Question 1 (Linear regression + regularization techniques + data augmentation) (+50)

In this problem, we approximate the polynomial from noisy data points. To that end, we generate a data pair $(x^{(i)}, y^{(i)})$ from the following noise-corrupted relation

$$y^{(i)} = .5 - x^{(i)} - .5{x^{(i)}}^2 - 2{x^{(i)}}^3 + 5{x^{(i)}}^4 + \delta\varphi^{(i)}x^{(i)}$$

of the following ground truth function

$$y = .5 - x - .5x^2 - 2x^3 + 5x^4$$

where $\delta = 0.05$ is the noise level, φ is drawn from standard normal distribution. We need to generate 8 holdout samples and 7 training samples.

- (1.a) (+5) Fill the generating data code and produce figures to visualize data and true functions.
- (1.b) (+5) Perform the LinearRegression() using polynomial approximation of order of 1, 10 and 4. Visualizing each cases to see which case are overfitting, underfitting and good fit. Adding the mean square error of the holdout data to support your observation.
- (1.c) (+10) For overfitting senario, applying L2 regularization. You need to do:
  - step 1: Plot the L2 regularization model versus no regularization model versus the ground truth function in one plot to see how L2 works (pick a reasonable λ).
  - step 2: Plot the mean square error of holdout data as a function of the regularization parameter λ over the range $\lambda \in [10^{-5}, 10]$. What is the optimal λ approximately?
  - step 3: Using L-curved technique (read [https://www.sintef.no/globalassets/project/evitameeting/2005/lcurve.pdf]) to find out the optimal λ. The L-curved line is the one represents the relationship between residual norm $||y - f(x, \theta)||_2$ and solution norm $||\theta||_2$. Plot the L-curve and compare the optimal values of λ from step 2 and step 3?
- (1.d) (+10) For overfitting senario, applying L1 regularization. Repeat the all 3 steps in the question (1.c).
- (1.e) (+10) For overfitting senario, applying Elasticnet regularization (a mixture of L1 and L2), i.e., the loss function with Elasticnet regularization is

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(y^{(i)}, \theta^\top x^{(i)}) + \lambda \cdot ||\theta||_1 + \frac{1}{2}\gamma||\theta||_2^2$$

We are interested in the performance of the mean square error of the holdout data as a function of λ and γ. One way to achieve this is to plot the contour of the mean square error of the holdout data as a function of λ and γ. Discuss the relationship between Elasticnet and L2, L1 regularization?

- (1.g) (bonous +10) The question is **can we do something else** and yet achieve the same regularization effect? One way to accomplish this is via **data augmentation technique**. The data augmentation technique that we study consists of 4 steps:
  - step 1: Clone 7 training data samples (X in the code) into 350 samples (using numpy.repeat). (Note: this is noise-free data).
  - step 2: We add noise these 350 samples by

$$\tilde{x}^i = x^i + \delta\varphi^i,$$

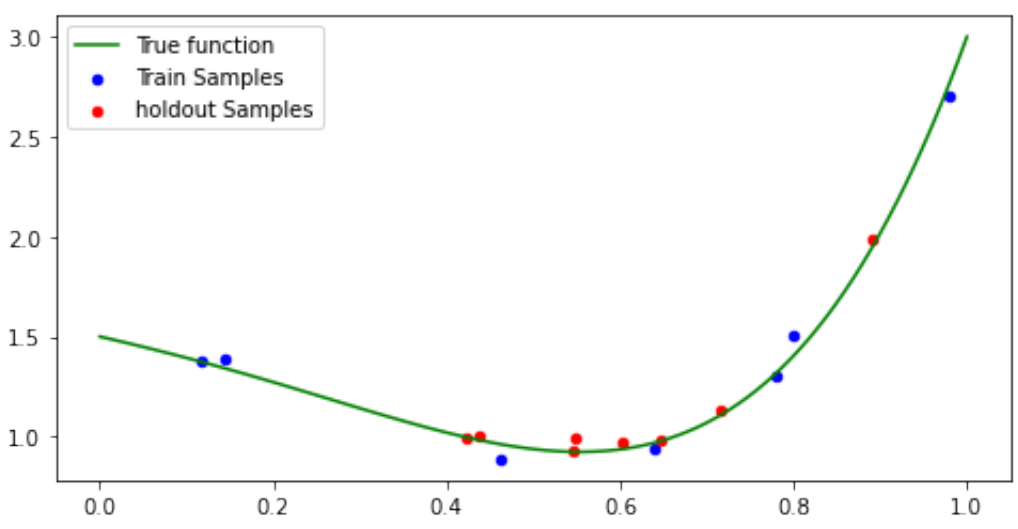
    where φ^i is drawn from standard normal distribution. We get X_data_augmented.
  - step 3: Similarly, clone 7 training label data into 350 samples (using numpy.repeat), denoted y_data_augmented.
  - step 4: Using Linearregression to train using data augmented data pairs (X_data_augmented, y_data_augmented).

  You need to (1) generate augmented data, (2) Determine a good noise level δ by trying different values of δ? Explain why too much or too small noise is not good? To answer the last question, you may need to read the folllowing references

1. Train Neural Networks With Noise to Reduce Overfitting, [https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/]
2. [https://arxiv.org/abs/2208.04995]

In [49]:
```
#1a -- Plotting true function, train samples and holdout samples
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings('ignore')

import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [8, 4]

import pandas as pd
from sklearn import datasets
np.random.seed(0)

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, ElasticNet, Lasso, LogisticRegression


order_true = 5
true_coefficient = np.array([.5, -1, -.5, -2, 5])

def true_fn(X):
    f = np.ones((X.shape))
    for i in range(order_true):
        f += true_coefficient[i] * X**i
    return f

# Test samples
n_holdout_samples = 8
X_holdout = np.sort(np.random.rand(n_holdout_samples))
Y_holdout = true_fn(X_holdout) + np.random.randn(n_holdout_samples) * 0.05 * true_fn(X_holdout)

# Training samples
n_samples = 7

X = np.sort(np.random.rand(n_samples))
Y = true_fn(X) + np.random.randn(n_samples) * 0.05 * true_fn(X)

X_test = np.linspace(0., 1., 100)

plt.plot(X_test,true_fn(X_test),'g',label = 'True function')
plt.scatter(X,Y, edgecolor='b', s=20, facecolor = 'b', label="Train Samples")
plt.scatter(X_holdout,Y_holdout, edgecolor='r', facecolor = 'r', s=20, label="holdout Samples")
plt.legend()
```

