ENGR110 Machine Learning Assignment

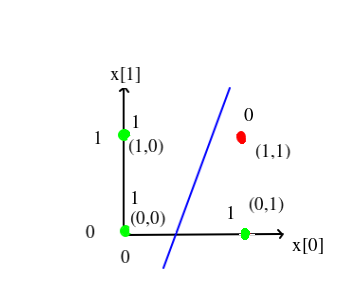
Username: leoanelge

Name: Elgene Menon Leo Anthony

ID: 300492604

**Exercise 1:**

Using Fig.1 as a template draw two regions: one region contains all output 0s and another all 1s. Boundary between regions should be the straight line. Include the picture into your report.

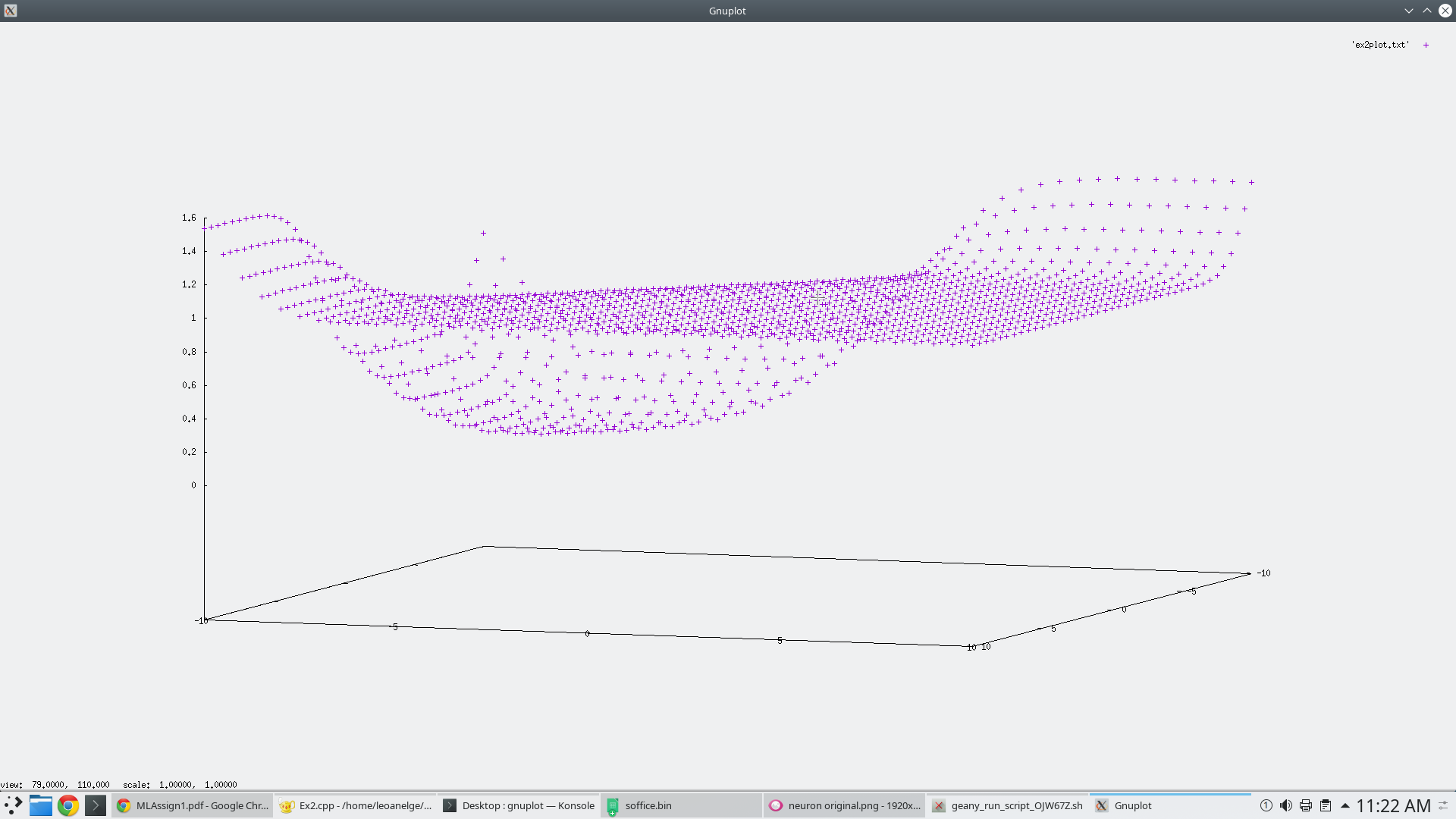


**Exercise 2:**

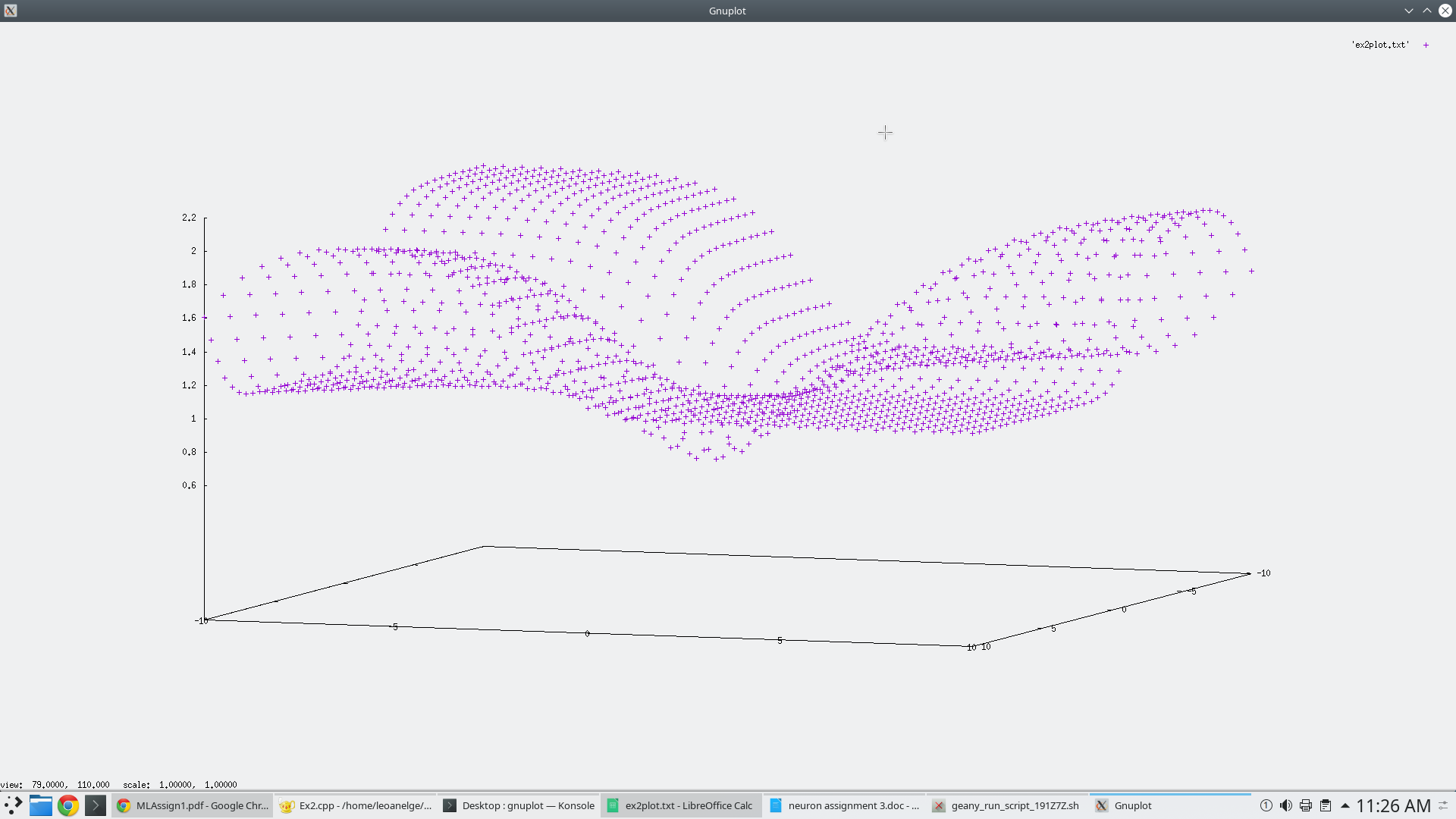
* Explain what an error variable represents and how it is calculated in Ex2.cpp?

