PHY407 Lab 3

Pierino Zindel (1002429703) and Hayden Johnson (1002103537)

September 28, 2018

Distribution of work: Question 1 was completed by Hayden, and question 2 was completed by Pierino. For question 3, code for extracting the altitude data from the file and for computing the gradient of the altitude was written by Pierino, while the remainder of the question was completed by Hayden.

1 More on the Error Function

1.1 Part a)

We seek to calculate the error function using the two methods from last time - trapezoid and Simpson's methods - and the new Gaussian quadrature method. Code for implementing the trapezoid and Simpson's methods was copied from our own submission for Lab02, as suggested. The code in gaussxw.py downloaded from Newman's website was used to compute the integral using Gaussian quadrature. Note that scipy.erf of course retains the same value, since its calculation is independent of the choice of N. We see that Gaussian integration is the best of the three methods, with Simpson's being second best and Trapezoid the worst, as expected.

Table 1: Values of erf(3) as output by the different methods of computing the integral.

N	scipy.erf	Trapezoid	Simpson's	Gaussian
8	0.999977909503	0.999968759667	0.999975957677	0.999977938559
100	0.999977909503	0.999977846867	0.99997790939	0.999977909503
1000	0.999977909503	0.999977908876	0.999977909503	0.999977909503

1.2 Part b)

Our goal is to calculate the error of the different methods and compare them graphically. The error of the different methods of integration is plotted as a function of N in figure 1. The actual error is calculated as the difference between the value of the integral and the value of scipy.erf(3), and the error estimate is calculated using equation (2) from the assignment sheet as suggested. As can be seen from the figure, the estimated and actual errors agree very well for all three methods, up until the error gets small enough ($\sim 10^{-15}$) that the round-off error starts to dominate. Also, we can see that the errors for all three methods decrease exponentially (until they hit round-off error), with Gaussian quadrature decreasing most quickly, followed by Simpson's method and with the trapezoid method performing the worst, which is the expected ordering for a general case function.

Note that the brown dotted line, representing the Gaussian error, goes down to zero at $N = 10^3$ because the error happened to be too small for python and it returned zero. This happened because in the region of 10^{-16} the roundoff error is completely dominant.

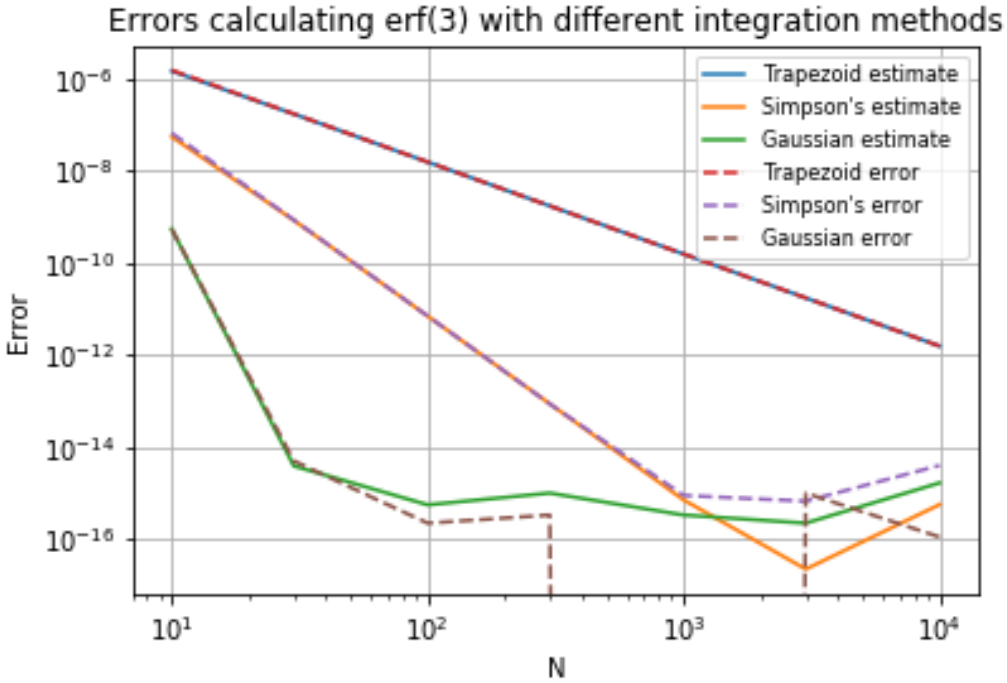

Figure 1: Graph showing calculated and estimated error for the integral of erf(3) using the three different integration methods, plotted as a function of the number of steps used.