Out[49]:
```
<matplotlib.legend.Legend at 0x7fea3bc45100>
```



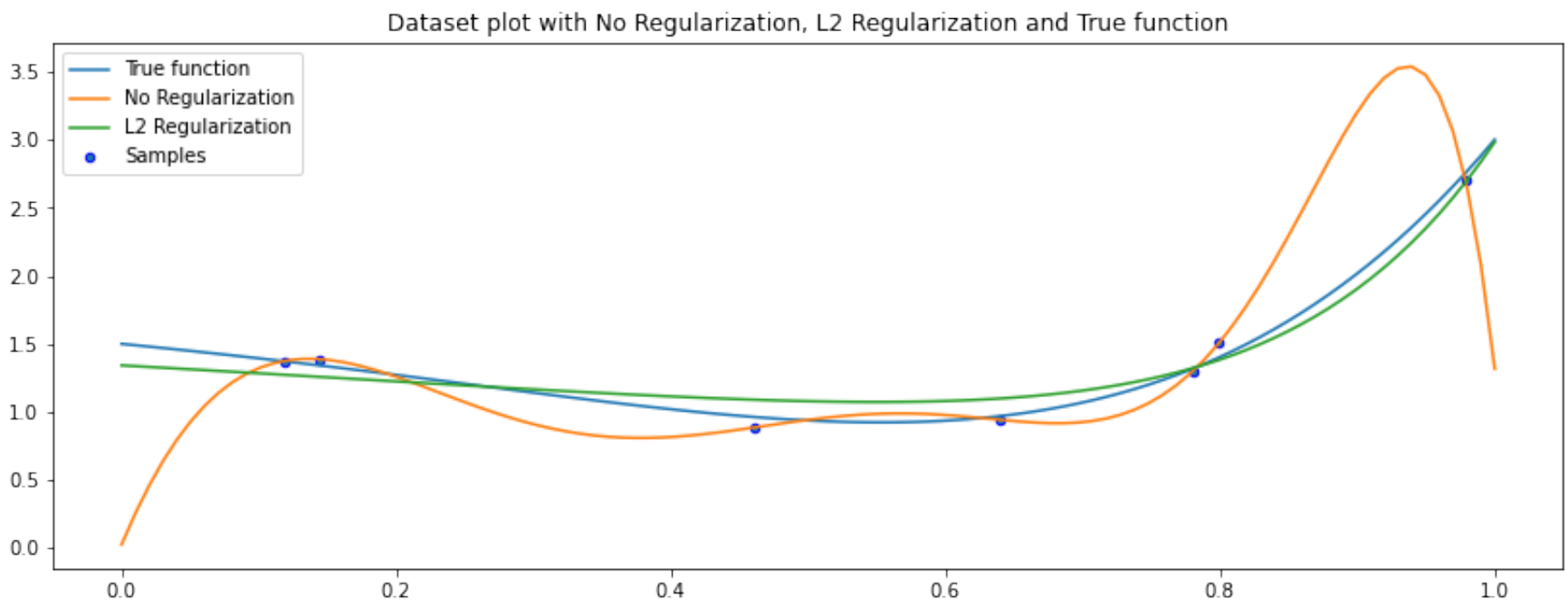
In [50]:
```
#1b --- Overfit, Underfit and Good Fit code
degrees = [1,10,4]

titles = ['Underfitting', 'Overfitting', 'A Good Fit']
plt.figure(figsize=(14, 5))
for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)

    # fit a least squares model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("pf", polynomial_features), ("lr", linear_regression)])
    pipeline.fit(X[:, np.newaxis], Y)

    ax.plot(X_test, true_fn(X_test), label="True function")
    ax.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
    ax.scatter(X, Y, edgecolor='b', s=20, label="Samples")
    ax.scatter(X_holdout[::3], Y_holdout[::3], edgecolor='r', s=20, label="Holdout Samples")
    ax.set_xlim((0, 1))
    ax.set_ylim((-2, 2))
    ax.legend(loc="best")
    ax.set_title("{} (Degree {})".format(titles[i], degrees[i]))
    Y_holdout = true_fn(X_holdout) + np.random.randn(n_holdout_samples) * 0.05 * true_fn(X_holdout)
    ax.text(0.12,-0.9, 'Holdout MSE: %.4f' % ((Y_holdout-pipeline.predict(X_holdout[:, np.newaxis]))**2).mean())
```

