# Fit function

* By coding equation 1,,

and plotting against the energy range 0-10 GeV, using the given values of the parameters, the **probability distribution** is obtained to be

Chart, histogram

Description automatically generated

* Also, we know from the given data the **simulated unoscillatory data** can be plotted to be: Chart, histogram

  Description automatically generated
* Therefore, the product of the upper two dist is the **theoretical oscillatory data dist**. Plotting along with the **given experimental data** we have: Chart, histogram

  Description automatically generated
* It can be seen that the gap at first is accounted, but the tail still deviates for some unknown reasons.

# Likelihood function

* By coding the NLL function

Text

Description automatically generated with medium confidence

for the given data, then taking u = as the only parameter, with m and L set as the previous given values, the NLL plot against to give

Chart, histogram

Description automatically generated

* The next step is to find the minimum and error of the minimum.
* For the minimum, the parabola minimization method is used and the **position of the minimum** is obtained and plotted to be 0.7853558966563745Chart, histogram

  Description automatically generated
* Then, to find the **error (standard deviation)**, the values corresponding to NLL\_min + 0.5 is obtained and plotted. Their difference is the standard deviation according to the instruction. Chart

  Description automatically generated
* Now we have the **estimated value** of



Note that I used

* Using the other method, estimating the error from curvature, where
  + The curvature is estimated using numerical calculus with h = 1e-4 (eqn 5.13 in note)
  + The error is estimated to be 0.07114480580876315

# 4. two dimensional minimization

## 4.1 The univariate method

* By coding the **univariate method** and **fitting m\_23\_2 first** (because it is steeper), the 2D minimum is obtained and the path is visualized and shown in the following plot.
* A picture containing graphical user interface

  Description automatically generated

Chart

Description automatically generatedZoomed in

* **The = 0.00227 +- 0.00002**
* **= 0.80 +- 0.03**
* **The value of minimized NLL = 51.7919659453035**
* **Text

  Description automatically generated**
* Using the **curvature error** method.
* The **initial guess** is 
* With changed initial guess : won’t converge
* Pass beyond a certain point the function won’t converge
* 
* Time = 90.1 ms

## 4.2 The Newton method

* Graphical user interface, text, application, email

  Description automatically generated
* By implementing the Newton method and using the same initial guess and appropriate step for gradient finding function, the minimization is accomplished and **path** is recorded and **visualized** as follows.
* A picture containing graphical user interface

  Description automatically generated
* Chart, line chart

  Description automatically generated
* The **fitting results** are
  + **The = 0.00227 +- 0.00002**
  + **= 0.80 +- 0.03**
  + **The value of minimized NLL = 51.8222826038462**
* The **default parameters** are:
  + Step for gradient finding: h0 = 1e-5; h1 = 1e-5
  + Convergent criterion: diff < 1e-5
  + Initial guess: ig = [0.9, 0.002]
* Clearly, the Newton’s method **converges faster** (straight to the minimum)
* Convergent results are nearly **the same**
* **Time**  = 2 ms
* Chart, histogram

  Description automatically generated
  + By replotting the distribution, we see that the fit is sight but the higher parts at the tail is still not accounted.

## 4.3 The Monte-Carlo minimization

* By implementing the Monte-Carlo minimization with chosen parameters using the same initial guess and **appropriate parameters** illustrated below, the minimization is accomplished and **path** is recorded and **visualized** as follows.
* A picture containing graphical user interface

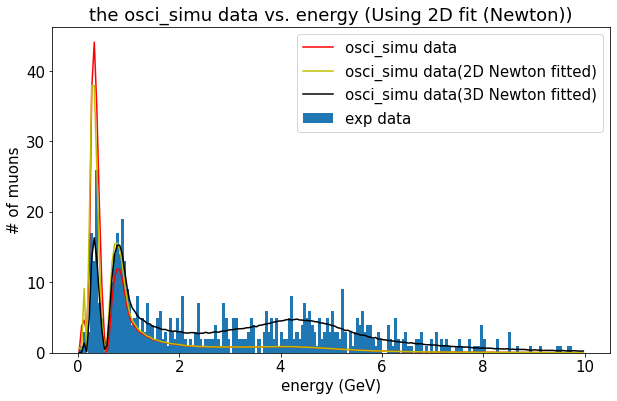
  Description automatically generated
* Chart, line chart

