd_dif(w[i+1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif i == 1200:
            dwdx = fwd_dif(w[i][j+1], w[i][j], h)
            dwdy = bwd_dif(w[i-1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        elif j == 1200:
            dwdx = bwd_dif(w[i][j-1], w[i][j], h)
            dwdy = fwd_dif(w[i+1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
        else:
            dwdx = fwd_dif(w[i][j+1], w[i][j], h)
            dwdy = fwd_dif(w[i+1][j], w[i][j], h)
            partials = (dwdx,dwdy)
            hold.append(partials)
    dw.append(hold)

#note that dw has dimensions below:
#dw[i][j][k] - i determines the row, j the position in the row (column), k dwdx[0] or dwdy[1]

#calculate I
I = np.zeros((1201,1201))
for i in range(1201):
    for j in range(1201):
        I[i][j] += - (np.cos(phi) * dw[i][j][0] + np.sin(phi) * dw[i][j][1])
        / np.sqrt(dw[i][j][0]**2 + dw[i][j][1]**2 + 1)

# fix broken values
for i in range(1201):
    for j in range(1201):
        if I[i][j] > 0.025 or I[i][j] < -0.025:
            I[i][j] = 0
        else:
            continue

plt.figure(figsize=[12,10])
plt.imshow(I)
```

```
plt.xticks([0,200,400,600,800,1000,1200],x,fontsize=14)
plt.yticks([1200,1000,800,600,400,200,0],x,fontsize=14)
plt.ylabel('x (km)',fontsize=18)
plt.xlabel('x (km)',fontsize=18)
plt.colorbar(label='I(x)')
#plt.savefig('Q3_b_2')
plt.show()
plt.close()

#set arrays for hawaii data
haw1 = np.zeros((1201,1201))
haw2 = np.zeros((1201,1201))

#get hawaii1 data
f = open('N21W159.hgt','rb')
for i in range(1201):
    for j in range(1201):
        buf = f.read(2)
        val = struct.unpack('>h',buf)[0]
        if val > 25000 or val < - 25000:
            haw1[i][j] = 0
        else:
            haw1[i][j] = val #set w[0][0] northwesternmost point, w[-1][-1] southeasternmost point

#get hawaii2 data
f = open('N21W158.hgt','rb')
for i in range(1201):
    for j in range(1201):
        buf = f.read(2)
        val = struct.unpack('>h',buf)[0]
        if val > 25000 or val < - 25000:
            haw2[i][j] = 0
        else:
            haw2[i][j] = val #set w[0][0] northwesternmost point, w[-1][-1] southeasternmost point

#calculate the derivatives of haw1 and haw2, should have made a function -.-
#set conditions
h = 420
phi = (-5 * np.pi) / 6
#initialize the list that will hold the derivatives
dw_haw1 = []
dw_haw2 = []
for i in range(1201):
    hold_haw1=[]
    hold_haw2=[]
    for j in range(1201):
        if i == 0 and j == 0:
            dwdx_haw1 = fwd_dif(haw1[i][j+1], haw1[i][j], h)
            dwdy_haw1 = fwd_dif(haw1[i+1][j], haw1[i][j], h)
            partials_haw1 = (dwdx_haw1,dwdy_haw1)
            hold_haw1.append(partials_haw1)

            dwdx_haw2 = fwd_dif(haw2[i][j+1], haw2[i][j], h)
            dwdy_haw2 = fwd_dif(haw2[i+1][j], haw2[i][j], h)
```