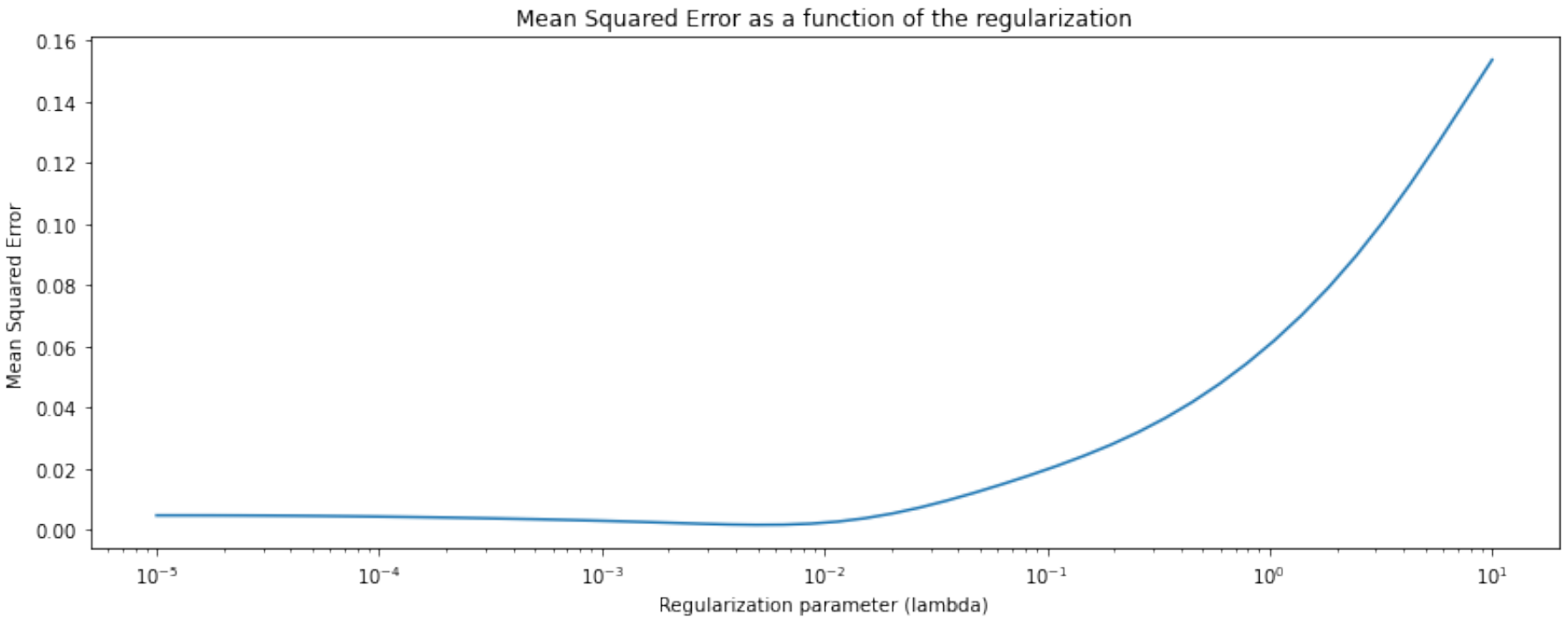
In [55]:
```
#1c--- L2 regularization for Overfitting Scenario
'''1c - Step 1'''

degrees = [10]
plt.figure(figsize=(14, 5))

for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)

    # fit a least squares model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("pf", polynomial_features), ("lr", linear_regression)])
    pipeline.fit(X[:, np.newaxis], Y)


    # fit a Ridge model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    ridge_regression = Ridge(alpha=0.1)  #sklearn uses alpha instead of lambda
    pipeline2 = Pipeline([("pf", polynomial_features), ("ri", ridge_regression)])
    pipeline2.fit(X[:, np.newaxis], Y)

    # visualize results
    ax.plot(X_test, true_fn(X_test), label="True function")
    ax.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="No Regularization")
    ax.plot(X_test, pipeline2.predict(X_test[:, np.newaxis]), label="L2 Regularization")
    ax.scatter(X, Y, edgecolor='b', s=20, label="Samples")
    ax.legend(loc="best")
    ax.set_title("Dataset plot with No Regularization, L2 Regularization and True function")


#1c---  Mean Squared Error of holdout data as a function of L2 Regularization parameter Lambda
'''1c - Step 2'''

alphas = np.logspace(-5, 1,  )
mse = []            #Initializing Mean Squared Error
```

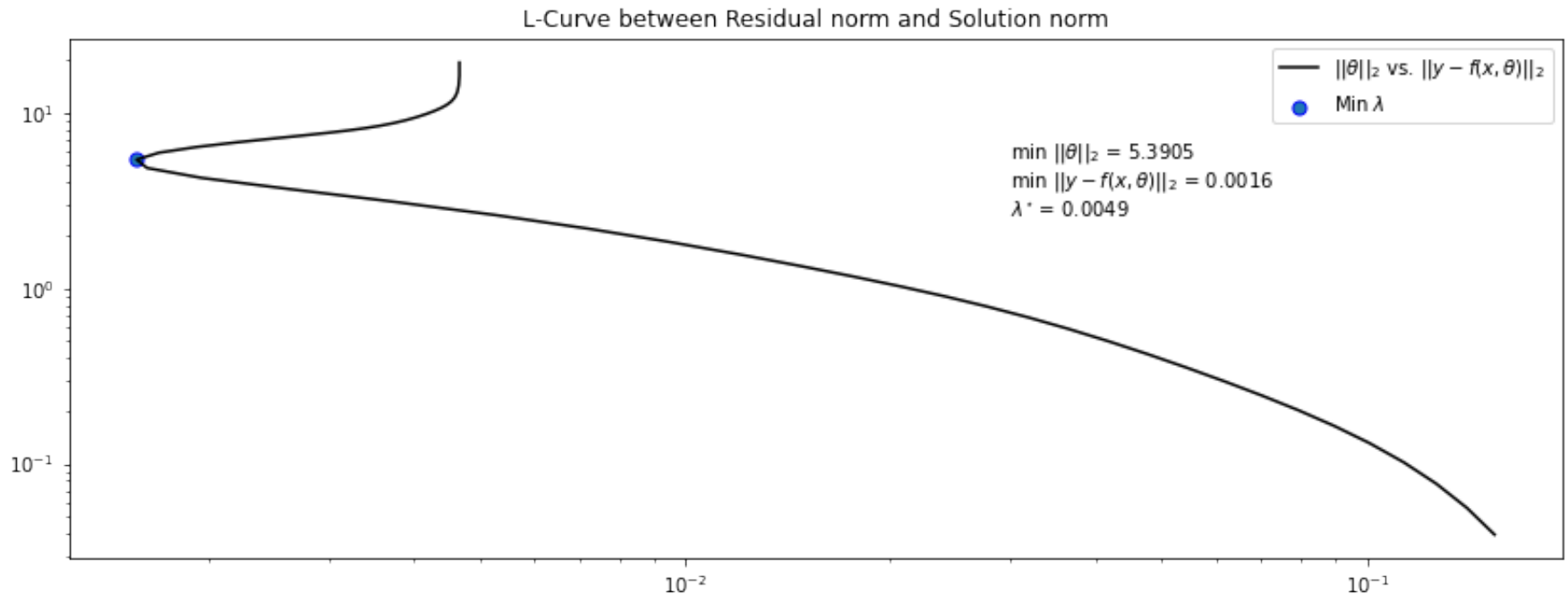
```
for a in alphas:
        polynomial_features = PolynomialFeatures(degree= 10, include_bias=False)
        ridge_regression = Ridge(alpha= a)  #sklearn uses alpha instead of lambda
        pipeline3 = Pipeline([("pf", polynomial_features), ("ri", ridge_regression)])
        pipeline3.fit(X[:, np.newaxis], Y)
        mse.append(((Y_holdout-pipeline3.predict(X_holdout[:, np.newaxis]))**2).mean())
#print(min(mse),mse.index(min(mse)),alphas[mse.index(min(mse))])
print('Optimal Lambda using L2 Regularization is:', alphas[mse.index(min(mse))])

# plot MSE wrt
plt.figure(figsize=(14, 5))
plt.plot(alphas, mse)
plt.xscale('log')
plt.xlabel('Regularization parameter (lambda)')
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error as a function of the regularization')
plt.axis('tight')

#1c --- L Plot using Residual and Solution Norm
'''1c - Step 3'''

residual_norm = []
solution_norm = []
alphas = np.logspace(-5, 1,  )

for a in alphas:
    polynomial_features = PolynomialFeatures(degree= 10, include_bias=False)
    ridge_regression = Ridge(alpha= a)  #sklearn uses alpha instead of lambda
    pipeline4 = Pipeline([("pf", polynomial_features), ("ri", ridge_regression)])
    pipeline4.fit(X[:, np.newaxis], Y)

    residual_norm += [((Y_holdout - pipeline4.predict(X_holdout[:, np.newaxis]))**2).mean()]
    solution_norm += [(ridge_regression.coef_**2).sum()]

plt.figure(figsize=(14, 5))
plt.loglog(residual_norm, solution_norm, 'k-', label = "$|| \\theta ||_{2}$ vs. $|| y - f(x,\\theta) ||_{2}$")
resmin = np.min(residual_norm)
solmin = solution_norm[np.argmin(residual_norm)]
alphamin = alphas[np.argmin(residual_norm)]
print('Optimal Lambda using L-Curve is:', alphamin)

plt.scatter(resmin, solmin, edgecolor = 'b', s= 50, label = "Min $\\lambda$")
plt.text(0.03,5.5,'min $|| \\theta ||_{2}$ = %.4f' % solmin)
plt.text(0.03,3.8,'min $|| y - f(x,\\theta) ||_{2}$ = %.4f' %resmin)
plt.text(0.03,2.5,'$\lambda^{\star}$ = %.4f' %alphamin)
plt.legend(loc = "best")
plt.title('L-Curve between Residual norm and Solution norm')
plt.axis('tight')

'''Lambda for both step 2 and 3 have the same optimal value which is approximately 0.0049 which implies that eith
of these two methods (L2 Regularization or L-Curve) can be used to find optimal regularization parameter of lambd
```

```
Optimal Lambda using L2 Regularization is: 0.004941713361323833
Optimal Lambda using L-Curve is: 0.004941713361323833
```

