ECE655 Project 01

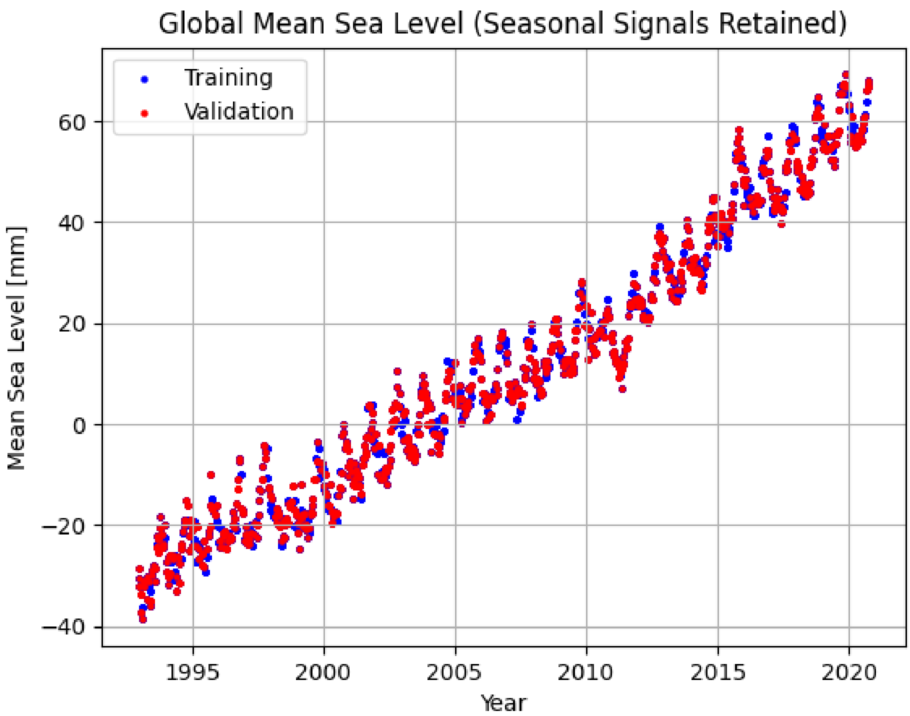
Author: Stewart Schuler

Due: 09/12/2025

Contents

1 Part A

The dataset chosen for this project compares the mean global sea level per year. That data is sourced from the *University of Colorado*[1].



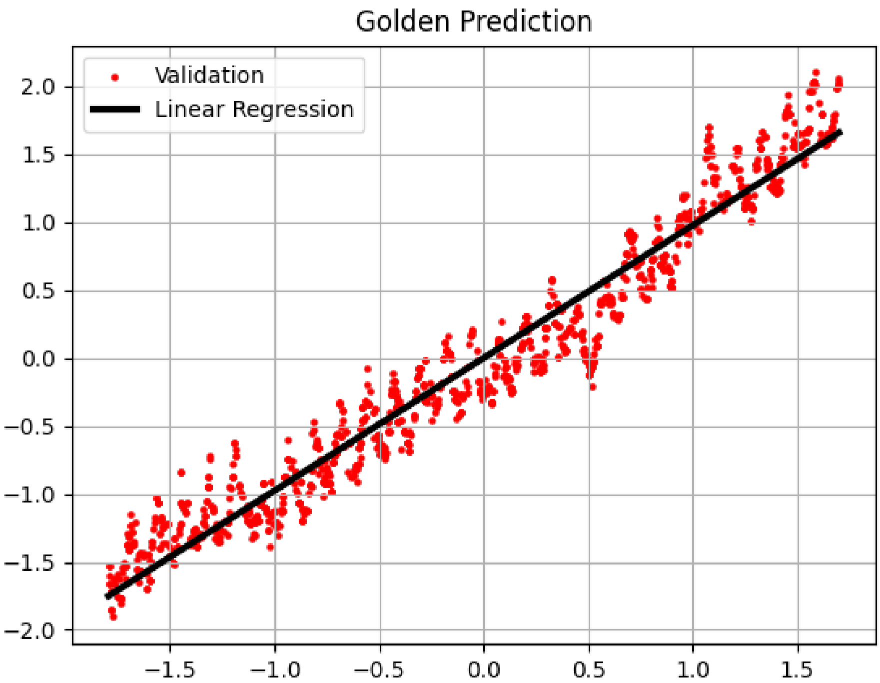
It can be seen there is a clear linear trend between year and mean sea level. The dataset was then partitioned into an 80/20 split of training and validation data. Applying linear regression to the (normalized) training data learned a line with an $b = 3.8E - 14$ and $w = 0.9758$.

```
lr = LinearRegression();
lr.fit(x_train, y_train);
b_lr , w_lr = lr.intercept_[0], lr.coef_[0][0]
y_pred = b_lr + x_val*w_lr;
mse = (( y_pred-y_val)**2).mean()
```

