CSPB FInal Project - Least Squares Analysis
Max score 100/100 for Course Final Project

The goal of this project is to use your Linear Algebra Knowledge and analytical skills to work through three applications of Least Squares. We are not looking for complete technical mastery or skill, but the ability to to synthesize, review and make sense of new material - as you will often be called to do in your professional career.

You may work with a partner. Both must submit an assignment and you must include an additional page about how you worked together and who did what.

Please only attempt questions you intend to answer fully. There will not be partial credit for incomplete responses - ie. don't attempt to do half of each example for a few points.

- Complete as many to as few questions as needed for the score you are aiming for.
- As per previous Study Guides, you may use solutions and work will be graded on how well solutions are communicated.
- All previous Study Guide instructions apply.
- In particular, just using the given solutions without additional insights will receive a 0.
- Use examples and additional explanations for full credit. Sloppy, rushed, or work that does not show evidence of engagement with the material will not receive full credit.

For this Project, you will review Chapters 12/13/14.

Name: Christopher Taylor

1. (40 Points)

Summarize the basic ideas of the Advertising and Illumination examples on pages 234 - 239 in your own words using the given information.

- Reproduce the data with code.
- A 2 -3 page summary rewriting the examples with the given images.

Advertising:

In the context of advertising, businesses often target different demographic groups to optimize/maximize their reach. If we use ***m*** to represent the demographic groups, each group has a desired number of impressions (views) specified by a vector $\boldsymbol{v}^{des}$, with only positive entries. To achieve the goals set for these views for different demographic groups, advertising is purchased across ***n*** different channels, with the amounts spent given by an n-vector ***s*** (where entries are nonnegative). The effectiveness of each channel is represented by an **m x n** matrix ***R***, where $\boldsymbol{R_{ij}}$ shows the number of views in group ***i*** per dollar spent on channel ***j***. The total views for a specific channel across demographic groups, represented by m-vector ***v***, is calculated by ***v = Rs***.
We need to determine the spending vector ***s*** that minimizes the difference between the actual views (***v***) and desired views ($\boldsymbol{v}^{des}$). To do so, we need to solve the least squares problem: minimize $\boldsymbol{||Rs - v^{des}||^2}$.
In the case where ***n = 3*** advertising channels and ***m = 10*** demographic groups. The effectiveness matrix ***R*** would be this:

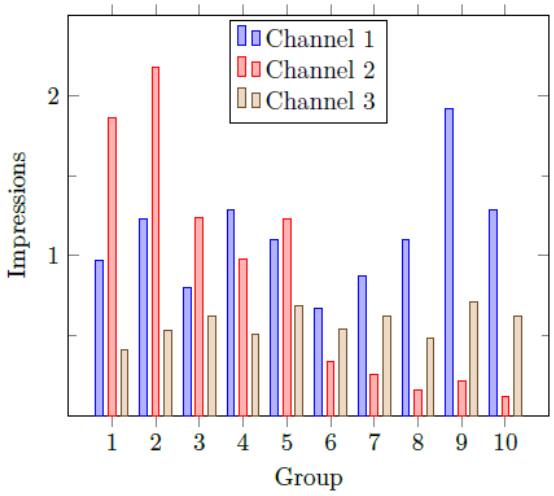
R =
0.97 1.86 0.41
1.23 2.18 0.53
0.80 1.24 0.62
1.29 0.98 0.51
1.10 1.23 0.69
0.67 0.34 0.54
0.87 0.26 0.62
1.10 0.16 0.48
1.92 0.22 0.71
1.29 0.12 0.62

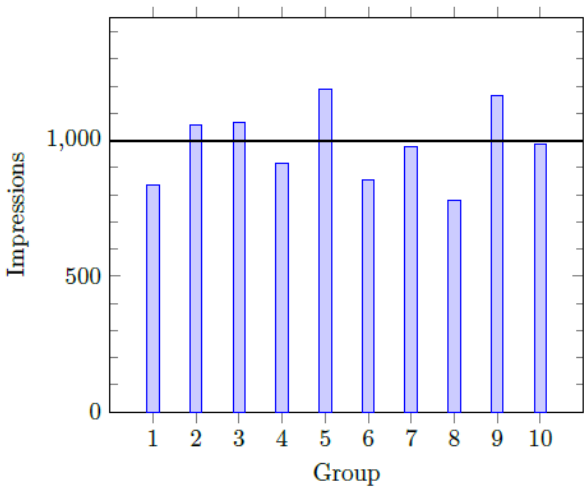
Units for ***R*** are in thousands of views per dollar. The entries of ***R*** show a significant variance, indicating different efficiencies of the advertising channels. The goal is to hit one million people in each demographic group, represented by $v^{des} = (10^3)\mathbf{1}$.
Using least squares, the optimal spending vector ***s*** is: $\hat{\boldsymbol{s}}$ = (62, 100, 1443)
This allocation achieves the desired impressions (1 million per group) with a Root Mean Square error of 132, 13.2% of the target values. The distribution of views across the demographic groups shows the variance in effectiveness and reach of the different channels used.

