You are *not* allowed to work with your peers. Show all work and derivations to receive credit. Write clearly and legibly for your own benefit.

Name : Varun Venkatesh

SID : varunven
1939023

Attention!!: You will have to upload the final Python notebook to Canvas in the midterm exam assignment to get credit in addition to attaching a print out pdf to the exam you submit! We grade the pdf; the notebook is for posterity (in case there is an academic integrity issue that arises). You must upload a legible pdf to get credit. This is easy to do. Simply use the `print` option under `file` menu, and choose save to pdf. If on JupyterHub, then under the `file` menu, chose to export to a pdf.

Problem 1 (Geometry of SVD) [3 pts]. Consider the 2×2 matrix

$$A = \frac{1}{\sqrt{10}} \begin{pmatrix} 2 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & -1 \end{pmatrix} + \frac{2}{\sqrt{10}} \begin{pmatrix} -1 \\ 2 \end{pmatrix} \begin{pmatrix} 1 & 1 \end{pmatrix}$$

a. **[1pt]** What is an SVD of A? Express it as $A = USV^\top$, with S the diagonal matrix of singular values ordered in a decreasing fashion. Make sure to check all the properties required for U, S, V.

Solution.

$$A = \frac{1}{\sqrt{10}} \begin{bmatrix} 2 & -2 \\ 1 & -1 \end{bmatrix} + \frac{2}{\sqrt{10}} \begin{bmatrix} -1 & -1 \\ 2 & 2 \end{bmatrix}$$

$$A = \frac{1}{\sqrt{10}} \begin{bmatrix} 2-2 & -2-2 \\ 1+4 & -1+4 \end{bmatrix} = \frac{1}{\sqrt{10}} \begin{bmatrix} 0 & -4 \\ 5 & 3 \end{bmatrix}$$

2 real nums & $\alpha_2 > \alpha_1$

$$AA^T = \frac{1}{5} \begin{bmatrix} 8 & -6 \\ -6 & 17 \end{bmatrix}, \quad \lambda_{AA^T} = 4, 1 \quad \sqrt{\lambda_{AA^T}} = 2, 1 \quad \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad U = \begin{bmatrix} -\sqrt{5}/5 & 2\sqrt{5}/5 \\ 2\sqrt{5}/5 & \sqrt{5}/5 \end{bmatrix}$$

$$\Theta_{AA^T} = [-1\ 2]^T, [2\ 1]^T$$

$$A^TA = \frac{1}{2} \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}, \quad \Theta_{A^TA} = [1\ 1]^T, [1\ -1]^T$$

$$V = \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & -\sqrt{2}/2 \end{bmatrix}$$

$$UU^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \qquad V^TV = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

$$A = U\Sigma V^T = \begin{bmatrix} -\sqrt{5}/5 & 2\sqrt{5}/5 \\ 2\sqrt{5}/5 & \sqrt{5}/5 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & -\sqrt{2}/2 \end{bmatrix}$$

b. **[1pt]** Find the semi-axis lengths and principal axes of the ellipse

$$\mathcal{E}(S) = \{Sx \mid x \in \mathbb{R}^2, \quad \|x\|_2 \leq 1\},$$

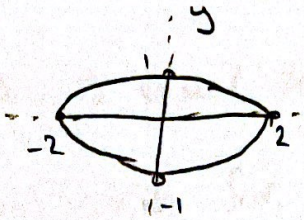
where S is the matrix of singular values above. Note: the principal axes and semi-axis lengths correspond to the largest and smallest distance from the center to the boundary of the ellipse.

Solution.

$S = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

For every $x \in \mathbb{R}^2$ where $\|x\|_2 \leq 1$, we know x represented as $[x_1, x_2]^T$ yields $\sqrt{x_1^2 + x_2^2} \leq 1$.

$Sx = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2x_1 \\ x_2 \end{bmatrix}$. Graphing the range of these values gives us:



, where the semi-axes are equal to 2 and 1 for the x and y axes respectively.

The principal axes are equal to the x and y axes in $\mathbb{R}^2$.

c. **[1pt]** Find the semi-axis lengths and principal axes of the ellipse

$$\mathcal{E}(A) = \{Ax \mid x \in \mathbb{R}^2, \quad \|x\|_2 \leq 1\}.$$

Hint: Show that $\mathcal{E}(A) = \{U\bar{y} : \bar{y} \in \mathcal{E}(S)\}$ to reduce to the previous subpart.

Solution.

$$Ax = \frac{1}{\sqrt{10}}\begin{bmatrix} 0 & -4 \\ 5 & 3 \end{bmatrix}\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = USV^T x = \begin{bmatrix} -\sqrt{5}/5 & 2\sqrt{5}/5 \\ 2\sqrt{5}/5 & \sqrt{5}/5 \end{bmatrix}\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \sqrt{2}/2 & \sqrt{2}/2 \\ \sqrt{2}/2 & -\sqrt{2}/2 \end{bmatrix}$$

$V^T x = \frac{1}{\sqrt{2}}\begin{bmatrix} x_1 + x_2 \\ x_1 - x_2 \end{bmatrix} = \bar{z}$, which we know is on $\mathcal{E}(S)$ because

if $\in \mathbb{R}^2$ and $\frac{1}{\sqrt{2}}\sqrt{(x_1+x_2)^2 + (x_1-x_2)^2} = \frac{\sqrt{2x_1^2 + 2x_2^2}}{\sqrt{2}}$, which has a maximum value of 1.

Therefore, this is equal to $U\bar{y}$, where $\bar{y} = S\bar{z}$, meaning the principal axis for $\mathcal{E}(A)$ is the y axis and the lengths of the semi-axes are $\frac{2}{\sqrt{5}}$ and $\frac{4}{\sqrt{5}}$ respectively for x and y.

Problem 2 (Convex sets)[6 pts]. For each of the sets described below, either show that the set is convex, or provide a counter-example (for example, give two points in the set that violate convexity).

To show convexity, you can either (1) verify the definition of convexity by checking the line segments connecting every pair of points in the set, or (2) use convexity of known sets shown in the lectures (hyperplanes, halfspaces, norm balls, ellipsoids, norm cones) together with properties that preserve convexity. Cite any properties or results you use from the lectures, and show how you use them.

a. **[2pt]**

$x_t \geq$ average of prev 3 elements in set vector where $t \geq 4 \ldots n$

$$S = \left\{ x \in \mathbb{R}^n \mid x_t \geq \frac{1}{3} \sum_{i=t-3}^{t-1} x_i, \text{ for } t = 4, 5, \ldots, n \right\}.$$

Solution.

minimum

The matrix used to represent this can be seen as

$$A = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ 0 & 0 & 1 & \ldots & 0 \\ 1/3 & 1/3 & 1/3 & \ldots & 0 \\ 0 & 1/3 & 1/3 & 1/3 & \ldots & 0 \\ & & \vdots & & & \\ 0 & 0 & \ldots & 1/3 & 1/3 & 1/3 & 0 \end{bmatrix},$$

where $S = \{ y \in R^n \mid \|Ax\|_i^2 \leq y_i^3 \}$

or where y_i is $\geq \|Ax\|_i^2$.

This can be seen as the convex hull where $(1/3 + 1/3 + 1/3) = 1$, $1/3 > 0$ meaning that S is convex.

b. [2pt] $S = \{y \in \mathbb{R}^n \mid \|y - a\|_2 \leq r \text{ for all } a \in C\}$, where $C \subset \mathbb{R}^n$ (convex ✓). There are no other assumptions on C.

Solution.

