1-C) Visualizing Regularization

Given the error function to be optimized:

$$minimise\ E(w) = \frac{1}{2}\sum_{n=1}^{N}(y_n - t_n)^2\ subject\ to\ |w_1|^q + |w_2|^q \leq \eta$$

Here, **t** is the actual observation and **y** is the predicted observation given as:

$$y_n = w_1 x_{n_1} + w_2 x_{n_2}$$

Where N is the total number of datapoints and w1 and w2 are the corresponding parameters for the independent variables x1 and x2 in the given model. Since regularization is done in order to reduce overfitting, which in turn helps in reducing the variance of the estimated regression parameters.

1. Error Contour Plots:

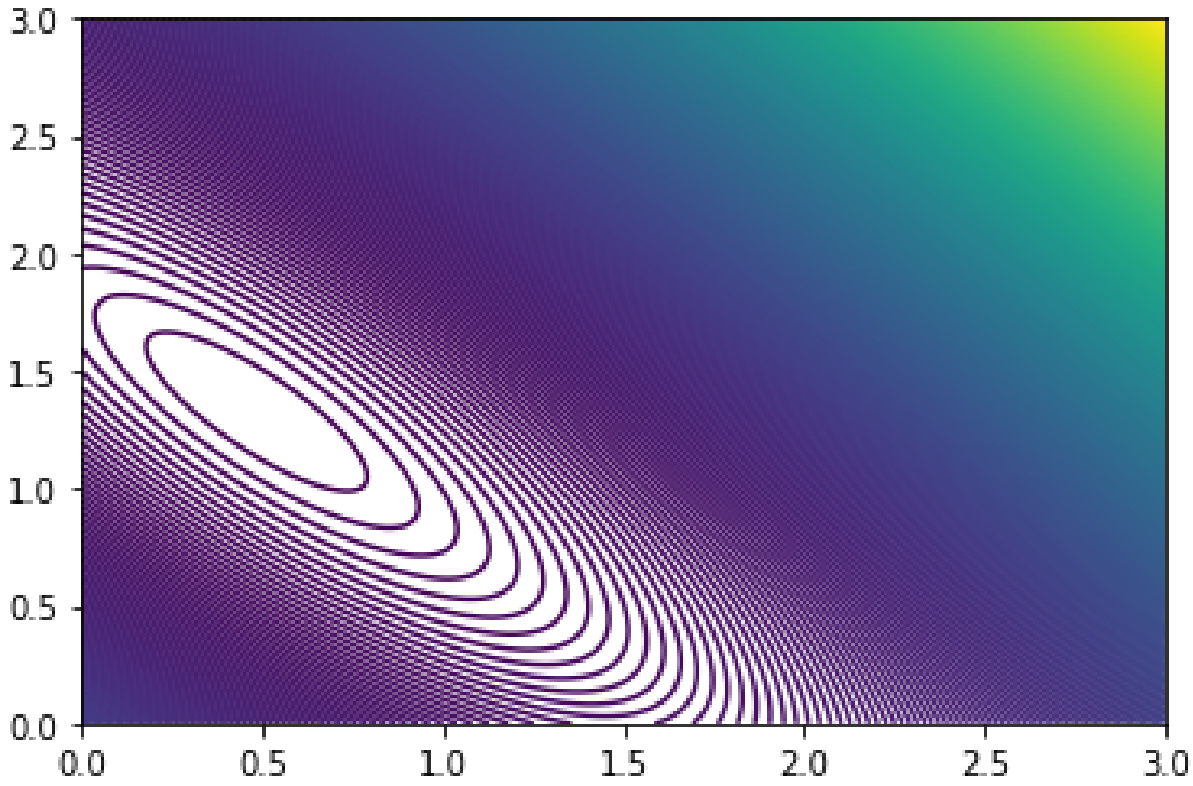
From the provided dataset, we took the corresponding values of **MLOGP(x1)** and **RDCHI(x2)** and assigned some w1 and w2 to them respectively.
We have ranged the values of the corresponding w from 0 to 3 and used the given function to calculate the error.

```
def error_calc(col1,col2,col3,w1,w2,n):
  total_error=0
  for i in range(n):
    total_error+=(col3[i]-(w1*col1[i]+w2*col2[i]))**2
  return total_error/2
```

Storing the corresponding error values in a multidimensional array, we plotted the contours.