The error variable represents the difference between the current value and the desired value. You want to minimise the error so that the current value is as close to the desired value as possible. The error is calculated in the ForwardProp function and is the sum of the squared error for each row.

* Include picture produced gnuplot taking output file of Ex2.cpp. Try different value of b to obtain another picture and include that too.



bias is 9.0



bias is 1.0

* Explain what values of weights w0 and w1 for given value of bias b are best to provide minimum of the error.

For the given bias of 9.0, the values w0 = -6 and w1 = -6 are best to provide the minimum of the error. For any given bias, the weights that give the minimum error which is 0.00674766 correspond to the points on the x and z axis of the graph, where the graph is at its lowest point.

**Exercise 3:**

* It asks for values of b, w0 and w1 (you can investigate C++ cin operator)

void NN1::UserInput(){

cout << "Enter the w0 value: ";

cin>>w[0];

cout << "Enter the w1 value: ";

cin>>w[1];

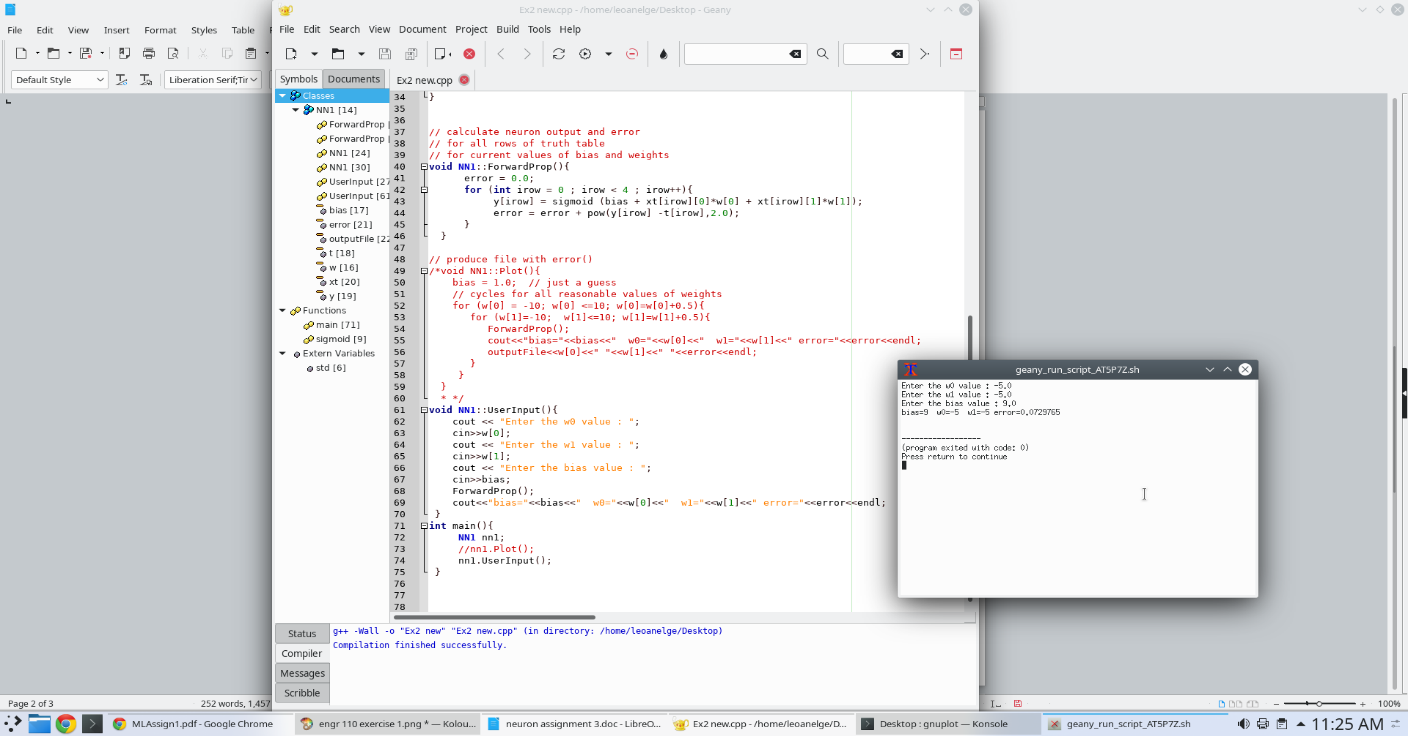
cout << "Enter the bias value: ";

cin>>bias;

