Lab2 Report

Zirui Wan (Question 2), Rundong Zhou (Question 1 and 3)

October 2, 2020

1 Question 1

1.1 Q1a

1.1.1 i

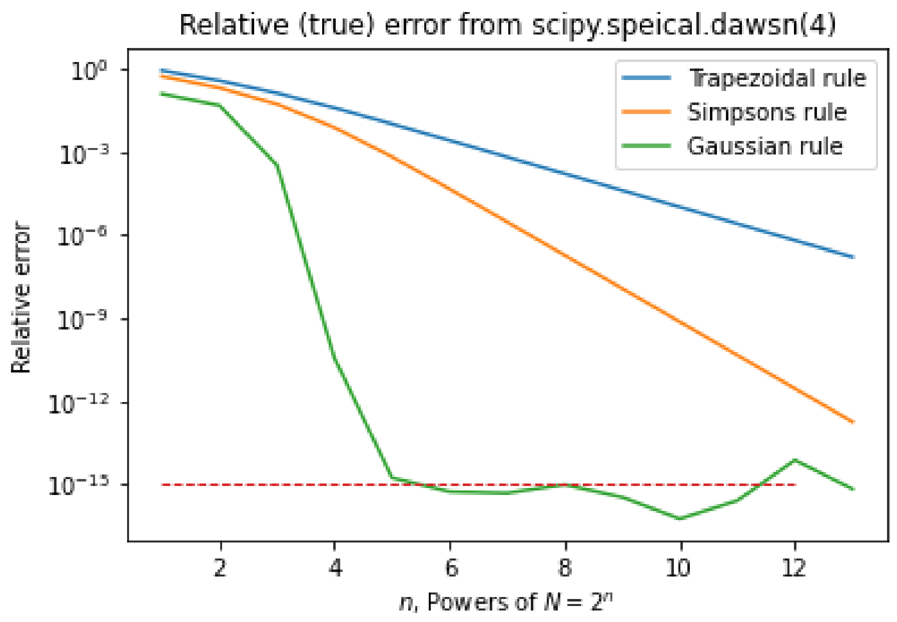
So I calculate values for $D(4)$ with four different methods with $N = 8$ up to $N = 2048$, I took the steps to be exponential with $N = 2^n$. the following lines are the printed output:

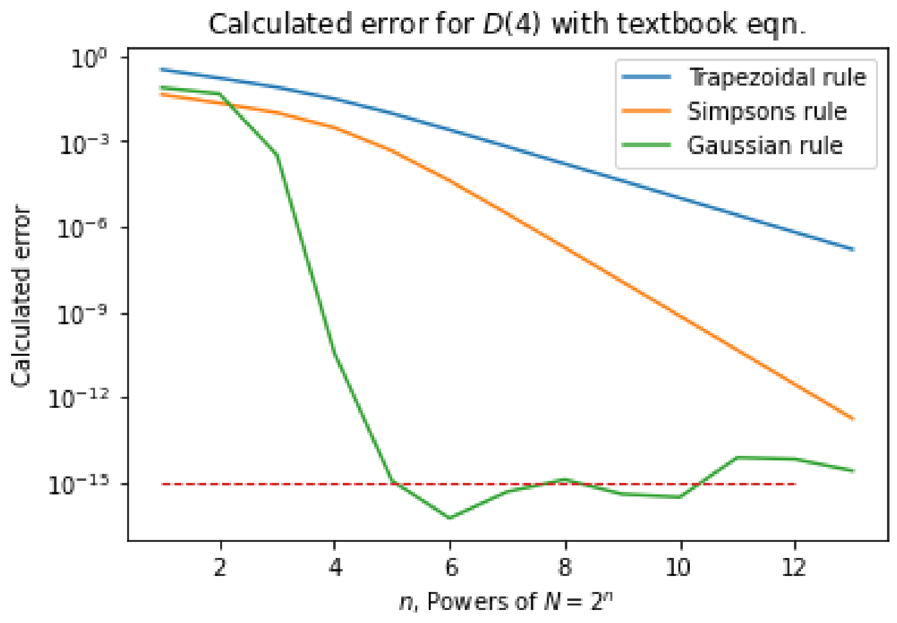
```
N= 8
Trap: 0.26224782053479523
Simp: 0.1826909645971217
Gauss: 0.12901067881706982
Scipy(True value): 0.1293480012360051
---------------
N= 16
Trap: 0.16828681895583716
Simp: 0.13696648509618445
Gauss: 0.12934800119977768
Scipy(True value): 0.1293480012360051
---------------
N= 32
Trap: 0.1395800909267732
Simp: 0.13001118158375183
Gauss: 0.1293480012360034
Scipy(True value): 0.1293480012360051
---------------
N= 64
Trap: 0.13194038496790617
Simp: 0.1293938163149505
Gauss: 0.12934800123600457
Scipy(True value): 0.1293480012360051
---------------
N= 128
Trap: 0.1299983024925397
```

```
Simp: 0.12935094166741756
Gauss: 0.12934800123600462
Scipy(True value): 0.1293480012360051
---------------
N= 256
Trap: 0.12951071531441982
Simp: 0.12934818625504652
Gauss: 0.12934800123600415
Scipy(True value): 0.1293480012360051
---------------
N= 512
Trap: 0.12938868844305068
Simp: 0.12934801281926095
Gauss: 0.12934800123600543
Scipy(True value): 0.1293480012360051
---------------
N= 1024
Trap: 0.12935817358096138
Simp: 0.12934800196026489
Gauss: 0.12934800123600504
Scipy(True value): 0.1293480012360051
---------------
N= 2048
Trap: 0.12935054435619742
Simp: 0.12934800128127624
Gauss: 0.12934800123600534
Scipy(True value): 0.1293480012360051
---------------
```

At $N = 8$, the Trapezoidal and Simpson's rules' result confirms with the result from Lab2. The Gaussian method gives a much accurate result, it has 3-digit accuracy even at N=8.
At $N = 16$, the Gaussian method's accurate digit doubles. This reflects the property of Gaussian method: it produces the accuracy of polynomials of degree $2N - 1$ with N points. However, since e^{t^2} has infinite degree, even the Gaussian method cannot provide the accurate answer.
At $N = 32$, the Gaussian method has 14 accurate digits, which means it reaches the accuracy limit $\mathcal{O}(10^{-15})$ of python.