1.3 Part c)

We now want to use our code to integrate the erf function using Gaussian quadrature in order to calculate the probability of blowing snow under different conditions. The error function is given by equation 1.

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{1}$$

Equation (3) from the assignment sheet reads:

$$P(u_{10}, T_a, t_h) = \frac{1}{\sqrt{2\pi}\delta} \int_0^{u_{10}} e^{\frac{-(\bar{u}-u)^2}{2\delta^2}} du$$

So if we substitute:

$$t = \frac{\bar{u} - u}{\sqrt{2}\delta}$$
$$\frac{dt}{du} = -\frac{1}{\sqrt{2}\delta}$$
$$\implies du = -\sqrt{2}\delta dt$$

Then we see that we can recast equation (3) to be:

$$P(u_{10}, T_a, t_h) = \frac{1}{\sqrt{\pi}} \int_{t_0}^{t_1} e^{-t^2} dt$$

Where:

$$t_0 = \frac{\bar{u} - u_{10}}{\sqrt{2}\delta}$$
$$t_1 = \frac{\bar{u}}{\sqrt{2}\delta}$$

We can now use our error function computing algorithm to compute this integral.

Using this methodology, the probability of plowing snow as a function of T_a with the suggest values for the additional parameters was plotted as shown in figure 2 (code in Lab03_Q1_c.py). As can be seen from the plot, the probability of blowing snow increases as the windspeed (u_{10}) increases, and decreases as the age of the snow (t_h) increases. These dependencies make sense: higher wind speed, and corresponding greater kinetic energy in the air, should correlate to a greater chance for the wind to pick up snow off of the surface and carry it; on the other hand, older snow is more likely to be packed down or solidified into ice - and thus more resistant to being picked up by the wind - than is freshly fallen powdered snow. The temperature at which the maximal likeliness of blowing snow occurs becomes lower as wind speed increases, as can be discerned from the fact that the peak of the function moves left as one looks from blue through green to red curves.