ForwardProp();

cout<<"bias="<<bias<<" w0="<<w[0]<<" w1="<<w[1]<<" error="<<error<<endl;

}



* It calculates output y using these values of b, w0 and w1 for 4 combinations of the inputs x0 and x1: {0,0}, {1,0}, {0,1}, {1,1} using equations (1) and (2). You can use some parts of Ex2.cpp if you like.
* It calculates error for each input combination
* It calculates and prints combined error Include code (as a text, not picture) in your submission.
* Run your program at several times for different values of b, w0 and w1

|  |  |  |  |
| --- | --- | --- | --- |
| bias | w0 | w1 | error |
| 9 | -6 | -6 | 0.00674766 |
| 30 | -234 | -312 | 2 |
| 4 | -21 | -45 | 2.00032 |
| 9 | -5 | -8 | 0.0729765 |

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | -9 | -12 | 1.90246 |
| 20 | 7 | 8 | 1 |

* Explain which combination is better?

The combination bias = 9, w0 = -6, w1 = -6 is the best because the error value is the lowest, meaning it is the closest to the desired value.

**Exercise 4:**

A screenshot of a cell phone

Description automatically generatedBias = 9.0, w0 = -6, w1=-6

This decision boundary is almost the same as the answer in exercise 1. The decision boundary is in a good place because the points (0,0), (0,1) and (1,0) are all classified as 1s and the point (1,1) is classified as a 0. All four points will be classified correctly.

Bias = 7, w0 = -3, w1=-A screenshot of a cell phone

Description automatically generated14

This decision boundary is also not ideal. While the points (0,0), (1,0) and (1,1) will be classified correctly, the point (0,1) will be incorrectly classified as a 0 rather than a 1. Three of the points will be classified correctly.

A screenshot of a cell phone

Description automatically generated

Bias =234, w0 = -342, w1= -423

This decision boundary is not ideal. Although the point (0,0) will be classified as a 1 and the point (1,1) will be classified as a 0, the points (0,1) and (1,0) will be incorrectly classified as 0s instead of 1s. Two of the points will be classified correctly.

A screenshot of a cell phone

Description automatically generated  
Bias = 20, w0 = -6, w1=-6

These values are far from optimum, because the decision boundary cannot even be seen on the image. This means that at least 3 of the points are being incorrectly classified. One of the points might be classified correctly.

**Exercise 5:**

void NN1::GlobalSearch(){

double w0;

double w1;

double b;

double minError = 1000;

for (w[0] = -10; w[0] <=10; w[0]=w[0]+0.1){

for (w[1]=-10; w[1]<=10; w[1]=w[1]+0.1){

for(bias = -10; bias <=10; bias = bias + 0.1){

ForwardProp();

//cout<<"bias="<<bias<<" w0="<<w[0]<<" w1="<<w[1]<<" error="<<error<<endl;

if(error<minError){

w0 = w[0];

w1 = w[1];

b = bias;

minError = error;

}

}

}

}

cout<<"bias="<<b<<" w0="<<w0<<" w1="<<w1<<" error="<<minError<<endl;

}

**Exercise 6:**

void NN1::CalculateGradient(){

double step = 0.1;

ForwardProp();

double e0 = error;

bias = bias + step;

ForwardProp();

double e1 = error;

bias = bias - step;

dBias = (e1 - e0) / step;

w[0] += step;

ForwardProp();

e1 = error;

dW[0] = (e1 - e0) / step;

w[0] -= step;

w[1] += step;

ForwardProp();

e1 = error;

dW[1] = (e1 - e0) / step;

w[1] -= step;

}

void NN1::GradientStep(){

double learning\_rate = 30.0;

bias = bias - learning\_rate \* dBias;

w[0] = w[0] - learning\_rate \* dW[0];

w[1] = w[1] - learning\_rate \* dW[1];

}

void NN1::GradientSearch(){

int nStep = 0;

ForwardProp();

while(nStep<400){

CalculateGradient();

GradientStep();

nStep++;