Out[55]:
```
'Lambda for both step 2 and 3 have the same optimal value which is approximately 0.0049 which implies that eithe
r\nof these two methods (L2 Regularization or L-Curve can be used to find optimal regularization parameter of la
mbda'
```





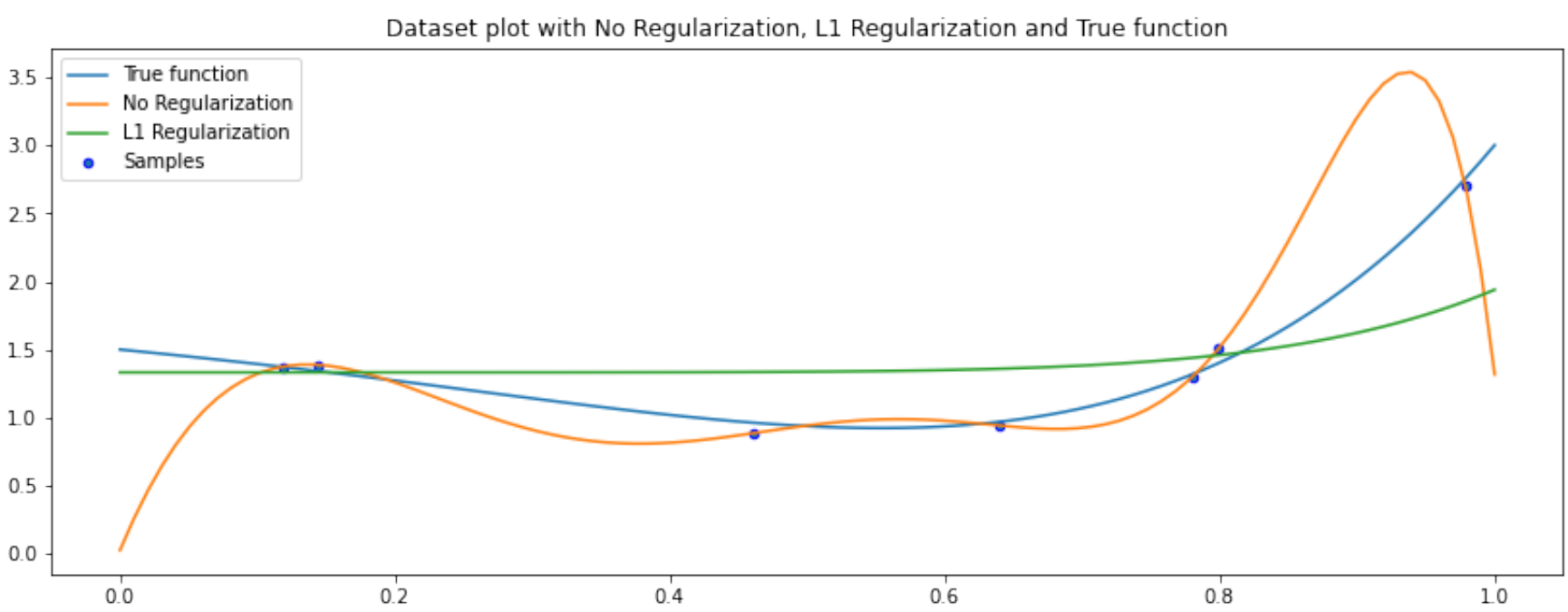


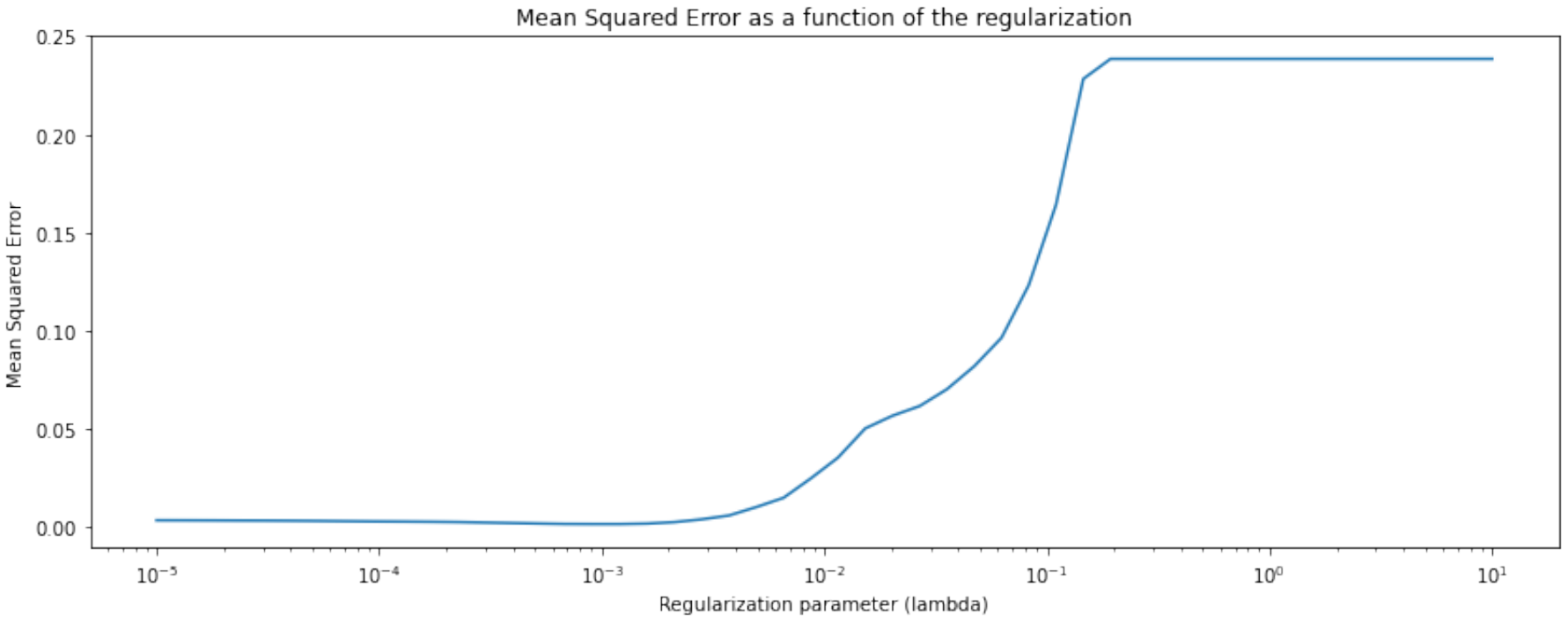
In [56]:
```
#1d--- L1 regularization for Overfitting Scenario
'''1d - Step 1'''

degrees = [10]
plt.figure(figsize=(14, 5))

for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)

    # fit a least squares model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline = Pipeline([("pf", polynomial_features), ("lr", linear_regression)])
    pipeline.fit(X[:, np.newaxis], Y)


    # fit a Ridge model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    Lasso_regression = Lasso(alpha=0.1)  #sklearn uses alpha instead of lambda
    pipeline2 = Pipeline([("pf", polynomial_features), ("la", Lasso_regression)])
    pipeline2.fit(X[:, np.newaxis], Y)

    # visualize results
    ax.plot(X_test, true_fn(X_test), label="True function")
    ax.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="No Regularization")
    ax.plot(X_test, pipeline2.predict(X_test[:, np.newaxis]), label="L1 Regularization")
    ax.scatter(X, Y, edgecolor='b', s=20, label="Samples")
    ax.legend(loc="best")
    ax.set_title("Dataset plot with No Regularization, L1 Regularization and True function")

#1d --- Mean Squared Error of holdout data as a function of L1 Regularization parameter Lambda
'''1d - Step 2'''

alphas = np.logspace(-5, 1,  )
mse = []

for a in alphas:
        polynomial_features = PolynomialFeatures(degree= 10, include_bias=False)
        linear_regression = Lasso(alpha= a)  #sklearn uses alpha instead of lambda
        pipeline3 = Pipeline([("pf", polynomial_features), ("lr", linear_regression)])
        pipeline3.fit(X[:, np.newaxis], Y)
```

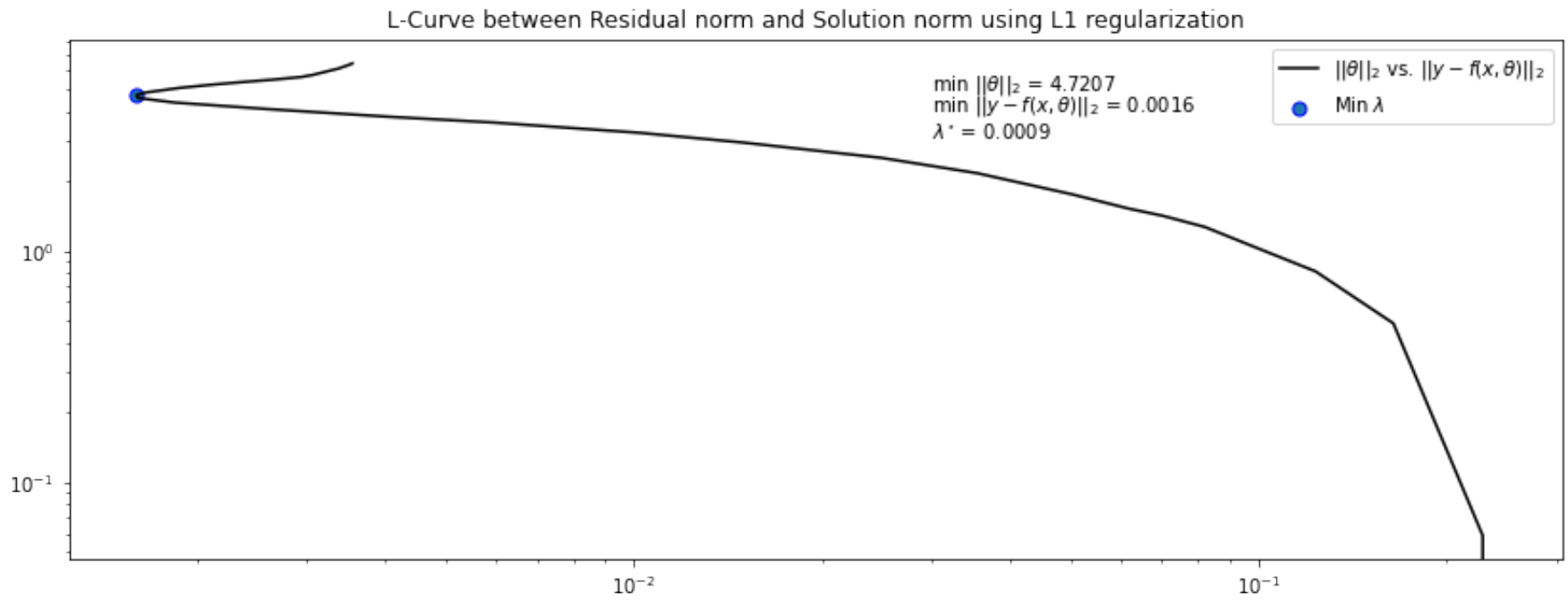
```
        mse.append(((Y_holdout-pipeline3.predict(X_holdout[:, np.newaxis]))**2).mean())

#print(min(mse),mse.index(min(mse)),alphas[mse.index(min(mse))])