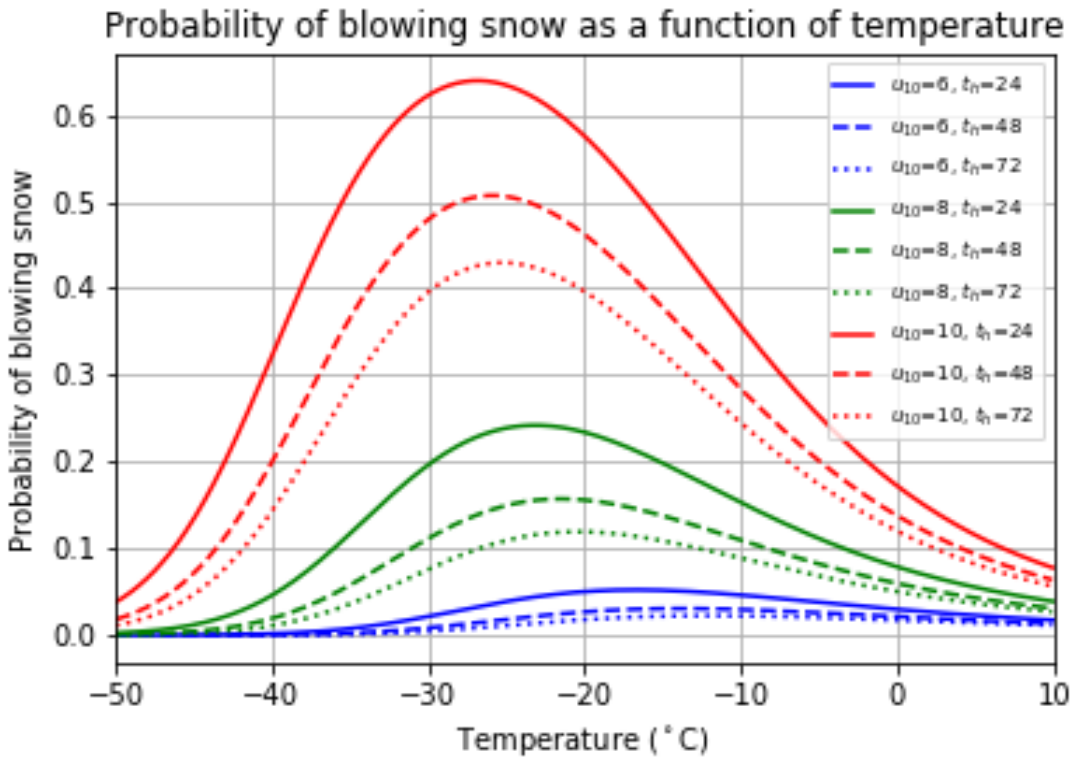

Figure 2: Graph showing the probability of blowing snow as a function of the temperature T_a for varying values of u_{10} (in m/s) and t_h (in hours).

2 Calculating Quantum Mechanical Observables

Modelling physics problems in python allows us to quickly solve and visualize problems that might otherwise require lots of manual computation. For this questions we seek to write a program that gives us wavefunction of the harmonic oscillator for any given order n and can then produce the energy, expectation values and their uncertainties for that wavefunction.

2.1 Part a)

First we wish to be able to numerically find and graph wavefunctions for the harmonic oscillator. Using the formulas given in the lab manual (equations 5 and 6) that relate the wavefunctions in terms of the Hermite polynomials, the wavefunctions were computed by composing two functions, one for the coefficient term $\frac{e^{-x^2/2}}{\sqrt{2^n n! \sqrt{\pi}}}$ and one for the Hermite polynomials $H_n(x)$. The Hermite polynomials were computed by defining the values for $n = 0$ and $n = 1$, to be $H_0(x) = 1$ and $H_1(x) = 2x$ respectively, and then recursively calling the function for all $n > 1$. The wavefunctions for $n = 0, 1, 2, 3$ and $-4 < x < 4$ were plotted and are depicted in figure 3.

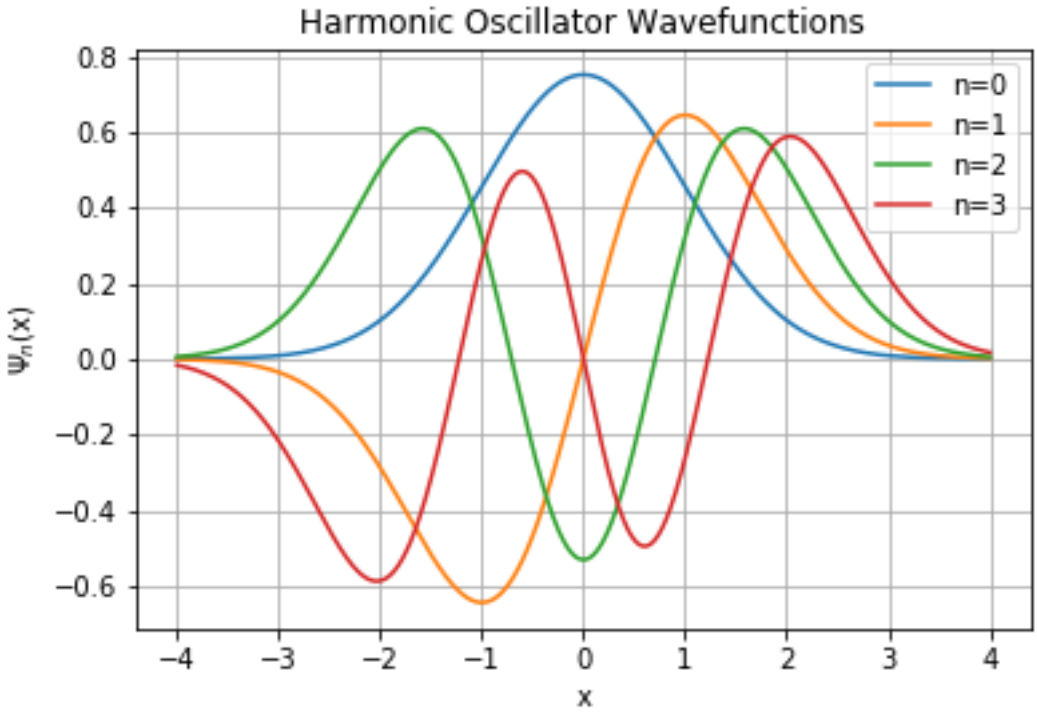

Figure 3: Wavefunctions of the harmonic oscillator for n=0,1,2,3

2.2 Part b)

We then tested our program for the case where $n = 30$ and $-10 < x10$ to ensure proper function in a timely manner for higher orders of n. The resulting wavefunction graph is shown in figure 4.