Listing 1: part_a.py

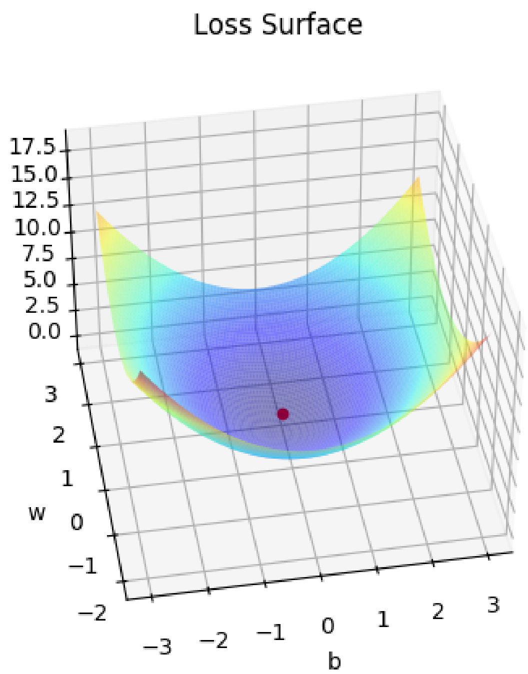
The regression line was found to have a MSE of 0.0489

[1] https://sealevel.colorado.edu/data/2020rel1-0


Golden Prediction
Validation
Linear Regression
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

2 Part B

A loss surface was created by computing the MSE for a grid of $10,000$ b and w pairs. Both variables were varied by ± 3 from the golden value found in Section 1.



The above surface was generated using the following code.

```
b_depth = 3;
w_depth = 3;
b_range = np.linspace( b_lr - b_depth, b_lr + b_depth, 100 );
w_range = np.linspace( w_lr - w_depth, b_lr + w_depth, 100 );
b_surf, w_surf = np.meshgrid( b_range, w_range );
y_surf = np.apply_along_axis( func1d=lambda x: b_surf + w_surf*x,
                              axis=1,
                              arr=x_train );

all_labels = y_train.reshape(-1,1,1);
all_errors = (y_surf-all_labels);
```

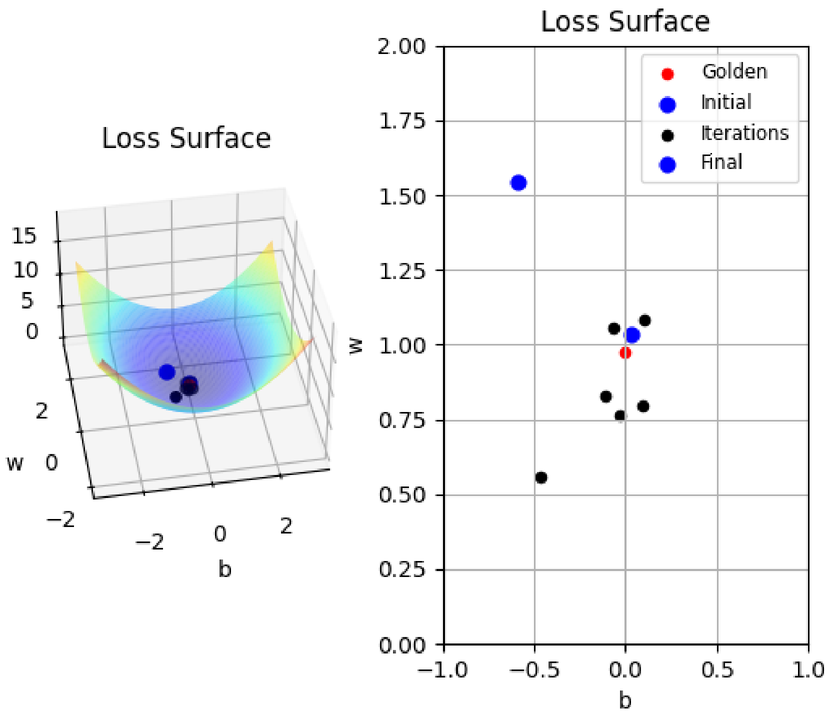
```
loss_surf = (all_errors**2).mean(axis=0);
```

Listing 2: part_b.py

3 Part C/D

A naive algorithm was applied to the data set, in which a random value for b and w is chosen. For each pair the MSE is computed, if it is lower than the previous best pair, the pair is saved and the best MSE is updated. This algorithm was let to run for 1000 iterations. Upon completion the best pair tested was $b = 0.0314$ and $w = 1.035$ which gave an MSE of 0.05487. In comparison to the "golden" values found in Section 1 it can be seen that the algorithm was able to produce similar values.