print('Optimal Lambda using L1 regularization is:', alphas[mse.index(min(mse))])

# plot MSE wrt
plt.figure(figsize=(14, 5))
plt.plot(alphas, mse)
plt.xscale('log')
plt.xlabel('Regularization parameter (lambda)')
plt.ylabel('Mean Squared Error')
plt.title('Mean Squared Error as a function of the regularization')
plt.axis('tight')

#1d --- L Plot using Residual and Solution Norm
'''1d - Step 3'''

residual_norm = []        #mse or loss function
solution_norm = []
alphas = np.logspace(-5, 1,  )

for a in alphas:
    polynomial_features = PolynomialFeatures(degree= 10, include_bias=False)
    lasso_regression = Lasso(alpha= a)  #sklearn uses alpha instead of lambda
    pipeline4 = Pipeline([("pf", polynomial_features), ("ls", lasso_regression)])
    pipeline4.fit(X[:, np.newaxis], Y)
    residual_norm += [((Y_holdout - pipeline4.predict(X_holdout[:, np.newaxis]))**2).mean()]
    solution_norm += [((abs(lasso_regression.coef_))).sum()]


#print(min(mse),mse.index(min(mse)),alphas[mse.index(min(mse))])

plt.figure(figsize=(14, 5))
plt.loglog(residual_norm, solution_norm,'k-', label = "$|| \\theta ||_{2}$ vs. $|| y - f(x,\\theta) ||_{2}$")
resmin = np.min(residual_norm)
solmin = solution_norm[np.argmin(residual_norm)]
alphamin = alphas[np.argmin(residual_norm)]
plt.scatter(resmin, solmin, edgecolor = 'b', s= 50, label = "Min $\\lambda$")
plt.legend(loc = "best")
print('Optimal Lambda using L-Curve is:', alphamin)
plt.text(0.03,5.,'min $|| \\theta ||_{2}$ = %.4f' % solmin)
plt.text(0.03,4,'min $|| y - f(x,\\theta) ||_{2}$ = %.4f' %resmin)
plt.text(0.03,3,'$\lambda^{\star}$ = %.4f' %alphamin)
plt.legend(loc = "best")
plt.title('L-Curve between Residual norm and Solution norm using L1 regularization')
plt.axis('tight')

'''Lambda for both step 2 and 3 have the same optimal value which is approximately 0.0009 which implies that eith
of these two methods (L1 regularization and L-Curve) can be used to find optimal regularization parameter of lamb
```

```
Optimal Lambda using L1 regularization is: 0.0009102981779915217
Optimal Lambda using L-Curve is: 0.0009102981779915217
```