1.1.2 ii





So I plot the relative (true) error and calculated result for all three methods on a log-log scale. The y-axis is the error in log_{10} scale. The x-axis is the value of n, which is the Power of $N = 2^n$. So the scale of x is respect to log_2.
All three methods' error shows a similar pattern in both graphs. The Trape-

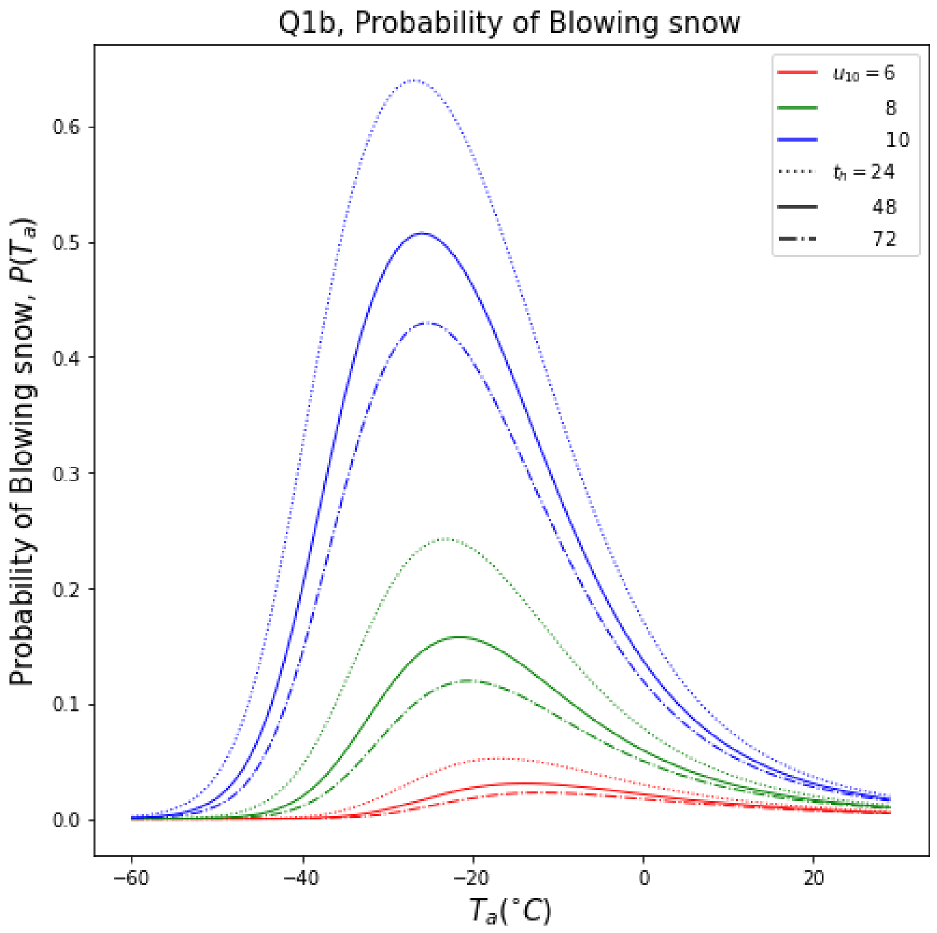
zoidal and Simpson's plot also confirms my Lab2 report, Simpson's method decreases faster than Trapezoidal. The error for Gaussian method decreases much faster than both Trapezoidal and Simpson's rule. It hits the python accuracy limit of $\mathcal{O}(10^{-15})$ at $n = 5$.
There is only one minor difference between the two plots. At $n = 1, 2$, the calculated error of Gaussian method is larger than Simpson's method's error, but in the true error plot, it is smaller. This can be easily confirmed from the equation:

$$\epsilon_N - \epsilon_{2N} \simeq \epsilon_N = I_{2N} - I_N$$

The estimation is not very accurate when ϵ_{2N} is still big.

1.2 Q1b

So I plot the probability of "Blowing snow" under nine different conditions against the average hourly temperature T_a:



From the graph, we can conclude that:

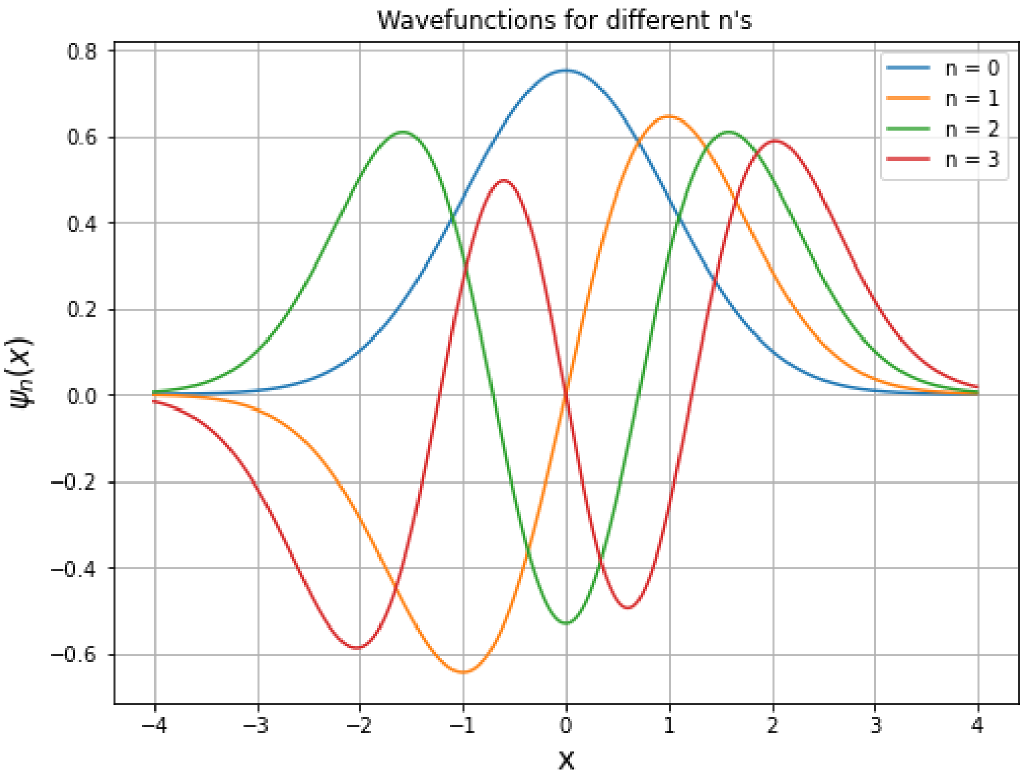
- u_{10}, the average hourly wind speed at a height of 10m, has the greatest impact on the probability curve. The greater the wind speed, the higher the probability.
- t_h, the snow surface, has a smaller impact on the probability curve. The smaller the snow surface, the higher the probability.
- Blowing snow is most likely to happen between the temperature of -40°C to -10 °C. The peaking temperature tends to shift right when the curve turns flatter. The higher wind speed, the lower peaking temperature.

These correlations make quite sense. With greater wind speed, the snow will be blown more fiercely, thus, higher probability for "blowing snow". Also, smaller snow surface makes it easier for the snow to be blown, thus, higher probability.