Python code for Advertising:

```
import numpy as np

R = np.array([
    [0.97, 1.86, 0.41],
    [1.23, 2.18, 0.53],
    [0.80, 1.24, 0.62],
    [1.29, 0.98, 0.51],
    [1.10, 1.23, 0.69],
    [0.67, 0.34, 0.54],
```

```
    [0.87, 0.26, 0.62],
    [1.10, 0.16, 0.48],
    [1.92, 0.22, 0.71],
    [1.29, 0.12, 0.62]
])

vdes = 1000 * np.ones(10)
s, residuals, rank, singular_values = np.linalg.lstsq(R, vdes, rcond=None)
v = R @ s
rmse = np.sqrt(np.mean((v - vdes) ** 2))
```

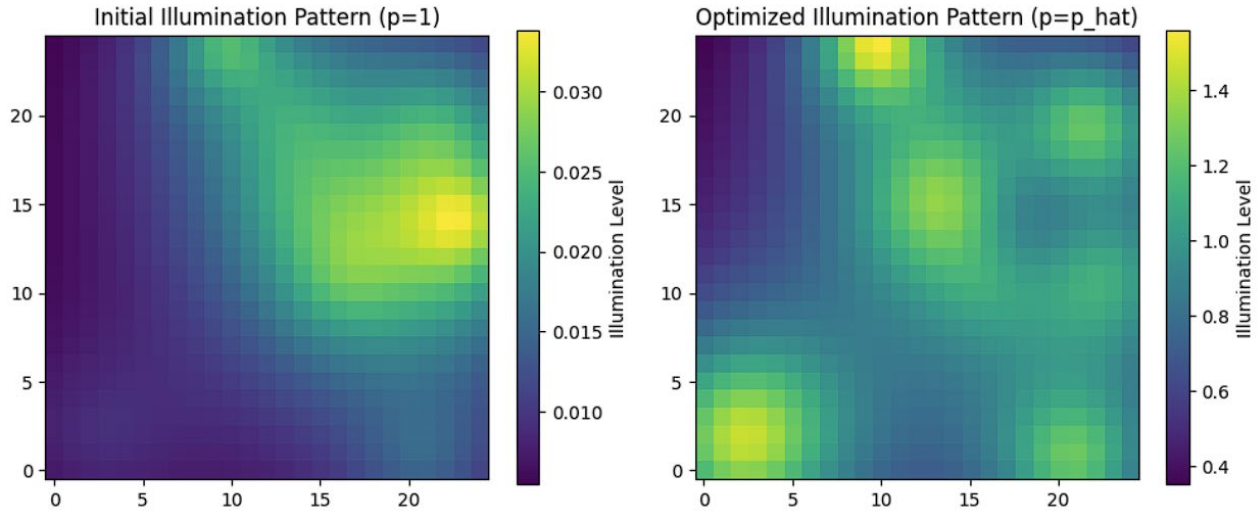




Illumination:
In lighting design, the goal is often to achieve a specific illumination pattern across a designated area using a set of ***n*** lamps. This area is divided into ***m*** regions or pixels, with the illumination in region ***i*** denoted by l_i. Vector ***l*** represents the lighting levels across all our regions. Each lamp operates at power level p_i, and the vector ***p*** shows the power levels of all the lamps.
Vector ***l*** is a linear function, ***l = Ap***, of the lamp powers, where ***A*** is an ***m x n*** matrix. Each column of ***A*** represents the illumination pattern of a single lamp when operating at full power, with all the other lamps turned off. The rows of ***A*** represent the sensitivity of each pixel to the power levels of the lamps.
We are trying to find the lamp powers that produce a desired illumination, l^{des}. For instance, $l^{des} = \alpha \mathbf{1}$ shows an equal lighting level α across all our regions. Least squares is used to reduce the sum of squared deviations from the target lighting level, $\|Ap - l^{des}\|^2$.
The least squares solution for the lamp power levels:
$\hat{p} = A^{\dagger} l^{des} = (A^T A)^{-1} A^T l^{des}$ where $A^{\dagger}$ is the pseudoinverse of ***A***.
If we consider an area divided into a 25×25 grid, resulting in ***m*** = 625 where ten lamps are placed at heights between 3m and 6m. If ***dij*** is the 3D distance between the pixel center and the lamp our lighting level wanes following an inverse square law: ***Aij*** is proportional to dij^{-2}. We scale ***A*** so when all lamps have power one, our mean lighting level is also one.
The target is equal illumination across the area: $l^{des} = 1$. Initially, setting p = 1 gives us an illumination with RMSE of 0.24.
With least squares, optimal lamp powers are:
$\hat{p}$ = (1.46, 0.79, 2.97, 0.74, 0.08, 0.21, 0.21, 2.05, 0.91, 1.47)
Our RMSE decreases to 0.14, much lower than the original RMSE, with most values close to our target level of 1.

Python code for Illumination:

```
import numpy as np

grid_size = 25
m = grid_size*grid_size
n = 10

np.random.seed(0)
lamp_positions = np.random.rand(n, 2) * grid_size
lamp_heights = 3 + 3 * np.random.rand(n)

A = np.zeros((m, n))
for i in range(m):
    for j in range(n):
        dx = pixel_positions[i, 0] - lamp_positions[j, 0]
        dy = pixel_positions[i, 1] - lamp_positions[j, 1]
        dz = lamp_heights[j]
        distance = np.sqrt(dx**2 + dy**2 + dz**2)
        A[i, j] = distance ** -2

A /= np.mean(np.sum(A, axis=0))

Ides = np.ones(m)  # Uniform illumination with value 1
p, residuals, rank, singular_values = np.linalg.lstsq(A, Ides, rcond=None)