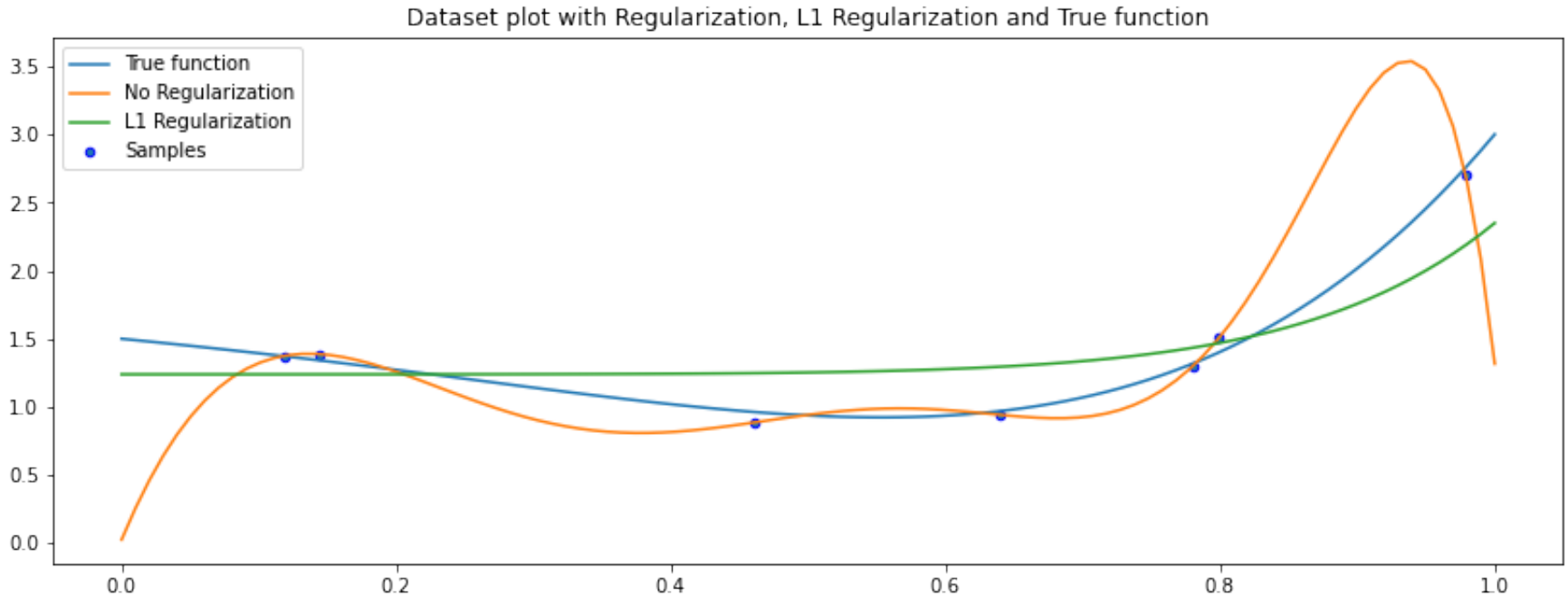
Out[56]:
```
'Lambda for both step 2 and 3 have the same optimal value which is approximately 0.0009 which implies that eithe
r\nof these two methods (L1 regularization and L-Curve) can be used to find optimal regularization parameter of
lambda'
```