Contour Plots:

A contour plot is a graphical technique for representing a 3-dimensional surface by plotting constant z slices, called contours, on a 2-dimensional format. That is, given a value for z, lines are drawn for connecting the (x,y) coordinates where that z value occurs.



To achieve this contour plot we have used the following code snippet and fetched the plot at 500 levels.

```
for i in range(N):
  for j in range(M):
    error[i][j] = error_calc(x1,x2,t,w1[i],w2[j],t.size)
  # print(f[i][j])

figure=plt.figure(figsize=(50,50))
fig, ax=plt.subplots()
ax.contour(X,Y,np.transpose(error),500)
plt.show()
```

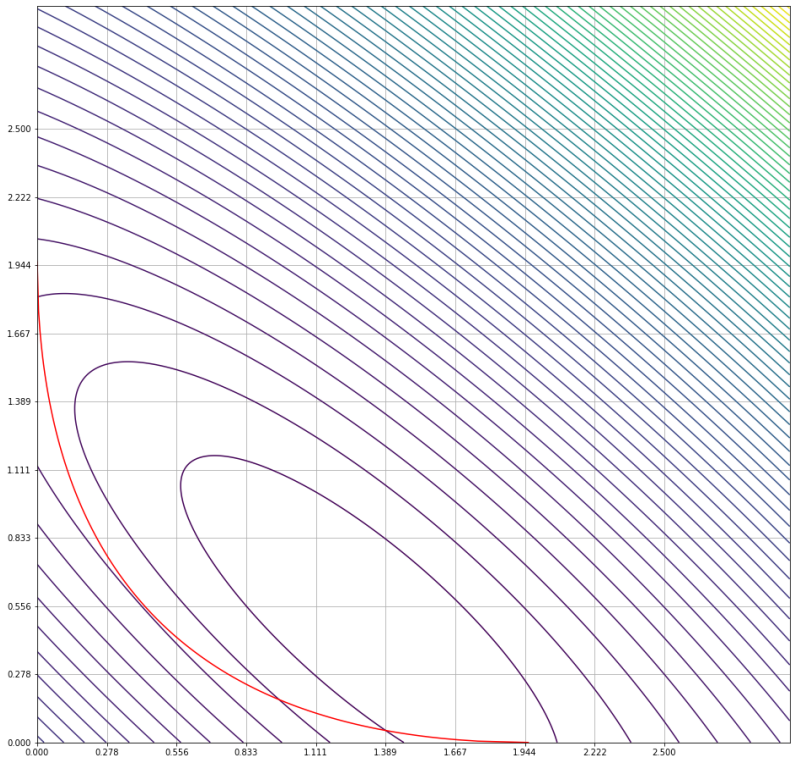
2. Constraint regions and error function contours, showing the tangential contour and the point of intersection where the minima occur:

We have limited the value of parameters by regularizing them using four constraint equations.