2 Question 2

2.1 Part (a)

2.1.1 Plot

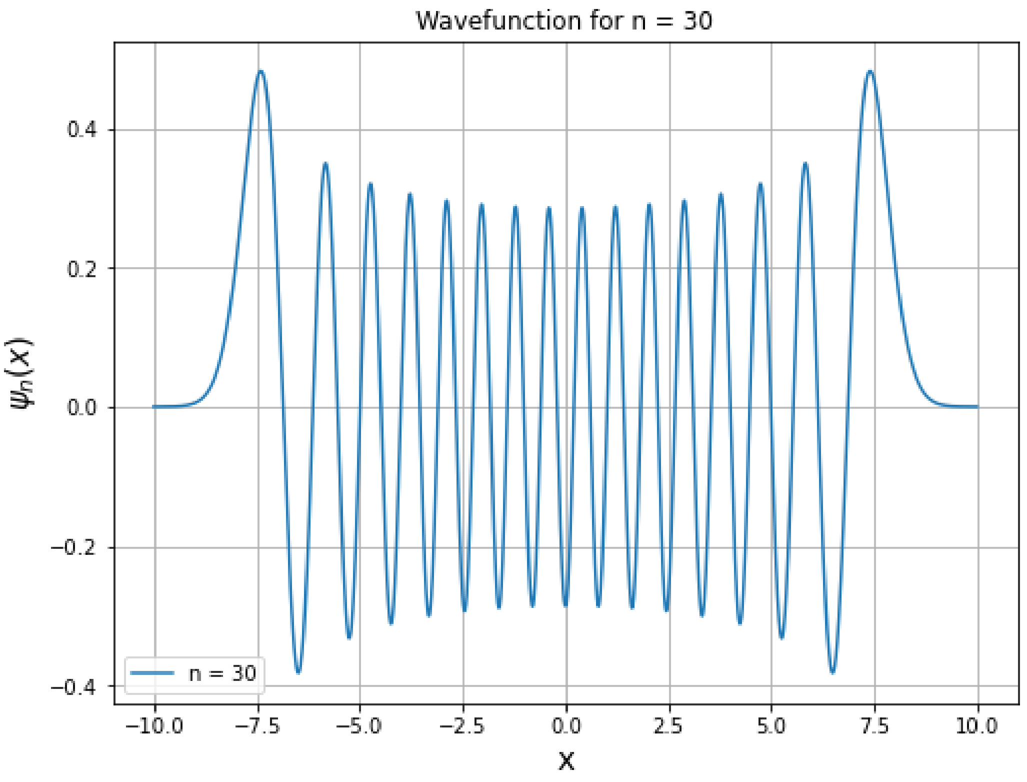


2.1.2 Explanatory Notes

The wavefunctions ψ_n for n = 0, 1, 2, 3 are plotted above. You can see that the shape satisfies the requirement for quantum wavefunctions. Firstly, they vanish when position goes to $-\infty$ and ∞. Secondly, the number of "nodes" (bounded by intersections with the base line where $\psi_n = 0$) is equal to the number of energy level n. Lastly, the amplitudes of wavefunctions decrease as n increases, because the energy is conserved, which is related to the integral of areas bounded by these wavefunctions and the base line.

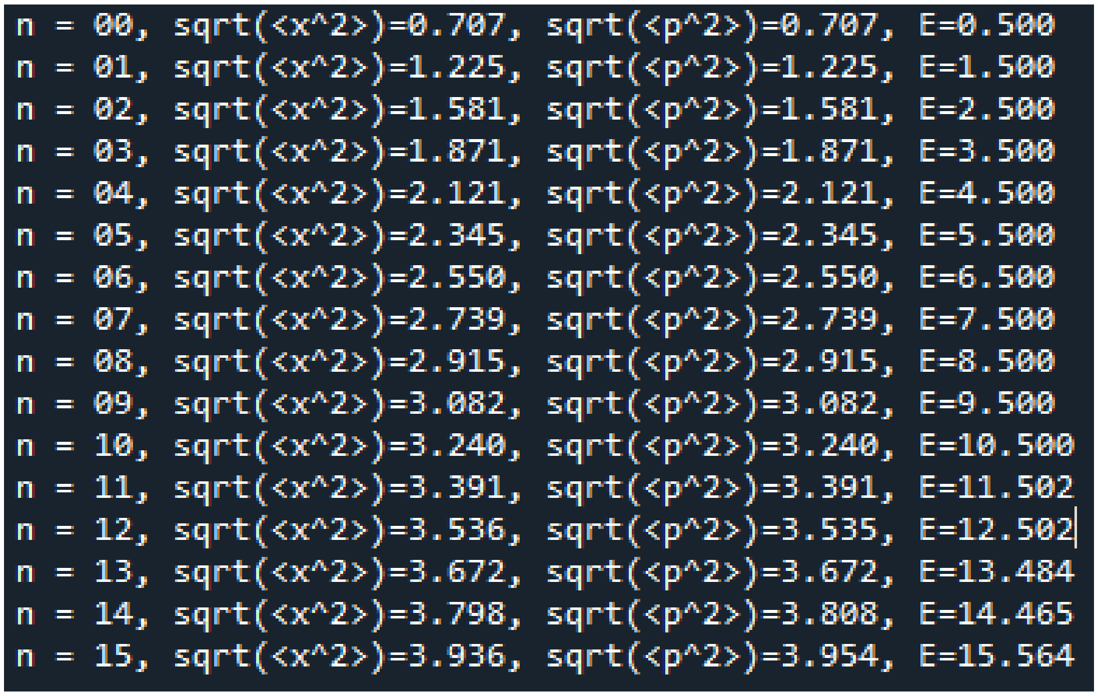
2.2 Part (b)

2.2.1 Plot



2.3 Part (c)

2.3.1 Printed Output



```
n = 00, sqrt(<x^2>)=0.707, sqrt(<p^2>)=0.707, E=0.500
n = 01, sqrt(<x^2>)=1.225, sqrt(<p^2>)=1.225, E=1.500
n = 02, sqrt(<x^2>)=1.581, sqrt(<p^2>)=1.581, E=2.500
n = 03, sqrt(<x^2>)=1.871, sqrt(<p^2>)=1.871, E=3.500
n = 04, sqrt(<x^2>)=2.121, sqrt(<p^2>)=2.121, E=4.500
n = 05, sqrt(<x^2>)=2.345, sqrt(<p^2>)=2.345, E=5.500
n = 06, sqrt(<x^2>)=2.550, sqrt(<p^2>)=2.550, E=6.500
n = 07, sqrt(<x^2>)=2.739, sqrt(<p^2>)=2.739, E=7.500
n = 08, sqrt(<x^2>)=2.915, sqrt(<p^2>)=2.915, E=8.500
n = 09, sqrt(<x^2>)=3.082, sqrt(<p^2>)=3.082, E=9.500
n = 10, sqrt(<x^2>)=3.240, sqrt(<p^2>)=3.240, E=10.500
n = 11, sqrt(<x^2>)=3.391, sqrt(<p^2>)=3.391, E=11.502
n = 12, sqrt(<x^2>)=3.536, sqrt(<p^2>)=3.535, E=12.502
n = 13, sqrt(<x^2>)=3.672, sqrt(<p^2>)=3.672, E=13.484
n = 14, sqrt(<x^2>)=3.798, sqrt(<p^2>)=3.808, E=14.465
n = 15, sqrt(<x^2>)=3.936, sqrt(<p^2>)=3.954, E=15.564
```