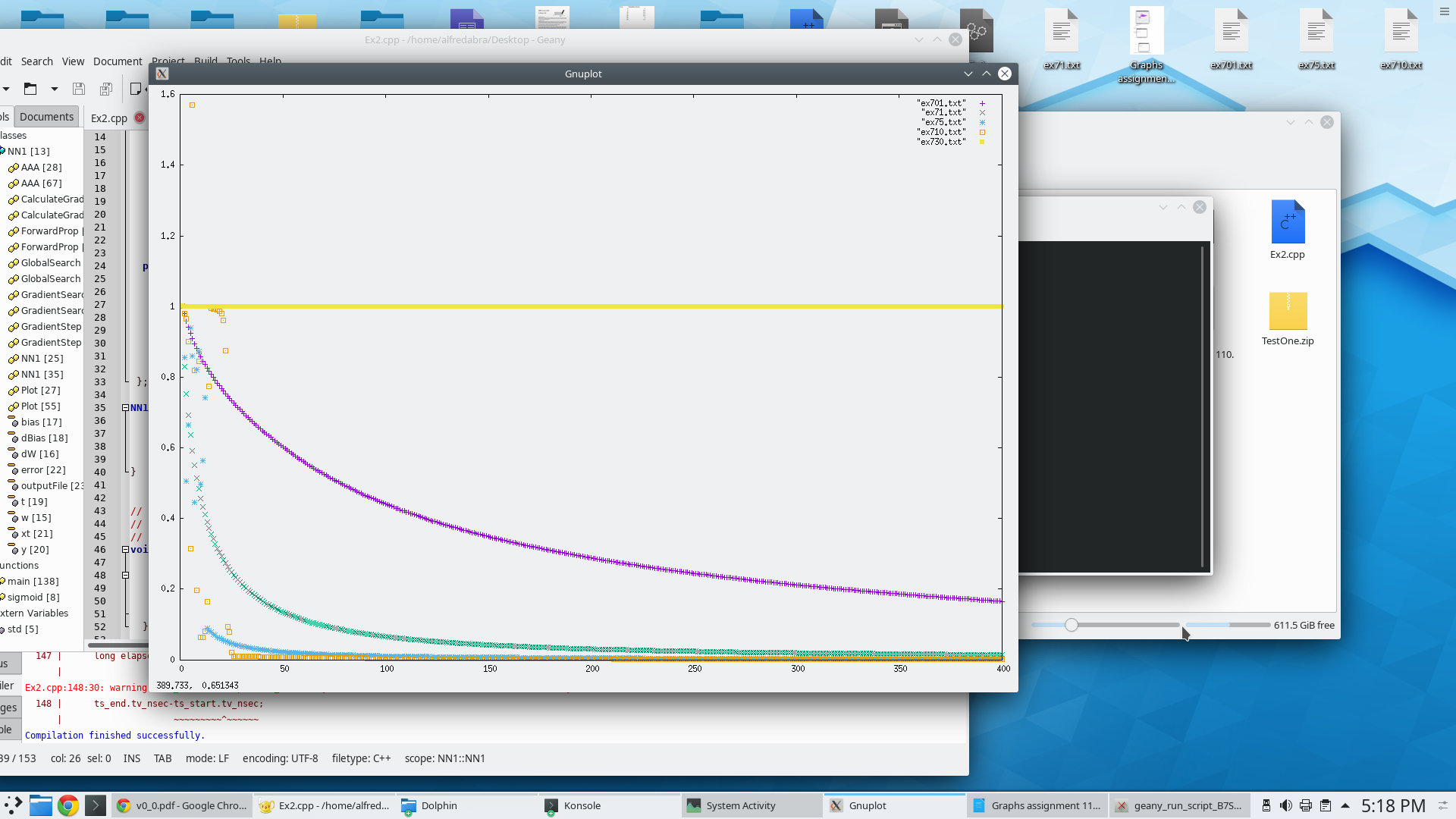
cout<<error<<endl;

outputFile<<nStep<<" "<<error<<endl;

}

}

**Exercise 7:**



* Explain what is happening with search convergence as learning rate increases.

If the learning rate is very small (purple line), the search convergence takes very small steps and takes a very long time to reach 0 error. If the learning rate is too high (yellow line), the search convergence will take very big steps that cause it to constantly miss the zero error and instead step back and forth across it. If the learning rate is at a good value, it takes steps that are large enough to reach zero quickly, but small enough not avoid missing the lowest error.