This is the norm ball repeated for every center a for $a \in C$. We know that the norm-ball is convex, so we need to show S is obtained from it by an operation that preserves convexity. Affine functions include shifting the location of the norm-ball, such as moving the center. This means this set is convex as it is an affine function applied to a convex function.

c. **[2pt]** $S = \{(x, y) \mid \|x\|_2 \geq \|y\|_2,\ x, y \in \mathbb{R}^n\}$.

Solution.

Let $\|x\|_2 = \sqrt{\sum x_1^2 + x_2^2 \ldots x_n^2}$ and $\|y\|_2 = \sqrt{\sum y_1^2 + y_2^2 \ldots y_n^2}$. If this set is convex, then that means for $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \in S$, then $\theta \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + (1-\theta)\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \in S$. If $\|x\|_2 \geq \|y\|_2$, then $\|x\|_2 - \|y\|_2 \geq 0$.

By the reverse triangle inequality, we know $\|x - y\|_2 \geq \|x\|_2 - \|y\|_2 \geq 0$.

This means $\theta\|x_1 - y_1\|_2 \geq \theta\|x_1\|_2 - \theta\|y_1\|_2$, so $\theta\|x_1 - y_1\|_2 + (1-\theta)\|x_2 - y_2\|_2 \geq \theta\|x_1\|_2 - \theta\|y_1\|_2 + (1-\theta)\|x_2\|_2 - (1-\theta)\|y_2\|_2$, showing that this set is not convex.

Problem 3 (Matrix Least Squares Variant) [10 pts]. Consider a data matrix $A \in \mathbb{R}^{m\times n}$ and observation matrix $Y \in \mathbb{R}^{m\times d}$. Recall that the (ordinary) matrix least squares problem $AX \approx Y$ can be seen as projecting the (columns of) Y onto the range of A—i.e., $AX = AA^{\dagger}Y$ where $AA^{\dagger}$ is a projection matrix. In this setting, our model $AX \approx Y$ is such that we are modeling errors in our measurements—that is,

$$Y + \varepsilon_Y = AX.$$

In many applications, however, it is more reasonable to model the errors in both the observations Y and the data A—i.e.,

$$Y + \varepsilon_Y = [A + \varepsilon_A]X \tag{1}$$

Hence, we need to formulate an optimization problem that captures trying to find the matrix $X \in \mathbb{R}^{n\times d}$ that is the best approximate solution to the above model. That is, the problem is to find the smallest perturbations $[\varepsilon_A \quad \varepsilon_Y]$ to the measured independent and dependent variables that satisfy (1), where $[\varepsilon_A \quad \varepsilon_Y]$ is the column-wise concatenation of the matrix ε_A with the matrix ε_Y. Indeed, this can be done with the Frobenius norm as follows:

$$\begin{aligned} &\min_{X,\varepsilon_Y,\varepsilon_A} \ \| [\varepsilon_A \quad \varepsilon_Y] \|_F^2 \\ &\text{subject to } \ Y + \varepsilon_Y = [A + \varepsilon_A]\, X \end{aligned} \tag{2}$$

Upon examining the constraint we can rewrite it as

$$[A + \varepsilon_A \quad Y + \varepsilon_Y] \begin{bmatrix} X \\ -I_d \end{bmatrix} = 0 \in \mathbb{R}^m, \tag{3}$$

where I_d is the $d \times d$ identity matrix.

Below, we are going to use several of the tools from **Modules 2–4** to solve this problem and implement it in Python. **Note that parts f–h are in the Python Notebook.**

a. **[1pt]** Let the matrix $A \in \mathbb{R}^{m\times n}$ and $Y \in \mathbb{R}^{m\times d}$. Note that this implies $\varepsilon_A \in \mathbb{R}^{m\times n}$ and $\varepsilon_Y \in \mathbb{R}^{m\times d}$. Assuming that $m > n$ and $\text{rank}(A + \varepsilon_A) = n$, using the constraint in the optimization problem (2) explain why $\text{rank}([A + \varepsilon_A \quad Y + \varepsilon_Y]) = n$.

Solution:

Since we are trying to minimize $\|[\varepsilon_A \ \varepsilon_Y]\|^2$, this means that if $Y + \varepsilon_y = [A + \varepsilon_A]X$, then $\text{rank}(Y + \varepsilon_y) \leq \min(\text{rank}(A+\varepsilon_A), \text{rank}(X))$. This is because we know $\text{rank}(AB) \leq \min(\text{rank}(A), \text{rank}(B))$. Therefore $\text{rank}([A + \varepsilon_A \ Y + \varepsilon_Y]) = \text{rank}(A + \varepsilon_A)$, as we know $\text{rank}(Y + \varepsilon_y)$ $= \text{rank}(A + \varepsilon_A)$ at most, $\hat{=} n$

b. **[1pt]** The conclusion that $\text{rank}([A+\varepsilon_A \quad Y+\varepsilon_Y]) = n$ from part a. tells us that the goal of solving the optimization problem (2) is to find the smallest matrix $[\varepsilon_A \quad \varepsilon_Y]$ that drops the rank of $[A \quad Y]$ to n when added to it.

Recalling the Eckart-Young Theorem from **Module 3**, formulate the optimization problem (2) as a low rank approximation optimization problem in terms of the matrix $C := [A \quad Y]$.

Solution:

Eckart-Young says that $A_k = \sum_{i=1}^{k} \sigma_i \cdot u_i v_i^T$ is the closest matrix of rank k to A. This means the closest matrix to $[A\ Y]$ of rank n is equal to $[A\ Y]_n = \sum_{i=1}^{\hat{}} \sigma_i \cdot u_i v_i^T$ where ~~this~~ is the SVD of $[A\ Y]$. USV^T

We want $[A+\varepsilon_A \quad Y+\varepsilon_i] = [A\ Y] + \sum_{i=1}^{\hat{}} \sigma_i \cdot u_i v_i^T$

c. **[2pt]** Suppose we express the SVD of $[A \quad Y]$ in terms of submatrices and vectors as follows:

$$C = [A \quad Y] = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}^T \tag{4}$$

where $\Sigma_1 \in \mathbb{R}^{n \times n}$ contains the top n singular values of C, and $\Sigma_2 \in \mathbb{R}^{d \times d}$ contains the next d singular values. The submatrices of V satisfy the following:

$$V_{11} \in \mathbb{R}^{n \times n}, \quad V_{12} \in \mathbb{R}^{n \times d}, \quad V_{21} \in \mathbb{R}^{d \times n}, \quad V_{22} \in \mathbb{R}^{d \times d}$$

Derive an expression for the solution $[\varepsilon_A \quad \varepsilon_y]$ to the problem in part b. in terms of the SVD of $C = [A \; y]$ as given in (4). In addition, derive an expression for the matrix $[A + \varepsilon_A \quad Y + \varepsilon_Y]$ in terms of the SVD and the solution $[\varepsilon_A \quad \varepsilon_y]$. Show your work to get credit.

Solution.

We know rank n approximation tells us $[A \; Y]_n = \sum_{i=1}^{n} \sigma_i \cdot u_i v_i^T$,

which is equal to $[A, Y]_n = \begin{bmatrix} U_{11} \\ U_{21} \end{bmatrix} \Sigma_1 \begin{bmatrix} V_{11} & V_{12} \end{bmatrix}$.

