Pr. 1.

(a) One application of the "**vec trick**" $\text{vec}(\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}^T) = (\boldsymbol{B} \otimes \boldsymbol{A})\text{vec}(\boldsymbol{X})$ is for computing a 2D **discrete Fourier transform** (DFT) of a 2D signal. The (1D) DFT of a vector $\boldsymbol{x} \in \mathbb{C}^N$ is the vector $\boldsymbol{f}_x \in \mathbb{C}^N$ with entries

$$[\boldsymbol{f}_x]_k = \sum_{n=1}^{N} x_n \exp\left(\frac{-2\pi i(k-1)(n-1)}{N}\right), \quad k = 1, \ldots, N.$$

(Here we are using the linear algebra (and `Julia`) convention where the first array index is 1, whereas DSP books usually use $0, \ldots, N-1$.) We can represent the above computation in matrix-vector form as $\boldsymbol{f}_x = \boldsymbol{F}_N \boldsymbol{x}$, where $\boldsymbol{F}_N$ is the $N \times N$ DFT matrix with entries

$$[\boldsymbol{F}_N]_{k,n} = \exp\left(\frac{-2\pi i(k-1)(n-1)}{N}\right), \quad k, n = 1, \ldots, N.$$

We can generate $\boldsymbol{F}_N$ in `Julia` as follows.

```
# N × N DFT matrix
using FFTW: fft
using LinearAlgebra: I
F = fft(I(N), 1)
```

One can verify that $\boldsymbol{F}_N' \boldsymbol{F}_N = N \boldsymbol{I}_N$, so the (1D) **inverse DFT** of $\boldsymbol{f}_x$ can be computed as $\boldsymbol{x} = (1/N)\boldsymbol{F}_N' \boldsymbol{f}_x$.

In two dimensions, we compute the **2D DFT**, call it $\boldsymbol{S}_{\boldsymbol{X}}$, of the $M \times N$ matrix $\boldsymbol{X}$, by computing the 1D DFT of each column of $\boldsymbol{X}$ followed by the 1D DFT of each row of the result (or vice versa).

Explain why the following expression computes the 2D DFT of $\boldsymbol{X}$:

$$\boldsymbol{S}_{\boldsymbol{X}} = \boldsymbol{F}_M \boldsymbol{X} \boldsymbol{F}_N^T.$$

Now write $\text{vec}(\boldsymbol{S}_{\boldsymbol{X}})$ as the product of a matrix and $\text{vec}(\boldsymbol{X})$.

(b) Show that the following expression computes the **2D inverse DFT** of $\boldsymbol{S}_{\boldsymbol{X}}$:

$$\boldsymbol{X} = \frac{1}{MN} \boldsymbol{F}_M' \boldsymbol{S}_{\boldsymbol{X}} \overline{\boldsymbol{F}}_N,$$

where $\overline{\boldsymbol{Y}}$ denotes (element-wise) complex conjugate of matrix $\boldsymbol{Y}$.

Hint: Use the fact that $\boldsymbol{F}_N' \boldsymbol{F}_N = \boldsymbol{F}_N \boldsymbol{F}_N' = N \boldsymbol{I}_N$.

Now write $\text{vec}(\boldsymbol{X})$ as the product of a matrix and $\text{vec}(\boldsymbol{S}_{\boldsymbol{X}})$.

Pr. 2.

(a) Find (by hand) all solutions of the **linear least squares problem** $\arg\min_{\boldsymbol{x} \in \mathbb{R}^2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$ when $\boldsymbol{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$.

(b) Describe briefly how you would modify your solution for this variation of the problem: $\arg\min_{\boldsymbol{x} \in \mathbb{C}^2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$.

Pr. 3.

In the ordinary **least-squares** problem we found the (minimum norm) $\hat{\boldsymbol{x}}$ that minimized $\|\boldsymbol{r}\|_2$ where $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$ is the residual. We saw that the optimal $\boldsymbol{x}$ can be expressed entirely in terms of $\boldsymbol{b}$ and an SVD of $\boldsymbol{A}$. Let $\boldsymbol{A}$ be a $M \times N$ matrix so that $\boldsymbol{x} \in \mathbb{F}^N$ and $\boldsymbol{b} \in \mathbb{F}^M$ and $\boldsymbol{r} \in \mathbb{F}^M$.