The loss surface is again plotted, marking the b and w pairs which improved the MSE it can be seen then each improvement is closer to the "golden" value. However due to the random nature of the algorithm the iterations do not travel along any predictable path.




```
iterations = [];
best_mse = float('inf');
N_ITR = 1000;
for itr in range(N_ITR):
    b_rand = np.random.randn();
```

```python
    w_rand = np.random.randn();
    y_pred = b_rand + x_val*w_rand;
    mse = (( y_pred-y_val)**2).mean()
    if( mse < best_mse ):
        iterations.append([b_rand, w_rand, float(mse)]);
        best_mse = mse;
```

Listing 3: part_c.py

4 Part E

Comparing the naive approach to gradient decent it can be seen that both, given enough iterations, will converge towards the "golden" variable values. However the naive approach will traverse a non-deterministic path, unlike gradient decent. Additionally, gradient decent (assuming reasonable hyper parameters) will improve it's MSE after each iteration. As was shown in the naive approach, only a small percentage of the iterations actually improve the MSE.

Lastly consider the impact of increasing the number of iterations when running the naive algorithm. The below table shows the algorithm's results for four different iteration numbers. It can be seen that as the number of iterations increase the MSE decreases, however this is only the *probabilistically likely* outcome of increasing iterations and not a guarantee due to the inherent randomness of the algorithm. Furthermore, it can be seen that the final computed values of b and w are not always closer to the "golden" values than their predecessors with fewer iterations, even if the overall MSE is lower.

Iterations	# of Hits	# of Misses	Final b	Final w	Final MSE
1000	4	996	0.02445	1.024	0.05297
2000	8	1992	0.03465	0.9427	0.05059
5000	8	4992	-0.01179	0.9886	0.04949
10000	11	9989	0.002239	0.9824	0.04911

```python
for N_ITR in [ 1000, 2000, 5000, 10000 ]:
    iterations = [];
    best_mse = float('inf');
    for itr in range(N_ITR):
        b_rand = np.random.randn();
        w_rand = np.random.randn();
        y_pred = b_rand + x_val*w_rand;
        mse = (( y_pred-y_val)**2).mean()
        if( mse < best_mse ):
            iterations.append([b_rand, w_rand, float(mse)]);
            best_mse = mse;

    iterations = np.array(iterations);
    print(f"[Naive-{N_ITR}] b: {iterations[-1][0]:.4}, w: {iterations[-1][1]:.4}, mse: {iterations[-1][2]:.4}\n\tHits: {iterations.shape[0]} Missed: {N_ITR-iterations.shape[0]}");
```

Listing 4: part_e.py

Appendix A - Complete Source Code

```python
import numpy as np
import matplotlib.pyplot as plt
#plt.style.use('fivethirtyeight')
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

FIGURES_DIR = "../report/figures/";

np.random.seed(1140);

dataset = np.loadtxt("../gmsl_2020rel1_seasons_retained.txt");
N = dataset.shape[0];
idx = np.arange(N);
np.random.shuffle(idx);

train_idx = idx[:int(N*0.8)];
val_idx   = idx[int(N*0.8):];
x_train, y_train = dataset[train_idx, 0], dataset[train_idx, 1];
x_val,   y_val   = dataset[  val_idx, 0], dataset[  val_idx, 1];
x_train = x_train.reshape(-1,1);
y_train = y_train.reshape(-1,1);
x_val = x_val.reshape(-1,1);
y_val = y_val.reshape(-1,1);

plt.figure()
plt.scatter( x_train, y_train, color='blue', label='Training', s=5 );
plt.scatter( x_val,   y_val,   color='red',  label='Validation', s=5 );
plt.title("Global Mean Sea Level (Seasonal Signals Retained)");
plt.xlabel("Year");
plt.ylabel("Mean Sea Level [mm]");
plt.legend();
plt.grid(True);
plt.savefig(f"{FIGURES_DIR}/dataset.png");

# Normalize dataset base on training data
ss_x = StandardScaler();
ss_x.fit( x_train );
x_train = ss_x.transform(x_train)
x_val   = ss_x.transform(x_val  )