This means if $[\varepsilon_A \; \varepsilon_y]$ is equal to the difference between full-rank and rank n, then:

$$[\varepsilon_A \quad \varepsilon_y] = \begin{bmatrix} U_{11} \\ U_{21} \end{bmatrix} \Sigma_1 \begin{bmatrix} V_{11} & V_{12} \end{bmatrix}$$

Then $[A+\varepsilon_A \quad Y+\varepsilon_Y] = [A \quad Y] + [\varepsilon_A \quad \varepsilon_Y] = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}^T + \begin{bmatrix} U_{11} & 0 \\ U_{21} & 0 \end{bmatrix} \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} \\ 0 & 0 \end{bmatrix}^T$

d. **[2pt]** Given (1) and your solution from the previous parts, derive an expression for X in terms of the SVD of C given in (4). Show your work to get credit.
Hint: Recall the constraint of (2), and the fact that the goal is to find X that satisfies that constraint.
Solution.

$[Y + \mathcal{E}_Y] = [A + \mathcal{E}_A]X$ with $\min \|[\mathcal{E}_A \ \mathcal{E}_Y]\|_F^2$ means we want to

minimize $\sqrt{\mathcal{E}_{A_{00}}^2 + \mathcal{E}_{A_{01}}^2 \cdots \mathcal{E}_{A_{nd}}^2 + \mathcal{E}_{Y_{00}}^2 + \mathcal{E}_{Y_{01}}^2 \cdots \mathcal{E}_{Y_{nd}}^2}$, or in other words,

make each element of $\mathcal{E}_A$ and $\mathcal{E}_Y$ as close to 0 as possible.

We want to make $[A + \mathcal{E}_A \ Y + \mathcal{E}_Y]\begin{bmatrix} X \\ -I_d \end{bmatrix} = 0$, which is shown by

$$\left(\begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}\begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}\begin{bmatrix} V_{11} & V_{12} \\ V_{21} & V_{22} \end{bmatrix}^T + \begin{bmatrix} U_{11} & 0 \\ U_{21} & 0 \end{bmatrix}\begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} V_{11} & V_{12} \\ 0 & 0 \end{bmatrix}^T\right)\begin{bmatrix} X \\ -I_d \end{bmatrix} = 0,$$

$$\begin{bmatrix} 2U_{11} & U_{12} \\ 2U_{21} & U_{22} \end{bmatrix}\begin{bmatrix} 2\Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}\begin{bmatrix} 2V_{11} & 2V_{12} \\ V_{21} & V_{22} \end{bmatrix}^T\begin{bmatrix} X \\ -I_d \end{bmatrix} = 0$$

$$(8\Sigma_1 U_{11}V_{11} + U_{12}V_{21}\Sigma_2)X - (8\Sigma_1 U_{12}V_{12} + U_{12}V_{22}\Sigma_2)I_d = 0$$

$$8\Sigma_1(U_{11}V_{11}X - U_{12}V_{12}I_d) + U_{12}(V_{21}\Sigma_2 X - V_{22}\Sigma_2 I_d)$$

① $U_{11}V_{11}X = U_{12}V_{12}$ ③ $V_{21}X = V_{22}$

$$(8\Sigma_1 U_{21}V_{11} + U_{12}V_{21}\Sigma_2)X - (8\Sigma_1 U_{21}V_{12} + U_{22}V_{22}\Sigma_2)I_d = 0$$

$8\Sigma_1 U_{21}(V_{11}X - V_{12}) = 0$ ④ $U_{12}V_{21}\Sigma_2 X - U_{22}V_{22}\Sigma_2 = 0$

② $V_{11}X = V_{12}$

X can be found from taking V_{11}^{-1} since V_{11} is square and

setting $X = V_{11}^{-1}V_{12}$ provides a solution

e. **[1pt]** Consider the case where $d = 1$. In this case Σ_2 is simply a scalar—in particular, it is σ_{n+1}. From the previous part, you can see that in this case $\begin{bmatrix} x^\top & -1 \end{bmatrix}^\top$ is a right singular vectors of $\begin{bmatrix} A & y \end{bmatrix}$ where $y \in \mathbb{R}^{m \times 1}$ is a vector since $d = 1$. Use this fact to show that the solution from the previous part solves the equation

$$(A^\top A - \sigma_{n+1}^2 I)x = A^\top y$$

Show your work to get credit.

Solution.

We said $X = V_{11}^{-1} V_{12}$, so we can say, then

$$A^T A X - \sigma_{n+1}^2 X = A^T Y$$

$$A^T A V_{11}^{-1} V_{12} - \sigma_{n+1}^2 V_{11}^{-1} V_{12} = A^T Y$$

$$(A^T A - \sigma_{n+1}^2) V_{11}^{-1} V_{12} = A^T Y$$

With the fact that $[x^T -1]^T$ is the right singular vector of $[A\ y]$, then since $\sigma_{n+1}^2 V_{11}^{-1} V_{12}$ = the right singular vector, then this shows that the value we got for X solves this.

Problem 4 (Gradient Descent for Linear Regression) [9 pts]. Consider the linear regression problem:

$$\min_x L(x) = \min_x \|Ax - b\|^2, \tag{5}$$

where $A \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^n$. When the problem is overdetermined i.e $n > m$ and A is full column-rank, we saw in Module 2 that $A^\top A$ is invertible and the solution to problem (5) is $\hat{x}_{\text{OD}} = (A^\top A)^{-1}A^\top b$.

In the midterm, we considered the underdetermined case i.e $n < m$; in this case, the problem (5) has infinitely many solutions with objective 0. If A is full row-rank, then $AA^\top$ is invertible and the minimum norm solution admits the closed form expression $\hat{x}_{\text{UD}} = A^\top(AA^\top)^{-1}b$.

In this problem, we will explore the solution found by gradient descent for problem (5) for both cases. Refer to the initital point of gradient descent as x_0 and the iterates as $\{x_t\}_{t=1}^T$. We say that the algorithm converges if there exists a t' such that $x_t = x_{t'}$ for all $t > t'$, and we refer to $\hat{x}_{\text{GD}} = x_{t'}$ as the solution obtained by gradient descent. This question has parts a - g, and parts f,g can be found on the python notebook.

a. **[1pt]** Consider first the overdetermined case, and assume full column-rank. Write down the solution obtained by gradient descent, assuming that training converges.

Solution.

while $(\|\nabla f(x_t)\| \geq \epsilon$:

$\quad sol = x_k - \alpha \cdot \nabla f(x_k)$

$\quad k = k+1$

If overdetermined, $\text{rank}(A) = m$ & $n > m$ for $A \in \mathbb{R}^{n\times m}$, meaning $x \in \mathbb{R}^m$. In this case, we have many equations to get through, so the solution obtained by gradient descent is equal to $x^{k+1} = x^k - \alpha_k \cdot A^T \cdot (Ax_k - b)$, giving us $\hat{x} = (A^TA)^{-1}A^Tb$

b. **[1pt]** Now consider the underdetermined case. For parts (b) - (d), we consider a simple special case to develop intuition: let $n = 1, m = 2$, and choose $A = [2, 1]$ and $b = [2]$.

Show that there exist infinitely many $\hat{x}$ satisfying $A^T\hat{x} = b$ on a line ℓ. Write the equation of the line ℓ.

Solution.

$A^T\hat{x} = b$, $\begin{bmatrix} 2 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [2]$, $2x_1 + x_2 = 2$. This gives us an equation, but with 2 variables, meaning x_2 is a free variable. This means there are infinite solutions, any of the values along the line $\ell = -2x_1 + 2 = x_2$.

c. **[1pt]** Starting from $x_0 = 0$, what is the direction of the negative gradient, written as a unit-norm vector? Does the direction change along the trajectory?

Solution.

$x_{k+1} = x_k - \mu A^T A x_k + \mu A^T b$, so $x_1 = 0 - 0 + \mu A^T b = \mu [2\ 1]^T [2] = \mu [4\ 2]$

we move in the direction of $[2, 1]$, or $\left[\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right]$.