$$|w_1|^q + |w_2|^q \leq \eta$$

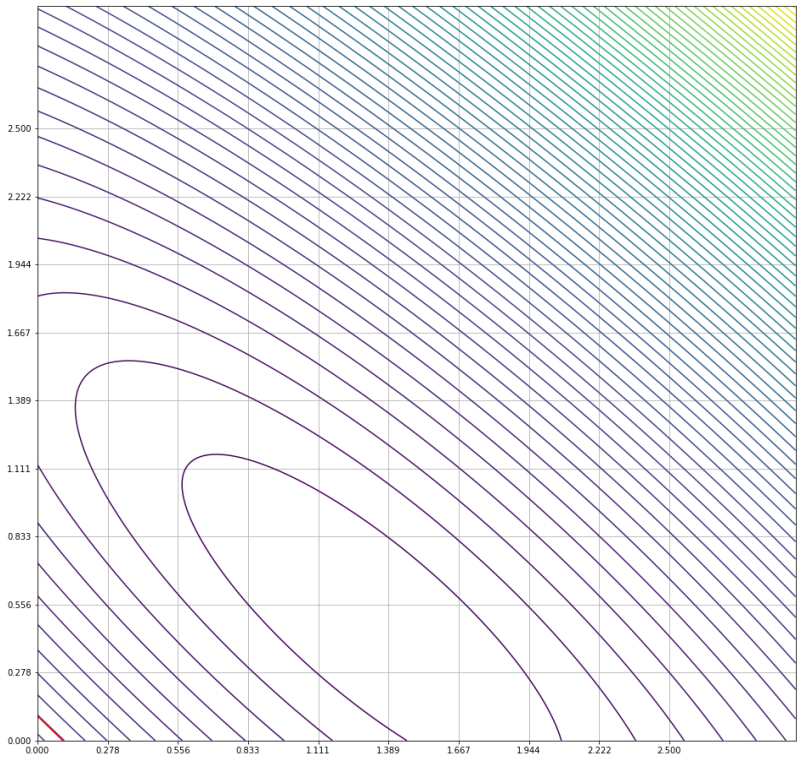
q = 0.5, 1, 2, 4 and η = 1.4, 0.1, 0.035, 0.052 respectively.

q = 0.5



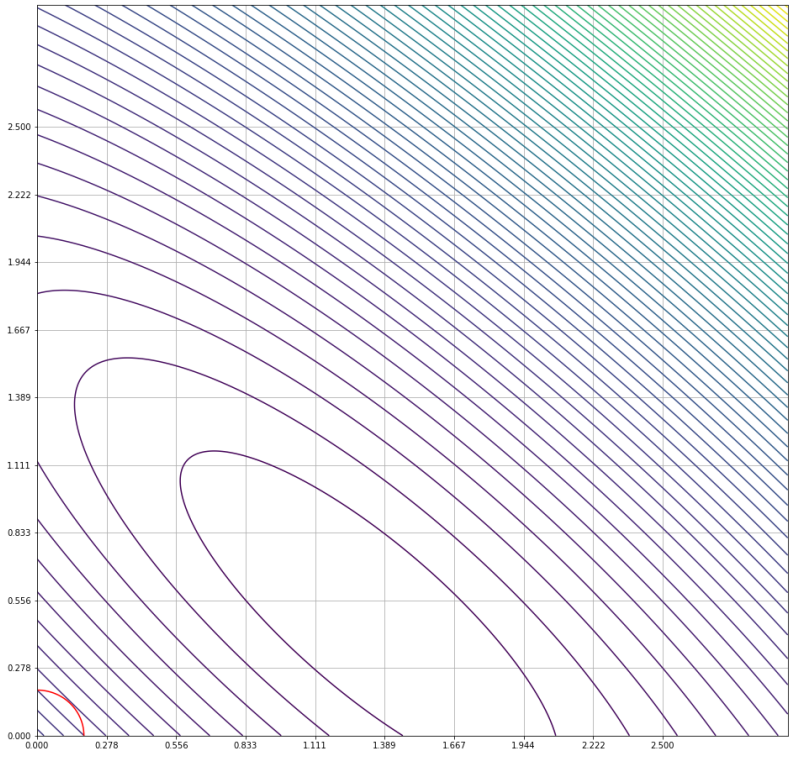
$$w_1^{0.5} + w_2^{0.5} \leq 1.4$$

q = 1 (Lasso Regression)