2.3.2 Explanatory Notes

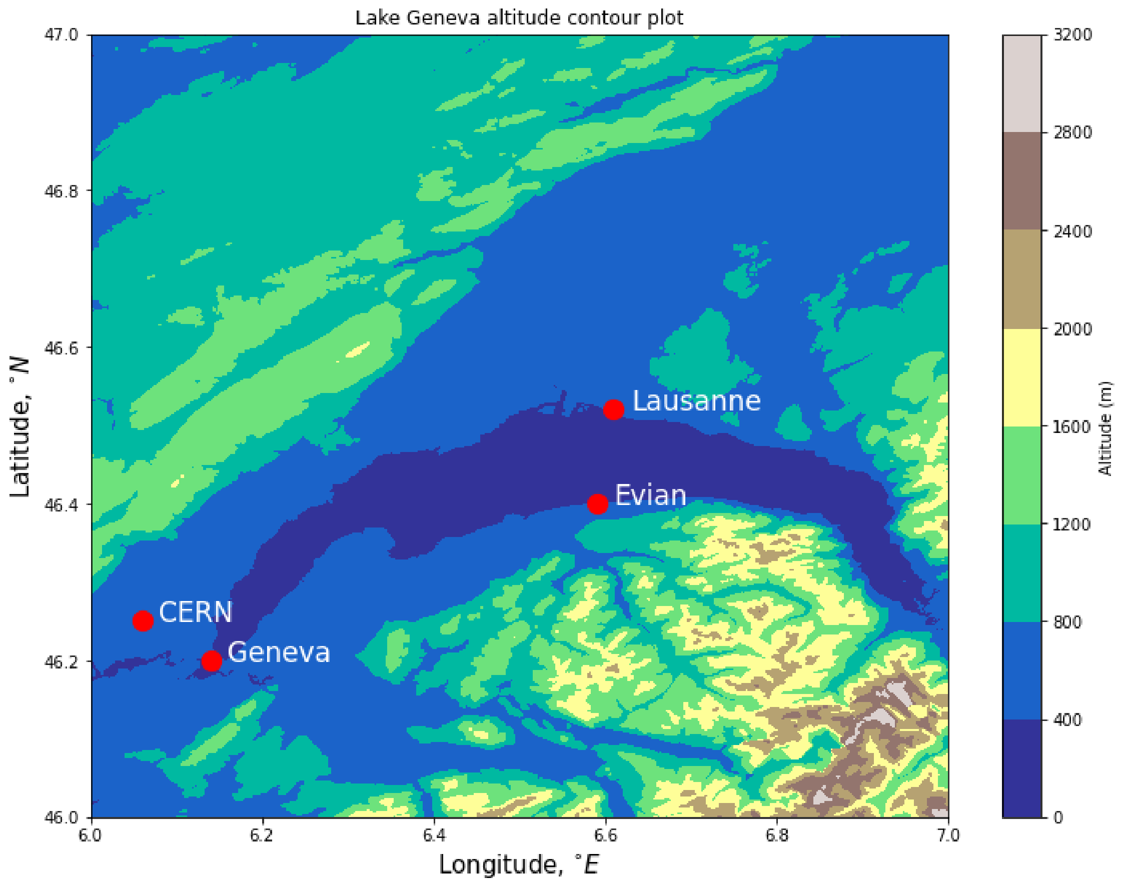
The printed outputs are attached above. You can see that indeed $\sqrt{< x^2 >} = 2.35$ when $n = 5$. Also the uncertainties of position and momentum, $\sqrt{< x^2 >}$ and $\sqrt{< p^2 >}$, are the same. Energy is quantumized, which means that when n increases by 1 energy is also increased by 1. It's remarkable that, when n is increased to relatively high levels and the wavefunction becomes highly nonlinear (as shown in the case for $n = 30$), the numerical errors also increase significantly such that $\sqrt{< x^2 >}$ and $\sqrt{< p^2 >}$ start to have slightly different values.

3 Question 3

3.1 Q3a Pseudo code

```
def read(file_name)
######first function for reading the file##########
    Use built in function to open the target file
    Initiate a 1201 by 1021 matrix for recording altitude, named w
    for loop, up to 1200 for y axis:
        for loop, up to 1200 for x axis
            buff variable to read 2 bytes from the file
            if the value is invalid: #ie, -2^15 in our file
                Assign an average value calculated from its left point and
                upper point to the matrix
            else, the value is valid:
                Assign the value from the file to the matrix
    return the matrix w which contents the altitude value.

def gradient(w, h)  #second function for calculate the gradient,
                    #takes the altitude matrix,
                    #and value for h, apply different method for edges
    Initiate a 1201 by 1201 matrix for recording gradient, named grad
    Convert elements in grad into list of length 2, 1 for x gradient,
    1 for y gradient
    for loop, i up to 1200 for y axis:
        if at top edge, i == 0:
            for loop, j up to 1200 for x xis:
                if j == 0, at left edge:
                    perform forward derivative for x, assign to grad matrix
                    perform forward derivative for y, assign to grad matrix
                elif j == 1200, at right edge:
                    perform backward derivative for x ..
                    perform forward derivative for y..
                else, x not at edge:
                    perform central derivative for x..
                    perform forward derivative for y..
        if at bottom edge, i == 1200:
                if j == 0, at left edge:
                    perform forward derivative for x..
                    perform backward derivative for y..
                elif j == 1200, at right edge:
                    perform backward derivative for x ..
                    perform backward derivative for y..
                else, x not at edge:
                    perform central derivative for x..
                    perform backward derivative for y..
        else, y not at edge:
                if j == 0, at left edge:
                    perform forward derivative for x..
                    perform central derivative for y..
                elif j == 1200, at right edge:
                    perform backward derivative for x ..
                    perform central derivative for y..
                else, x not at edge:
                    perform central derivative for x..
                    perform central derivative for y..
    return grad matrix

def relief_map(gradient, phi) #function to calculate the relief map,
                            #takes the gradient matrix and the angle phi
    initiate a 1201 by 1201 matrix, named relief
    for loop, up to 1200 for y axis:
        for loop, up to 1200 for x axis:
            calculate the norm by sqrt(grad_x^2 + grad_y^2 +1)
            calculate the relief value:
            - ( - cos(phi)*grad_x + sin(phi)*grad_y) / n
            ########PLEASE NOTE the MINUS sign in front of grad_x######
            ########we calculate the x gradient from west to east######
            ########need to flip the sign when calculate relief########
            ########or E and W axis will be flipped###################
    return relief matrix

w = read(file_name)     #read the file and produce the elevation matrix
plot matrix w, altitude map

grad = gradient(w, h)      #calculate gradient from elevation
map = relief(grad, phi)     #calculate the relief map with gradient and phi angle
plot matrix relief, relief map
```