In applications with **heteroscedastic measurement errors**, we prefer to minimize $\|\boldsymbol{W}\boldsymbol{r}\|_2$, *i.e.*, a weighted squared error, where $\boldsymbol{W}$ is a diagonal matrix having diagonal entries $w_1, \ldots w_M \geq 0$. Determine the optimal $\boldsymbol{x}$ for this **weighted least-squares** problem. (Again, you should express the answer in terms of $\boldsymbol{b}$, $\boldsymbol{W}$, and an SVD of an appropriate matrix.) Your answer must not have any pseudo-inverse in it! (It may have an inverse as long as you are sure that the matrix is invertible and trivial to invert.) You may assume that `W*A == zeros(size(W*A))` is false in Julia.

Pr. 4.
Recall that the **least squares** problem

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2,$$

has the solution $\hat{\boldsymbol{x}} = \boldsymbol{A}^+ \boldsymbol{b}$ where $\boldsymbol{A}^+$ denotes the **pseudo-inverse**. When the nullspace of $\boldsymbol{A}$ is non-trivial then $\hat{\boldsymbol{x}}$ is the **minimum norm** solution. When $\boldsymbol{A}$ is large, it can be computationally prohibitive to compute an SVD of $\boldsymbol{A}$ and then its pseudo-inverse before computing $\hat{\boldsymbol{x}}$. In such settings, the **gradient descent** iteration given by

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \mu \boldsymbol{A}' (\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}),$$

will converge to a minimizer of $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$ as iteration $k \to \infty$ when $0 < \mu < 2/\sigma_1^2(\boldsymbol{A})$. Note that the iteration has a fixed point, *i.e.*, $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k$ if

$$\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{b}) = \boldsymbol{0},$$

which are exactly the normal equations. So any fixed point minimizes the least squares cost function.

(a) Write a function called `lsgd` that implements the above least squares gradient descent algorithm.

In `Julia`, your file should be named `lsgd.jl` and should contain the following function:

```
"""
    x = lsgd(A, b ; mu=0, x0=zeros(size(A,2)), nIters::Int=200)

Performs gradient descent to solve the least squares problem:
``\\argmin_x 0.5 \\| b - A x \\|_2``

In:
- `A` `m × n` matrix
- `b` vector of length `m`

Option:
- `mu` step size to use, and must satisfy ``0 < mu < 2 / \\sigma_1(A)^2``
to guarantee convergence,
where ``\\sigma_1(A)`` is the first (largest) singular value.
Ch.5 will explain a default value for `mu`
- `x0` is the initial starting vector (of length `n`) to use.
Its default value is all zeros for simplicity.
- `nIters` is the number of iterations to perform (default 200)

Out:
- `x` vector of length `n` containing the approximate LS solution
"""
function lsgd(A, b ; mu::Real=0, x0=zeros(size(A,2)), nIters::Int=200)
```

Email your solution as an attachment to `eecs551@autograder.eecs.umich.edu`.

The function specification above uses a powerful feature of `Julia` where functions can have optional arguments with specified default values. If you simply call `lsgd(A,b)` then `mu`, `x0` and `nIters` will all have their default values. But if you call, say, `lsgd(A, b, nIters=5, mu=7)` then it will use the specified values for `nIters` and `mu` and the default for `x0`. Note that these named optional arguments can appear in any order. This approach is very convenient for functions with multiple arguments.

(b) After your code passes, use it to generate a plot of $\log_{10}(\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|)$ versus $k = 0, 1 \ldots, 200$ using $\mu = 1/\sigma_1^2(\boldsymbol{A})$ for $\boldsymbol{A}$ and $\boldsymbol{b}$ generated as follows

```
using Random: seed!
m = 100; n = 50; sigma = 0.1
seed!(0) # seed random number generator
A = randn(m, n); xtrue = rand(n); noise = randn(m)
b = A * xtrue + sigma * noise # b and xhat change when sigma changes
```

Repeat for $\sigma = 0.5, 1, 2$ and turn in one plot with all four curves on it, using a logarithmic scale for the vertical axis. Does $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$ decrease monotonically with k in the plots? Note that $\sigma =$ `sigma` here is a noise standard deviation unrelated to singular values.

Pr. 5.

(a) For $\delta > 0$, determine the solution of the **regularized least-squares** problem

$$\hat{\boldsymbol{x}}(\delta) = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \delta^2 \frac{1}{2}\|\boldsymbol{x}\|_2^2.$$

Express the answer in terms of the SVD of an appropriate matrix.
Use the fact that $\boldsymbol{B}^+ = (\boldsymbol{B}'\boldsymbol{B})^{-1}\boldsymbol{B}'$ when $\boldsymbol{B}'\boldsymbol{B}$ is invertible to simplify the expression.
Hint: Rewrite the cost function so it looks like a usual least-squares problem.