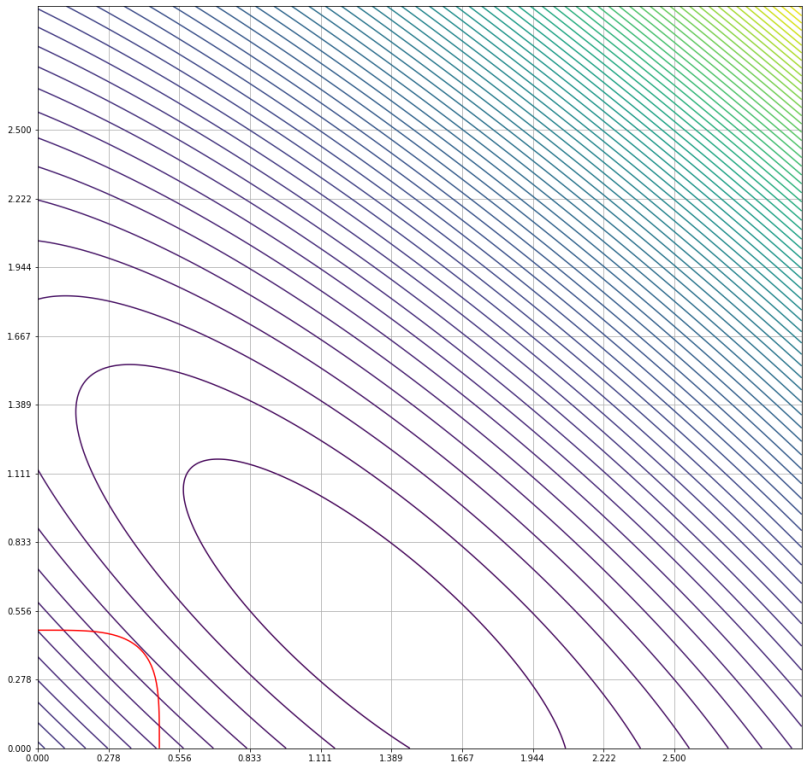
$$w_1^1 + w_2^1 \leq 0.1$$

q = 2 (Ridged Regression)



$$w_1^2 + w_2^2 \leq 0.035$$

q = 4



$$w_1^4 + w_2^4 \leq 0.052$$

Minima Occurs at:

	$q = 0.5$	$q = 1$	$q = 2$	$q = 4$
$w1$	0.005	0.100	0.132	0.402
$w2$	1.767	0.000	0.132	0.402

These values of w1 and w2 are approximated from the above contour plots.

3. Mean Squared Errors where the hence obtained weights are used to make polynomial model for regression for each case:

The Mean Squared Error **measures how close a regression line is to a set of data points**. Acts as a function corresponding to the expected value of the squared error loss. Mean square error is calculated by taking the average, specifically the mean, of errors squared from data as it relates to a function.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Here, n is the number of data points; Yi is the observed value and $\widehat{Yi}$ is the predicted value.

We have calculated MSE corresponding to different q values(0.5,1,2,4) at η values being (1.4,0.1,0.035,0.052) respectively. For which, the **output received** is as followed:

	$q = 0.5$	$q = 1$	$q = 2$	$q = 4$
MSE	3.009	22.087	18.493	9.276

To obtain these results, we used the same error function used above and then divided the error by the total number of observations as:

```
t_ = t.size/2

print("For q = 0.5")
print(error_calc(x1,x2,t,0.005,1.767,t.size) / t_)

print("\nFor q = 1")
print(error_calc(x1,x2,t,0.1,0,t.size) / t_)

print("\nFor q = 2")
print(error_calc(x1,x2,t,0.132,0.132,t.size) / t_)

print("\nFor q = 4")
print(error_calc(x1,x2,t,0.402,0.402,t.size) / t_)
```

Where the error_calc function is:

```
def error_calc(col1,col2,col3,w1,w2,n):
  total_error=0
  for i in range(n):
    total_error+=(col3[i]-(w1*col1[i]+w2*col2[i]))**2
  return total_error/2
```