3.2 Q3b

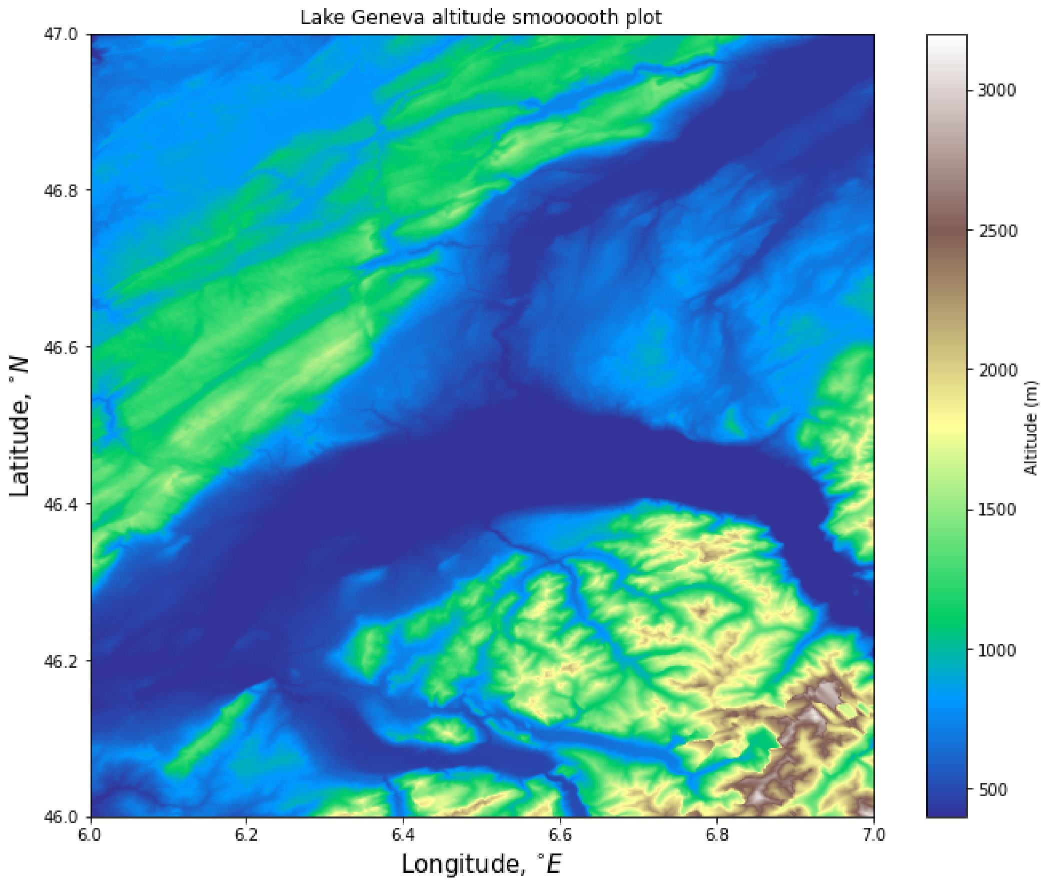


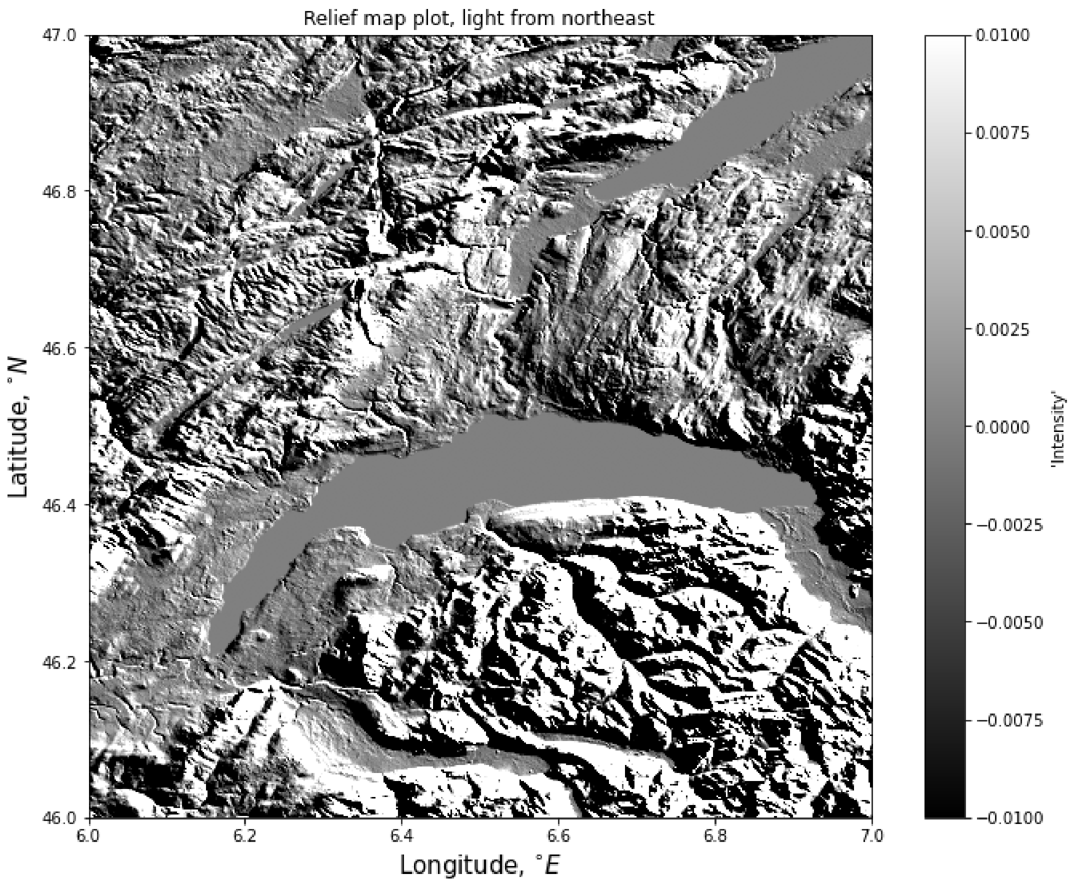
So I provide three plots:

- A discrete contour map for Lake Geneva, using `matplotlib.pyplot.contourf()`
- A smoother map for Lake Geneva, using `matplotlib.pylot.imshow()`
- A relief map for Lake Geneva, using `matplotlib.pylot.imshow()`. I set `vmax = 0.01` and `vmin = -0.01` for best sensitivity, since the gradient at the plain and hills is way smaller than the gradient at the Alps.