The direction does not change with the trajectory because we are moving on line with it till we hit the line with minimized norm dist.

d. **[2pt]** Now, we explore for the simple example which solution gradient descent finds. Assume that gradient descent converges.

(i) Using your findings from the previous part, which solution $\hat{x}_{\text{GD}}$ on line ℓ does gradient descent find?

(ii) Verify that for any other solution $\hat{x} \in \ell$, it holds that $(\hat{x} - \hat{x}_{\text{GD}}) \perp \hat{x}_{\text{GD}}$.

(iii) Through a pictorial sketch, argue that $\hat{x}_{\text{GD}}$ has the smallest Euclidean norm across all the solutions on the line ℓ.

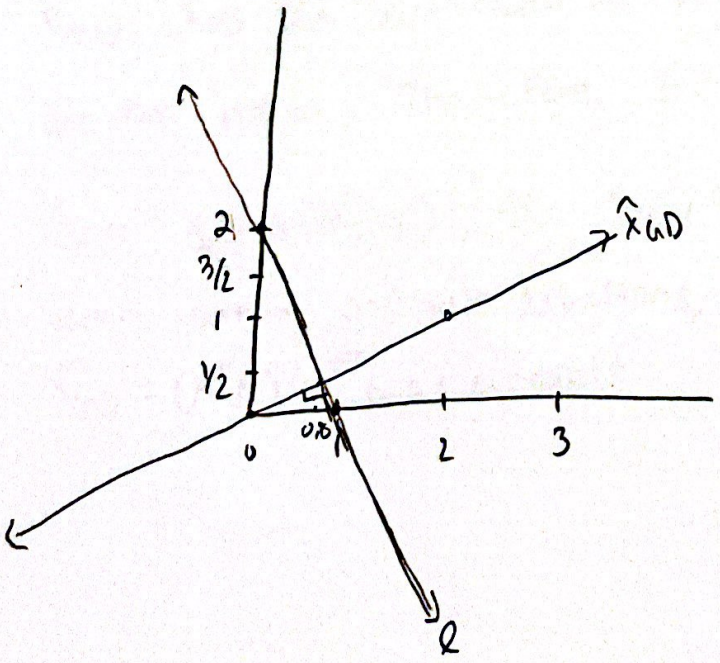
Hint: Pythagoras theorem may be helpful here.

Solution.

i) From $x_0 = 0$, the solution we find is the point $(\frac{4}{5}, \frac{2}{5})$.

ii) For any point $(x_1, -2x_1 + 2)$, we know this is $\perp$ to $(\frac{4}{5}, \frac{2}{5})$ because $(\frac{4}{5}, \frac{2}{5}) \cdot (x_1, -2x_1) = \frac{4}{5}x_1 - \frac{4}{5}x_1 = 0$.

iii)



We can see the two form a right angle, where the dist is $\sqrt{0.8^2 + 0.4^2} = \sqrt{0.8} = 0.894$, any other angle will yield a longer distance since the shortest distance between two points is equal to the distance of the point at a $\perp$ angle.

e. **[2pt]** Now, we generalize the argument from the simple example to general underdetermined systems. Assume full row-rank.

(i) Show that for $x_0 = 0$, the gradient vector is always spanned by the rows of A.

(ii) Show that $\hat{x}_{\text{GD}} = \hat{x}_{\text{UD}}$

This result is surprising and nontrivial. It states that amongst infinitely many solutions that attain the optimal objective value, gradient descent chooses the one with the minimum norm, which is a nice desirable property. This phenomenon, know as *implicit regularization*, helps explain the success of using gradient-based methods to train overparameterized models, like deep neural networks; this has been a very active area of research over the last 5 years!

Solution.

i) For $x_0 = 0$, $\nabla f(x) = \mu A^T b$, which has rank = rank(A), which on this case with full-row rank = row(A)

ii) $\hat{x}_{GD} = \hat{x}_{UD}$ because as we iterate through possible solutions, $\hat{x}_{GD}$ has the full-rank of A and minimizes the distance for the norm. Therefore, for x_{k+1} that is the solution, $x_{k+1} = x_k - \mu \nabla f(x_k)$, the $x_{k+1} \neq x_k$ will be the solution, meaning even among infinite solutions, $\hat{x}_{GD}$ is the optimal choice. $\hat{x}_{GD} = (A^T A)^{-1} A^T b$ as a result

f. **[1pt]** See Notebook

g. **[1pt]** See Notebook

Problem 5 (Convexity and income inequality measures)[4 pts]. Let $x \in \mathbb{R}^n$ represent the vector of incomes of n individuals, where $x_i > 0$, $i = 1, \ldots, n$. Let g_i be the fraction of the total income earned by i individuals who have the lowest income,

g_1 = lowest frac
g_2 = lowest frac of bottom 2
...
g_n = 100% = 1

$$g_i = \frac{1}{\mathbf{1}^T x} \sum_{j=1}^{i} x_{(j)}, \quad i = 1, \ldots, n,$$

where $x_{(j)}$ denotes the jth *smallest* number among $\{x_1, \ldots, x_n\}$. Note that if income is distributed equally among all individuals we have $g_i = i/n$ for all i. One measure of *income inequality* in a society is given by

$$f(x) = \frac{2}{n} \sum_{i=1}^{n} \left(\frac{i}{n} - g_i \right).$$

This measure ranges between 0 (for equal distribution) and $1 - 1/n$ (when all income is earned by one person). Here we are interested to examine the properties of the set of bounded-inequality incomes: $C = \{x \in \mathbb{R}^n_{++} \mid f(x) \leq \frac{1}{2}\}$ (where $\mathbb{R}_{++}$ denotes positive real numbers).

C $x \geq 0$, $f(x) \in [0, \frac{1}{2}]$ → runs where

Interesting Aside: this leads to interesting economic measures that are beyond our scope here.

Answer the following questions:

a. **[2pt]** Recall that the "sum-k-largest" function $\sum_{i=1}^{k} x_{[i]}$ is a convex function of x, where $x_{[i]}$ denotes the ith largest entry in vector x. Using this, answer the following: consider the function $S_i(x) = \sum_{j=1}^{i} x_{(j)}$, that is, the sum of the i lowest incomes in x. For some fixed i, is the function $S_i(x)$ convex, concave, or neither? Justify your answer (cite any property/definition you use from class and show how it is used).

Solution.

The sum k-largest function is the reverse of $S_i(x)$ because it adds the k largest, not k-smallest. This means $S_i(x)$ is not convex, as the reason sum-k-largest is convex is the pointwise max for it is convex - this does not hold for $S_i(x)$. $S_i(x)$ is instead concave because if we take $\sum_{i=1}^{p} x_i - (-S_p(x)) = \sum_{i=1}^{n-p} x_i$, where $0 \leq p \leq n$, then we get that the sum of 2 convex functions is convex, since $-(-) = +$. Therefore, the negation of $S_i(x)$ is convex, so $S_i(x)$ is concave.

b. **[2pt]** It can be shown that (you don't need to show this) the set C defined above can be equivalently expressed as

$$\left\{ x \in \mathbb{R}^n_{++} \;\middle|\; \frac{\sum_{i=1}^n S_i(x)}{\mathbf{1}^\top x} > \left(\frac{n}{4} + \frac{1}{2}\right) \right\}.$$

Is this set convex or nonconvex? Justify your answer. **Hint:** use the result of part a about $S_i(x)$.

Solution.

This set is not convex because S_i is concave, meaning the set contains elements x that are passed through a concave, non-convex function.

Final Exam : Numerical Part

You will need to ensure that you have `utilsfinal` in the folder in which you are running the notebook, and also have the provided figures in a folder called `finalfigs` also in the same directory.