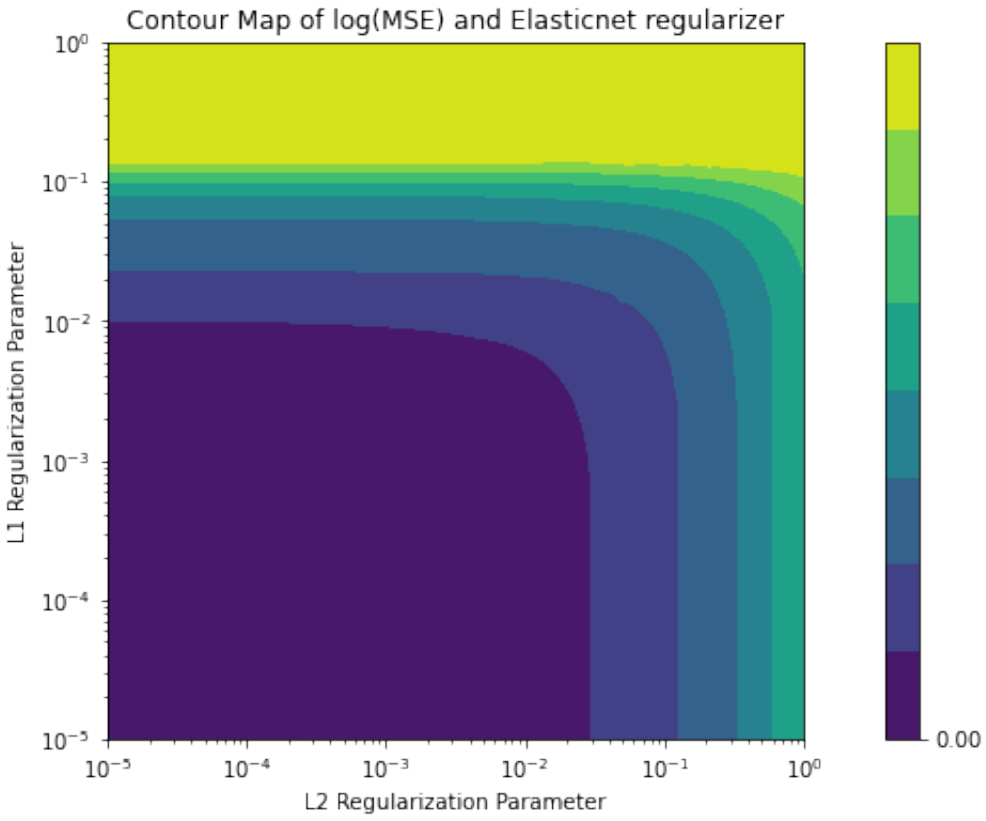


In [21]:
```python
#1e--- Part 1 --- Elasticnet -- Mean square error of the holdout data as a function of lambda and gamma

degrees = [10]
plt.figure(figsize=(14, 5))

for i in range(len(degrees)):
    ax = plt.subplot(1, len(degrees), i + 1)

    # fit a least squares model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    linear_regression = LinearRegression()
    pipeline6 = Pipeline([("pf", polynomial_features), ("lr", linear_regression)])
    pipeline6.fit(X[:, np.newaxis], Y)


    # fit a Elastic Net model
    polynomial_features = PolynomialFeatures(degree=degrees[i], include_bias=False)
    ElasticNet = ElasticNet(alpha=0.1, l1_ratio = 0.5)  #alpha denotes lambda (1-l1_ratio)*alpha and gamma is (l
    pipeline2 = Pipeline([("pf", polynomial_features), ("en", ElasticNet)])
    pipeline2.fit(X[:, np.newaxis], Y)

    # visualize results
    ax.plot(X_test, true_fn(X_test), label="True function")
    ax.plot(X_test, pipeline6.predict(X_test[:, np.newaxis]), label="No Regularization")
    ax.plot(X_test, pipeline2.predict(X_test[:, np.newaxis]), label="L1 Regularization")
    ax.scatter(X, Y, edgecolor='b', s=20, label="Samples")
    ax.legend(loc="best")
    ax.set_title("Dataset plot with Regularization, L1 Regularization and True function")
```



In [9]:
```
#1e--- Part 2 --- Contour plot for ElasticNet - Lambda and Gamma

#degrees = [10]
from sklearn.linear_model import LinearRegression, Ridge, ElasticNet, Lasso, LogisticRegression
a1 = np.logspace(-5,1,50)
a2 = np.linspace(0,1,50)
A1, A2 = np.meshgrid(a1,a2)
plt.figure(figsize = (8,6))

J_grid = np.zeros((50,50))
mse2 = []
ones = np.ones((50,50))
al =[]
l1 = []

for i in range(50):
    for j in range(50):
        polynomial_features = PolynomialFeatures(degree= 10, include_bias=False)
        linear_regression = ElasticNet(alpha = A1[i][j], l1_ratio= A2[i][j])  #alpha denotes lambda (1-l1_ratio*a
        pipeline = Pipeline([("pf", polynomial_features), ("lr", linear_regression)])
        pipeline.fit(X[:, np.newaxis], Y)
        mse2 = ((Y_holdout-pipeline.predict(X_holdout[:, np.newaxis]))**2).mean()
        J_grid[i, j] = mse2

gamma = np.multiply(A1, A2)
f = ones-A2
l = np.multiply(f, A1)

lvls = np.linspace(-8,0,17)
contoursf = plt.contourf(l, gamma, J_grid, level= lvls)
plt.colorbar(contoursf, ticks=lvls, format = '%.2f')

plt.xscale('log')
plt.yscale('log')
plt.axis('square')
plt.xlim((1e-5,1)), plt.xlabel('L2 Regularization Parameter')
plt.ylim((1e-5,1)), plt.ylabel('L1 Regularization Parameter')
plt.title('Contour Map of log(MSE) and Elasticnet regularizer')
#print(gamma[np.argmin(gamma)])
```

Out[9]:
```
Text(0.5, 1.0, 'Contour Map of log(MSE) and Elasticnet regularizer')
```



Solution to 1e Part 3 --- Relationship between L2, L1 and ElasticNet

L1 regularization produces 0.0009 optimal value of lambda vs L2 regularization has 0.0049 value of lambda. Here, lambda is the regularization parameter whose value is optimized for better results. So the relationship of L1 and L2 with ElasticNet is as follows -

elastic_net_penalty = (alpha *l1_penalty) + ((1 – alpha)* l2_penalty)