(b) What does $\hat{\boldsymbol{x}}(\delta)$ tend to as $\delta \to \infty$? Does this answer make sense?

(c) Write (by hand, not code) an iteration based on the **gradient descent** (GD) method such that the iterates converge to the minimizer $\hat{\boldsymbol{x}}(\delta)$.

(d) Determine a condition on the step size μ that guarantees convergence of your GD method. Express the condition in terms of the original problem quantities: $\boldsymbol{A}$, $\boldsymbol{b}$, and δ.

Pr. 6.
Consider the problem of finding the **minimum 2-norm solution** of the linear **least squares** problem $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2$ when $\boldsymbol{A} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. The solution is: $\hat{\boldsymbol{x}} = \boldsymbol{A}^+\boldsymbol{b} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

(a) Consider a perturbation $\boldsymbol{E}_1 = \begin{bmatrix} 0 & \delta \\ 0 & 0 \end{bmatrix}$ of $\boldsymbol{A}$, where δ is a small positive number. Solve the perturbed version of the above problem: $\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{A}_1\boldsymbol{z} - \boldsymbol{b}\|_2$, where $\boldsymbol{A}_1 = \boldsymbol{A} + \boldsymbol{E}_1$.
What happens to $\|\hat{\boldsymbol{x}} - \boldsymbol{z}^*\|_2$ as δ approaches 0?

(b) Now consider the perturbation $\boldsymbol{E}_2 = \begin{bmatrix} 0 & 0 \\ 0 & \delta \end{bmatrix}$ where again δ is a small positive number. Solve the perturbed problem $\boldsymbol{z}^* = \arg\min_{\boldsymbol{z}} \|\boldsymbol{A}_2\boldsymbol{z} - \boldsymbol{b}\|_2$ where $\boldsymbol{A}_2 = \boldsymbol{A} + \boldsymbol{E}_2$.
What happens to $\|\hat{\boldsymbol{x}} - \boldsymbol{z}^*\|_2$ here as $\delta \to 0$?

Pr. 7.
Given constants a, b, c, consider the plane $\left\{(x, y, z) \in \mathbb{R}^3 \,:\, ax + by + cz = 0\right\}$ that intersects the origin.

(a) Describe how you would use an **SVD** to find **basis vectors** for the plane.
How many bases vectors are required to express a point on the plane?

(b) Find the point on the plane that is closest to an arbitrary point $(\alpha, \beta, \gamma) \in \mathbb{R}^3$. Your expression should be general and reasonably simple.
Anytime a problem says "find" something, it is also implied that you must also show how *you found it.*

(c) Using your preceding answer (and probably `Julia` or a calculator), find the point on the plane $x + 2y + 3z = 0$ that is closest to the point $(4, 5, 6)$.

Pr. 8.

(Photometric stereo: compute normals)

In this problem you implement another tool needed for a computer vision method called **photometric stereo**, allowing us (eventually) to reconstruct a 3D object's surface from 2D images of it under different lighting conditions. Figure 1 gives a preview at what you will be able to do after completing all the pieces.

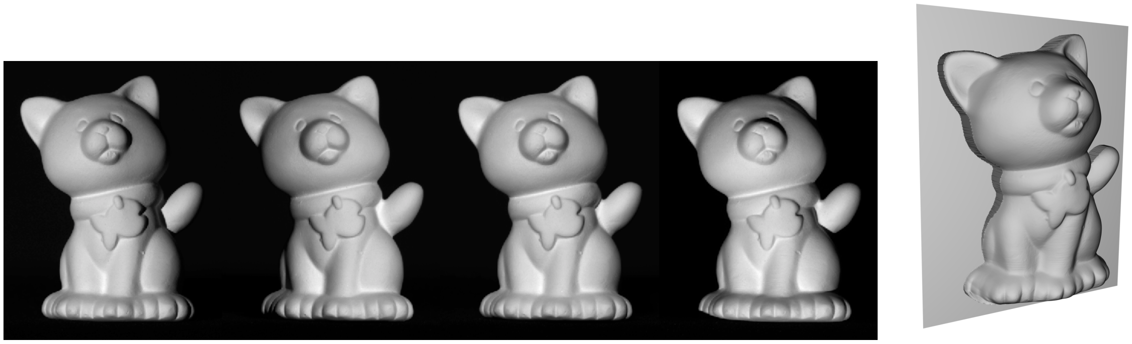


Figure 1: Photometric stereo example. Left (a): 2D images of a common scene under different lighting conditions. Right (b): 3D surface reconstruction computed from the input images.