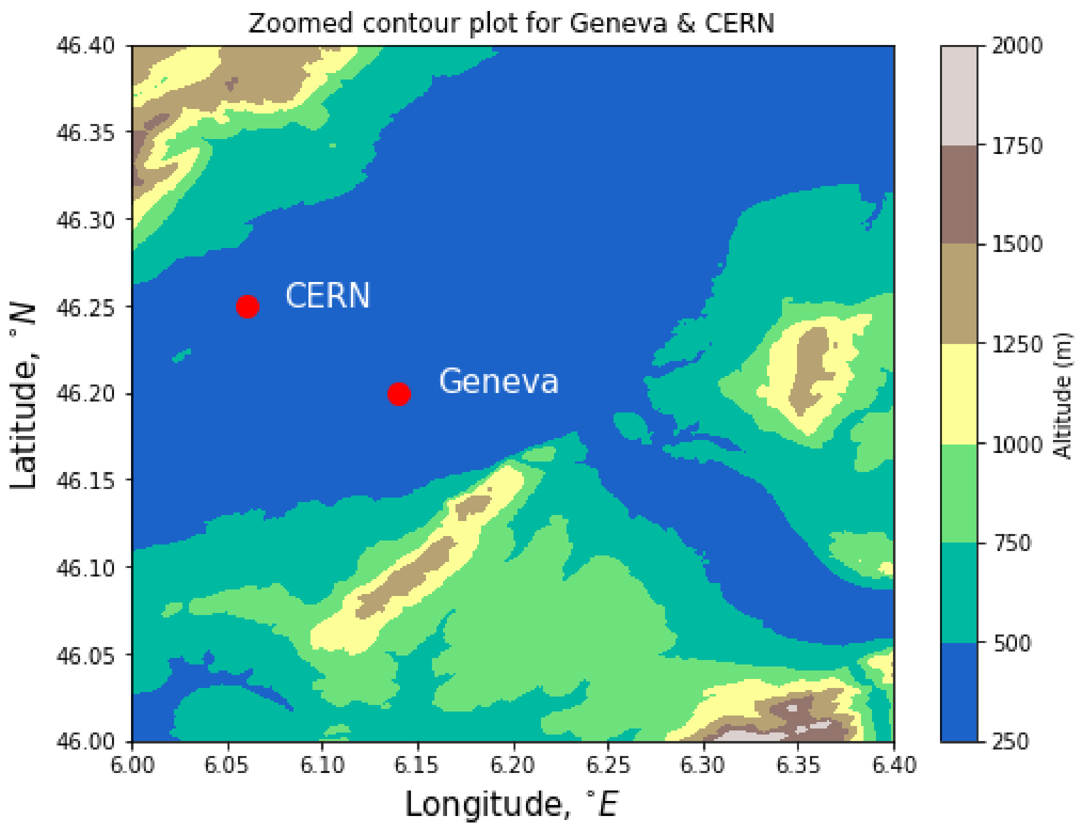
The main difference between function `contourf()` and `imshow()` is, `contourf()` plots the graph from bottom to top, like `pcolormesh()`, while `imshow()` plots the graph from top to bottom.
In our case, `imshow()` gives a natural axis orientation (top to bottom), and I need to flip the y-axis manually for `contourf()`.
The relief map provides a more straight forward three-dimensional view of the terrain, since the depth of color is determined by the slop (gradient). In the

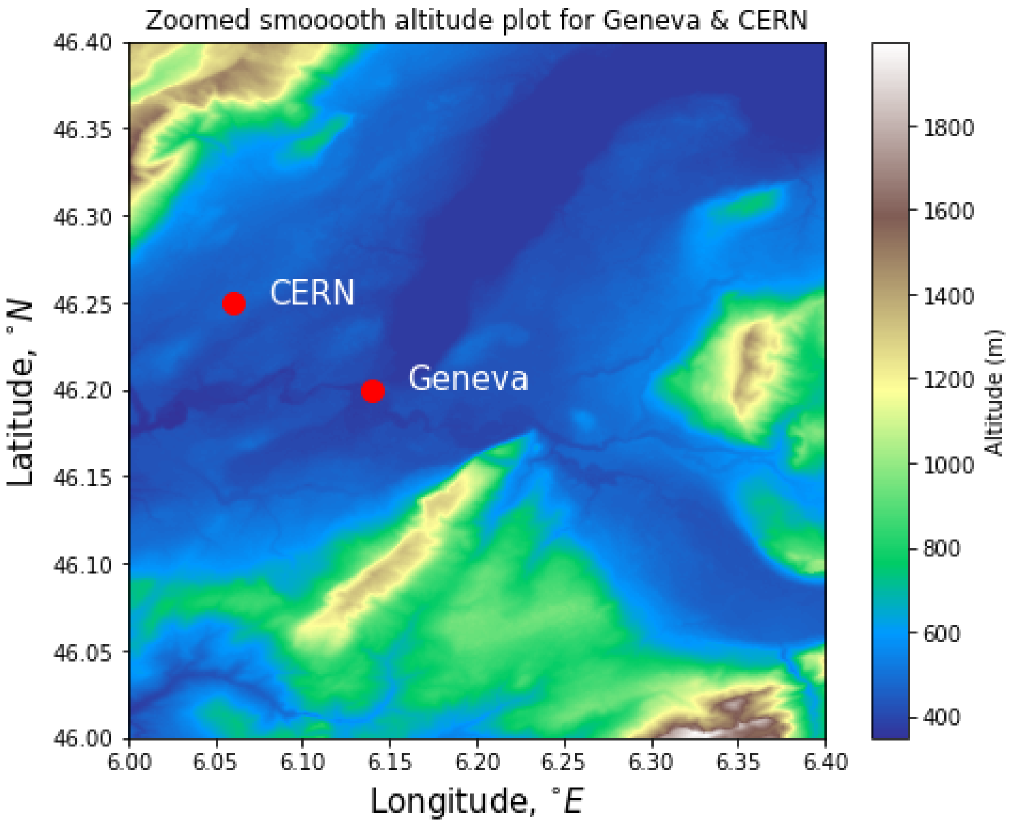
elevation map, we can still see some terrain pattern from it, but it looks flat comparing to the relief map.
In the relief map, we can see clearly the exact outline of Lake Geneva, but not from the elevation map (maybe a little bit from the contour map, however the outline is still not exact).