Attention:

you need to upload a ziped file with the notebook and the images you generated to this Canvas assignment to get credit!! Just zip up the file you open from here (called 'finalexam') after you edit the code and add your name to the title --- e.g., `finalexam_lastname_firstname.zip`

and you need to upload your pdf including the **print out of the pdf of the notebook** to gradescope!!

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as plt
import scipy.linalg as sla
from skimage import img_as_ubyte
from imageio import imread, imwrite
from sklearn.preprocessing import StandardScaler

%matplotlib inline

%load_ext autoreload
%autoreload 2

from utilsfinal import *
floc='./finalfigs/'
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[Problem 1] Parts f-h: Least Squares Variant

In this problem, we will use the least squares variant from **Problem 1** to approximately learn the lighting in a photograph, which we can then use to paste new objects into the image while still maintaining the realism of the image. You will be estimating the lighting coefficients for the interior of St. Peter's Basillica, and you will then use these coefficients to change the lighting of an image of a tennis ball so that it can be pasted into the image.

In the figure below, on the left, we show the result of pasting the tennis ball in the image without adjusting the lighting on the ball. The ball looks too bright for the scene and does

not look like it would fit in with other objects in the image. On the right we see the same figure with a lighting adjustment so that the ball looks more natural in the scene.

To convincingly add a tennis ball to an image, we need to need to apply the appropriate lighting from the environment onto the added ball. To start, we will represent environment lighting as a spherical function $f(v)$ where v is a three dimensional unit vector (i.e. $\| v \|_2=1$), and f outputs a three dimensional color vector, one component for red, green, and blue light intensities.

Because $f(v)$ is a spherical function, the input v must correspond to a point on a sphere. The function $f(v)$ represents the total incoming light from the direction v in the scene. The lighting function of a spherical object $f(v)$ can be approximated by the first nine spherical harmonic basis functions which are given by \begin{equation} \begin{array}{lllll} &L_1=1, \quad &L_2=y, \quad &L_3=x, \quad &L_4=z, \quad &L_5=xy\ &L_6=yz, \quad &L_7=3x^2-1,\quad &L_8=xz, \quad &L_9=x^2-y^2 & \end{array} \end{equation} where $v=\begin{bmatrix} x & y & z \end{bmatrix}^\top$.

The lighting function can then be approximated as

$$f(v) \approx \sum_{i=1}^{9} \gamma_i L_i(v)$$

i.e.

$$\begin{bmatrix} — & f(v_1) & — \\ \vdots & ¿ — & ¿ — ¿ \end{bmatrix} = \begin{bmatrix} L_1(v_1) & L_2(v_1) & \cdots & L_9(v_1) \\ L_1(v_2) & L_2(v_2) & \cdots & L_9(v_2) \\ \vdots & \vdots & \cdots & \vdots \\ L_1(v_m) & L_2(v_m) & \cdots & L_9(v_m) \end{bmatrix} \begin{bmatrix} — & \gamma_1 & — \\ \vdots & ¿ — & ¿ — ¿ \end{bmatrix}$$

wher $L_i(v)$ is the i--th basis function from the list above.

The function of incoming light $f(v)$ can be measured by photographing a spherical mirror placed in the scene of interest. In this case, we provide you with an image of the sphere as seen in the figure below.

In the code provided in `utilsfinal.py`, there is a function `extractNormals(img)` that will extract the training pairs $(v_i, f(v_i))$ from the image. There is also a provided function `computeBasis(vs)` which will take in the `vs` values from `extractNormal(img)` and return the A matrix. An example using this function is given in the next block.

```
data,tennis,target = loadImages()
vs, ys = extractNormals(data) # ys are the observed intesities, vs are
the normal vectors
A = computeBasis(vs) # computes matrix of basis functions applied to
the normal vectors v_i
```

Part f

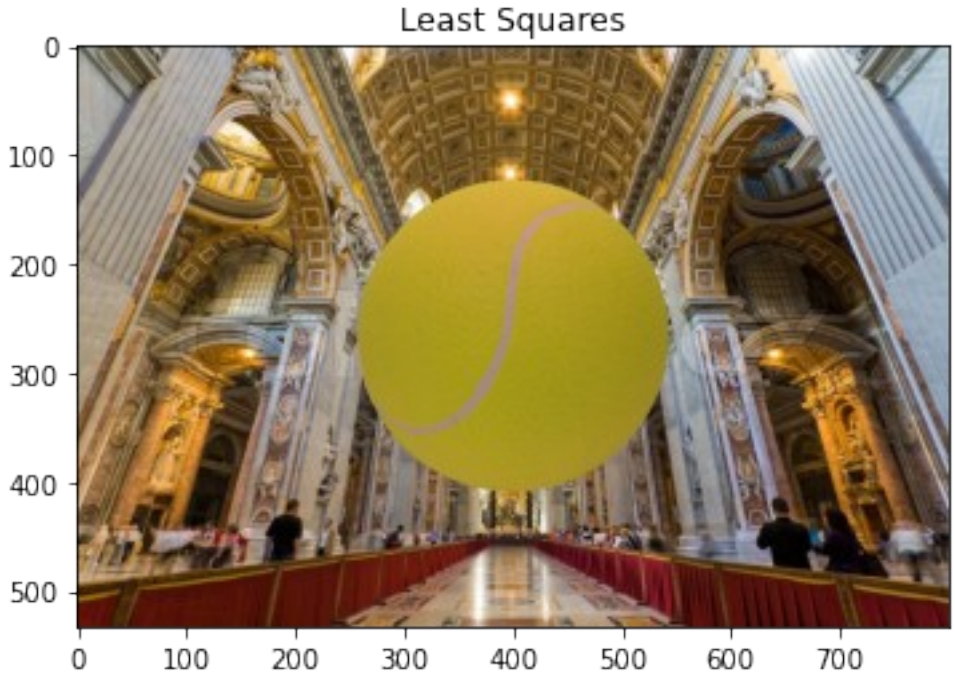
TO DO: Use the spherical harmonic basis functions to create a nine dimensional feature vector for each sample. Use this to formulate an **ordinary least squares** problem and solve for the unknown coefficients γ_i. Report the estimated values for γ_i and include a visualization of the approximation using the provided code. Use the solution using the formula we learned in Module 2 for matrix least squares.

Helpful Hint: Make sure you check dimensions as you are working through the code. This is a good debugging technique.

```
def OLS(A, ys):
    '''
    Ordinary Matrix Least Squares Solver
    Inputs:
    - A is m x 9
    - vs is m x 3

    Outputs:
    - coeff: coefficients of least squares, should be 9 x 3
    '''

    ## Your code goes here
    coeff = np.linalg.lstsq(A, ys, rcond=None)[0]
    return coeff

# reduce the number of samples because computing the SVD on
# the entire data set takes too long
Ap = A[::50]
ysp = ys[::50]

coeffLSQ = OLS(Ap,ysp) # run OLS
imgLSQ = relightSphere(tennis,coeffLSQ) # calls a function to render
the sphere given the relighting coefficients
targetLSQ = compositeImages(imgLSQ,target) # build the image of the
relight sphere in St. Peters

print('Least squares:\n'+str(coeffLSQ))
plt.figure()
plt.imshow(targetLSQ)
plt.title('Least Squares')
plt.show()

imwrite(floc+'OLS_output.png',img_as_ubyte(targetLSQ))
```

```
Least squares:
[[202.31845431 162.41956802 149.07075034]
 [-27.66555164 -17.88905339 -12.92356688]
 [ -5.15203925  -4.51375871  -4.24262639]
 [ -1.08629293   0.42947012   1.15475569]
```

```
[ -3.14053107  -3.70269907  -3.74382934]
[ 23.67671768  23.15698002  21.94638397]
[ -3.82167171   0.57606634   1.81637483]
[  4.7346737    1.4677692   -1.12253649]
[ -9.72739616  -5.75691108  -4.8395598 ]]
```