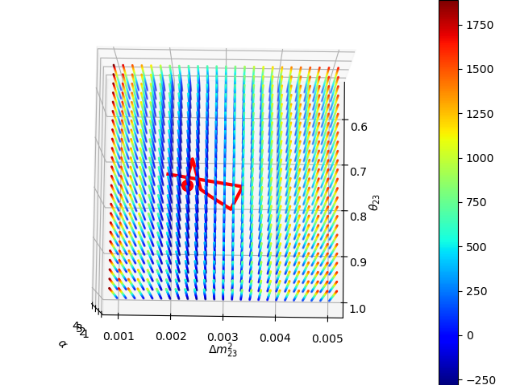
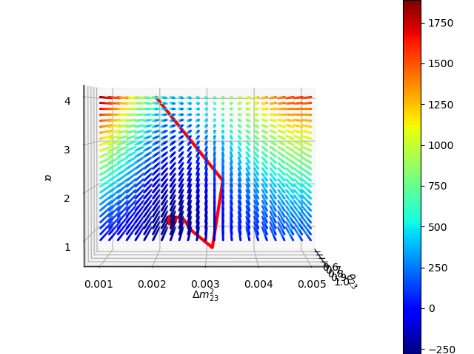
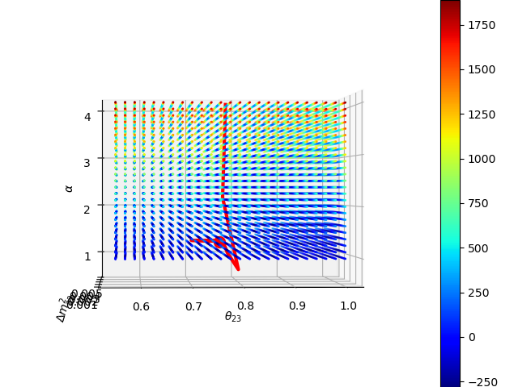
  Description automatically generated
* Note that each time it runs a different version of path will be generated
* The **fitting results** are
  + The = 0.00228 +- 0.00002
  + = 0.78 +- 0.05
  + The value of minimized **NLL** = 51.79702117804046
  + Time = 14ms
* The **default parameters** are:
  + K = 1e-5: corresponding to Boltzmann constant in TD, higher k ---- easier to accept the random change
  + H = 0.1: the step for each increment, normalized with the size of each parameter in code
  + T\_max = 1000: how many times the algorithm will iterate to find a minimum. Corresponds to temperature

# 5. 3D minimization

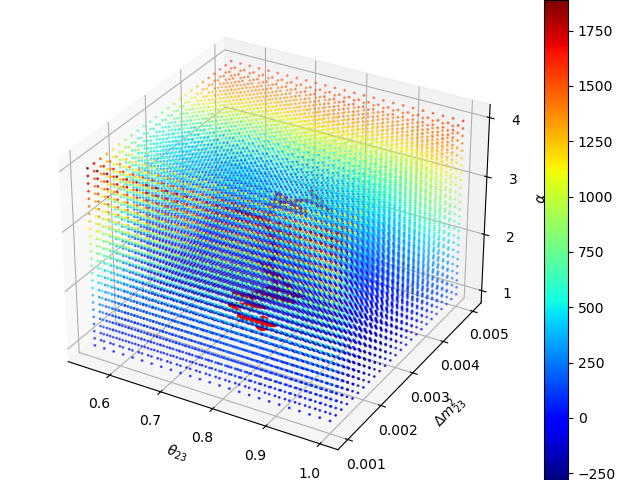
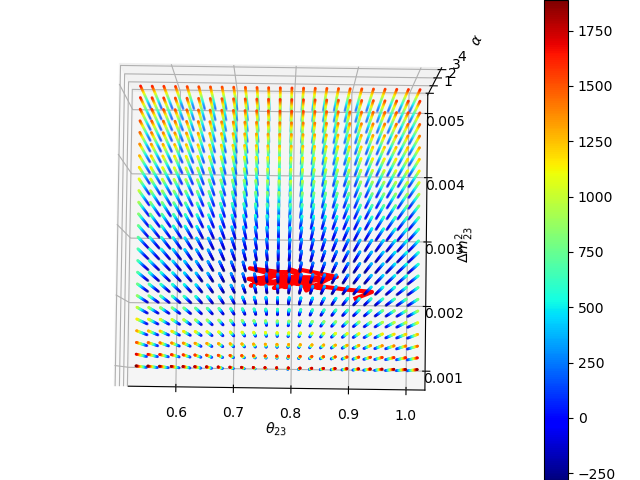
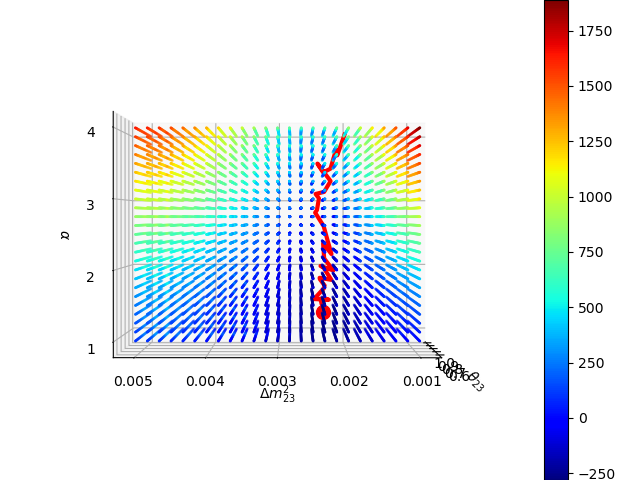
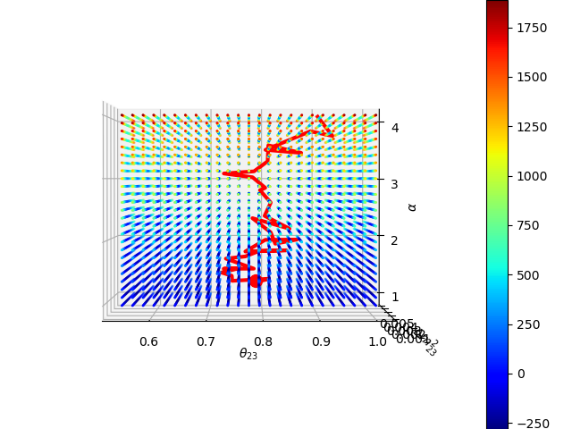
# 5.1 3D Newton minimization

