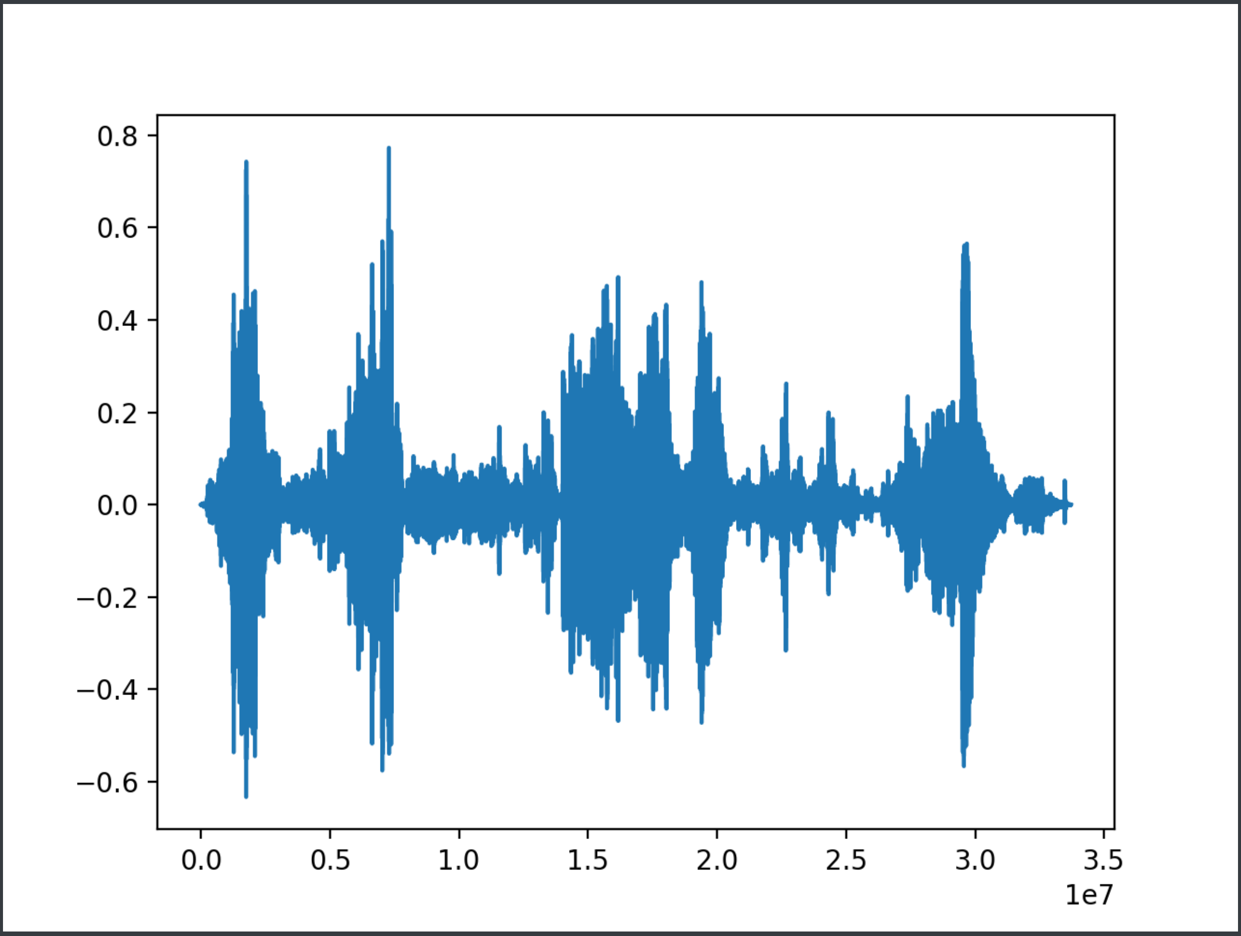
MLPR Assignment 1

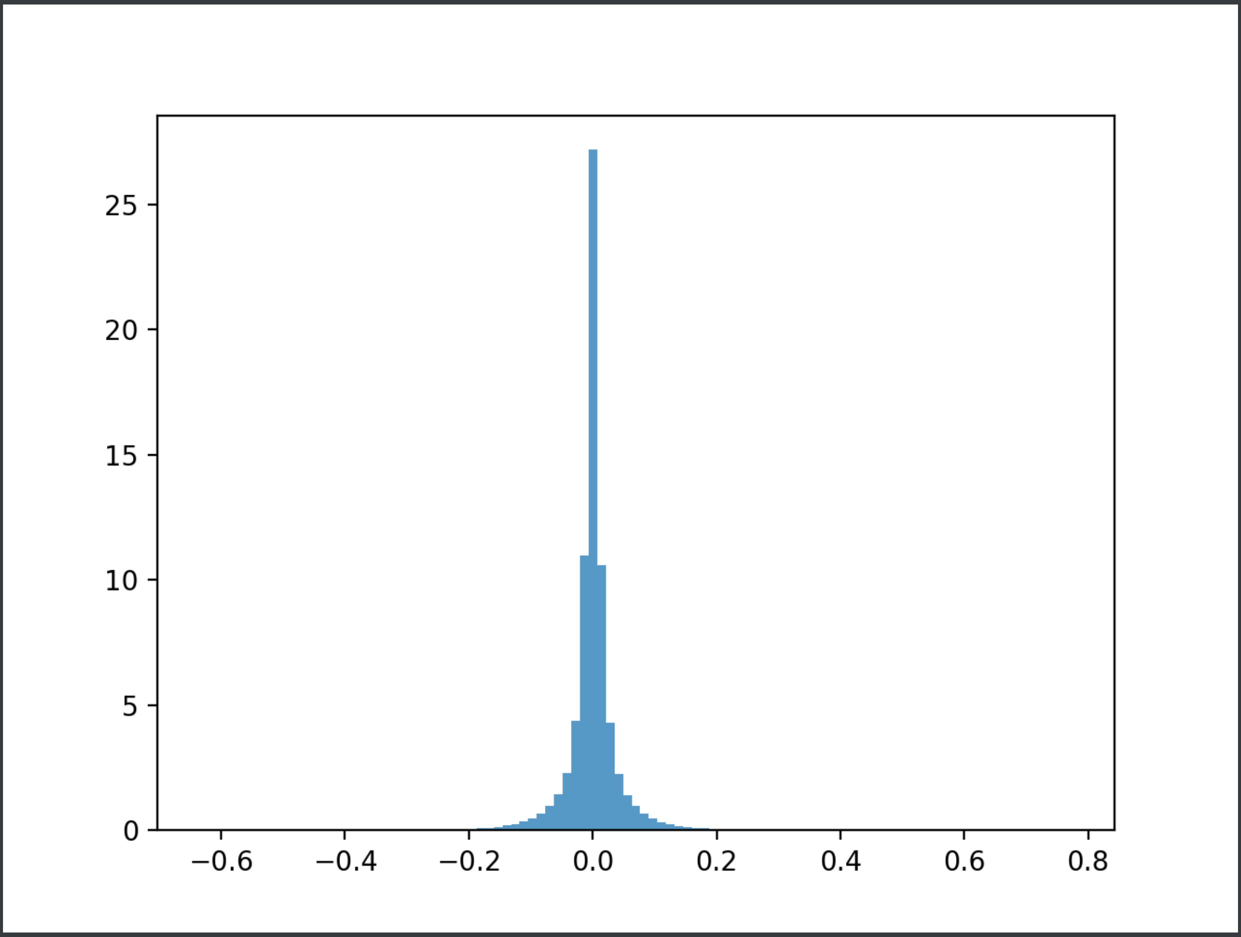
1.

a)

line graph



histogram



The values of aptitudes lie in (-1, 1) and changes slowly as time increases. The distribution of aptitudes values is similar to a Gaussian distribution with mean = 0 and most of the data lies in (-0.2, 0.2).