Part g

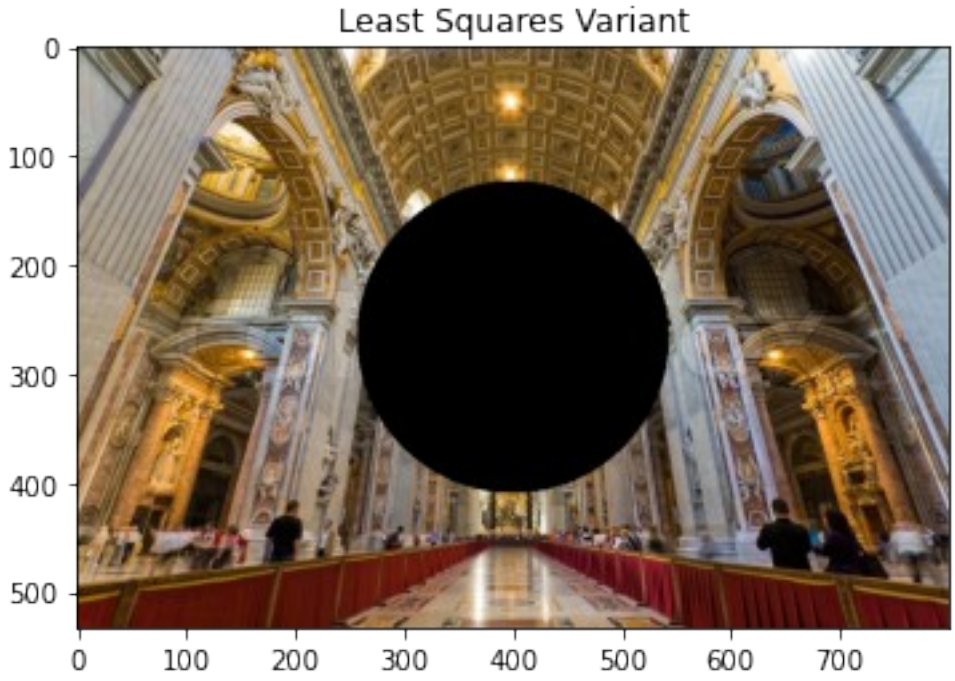
When we extract from the data the direction n to compute $(v_i, f(v_i))$, we make some approximations about how the light is captured on the image. We also assume that the spherical mirror is a perfect sphere, but in reality, there will always be small imperfections. Thus, our measurement for v contains some error, which makes this an ideal problem to apply the least squares variant described in the problem statement for **Problem 1** given in the exam pdf.

TO DO: Solve this problem with the **least squares variant** by allowing perturbations in the matrix of basis functions. Report the estimated values for γ_i and include a visualization of the approximation. The output image will be visibly wrong, and we'll explore how to fix this problem in the next part. Your implementation may only use `numpy.linalg.svd` and the matrix inverse functions from the linear algebra library in `numpy` as well as `np.linalg.solve`.

$$\begin{bmatrix} — & f(v_1) & — \\ \vdots & ¿— & ¿ \\ & ¿ & \end{bmatrix} = \begin{bmatrix} L_1(v_1) & L_2(v_1) & \cdots & L_9(v_1) \\ L_1(v_2) & L_2(v_2) & \cdots & L_9(v_2) \\ \vdots & \vdots & \cdots & \vdots \\ L_1(v_m) & L_2(v_m) & \cdots & L_9(v_m) \end{bmatrix} \begin{bmatrix} — & \gamma_1 & — \\ \vdots & ¿— & ¿ \\ & ¿ & \end{bmatrix}$$

```
def LSV(A, ys):
    '''
    Least Squares Variant
    Inputs:
    - A: m x 9 data matrix
    - y: m x 3 observation matrix

    Outputs:
    - coeff: 9 x 3 matrix of coefficients "gamma"
    '''
    # Build the combined matrices
    XY = A
    for i in range(ys.shape[1]):
        np.append(XY, ys[i])

    # run svd on combined data matrix
    # hint the v output of svd is V^T
    u,s,v = la.svd(XY)
    d = ys.shape[1]
    n = A.shape[1]
    v11 = v[:n, :n]
    v12 = v[:n, n-d:]
    print(v.shape)
    print(v11.shape)
    print(v12.shape)
    v11 = la.inv(v11)
    # extract the coefficients from the svd using the formula you
derived in parts a--e
    coeff = np.matmul(v11, v12)
    return coeff

data,tennis,target = loadImages()
vs, ys = extractNormals(data)
A = computeBasis(vs)
# reduce the number of samples because computing the SVD on
# the entire data set takes too long
Ap = A[::50]
ysp = ys[::50]

coeffLSV = LSV(Ap,ysp)
imgLSV = relightSphere(tennis,coeffLSV)
targetLSV = compositeImages(imgLSV,target)

print('Least Squares Variant Coefficients:\n'+str(coeffLSV))
plt.figure()
plt.imshow(targetLSV)
plt.title('Least Squares Variant')
plt.show()
imwrite(floc+'lsv_output.png',img_as_ubyte(targetLSV))
```

```
(9, 9)
(9, 9)
(9, 3)
Least Squares Variant Coefficients:
[[ 5.91971997e-19 -7.26871312e-20 -7.56149567e-22]
 [-5.57061079e-20  1.56363804e-19 -1.39562872e-19]
 [ 5.35830449e-21  2.59358586e-19  3.07374411e-20]
 [ 2.96433041e-20 -1.01770545e-19  2.63435166e-20]
 [ 5.31946981e-20  1.05210974e-17 -5.57832970e-21]
 [-8.32139655e-21  2.01109488e-16  2.25865461e-20]
 [ 1.00000000e+00 -4.75130261e-20 -5.56485494e-20]
 [-2.31873671e-20  1.00000000e+00  2.81858127e-20]
 [-7.16512304e-20  1.37936313e-20  1.00000000e+00]]
```



Part h

In the previous part, you should have noticed that the visualization is drastically different than the one generated using least squares. Recall that in total least squares we are minimizing $\| \begin{bmatrix} \varepsilon_A & \varepsilon_y \end{bmatrix} \|_F^2$. Intuitively, to minimize the Frobenius norm of components of both the inputs and outputs, the inputs and outputs should be on the same scale. However, this is not the case here.

Color values in an image will typically be in $[0, 255]$, but the original image had a much larger range. We compressed the range to a smaller scale using tone mapping, but the effect of the compression is that relatively bright areas of the image become less bright.

As a compromise, we scaled the image colors down to a maximum color value of 384 instead of 255. Thus, the inputs here are all unit vectors, and the outputs are three dimensional vectors where each value is in $[0, 384]$.