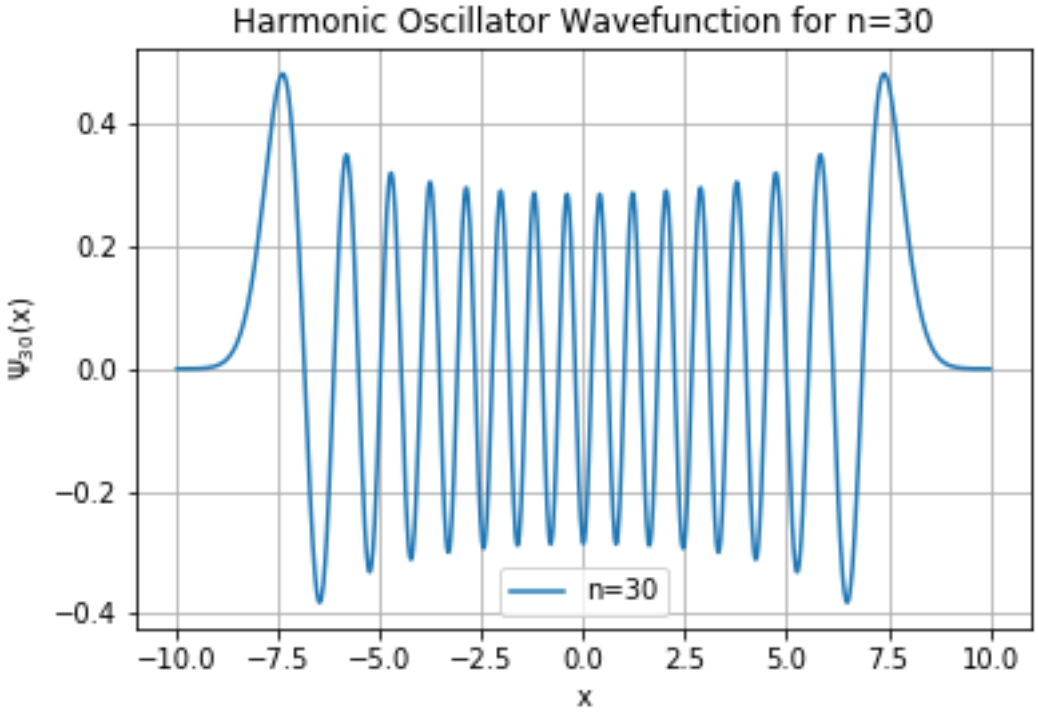

Figure 4: Wavefunction of the harmonic oscillator for n=30

2.3 Part c)

For the last part of the problem we seek to write a program that uses the wavefunctions of part a) to compute the expectation values of the position-squared and momentum-squared, and to report the uncertainty in each along with the energy of the wavefunction. The position and momentum expectation values were computed using the equation 10 and 11 from the lab manual with a substitution of $x = \frac{z}{1-z^2}$ and $dx = \frac{1+z^2}{(1-z^2)^2}dz$ which allowed for the alteration of the integral endpoints from $\pm infinity$ to ± 1 which can be easily evaluated numerically. The integral was computed using Gaussian quadrature with 100 points. The Gaussian weight function used in our program was part of an imported module written by Mark Newman and downloaded from: http://www-personal.umich.edu/ mejn/cp/programs/gaussxw.py. The wavefunction and their desired values were computed for $n = 0, .., 15$. The uncertainties for the momentum and position along with the energy for each are given in table 2. From the table we see that the uncertainties are identical up to the sixth decimal place for $n = 0$, and begin to diverge as n increases. Moreover, they obey the uncertainty relation as defined for the harmonic oscillator to be $\Delta x \Delta p = (2n+1)$, where the natural length is $\hbar$. The energy also obeys the know relation for the harmonic oscillator which is $E = n + \frac{1}{2}$, with some error as n increases give the numerical bases of the calculation. This discrepancy can be easily be accounted for by increasing the number of points of integration for the wavefunctions as n increases. A quick calculation with twice as many integration points shows this, as well as showing that the expectation values and uncertainties remain equal for position and momentum for all n.

