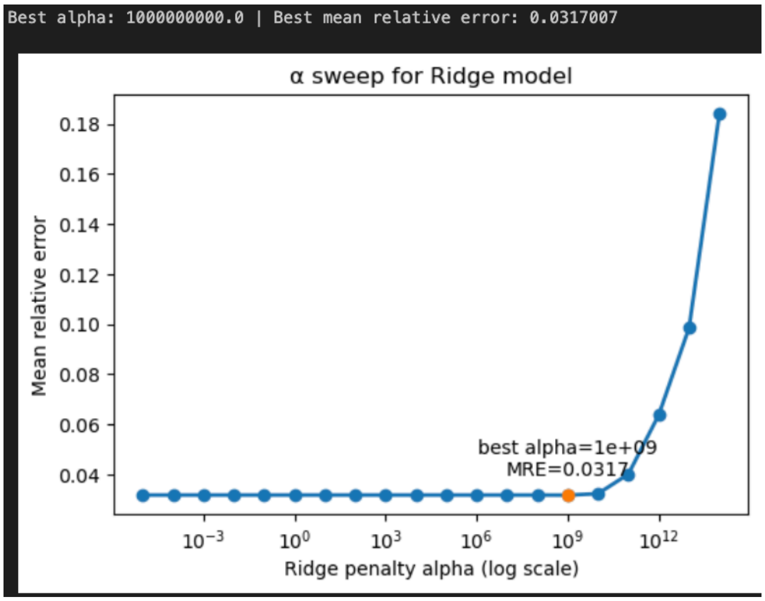
<u>Census Prediction Minproject Report (updated)</u>

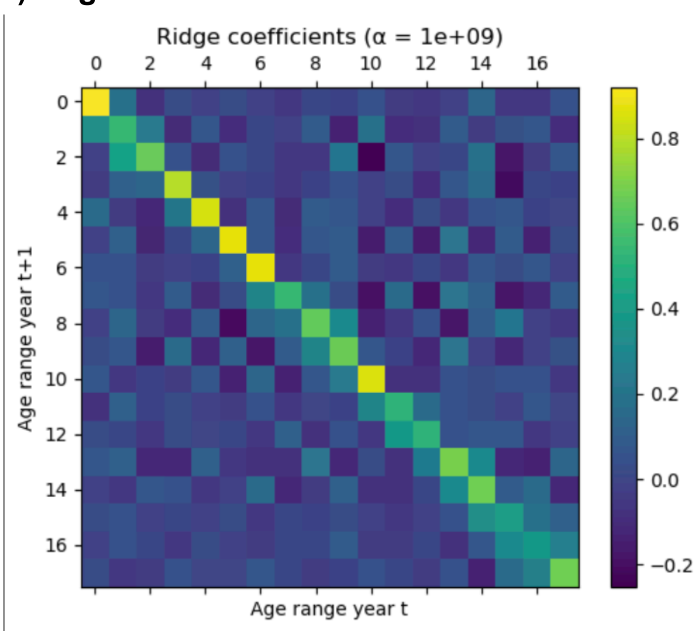
1) Optimal alpha and test mean relative error (only using power of 10's):

- **Alpha = 10^9**
- **Test mean relative error: 0.0317007**



(fig 1)

2) Regression coefficient visualization:

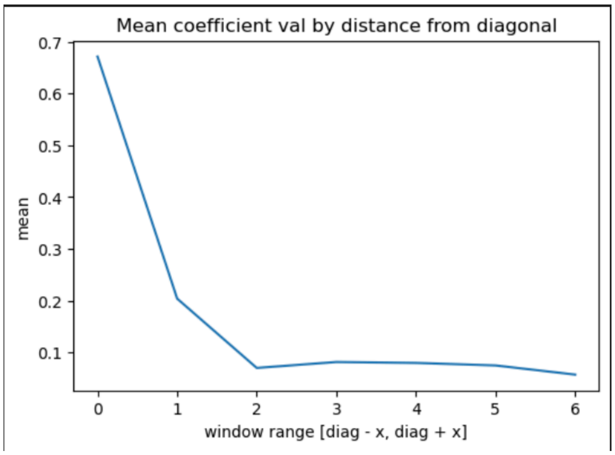


(fig 2)

3) Explanation:

The 18x18 grid comparison represents a transition matrix from year **t** to year **t+1**. There is an obvious pattern of concentration on the diagonals, which represents a similar coefficient from one year to the next.

I made an additional graph to quantify this pattern:



(fig 3)

This image plots the mean values on different windows from the diagonal of the graph in *(fig 2)*. The average mean absolute value along the diagonal is 0.6714. Taking 1 step out in both directions [diag - 1, diag + 1], the value shrinks to 0.204, and taking an additional step [diag - 2, diag + 2], we get an even lower 0.070. This supports the pattern of concentration around the diagonal band.

This pattern makes logical sense, as with 5 year age brackets and 1 year step size, most people would remain within their bracket from year **t** to **t+1**. There is a weaker diagonal concentration at the beginning and end, which would represent more variability from year to year in birth and death rates.

4) Code submission (also attached in email):

https://github.com/cpdiprete/Census_Statistical_Analysis

5) Previous results:

I previously got an optimal alpha value of

- **Alpha = 10^{-6}**
- **Best MRE = 0.0304648**

After my submission I was instructed to filter out the 2 non states (District of Columbia and Puerto Rico). Here are the edits I made to my code to perform this:

1) Filtering out DC and Puerto Rico. This leaves me with 50 states as opposed to 52.

```
def total_cols(df):
    total_estimate_columns = []
    for col in df.columns:
        if col.endswith("!!Total!!Estimate") and not (col.startswith("District of Columbia") or col.startswith("Puerto Rico")):
            total_estimate_columns.append(col)
    return total_estimate_columns
```

2) Removing normalization (new code)

```
def make_pairs(train_array, test_array):
    x_train = train_array[:9] # years 0-8
    y_train = train_array[1:10] # years 1-9

    x_train_reshaped = x_train.reshape(9*50, 18)
    y_train_reshaped = y_train.reshape(9*50, 18)

    x_test_reshaped = test_array[:2].reshape(2*50, 18)
    y_test_reshaped = test_array[1:3].reshape(2*50, 18)
    return x_train_reshaped, y_train_reshaped, x_test_reshaped, y_test_reshaped
```

Old/revised code

```
def make_pairs_normalized(train_array, test_array):
    # normalize to proportions
    train_sum = np.clip(train_array.sum(axis=2, keepdims=True), 1e-12, a_max=None)
    test_sum = np.clip(test_array.sum(axis=2, keepdims=True),  1e-12, a_max=None)
    train_normalized = train_array / train_sum
    test_normalized = test_array / test_sum

    x_train = train_normalized[:9] # years 0-8
    y_train = train_normalized[1:10] # years 1-9
    x_train_reshaped = x_train.reshape(9*50, 18)
    y_train_reshaped = y_train.reshape(9*50, 18)

    x_test_reshaped = test_normalized[:2].reshape(2*50, 18)
    y_test_reshaped = test_normalized[1:3].reshape(2*50, 18)

    return x_train_reshaped, y_train_reshaped, x_test_reshaped, y_test_reshaped
```

The idea behind normalizing the populations was to get a better representation of the changes in overall shape between years. Without normalizing, a state like California or Texas impacts the model much more than a less populated one.

But since the goal is to predict the next year's total counts given the previous year's, it makes sense to not normalize.