To do: Rewrite the LSV function with an optional scale parameter s that will be used to scale the y values as follows

$$y_{\text{scaled}} = \frac{y}{s}$$

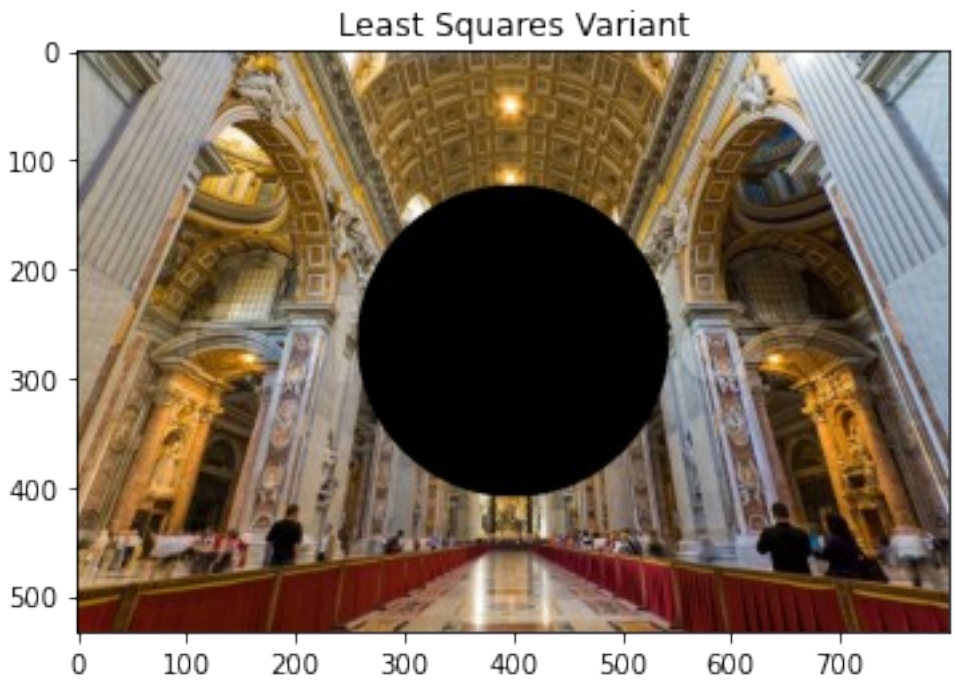
Propose a value by which to scale the outputs $f(v_i)$ such that the values of the inputs and outputs are roughly on the same scale. Solve this scaled version of the problem using the least squares variant, and report the computed spherical harmonic coefficients and provide a rendering of the relit sphere.

Solution to Part h

Recall: the least squares variant assumes that the noise is the same in all directions. When we just have some data, it can be hard to think about what the noise model should be. The default is to take a "significant figures" mentality and assume that everything should be on the same scale, assuming that noise is roughly proportional to the size of the inputs. Because most of the basis functions lie within $[-1, 1]$, we want to scale the image pixel values so that they lie in a similar range. For these results, we can scale the values by 1/384, but any reasonable value that scales the pixel values to a similar range is acceptable

```
def LSV(A, ys, s=1):
    '''
    Least Squares Variant
    Inputs:
    - A: m x 9 data matrix
    - y: m x 3 observation matrix
    - s: this scale parameter will scale the ys
             i.e. ys/s

    Outputs:
    - coeff: 9 x 3 matrix of coefficients "gamma"
    '''
    # Build the combined matrices
    XY = A
    ys = ys/s
    for i in range(ys.shape[1]):
        np.append(XY, ys[i])

    # run svd on combined data matrix
    # hint the v output of svd is V^T
    u,S,v = la.svd(XY)
    d = ys.shape[1]
    n = A.shape[1]
    v11 = v[:n, :n]
    v12 = v[:n, n-d:]
```

```
    print(v.shape)
    print(v11.shape)
    print(v12.shape)
    v11 = la.inv(v11)
    # extract the coefficients from the svd using the formula you
derived in parts a--e
    coeff = np.matmul(v11, v12)
    return coeff

data,tennis,target = loadImages()
vs, ys = extractNormals(data)
A = computeBasis(vs)
# reduce the number of samples because computing the SVD on
# the entire data set takes too long
Ap = A[::50]
ysp = ys[::50]

coeffLSVScaled = LSV(Ap,ysp,s=384) # TODO fill in a value for s

imgLSV = relightSphere(tennis,coeffLSVScaled)
targetLSV = compositeImages(imgLSV,target)

plt.figure()
plt.imshow(targetLSV)
plt.title('Least Squares Variant')
plt.show()
imwrite(floc+'lsv_output.png',img_as_ubyte(targetLSV))

(9, 9)
(9, 9)
(9, 3)
```



[Problem 2]: Gradient Descent on Linear Regression: Parts f, g

In this problem, we will apply gradient descent to an underdetermined linear regression problem on synthetic data. We will verify that the iterates of gradient descent converge to the least norm solution x_{UD}

Synthetic Data

First, we generate a random train matrix A and labels b

```
n, m = 200, 1000
```

```
A = np.random.normal(size = (n,m), scale = 3)
x_gt = 10*np.ones(m)
b = A @ x_gt
```

```
x_UD = A.T @ np.linalg.inv (A @ A.T) @ b
```

Part (f): Implement GD

(f) Below, implement gradient descent on the linear regression problem with initial point _start, constant stepsize _alpha and stopping threshold _eps.

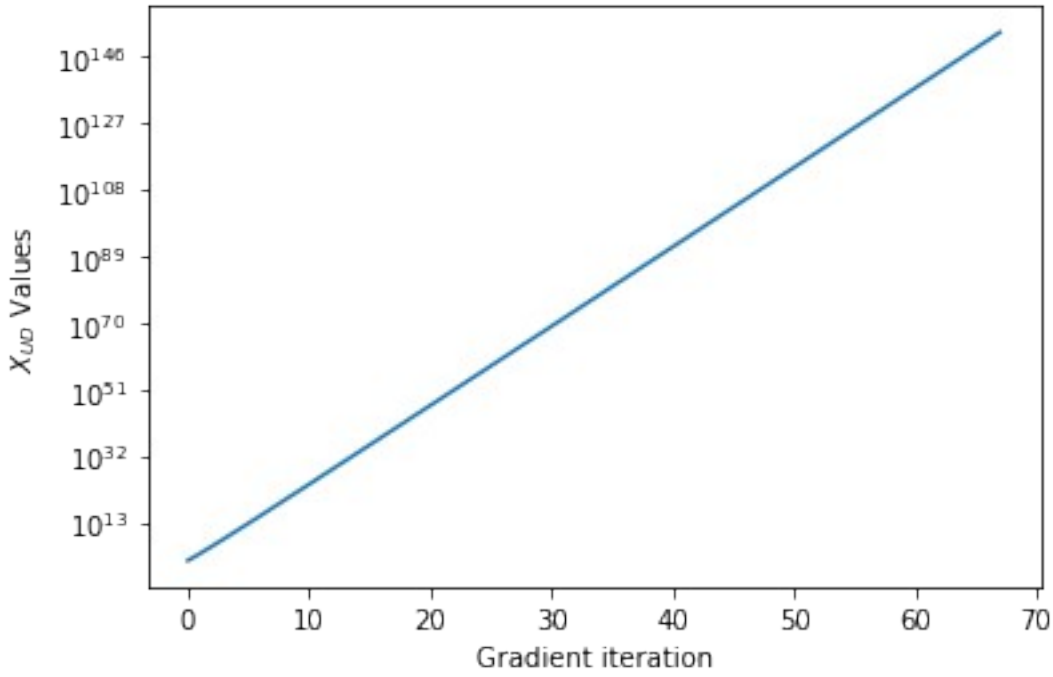
The stopping rule we adopt is to terminate the algorithm when the least squares objective is smaller than _eps. **Note that** this is a different stopping rule than the last python assignment.

```
_start = np.zeros(m)
_alpha = 0.01
_eps = 1e-14

def gradF(x):
    #TODO
    return np.matmul(np.transpose(A), (np.matmul(A, x)-b))

def runGradientDescent(start, alpha, eps):
    """
    input: start (initial guess), alpha (step-size), eps (tolerance)
    output: iterates (all iterates of gradient descent), counter
(number of iterations taken)
    """
    solution = []
    solution.append(start)
    k = 1
    while(np.linalg.norm(np.matmul(A, solution[k-1])-b) >= eps and
np.linalg.norm(np.matmul(A, solution[k-1])-b) < float("inf")):
        news = solution[k-1] - alpha*gradF(solution[k-1])
        solution.append(news)
        k+=1
    return solution, k

iterates_gd, counter_gd = runGradientDescent(_start, _alpha, _eps)
```