```

```
ss_y = StandardScaler();
ss_y.fit( y_train );
y_train = ss_y.transform(y_train)
y_val   = ss_y.transform(y_val  )



# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Part A
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

lr = LinearRegression();
lr.fit(x_train, y_train);
b_lr , w_lr = lr.intercept_[0], lr.coef_[0][0]
y_pred = b_lr + x_val*w_lr;
mse = (( y_pred-y_val)**2).mean()
print(f"[LR] b: {b_lr:.4} w: {w_lr:.4} msg: {mse:.4}");

x_lr = np.linspace( min(x_train), max(x_train), 10 );
y_lr = b_lr + (x_lr*w_lr);

plt.figure()
#plt.scatter( x_train, y_train, color='blue', label='Training',   s=5 );
plt.scatter( x_val, y_val, color='red', label='Validation', s=5 );
plt.plot( x_lr, y_lr, color='black', label='Linear Regression', linewidth=3 );
plt.title("Golden Prediction");
plt.grid(True);
plt.legend();
plt.savefig(f"{FIGURES_DIR}/linear_regression.png");

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Part B
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

b_depth = 3;
w_depth = 3;
b_range = np.linspace( b_lr - b_depth, b_lr + b_depth, 100 );
w_range = np.linspace( w_lr - w_depth, b_lr + w_depth, 100 );
b_surf, w_surf = np.meshgrid( b_range, w_range );
y_surf = np.apply_along_axis( func1d=lambda x: b_surf + w_surf*x,
                              axis=1,
                              arr=x_train );

all_labels = y_train.reshape(-1,1,1);
all_errors = (y_surf-all_labels);
```

```python
loss_surf = (all_errors**2).mean(axis=0);

fig = plt.figure();
ax = fig.add_subplot( 111, projection='3d')
ax.plot_surface( b_surf, w_surf, loss_surf, rstride=1,
    cstride=1, alpha=.5, cmap=plt.cm.jet, linewidth=0,
    antialiased=True);
ax.scatter(b_lr, w_lr, color='red', s=20);
ax.set_xlabel('b');
ax.set_ylabel('w');
ax.set_title('Loss Surface');
ax.view_init(40, 260)
plt.savefig(f"{FIGURES_DIR}/loss_surface.png");

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Part C
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
iterations = [];
best_mse = float('inf');
N_ITR = 1000;
for itr in range(N_ITR):
    b_rand = np.random.randn();
    w_rand = np.random.randn();
    y_pred = b_rand + x_val*w_rand;
    mse = (( y_pred-y_val)**2).mean()
    if( mse < best_mse ):
        iterations.append([b_rand, w_rand, float(mse)]);
        best_mse = mse;

iterations = np.array(iterations);
print(f"[Naive] b: {iterations[-1][0]:.4}, w: {iterations
    [-1][1]:.4}, mse: {iterations[-1][2]:.4}");

# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Part D
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
fig = plt.figure();
ax = fig.add_subplot( 121, projection='3d')
ax.plot_surface( b_surf, w_surf, loss_surf, rstride=1,
    cstride=1, alpha=.5, cmap=plt.cm.jet, linewidth=0,
    antialiased=True);
ax.scatter(b_lr, w_lr, color='red', s=20, label="Golden");
ax.scatter(iterations[0][0], iterations[0][1], color='blue',
    s=40, label="Initial");
ax.scatter(iterations[1:-1:,0], iterations[1:-1:,1,], color='
    black', s=20, label="Iterations");
ax.scatter(iterations[-1][0], iterations[-1][1], color='blue'
    , s=40, label="Final");
ax.set_xlabel('b');
```

```python
ax.set_ylabel('w');
ax.set_title('Loss Surface');
ax.view_init(40, 260);

ax = fig.add_subplot( 122 )
# ax.plot_surface( b_surf, w_surf, loss_surf, rstride=1,
    cstride=1, alpha=.5, cmap=plt.cm.jet, linewidth=0,
    antialiased=True);
ax.scatter(b_lr, w_lr, color='red', s=20, label="Golden");
ax.scatter(iterations[0][0], iterations[0][1], color='blue',
    s=40, label="Initial");
ax.scatter(iterations[1:-1:,0], iterations[1:-1:,1,], color='
    black', s=20, label="Iterations");
ax.scatter(iterations[-1][0], iterations[-1][1], color='blue'
    , s=40, label="Final");
ax.set_xlabel('b');
ax.set_ylabel('w');
ax.set_xlim(-1, 1 );
ax.set_ylim(0, 2);
ax.set_title('Loss Surface');
plt.grid(True);
plt.legend(fontsize="small");
plt.savefig(f"{FIGURES_DIR}/loss_surface_naive.png");




# ~~~~~~~~~~~~~~~~~~~~~~~~~~~
# Part E
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~

for N_ITR in [ 1000, 2000, 5000, 10000 ]:
    iterations = [];
    best_mse = float('inf');
    for itr in range(N_ITR):
        b_rand = np.random.randn();
        w_rand = np.random.randn();
        y_pred = b_rand + x_val*w_rand;
        mse = (( y_pred-y_val)**2).mean()
        if( mse < best_mse ):
            iterations.append([b_rand, w_rand, float(mse)]);
            best_mse = mse;

    iterations = np.array(iterations);
    print(f"[Naive-{N_ITR}] b: {iterations[-1][0]:.4}, w: {
   iterations[-1][1]:.4}, mse: {iterations[-1][2]:.4}\n\tHits
   : {iterations.shape[0]} Missed: {N_ITR-iterations.shape
   [0]}");

```

```
plt.show();
```
Listing 5: complete.py