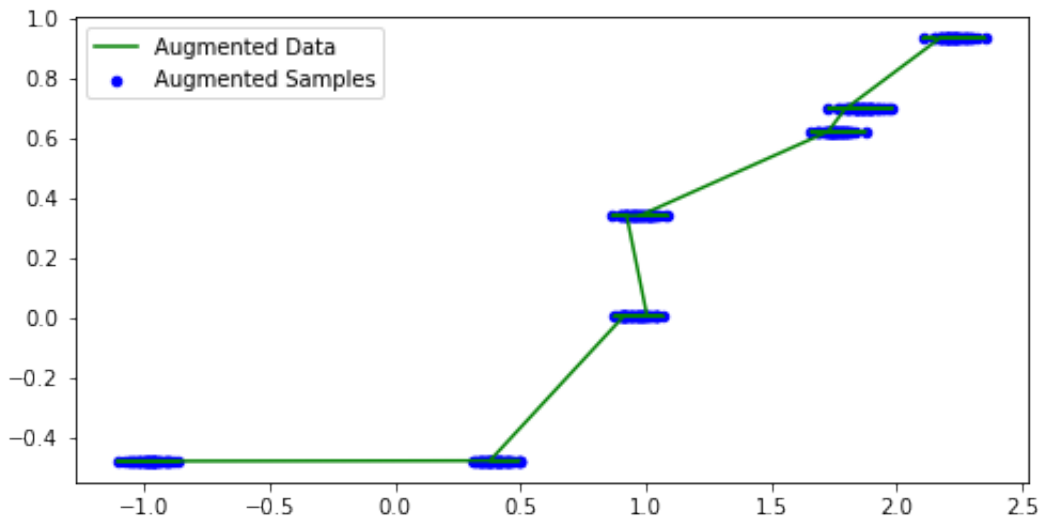
where L1 ratio is the ratio between L1 and L2 penalty, ranging from 0 (ridge) to 1 (lasso). L2 penalty is used to penalize a model based on the minimum sum of the squared coefficient values. Whereas, L1 penalty penalizes a model based on the minimum sum of the absolute values.

Hyperparameter - "alpha" is provided to assign weightage to the L1 penalty and one minus alpha value is used to weight the L2 penalty. Lambda represents the penalty term in the cost function.

L2 regularization forces the weights to decay towards zero (but not exactly zero) but L1 penalize the absolute value of the weights as seen in my values. Unlike L2, the weights may be reduced to zero here. Hence, it is very useful when we are trying to compress our model. Otherwise, we usually prefer L2 over it. The benefit of elastic net is that it gives a balance of both penalties, which results in better performance than a model with either one or the other penalty like L1 or L2.

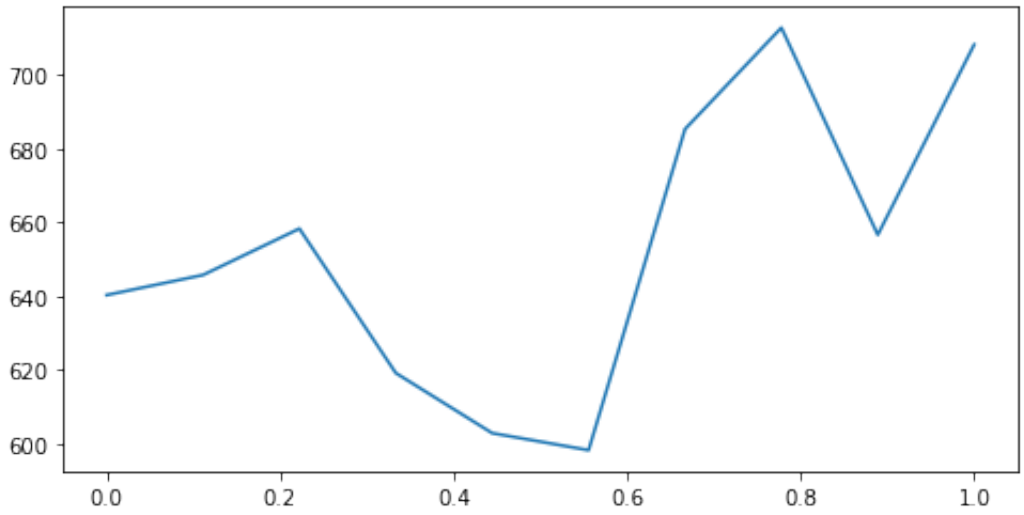
In [57]:
```
#1g -- Data Augmentation Technique -- Bonus
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings('ignore')

import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [8, 4]

import pandas as pd
from sklearn import datasets
np.random.seed(0)

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, ElasticNet, Lasso, LogisticRegression

np.random.seed(0)

# Training samples
n_samples = 7

'''1g -- Part 1'''
X7 = np.sort(np.random.randn(n_samples))
X = np.repeat(X7,50)
#delta = 0.05

final_noise = (np.multiply(np.random.normal(0,1,350),0.05))
X_data_augmented = X + final_noise

Y7 = np.sort(np.random.randn(n_samples))
y_data_augmented = np.repeat(Y7,50)

#print(final_noise)

plt.plot(X_data_augmented,y_data_augmented,'g',label = 'Augmented Data')
plt.scatter(X_data_augmented,y_data_augmented, edgecolor='b', s=20, facecolor = 'b', label="Augmented Samples")
plt.legend()

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

model = LinearRegression()
y_data_augmented = y_data_augmented.reshape(-1,1)
X_data_augmented = X_data_augmented.reshape(-1,1)
model.fit(y_data_augmented,X_data_augmented)
```

Out[57]:
```
LinearRegression()
```



In [59]:
```
'''1g -- Part 2 '''
mse = []
delta = np.linspace(0., 1, 10)
X_test = np.linspace(0., 35., 350)

def mean_squared_error(y, y_pred):
    return 0.5*np.mean((y-y_pred)**2)

for i in delta:
    final_noise = (np.multiply(np.random.normal(0,1,350),i))
    X_data_augmented = X + final_noise
    y_data_augmented = np.repeat(Y7,50)
    model = LinearRegression()
    y_data_augmented = y_data_augmented.reshape(-1,1)
    X_data_augmented = X_data_augmented.reshape(-1,1)
    model.fit(y_data_augmented,X_data_augmented)
    y_data_augmented_pred = model.predict(X_test[:,np.newaxis])
    mse.append((mean_squared_error(y_data_augmented_pred,y_data_augmented)))
plt.plot(delta,mse)

min_delta = delta[np.argmin(mse)]
print(min_delta)
```

```
0.5555555555555556
```



Solution to 1g --- Part 3 ---

A good level of noise is close to 0.55 which provides the minimum Mean Squared Error as shown in the plot above. Too high or too small noise is not good because at first, very small amounts of input noise leads to much more generalization and fault tolerance which will make it difficult for individual data points to fit precisely, and hence will reduce over-fitting. Adding noise makes it easier for the model to learn, resulting in better and faster learning, as discussed in the link attached in the question. But, if you notice, after the minimum Mean Squared Error (delta = 0.55) rises again because too much noise increases the complexity and degrades the performance of the model. So the amount of noise added should be optimized.