Part (g) Visualize trajectory

(g.1) Plot the distance between the gradient descent iterates and $x_U D$ over time. Use a logarithmic scale on the y-axis. Remark what you observe, and how this connects to the previous parts of the question.

```
#TODO
dist = []
for i in range(len(iterates_gd)):
    dist.append(la.norm(iterates_gd[i]-x_UD))
plt.figure()
plt.plot(range(counter_gd), dist)
plt.xlabel("Gradient iteration")
plt.ylabel("$X_{UD}$ Values")
plt.yscale('log')
plt.show()

#This has a steady increase with the log scale that we used, meaning
it is a logarithmic increase until the XUD values are matched
```




```
trajectory = np.array(iterates_gd)
scaler = StandardScaler()
trajectory_norm = scaler.fit_transform(trajectory[:, :])
limit_point_norm = scaler.transform([x_UD])
```

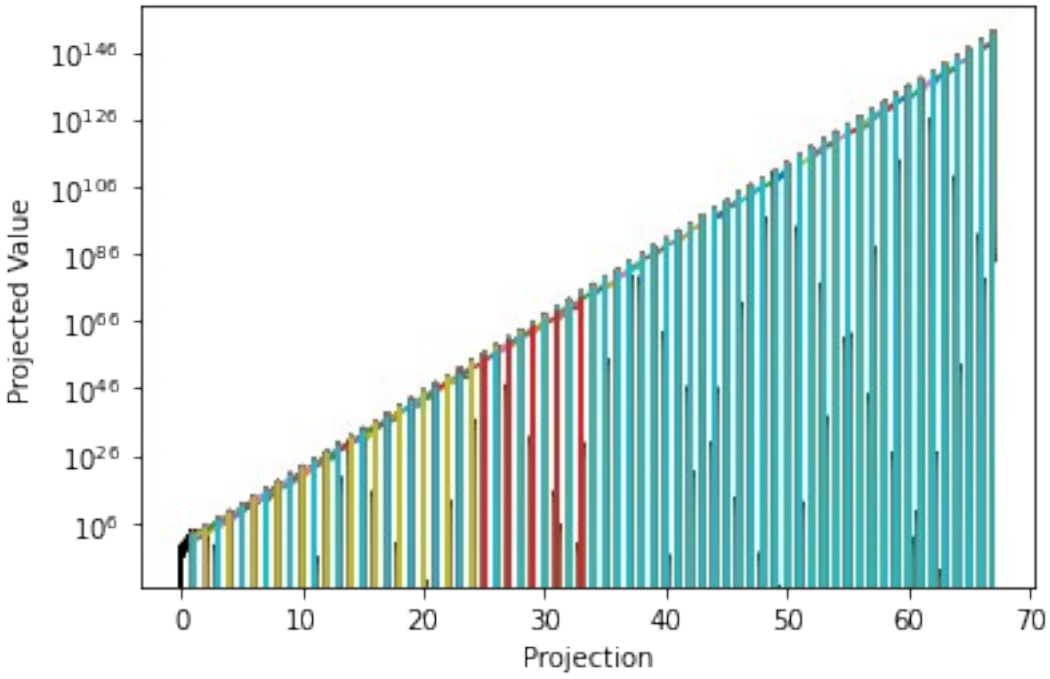
(g.2) Pre-process the above "trajectory" matrix for PCA, as discussed in lecture.

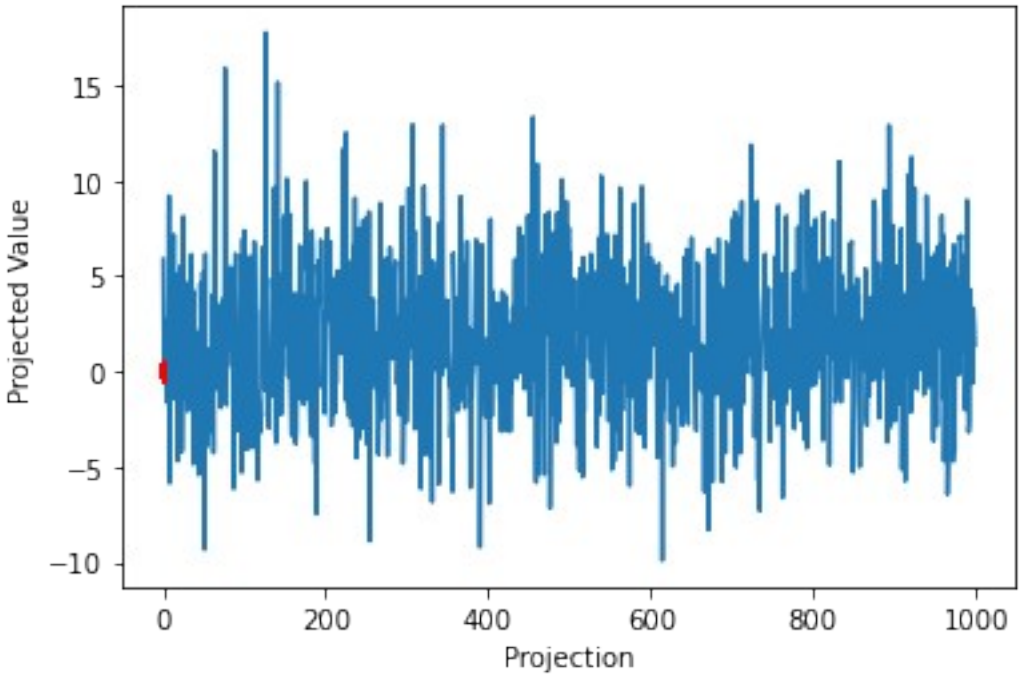
Then perform PCA to reduce the dimensionality of the iterates to 2D. Similarly pre-process "limit_point_norm" (which contains x_{UD}) and project it onto the top-2 principal components from above.

Now, plot the projected iterates, with subsequent iterates connected by lines. Also plot the projected x_{UD}. Remark on what you observe

```
#TODO
covar_mat = np.matmul(trajectory_norm.T, trajectory_norm)
values, vectors = sla.eigh(covar_mat, eigvals = (998, 999))
vectors = vectors.T
projected_features = np.matmul(vectors, trajectory_norm.T)
res_traj = projected_features.dot(trajectory_norm)

covar_mat = np.matmul(limit_point_norm.T, limit_point_norm)
values, vectors = sla.eigh(covar_mat, eigvals = (998, 999))
vectors = vectors.T
projected_features = np.matmul(vectors, limit_point_norm.T)
res_lim = projected_features.dot(limit_point_norm)

plt.figure()
plt.plot(res_traj, color='black')
```

```
plt.plot(trajectory)
plt.xlabel("Projected Feature Number")
plt.ylabel("Projected Value")
plt.yscale('log')
plt.show()

plt.figure()
plt.plot(x_UD)
plt.plot(res_lim, color='r')
plt.xlabel("Projected Feature Number")
plt.ylabel("Projected Value")
plt.show()

#For the iterates, we see a steady increase on the log graph, similar
to the expected behavior from the previous graph.
# We see the lowest value is the 2 projected features marked black at
the bottom.
# For the res_lim, we still see the minimum value is the projected
features at the top. It falls directly on the minimum
# xUD value on the graph
```




15
10
5
0
−5
−10
Projected Value
0
200
400
600
800
1000
Projection