Table 2: Uncertainties in position and momentum, and energies for wavefunctions of the harmonic oscillator for n=0,..,15

n	$\sqrt{\langle x^2 \rangle}$	$\sqrt{\langle p^2 \rangle}$	Energy
0	0.707107	0.707107	0.500000
1	1.224745	1.224745	1.500000
2	1.581139	1.581139	2.500000
3	1.870829	1.870829	3.500000
4	2.121320	2.121320	4.500000
5	2.345208	2.345208	5.500000
6	2.549510	2.549510	6.500001
7	2.738614	2.738613	7.500003
8	2.915474	2.915472	8.499981
9	3.082182	3.082195	9.499888
10	3.240355	3.240425	10.500125
11	3.391493	3.391399	11.501906
12	3.536339	3.535189	12.501629
13	3.672389	3.671681	13.483845
14	3.798449	3.808251	14.465494
15	3.936250	3.954023	15.564184

3 Generating a Relief Map

3.1 Part a)

We seek to write pseudo-code for a program to download the altitude data from the file, calculate the gradient of the altitude data, use this gradient to calculate the illumination at a certain incident angle of light, and then plot the altitudes and illumination.

3.1.1 Pseudo-code for reading the file and storing the values to an array:

1. Define the size of the array of values, N
2. Create empty array to store height values
3. Open file
4. Iterate over the rows of the array

4.1 Create empty array to store values for the row

4.2 Iterate over the elements of the row

4.21 Read each value out of the file and store it to the row array

4.3 Write the completed row into the 2d array storing the values

5. Close the file

3.1.2 Pseudo-code for calculating the gradient of the heights w:

1. Define value of h which specifies distance between grid points
2. create empty array to store gradient values
3. iterate over rows of the coordinate grid

3.1 define an empty array to store gradient values for the row

3.2 iterate over the columns of the grid

3.21 if in first column, compute $\frac{\partial w}{\partial x}$ with forward derivative

3.22 if in last column, compute $\frac{\partial w}{\partial x}$ with backward derivative

3.23 otherwise, compute $\frac{\partial w}{\partial x}$ using central difference

3.24 if in first row, compute $\frac{\partial w}{\partial y}$ with forward derivative

3.25 if in last row, compute $\frac{\partial w}{\partial y}$ with backward derivative

3.26 otherwise, compute $\frac{\partial w}{\partial y}$ using central difference

3.27 store computed partial derivatives to row array

3.3 store computed row to gradient array

3.1.3 Pseudo-code for plotting w and I:

1. Define values of a_x, a_y, a_z
2. Extract values of dw/dx and dy/dx from gradient array
3. Compute intensity for all points on the grid, as a function of the gradient, using equation (15) from the assignment sheet
4. Clip any negative values from the array (replace them with zero)
5. Specify bounds of plot and coordinate grid
6. Plot height array using pcolormesh
7. Plot illumination using pcolormesh

3.2 Part b)

We now write the program described by the pseudo-code from part a, and present the plots created by it.

The altitude above sea level for the square of latitude and longitude containing Hawaii's Big Island are plotted in figure 5. The island's shape and vertical features are quite recognizable from the plot, with Mauna Kea and Mauna Loa being the two large peaks at the Northern end and center of the island, respectively, and Haulalai being the smaller peak on the west.

The expected illumination of the Big Island of Hawaii at sunset is plotted in figure 6. This plot was created by first calculating the gradient of the altitude data that is plotted in figure 5, and then using that information in conjunction with equation (15) from the assignment sheet, plotted with $\phi = \pi$ as suggested, in order to simulate illumination by a light source directly to the west. Note that the plot agrees quite well with what ought to be expected just by looking at figure 5. One shortcoming to note is that the methodology used here simply calculates intensity of illumination as a function of the gradient, and does not account for the fact that the incident light might be blocked completely by other ground before reaching a given point. Because the island consists of only three peaks, none of which blocks light

incident from the West from reaching the others, this defect is not particularly noticeable, although there are a few points in the shadow of the peaks where the gradient is briefly positive which are erroneously illuminated.

In figure 7, we plot the illumination of the Big Island at sunrise, as opposed to sunset, by taking the incident light to be coming directly from the east.