```
    partials_haw2 = (dwdx_haw2,dwdy_haw2)
    hold_haw2.append(partials_haw2)
elif i == 0 and j == 1200:
    dwdx_haw1 = bwd_dif(haw1[i][j-1], haw1[i][j], h)
    dwdy_haw1 = fwd_dif(haw1[i+1][j], haw1[i][j], h)
    partials_haw1 = (dwdx_haw1,dwdy_haw1)
    hold_haw1.append(partials_haw1)

    dwdx_haw2 = bwd_dif(haw2[i][j-1], haw2[i][j], h)
    dwdy_haw2 = fwd_dif(haw2[i+1][j], haw2[i][j], h)
    partials_haw2 = (dwdx_haw2,dwdy_haw2)
    hold_haw2.append(partials_haw2)
elif i == 1200 and j == 0:
    dwdx_haw1 = fwd_dif(haw1[i][j+1], haw1[i][j], h)
    dwdy_haw1 = bwd_dif(haw1[i-1][j], haw1[i][j], h)
    partials_haw1 = (dwdx_haw1,dwdy_haw1)
    hold_haw1.append(partials_haw1)

    dwdx_haw2 = fwd_dif(haw2[i][j+1], haw2[i][j], h)
    dwdy_haw2 = bwd_dif(haw2[i-1][j], haw2[i][j], h)
    partials_haw2 = (dwdx_haw2,dwdy_haw2)
    hold_haw2.append(partials_haw2)
elif i == 1200 and j == 1200:
    dwdx_haw1 = bwd_dif(haw1[i][j-1], haw1[i][j], h)
    dwdy_haw1 = bwd_dif(haw1[i-1][j], haw1[i][j], h)
    partials_haw1 = (dwdx_haw1,dwdy_haw1)
    hold_haw1.append(partials_haw1)

    dwdx_haw2 = bwd_dif(haw2[i][j-1], haw2[i][j], h)
    dwdy_haw2 = bwd_dif(haw2[i-1][j], haw2[i][j], h)
    partials_haw2 = (dwdx_haw2,dwdy_haw2)
    hold_haw2.append(partials_haw2)
elif i == 0:
    dwdx_haw1 = fwd_dif(haw1[i][j+1], haw1[i][j], h)
    dwdy_haw1 = fwd_dif(haw1[i+1][j], haw1[i][j], h)
    partials_haw1 = (dwdx_haw1,dwdy_haw1)
    hold_haw1.append(partials_haw1)

    dwdx_haw2 = fwd_dif(haw2[i][j+1], haw2[i][j], h)
    dwdy_haw2 = fwd_dif(haw2[i+1][j], haw2[i][j], h)
    partials_haw2 = (dwdx_haw2,dwdy_haw2)
    hold_haw2.append(partials_haw2)
elif j == 0:
    dwdx_haw1 = fwd_dif(haw1[i][j+1], haw1[i][j], h)
    dwdy_haw1 = fwd_dif(haw1[i+1][j], haw1[i][j], h)
    partials_haw1 = (dwdx_haw1,dwdy_haw1)
    hold_haw1.append(partials_haw1)

    dwdx_haw2 = fwd_dif(haw2[i][j+1], haw2[i][j], h)
    dwdy_haw2 = fwd_dif(haw2[i+1][j], haw2[i][j], h)
    partials_haw2 = (dwdx_haw2,dwdy_haw2)
    hold_haw2.append(partials_haw2)
elif i == 1200:
    dwdx_haw1 = fwd_dif(haw1[i][j+1], haw1[i][j], h)
```

```
            dwdy_haw1 = bwd_dif(haw1[i-1][j], haw1[i][j], h)
            partials_haw1 = (dwdx_haw1,dwdy_haw1)
            hold_haw1.append(partials_haw1)

            dwdx_haw2 = fwd_dif(haw2[i][j+1], haw2[i][j], h)
            dwdy_haw2 = bwd_dif(haw2[i-1][j], haw2[i][j], h)
            partials_haw2 = (dwdx_haw2,dwdy_haw2)
            hold_haw2.append(partials_haw2)
        elif j == 1200:
            dwdx_haw1 = bwd_dif(haw1[i][j-1], haw1[i][j], h)
            dwdy_haw1 = fwd_dif(haw1[i+1][j], haw1[i][j], h)
            partials_haw1 = (dwdx_haw1,dwdy_haw1)
            hold_haw1.append(partials_haw1)