* By adding the variable and implementing the 3D Newton method, the minimization is accomplished with a small enough step h (or else it will not converge).
* First I fixed the m and theta with previously determined minimum parameters to observe the effect of alpha on NLL.
* Chart, histogram

  Description automatically generated
* Then the Newton 3D is carried out. I have not find a way to visualize the path and NLL( ), so the pdf comparison is plotted instead.
* 
* Clearly, the bump at the end can be accounted better by the extra dimension.
* The **fitted parameter** with errors are
  + Newton Method 3D: The value of **m\_23\_2** is 0.00229 +- 0.00003
  + Newton Method 3D: The value of **theta\_23** is 0.76 +- 0.03
  + Newton Method 3D: The value of **a** is 1.29 +- 0.05
  + Newton Method 3D: The value of **NLL** is: -279.28902645700504
* Also the **running time** is 15 ms
* Okay, now I found a way to visualize the path of the NLL\_3D, as follows:
* Chart

  Description automatically generated

## 5.2 Monte Carlo minimization

* By revising the Monte – Carlo minimization to add another dimension, the minimization is implemented rather simply, and the results are shown below.
* The parameters for fitting are
  + K = 1e-5: corresponding to Boltzmann constant in TD, higher k ---- easier to accept the random change
  + H = 0.1: the step for each increment, normalized with the size of each parameter in code
  + T\_max = 500: how many times the algorithm will iterate to find a minimum. Corresponds to temperature
* The results are
  + Monte Method 3D: The value of m\_23\_2 is 0.00234 +- 0.00002
  + Monte Method 3D: The value of theta\_23 is 0.79 +- 0.10
  + Monte Method 3D: The value of a is 1.33 +- 0.05
  + Monte Method 3D: The value of NLL is: -281.1539290398881
* The **running time** is 47ms
* **Not minimum:**
  + when setting T\_max = 50, the function is still going in the correct direction, but did not converge closely to the minimum
  + Monte Method 3D: The value of m\_23\_2 is 0.00229 +- 0.00003
  + Monte Method 3D: The value of theta\_23 is 0.86 +- 0.01
  + Monte Method 3D: The value of a is 1.98 +- 0.08
  + Monte Method 3D: The value of NLL is: -205.3115458258467
* When T\_max = 100, it goes close to minimum
  + Monte Method 3D: The value of m\_23\_2 is 0.00236 +- 0.00003
  + Monte Method 3D: The value of theta\_23 is 0.79 +- 0.03
  + Monte Method 3D: The value of a is 1.31 +- 0.05
  + Monte Method 3D: The value of NLL is: -280.66802379366464

## 5.3 The gradient Method

* The gradient method is highly dependent on the initial parameters(convergent condition and step size),
* Even on simple function like f = x^2 + y^2, the result is unstable:
  + **When** diff(convergent criterion) **= 1e-6, a = 1e-3**
  + Minimized position of simple f is array([0.00035268, 0.00035268])
  + Time taken is 2.12s
  + Initial guess = [1, 1]
  + **When** diff(convergent criterion) **= 1e-4, a = 1e-2**
  + Minimized position of simple f is array [0.00342395, 0.00342395]
  + Time taken is 140 ms
  + Initial guess = [1, 1]
* Therefore, this method is time-consuming to find appropriate parameters ,takes long time to run and possibly with lower accuracy.
* As a result, I will not use this method for the latter analysis.