I = A @ p
rmse = np.sqrt(np.mean((I - Ides) ** 2))
```



2. (30 points) Review Chapter 13 and pick one of 13.2, 13.3, or **13.11** to summarize, demonstrate or illuminate (you may use the given solutions)

- A 2 -3 page summary rewriting the examples with the given images.

To evaluate performance of least squares models, it's important to consider both the training and test root mean square (RMS) prediction errors because they show how well the model fits the training data and how well it handles unseen (test) data.

Model A

Train RMS: 1.355 Test RMS: 1.423

There's a small difference between the training RMS and the test RMS, showing that it performs consistently on both training and test datasets. RMS errors are comparatively low, so the model probably makes fairly accurate predictions, which means that Model A generalizes well to unseen data and is probably not overfit. The small increase we see between training and testing RMS is common, so the model captures the patterns in the data without being overfit.
Model A's predictions are good, and it is likely to generalize well to new data. The reported numbers appear credible and mean we probably have a well-balanced model.

Model B

Train RMS: 9.760 Test RMS: 9.165

Both of Model B's RMS errors are too high. The test RMS might be a little lower than the training RMS, but overall high error rates mean the model's predictions aren't very good. The minor decrease in test RMS compared to train RMS could be from random variation or maybe even characteristics of the test data that just happen to fit the model a little better, but it doesn't mean the model will generalize well to other data.
Model B's predictions are bad, but since the two RMSE values are similar, it would probably generalize well to new data. These high error rates mean the model is probably underfitting, failing to "learn" the complexities from our training data.

Model C

Train RMS: 5.033 Test RMS: 0.889

Here we have a big difference between training and testing RMSE. Usually, the test RMS is higher than the training RMS, even if only slightly. Since our test RMSE is much lower, it probably means we need to rework this model. This could possibly mean we have data leakage, where our test set somehow influences the training process, giving us really low test errors.
Model C's results are suspicious and possibly mean we have some sort of data leakage or some other issue. Test RMS so much lower than the train RMS means the numbers are probably not reliable.

Model D

Train RMS: 0.211 Test RMS: 5.072

This model has a low train RMS with a much higher test RMS. A large difference like this means we probably have overfitting happening. The model might fit the training data really well, but it won't really generalize to other test data well at all. The model probably captured some sort of noise and/or patterns in the training data that don't really apply to the test data at all, so it will perform poorly on any new data.
Model D's predictions are no good for unseen data and it's overfit to the training data, so the model is not going to generalize well, possibly because it's relying too much on its training data.