            dwdx_haw2 = bwd_dif(haw2[i][j-1], haw2[i][j], h)
            dwdy_haw2 = fwd_dif(haw2[i+1][j], haw2[i][j], h)
            partials_haw2 = (dwdx_haw2,dwdy_haw2)
            hold_haw2.append(partials_haw2)
        else:
            dwdx_haw1 = fwd_dif(haw1[i][j+1], haw1[i][j], h)
            dwdy_haw1 = fwd_dif(haw1[i+1][j], haw1[i][j], h)
            partials_haw1 = (dwdx_haw1,dwdy_haw1)
            hold_haw1.append(partials_haw1)

            dwdx_haw2 = fwd_dif(haw2[i][j+1], haw2[i][j], h)
            dwdy_haw2 = fwd_dif(haw2[i+1][j], haw2[i][j], h)
            partials_haw2 = (dwdx_haw2,dwdy_haw2)
            hold_haw2.append(partials_haw2)
    dw_haw1.append(hold_haw1)
    dw_haw2.append(hold_haw2)

#set y labels
y = [0,1200,1000,800,600,400,200,0]
y = np.array(y)*420/1000
#set x labels
x1 = np.array([0,0,200,400,600,800,1000])
x1 = x1*420/1000
x2 = np.array([1200,1200,1400,1600,1800,2000,2200,2400])
x2 = x2*420/1000

#subplot hawaii
fig = plt.figure(figsize=[12,10])
ax1 = fig.add_axes([0,0.5,0.5,0.5])
ax2 = fig.add_axes([0.42,0.5,0.5,0.5],yticklabels=[])
ax1.imshow(haw1)
ax2.imshow(haw2)
ax1.set_xticklabels(x1,fontsize=14)
ax1.set_yticklabels(y,fontsize=14)
ax2.set_xticklabels(x2,fontsize=14)
plt.savefig('Q3_haw_2')
plt.show()
plt.close()

#calculate I_haw1, I_haw2
```

```
I_haw1 = np.zeros((1201,1201))
I_haw2 = np.zeros((1201,1201))
for i in range(1201):
    for j in range(1201):
        I_haw1[i][j] += - (np.cos(phi) * dw_haw1[i][j][0] + np.sin(phi) * dw_haw1[i][j][1])
        / np.sqrt(dw_haw1[i][j][0]**2 + dw_haw1[i][j][1]**2 + 1)
        I_haw2[i][j] += - (np.cos(phi) * dw_haw2[i][j][0] + np.sin(phi) * dw_haw2[i][j][1])
        / np.sqrt(dw_haw2[i][j][0]**2 + dw_haw2[i][j][1]**2 + 1)

# fix broken values
for i in range(1201):
    for j in range(1201):
        if I_haw1[i][j] > 0.2 or I_haw1[i][j] < -0.2:
            I_haw1[i][j] = 0
        else:
            continue

# fix broken values
for i in range(1201):
    for j in range(1201):
        if I_haw2[i][j] > 0.2 or I_haw2[i][j] < -0.2:
            I_haw2[i][j] = 0
        else:
            continue
#subplot hawaii
fig = plt.figure(figsize=[12,10])
ax1 = fig.add_axes([0,0.5,0.5,0.5])
ax2 = fig.add_axes([0.42,0.5,0.5,0.5],yticklabels=[])
ax1.imshow(I_haw1)
ax2.imshow(I_haw2)
ax1.set_xticklabels(x1,fontsize=14)
ax1.set_yticklabels(y,fontsize=14)
ax2.set_xticklabels(x2,fontsize=14)
ax1.set(ylabel= 'x (km)')
ax1.set(xlabel="
plt.savefig('Q3_haw')
```

References

[1] Newman, Mark, *Computational Physics.* 2013.