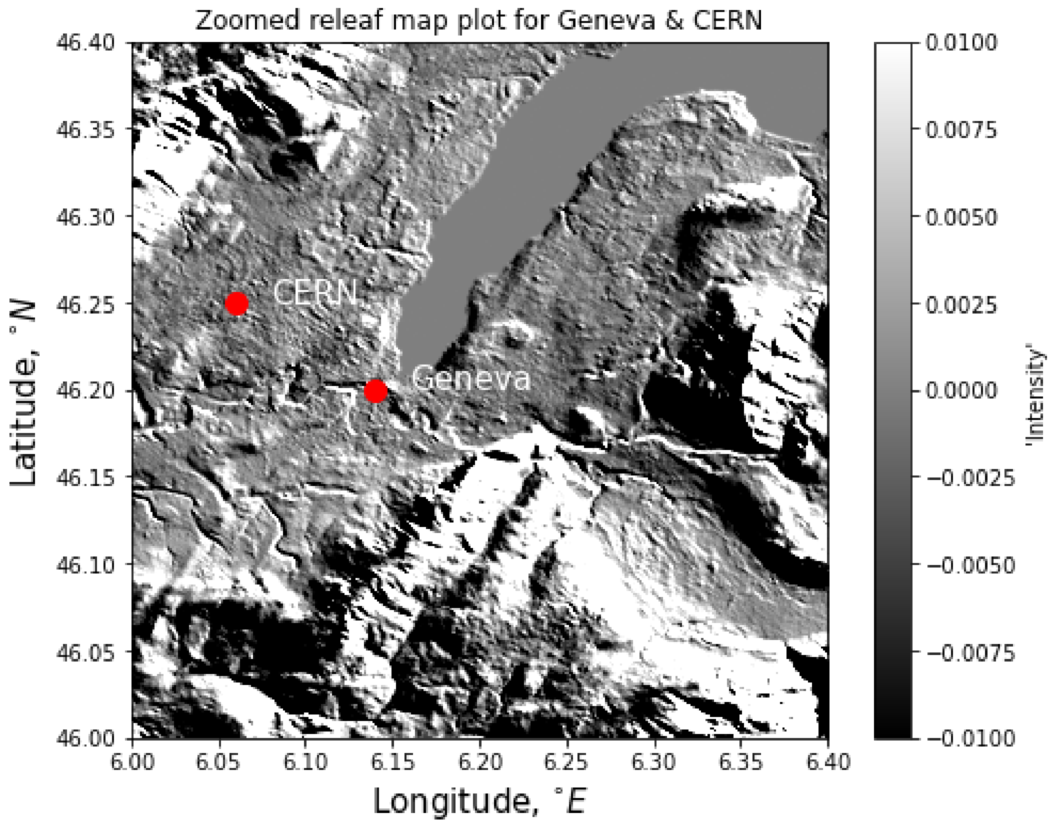


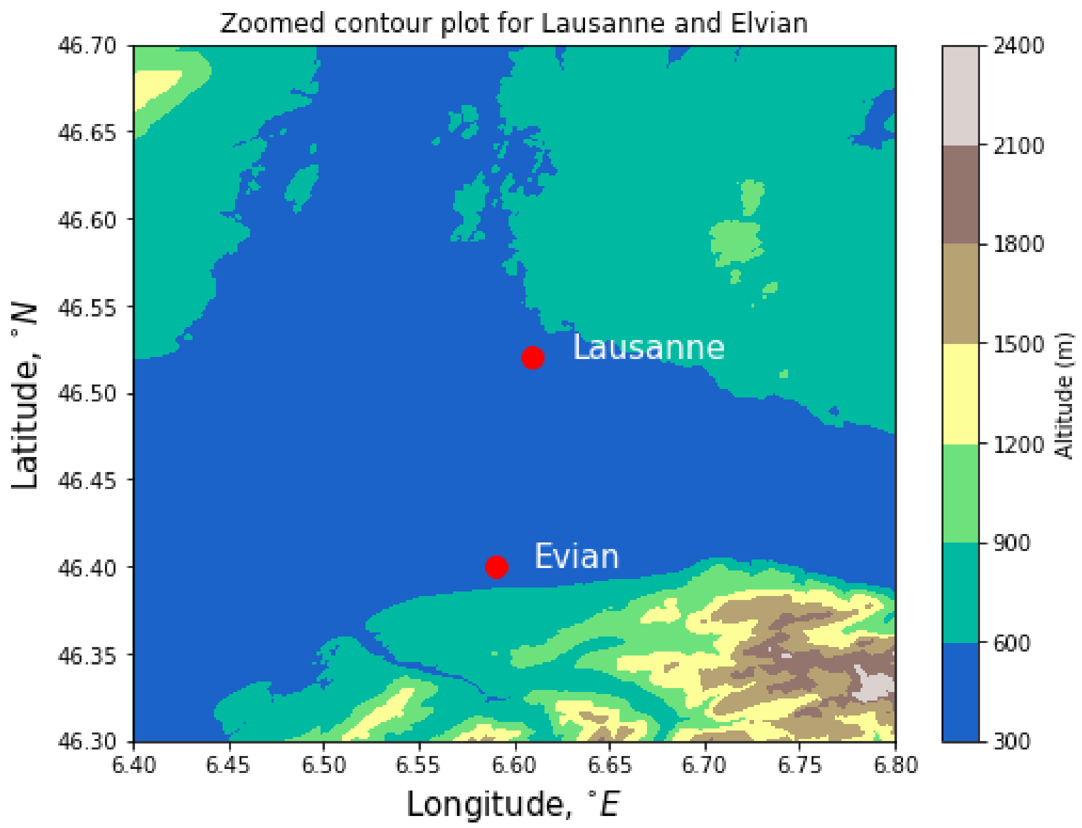


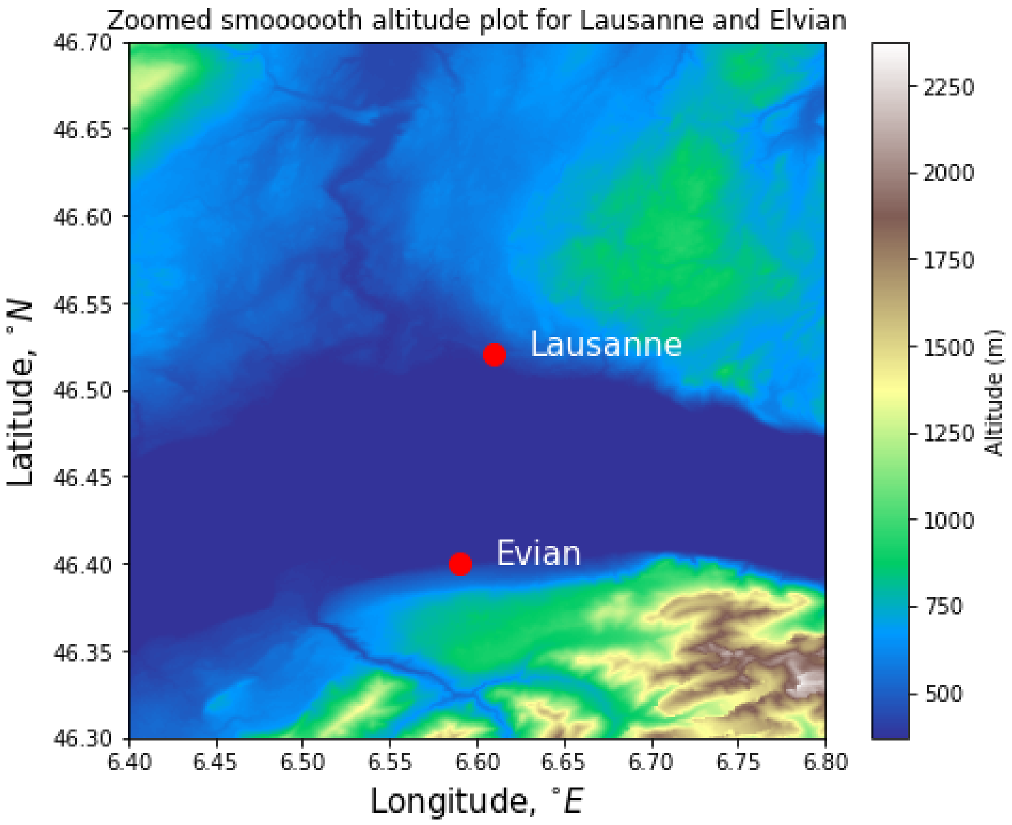
The following are the zoomed maps for Geneva, CERN, Lausanne and Elvian. Still, one contour map, one smooth elevation map, and one relief map for each.

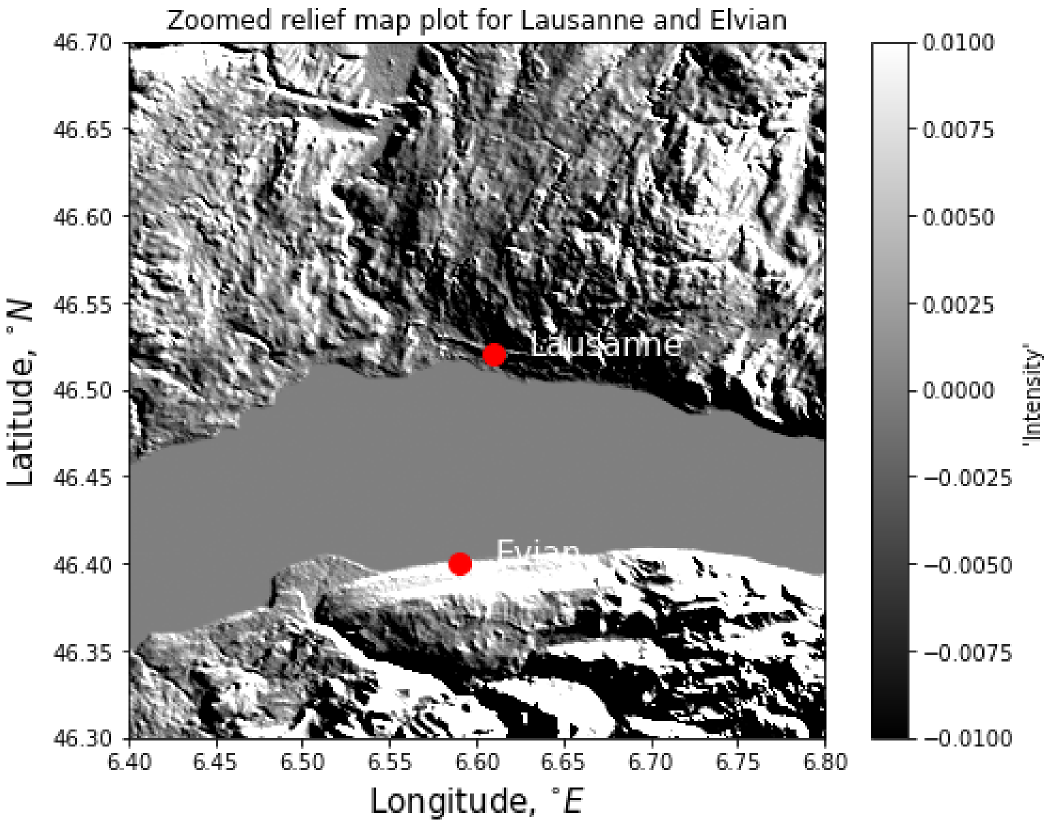

Zoomed contour plot for Geneva & CERN
CERN
Geneva
Latitude, °N
Longitude, °E
Altitude (m)
46.40
46.35
46.30
46.25
46.20
46.15
46.10
46.05
46.00
6.00
6.05
6.10
6.15
6.20
6.25
6.30
6.35
6.40
2000
1750
1500
1250
1000
750
500
250


Zoomed smooooth altitude plot for Geneva & CERN
CERN
Geneva
Latitude, $^\circ N$
Longitude, $^\circ E$
Altitude (m)
46.40
46.35
46.30
46.25
46.20
46.15
46.10
46.05
46.00
6.00
6.05
6.10
6.15
6.20
6.25
6.30
6.35
6.40
1800
1600
1400
1200
1000
800
600
400


Zoomed releaf map plot for Geneva & CERN
CERN
Geneva
Latitude, $^{\circ}N$
Longitude, $^{\circ}E$
46.40
46.35
46.30
46.25
46.20