Model E

Train RMS: 0.633 Test RMS: 0.633

Here we have the exact same training RMSE as we have testing RMSE, which is highly improbable, but still can be possible. This means our model's performance is the same for both sets of data. An RMSE of 0.633 is fairly low, so we probably have decent estimate precision. These training and testing RMS values being equal to one another means the model not only generalizes well, but also doesn't over or under fit.
If these numbers are correct, which isn't very likely, then Model E's predictions are solid, and it is likely to generalize well to new data. I would probably try testing it on a different data set to make sure I didn't accidentally train it on the testing data.

3. (30 Points)

Pick one example from Chapter 14 to explore.

- A 2 -3 page summary rewriting the examples with the given images.

Iris Flower Classification

The Iris flower data set is well-known and first used by Ronald Fisher in the 1930s. It has of measurements (in cm) of four attributes (Sepal length, Sepal width, Petal length, and Pedal width) from three species of iris flowers (Iris Setosa, Iris Versicolour, and Iris Virginica). Each species has 50 samples, so 3x50 = 150 samples.

To classify Iris Virginica from the other two species, a Boolean classifier was constructed using the entire dataset: $\hat{f}(x) = \mathbf{sign}(x^T\beta + v)$
With these coefficients: $v = -2.39$, $\beta_1 = -0.0918$, $\beta_2 = 0.406$, $\beta_3 = 0.00798$, $\beta_4 = 1.10$

This classifier's performance was evaluated with a confusion matrix:
- True positives (correctly classified Iris Virginica): 46
- False positives (other species classified as Iris Virginica): 4
- False negatives (Iris Virginica classified as other species): 7
- True negatives (correctly classified as not Iris Virginica): 93

This confusion matrix and error rates show that the classifier had an overall error rate of 7.3%, so the classifier is quite accurate but not perfect, with a fairly small number of misclassifications.

To validate the classifier's performance, a five-fold cross-validation was performed. The dataset was divided into five groups (folds), each containing 10 examples from each species for a total of 30/fold. The results showed some variability in error rates across different folds, but we expect that since the test sets only have 30 examples, so even 1 prediction error changes the error rate by 3.3%.

The error rates for the training and test sets across the five folds were as follows:

- Fold 1: Training 6.7%, Test 3.3%
- Fold 2: Training 5.8%, Test 10.0%
- Fold 3: Training 7.5%, Test 3.3%
- Fold 4: Training 6.7%, Test 16.7%
- Fold 5: Training 8.3%, Test 3.3%

This variability highlights some limits of performing cross-validation with smaller datasets, where a single prediction error can have a big effect on test error rate. The overall error rate on unseen data is estimated to be between 7% and 10%, but due to the small test size this estimate isn't a strong one.

A multi-class classifier was also developed to distinguish between all three iris species. The dataset was split into a training set (120 samples, 40 from each species) and a test set (30 samples, 10 from each species).

Confusion matrices for the training and test sets were as follows:
Training Set Confusion Matrix:
- Setosa: 100% correctly classified.
- Versicolour: 67.5% correctly classified.
- Virginica: 90% correctly classified.

Test Set Confusion Matrix:
- Setosa: 100% correctly classified.
- Versicolour: 60% correctly classified.
- Virginica: 100% correctly classified.

The training set had an error rate of 14.2%, while the test set had a slightly lower error rate of 13.3%. The classifier was very effective at identifying Iris Setosa, with perfect accuracy for both sets. It also performed well with Iris Virginica, especially on the test set, but had more difficulty with Iris Versicolour.

Cross-validation and out-of-sample testing are valuable for assessing model generalization. However, their effectiveness can be limited by small datasets, leading to significant variability in test error rates due to the random selection of test samples. This limitation can result in either overestimating or underestimating a model's performance. Moreover, assumptions that test data will always represent future data can fail in real-world applications where data patterns shift over time.

In summary, the Iris dataset classifiers performed reasonably well, particularly for distinguishing Iris Virginica and Iris Setosa. However, variability in cross-validation results and the difficulty in classifying Iris Versicolour indicate areas for potential improvement and highlight the importance of robust validation techniques.