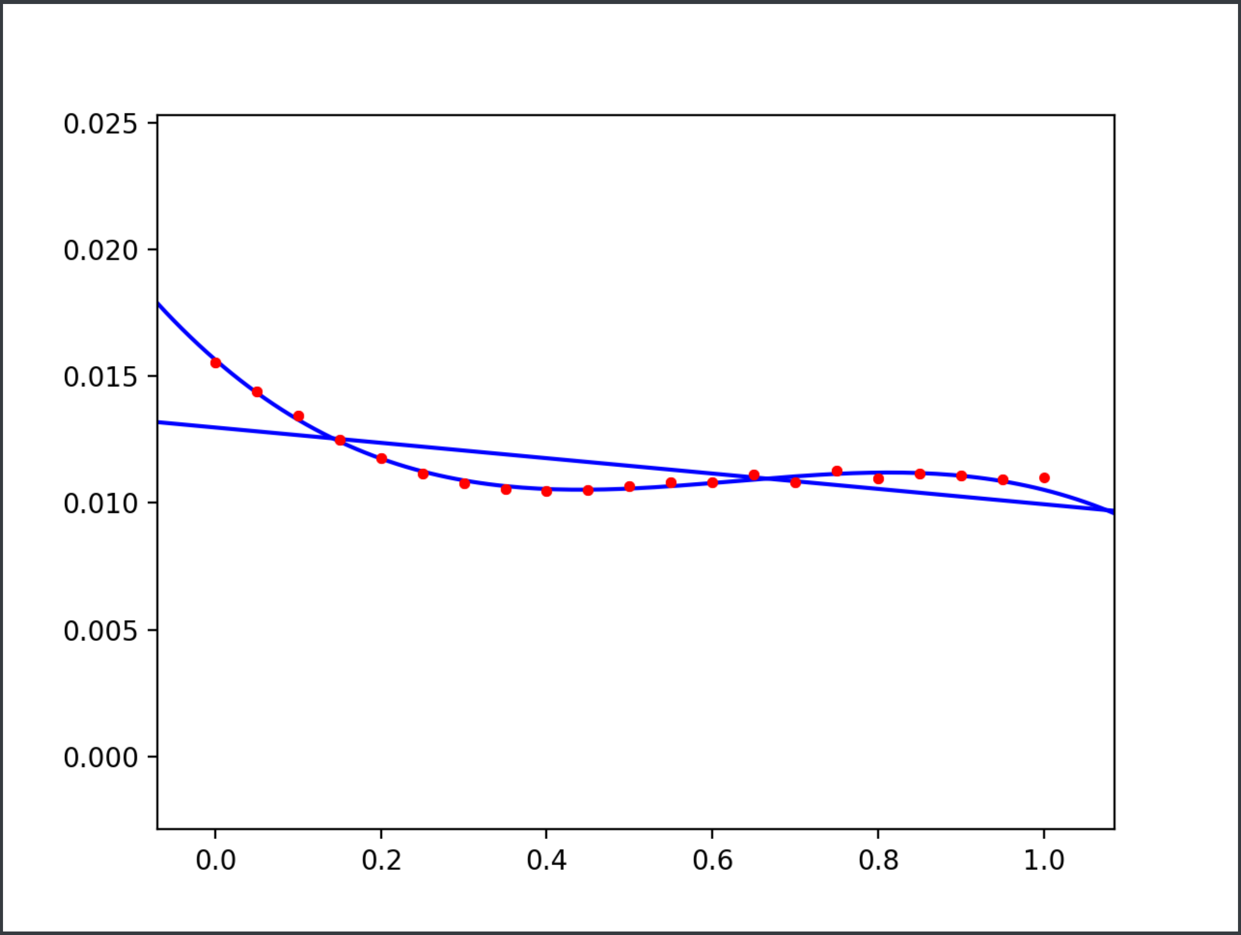
b)

```
#path parameters
dataset_path = '../data/amp_data.mat'

#load amp data
amp_file = scipy.io.loadmat(dataset_path)
amp_dataseq = amp_file['amp_data']
amp_dataset = []

#preprocess
for i in range(len(amp_dataseq) // 21):
    amp_dataset.append(amp_dataseq[i*21 : (i+1)*21])
amp_mat = np.array(amp_dataset).reshape(-1, 21)
#print(amp_mat.shape)
np.random.shuffle(amp_mat)

X_shuf_train = amp_mat[:math.ceil(0.7*amp_mat.shape[0]), :20]
```

```
X_shuf_val =
amp_mat[math.ceil(0.7*amp_mat.shape[0]):math.ceil(0.85*amp_mat.shape[0
]), :20]
X_shuf_test = amp_mat[math.ceil(0.85*amp_mat.shape[0]):, :20]

y_shuf_train = amp_mat[:math.ceil(0.7*amp_mat.shape[0]), 20]
y_shuf_val =
amp_mat[math.ceil(0.7*amp_mat.shape[0]):math.ceil(0.85*amp_mat.shape[0
]), 20]
y_shuf_test = amp_mat[math.ceil(0.85*amp_mat.shape[0]):, 20]
```

2.

a)



```
def linear_transform_1d(X):
    no_bias_linear = np.reshape(np.array(X), (-1, 1))
    #print(no_bias_linear.shape)
    return np.concatenate((no_bias_linear,
np.ones((no_bias_linear.shape[0], 1))), axis = 1)

def polynomial_transform_1d(X, order):
    quad_n = [np.array(np.power(X, i)).reshape(-1, 1) for i in
range(1, order + 1)]
    quad_n.append(np.ones((len(X), 1)))
    return np.concatenate(tuple(quad_n), axis = 1)

def fit_plot(X, input, yy, transform_function, grid_size = 0.01, order
= None):
    yy = np.reshape(np.array(yy), (-1, 1))
    W = np.linalg.lstsq(X, yy, rcond = None)[0]
    #yy_pred = np.dot(X, W).tolist()

    x_grid = np.arange(-2, 2, grid_size)
    y_grid = np.dot(transform_function(x_grid, order = order),
W).tolist() if order != None else np.dot(transform_function(x_grid),
W).tolist()
    #plt.clf()
    plt.plot(x_grid, y_grid, 'b-')
    plt.plot(input, yy, 'r.')
    #plt.show()

time_steps = [i * 0.05 for i in range(20)]
linear_transform = linear_transform_1d(time_steps)
polynomial_transform = polynomial_transform_1d(time_steps, order = 4)

plt.plot([1.0], [y_shuf_train[0]], 'r.')
fit_plot(polynomial_transform, time_steps, X_shuf_train[0],
polynomial_transform_1d, order = 4)

fit_plot(linear_transform, time_steps, X_shuf_train[0],
linear_transform_1d)
plt.show()
```

b)

The true function is not linear so straight line cannot fit 20 points, but since the time slice and the variance between two adjacent aptitudes is very small, gradient between any three adjacent points is approximately the same.

quartic model has more parameters and higher capacity than the straight line and thus is able to fit the data better.

c)

5-order polynomial model with 20 context length should be best for prediction. This guess is because when we use quartic polynomial some training data cannot be fitted well and the model is undercutting, but when we use 6-order polynomial, the test point can not be fitted well and the model suffers from overfitting.