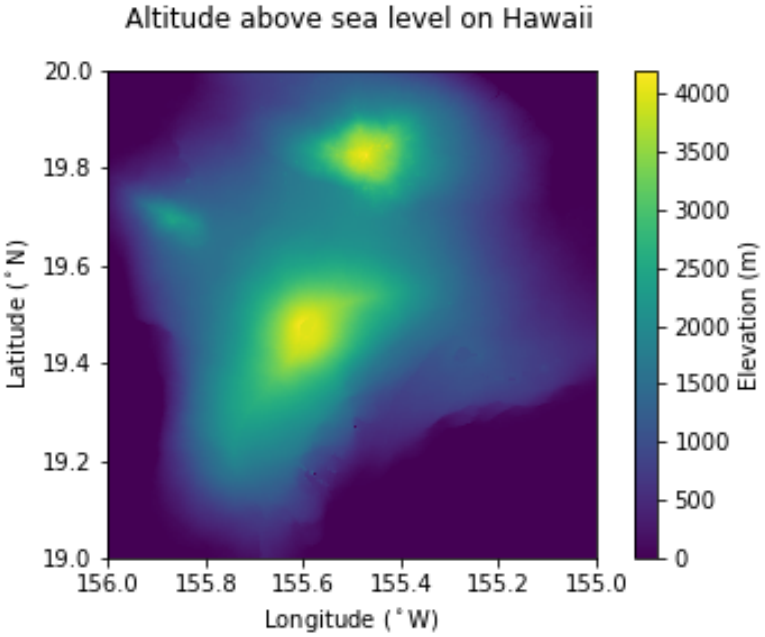

Figure 5: Plot of altitude above sea level, in meters, on Hawaii's Big Island generated using data from the USGS website.

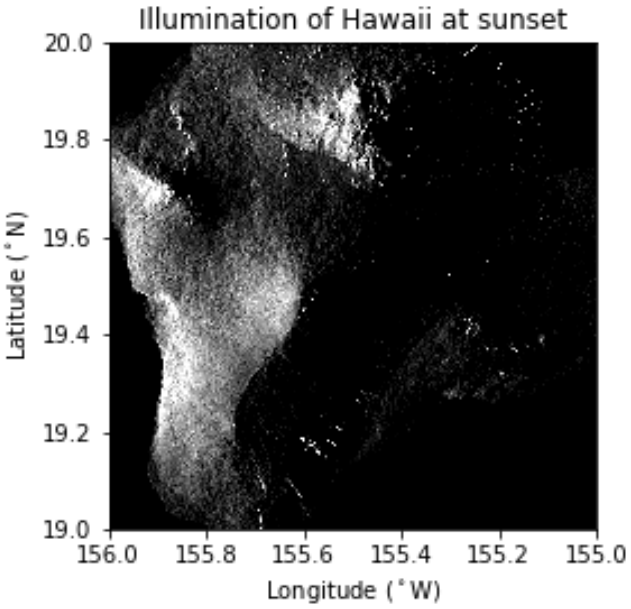

Figure 6: Plot of illumination of Hawaii's Big Island at sunset, as seen from directly above, made using the gradient of the altitude data plotted in figure 5.

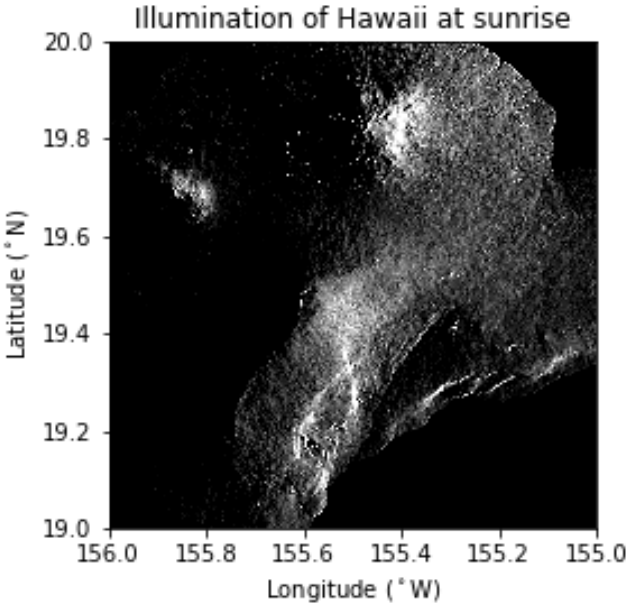

Figure 7: Plot of illumination of Hawaii's Big Island at sunrise, as seen from directly above, made using the gradient of the altitude data plotted in figure 5.