46.15
46.10
46.05
46.00
6.00
6.05
6.10
6.15
6.20
6.25
6.30
6.35
6.40
0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
−0.0100
'Intensity'


Zoomed contour plot for Lausanne and Elvian
Latitude, $^\circ N$
Longitude, $^\circ E$
Altitude (m)
Lausanne
Evian
46.70
46.65
46.60
46.55
46.50
46.45
46.40
46.35
46.30
6.40
6.45
6.50
6.55
6.60
6.65
6.70
6.75
6.80
2400
2100
1800
1500
1200
900
600
300


Zoomed smoooooth altitude plot for Lausanne and Elvian
Lausanne
Evian
Latitude, $^\circ N$
Longitude, $^\circ E$
Altitude (m)
46.70
46.65
46.60
46.55
46.50
46.45
46.40
46.35
46.30
6.40
6.45
6.50
6.55
6.60
6.65
6.70
6.75
6.80
2250
2000
1750
1500
1250
1000
750
500


Zoomed relief map plot for Lausanne and Elvian
Latitude, °N
Longitude, °E
Lausanne
Evian
'Intensity'
46.70
46.65
46.60
46.55
46.50
46.45
46.40
46.35
46.30
6.40
6.45
6.50
6.55
6.60
6.65
6.70
6.75
6.80
0.0100
0.0075
0.0050
0.0025
0.0000
−0.0025
−0.0050
−0.0075
−0.0100