Suppose we are in a dark room with an object on a dark table, a camera fixed above it, and a moveable light source. We model the object surface as $z = f(x, y)$, where (x, y) denotes coordinates on the table and z denotes height above the table. Assume that a $m \times n$ sized image $I(x, y)$ is a representation of $f(x, y)$ for each (x, y) tuple. (Given one light source, $f(x, y)$ is the z coordinate of the position where a ray of light hits the surface at (x, y, z).)
As seen in Figure 1a, the pixel intensity $I(x, y)$ indicates how much light reflects off the surface $f(x, y)$. If our object is diffuse (also called matte or **Lambertian reflectance**), one can derive the relationship

$$I(x, y) = \alpha(x, y) \, (\boldsymbol{\ell}^T \boldsymbol{n}(x, y)), \tag{1}$$

where $\boldsymbol{\ell} \in \mathbb{R}^3$ is a unit vector describing the orientation of the incident light rays on the surface, $\boldsymbol{n}(x, y) \in \mathbb{R}^3$ is the unit-norm surface normal vector of f at $(x, y, f(x, y))$, and $\alpha(x, y) > 0$ is a scaling constant called the **surface albedo**.

Now suppose that we take d images $I_1, \ldots, I_d$ of our object, with lighting directions $\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_d \in \mathbb{R}^3$. We are going to work on one pixel at a time. For any pixel location (x, y), we can stack (1) into an **overdetermined** system of equations:

$$\underbrace{\begin{bmatrix} I_1(x, y) \\ \vdots \\ I_d(x, y) \end{bmatrix}}_{\triangleq \boldsymbol{g}_{xy} \in \mathbb{R}^d} \approx \underbrace{\begin{bmatrix} \boldsymbol{\ell}_1 & \ldots & \boldsymbol{\ell}_d \end{bmatrix}^T}_{\triangleq \boldsymbol{L}^T} \underbrace{(\alpha(x, y) \boldsymbol{n}(x, y))}_{\triangleq \boldsymbol{\rho}(x,y) \in \mathbb{R}^3}. \tag{2}$$

We could solve (2) for $\boldsymbol{\rho}(x, y)$ for each (x, y) when $d = 3$, but, in practice, when there is noise and our assumptions do not hold exactly, a more robust approach is to take $d > 3$ images in the **least-squares** problem

$$\hat{\boldsymbol{\rho}}(x, y) = \underset{\boldsymbol{r} \in \mathbb{R}^3}{\arg\min} \, \|\boldsymbol{g}_{xy} - \boldsymbol{L}^T \boldsymbol{r}\|_2^2. \tag{3}$$

We then approximate the surface normal $\boldsymbol{n}$ by the following normalized estimate:

$$\hat{\boldsymbol{n}}(x, y) \triangleq \frac{\hat{\boldsymbol{\rho}}(x, y)}{\|\hat{\boldsymbol{\rho}}(x, y)\|_2}. \tag{4}$$

Your task for this problem is to write a function called `compute_normals` that computes the unit-norm surface normal vectors for each pixel in a scene, given $\{I_i(x, y)\}$ and $\{\boldsymbol{\ell}_i\}$, by solving (3). and applying (4).
Hint: Julia's `normalize` and `mapslices` functions are useful.
In `Julia`, your file should be named `compute_normals.jl` and should contain the following function:

```
"""
    N = compute_normals(data, L)

In:
- `data` `m × n × d` matrix whose `d` slices contain `m × n` images
of a common scene under different lighting conditions

- `L` `3 × d` matrix whose columns are the lighting direction vectors
for the images in data, with `d ≥ 3`

Out:
- `N` `m × n × 3` matrix containing the unit-norm surface normal vectors
for each pixel in the scene
"""
function compute_normals(data, L)
```

Email your solution as an attachment to `eecs551@autograder.eecs.umich.edu`.
Note: The surface normals at each pixel are independent, so you can compute them all simultaneously by stacking the solutions to (3) for each pixel into a single matrix expression, using `reshape` to help. This solution can be expressed elegantly without using loops! (But it is OK to use loops if you prefer.)

Hint. You can test your code by applying it to synthetic data like the following.

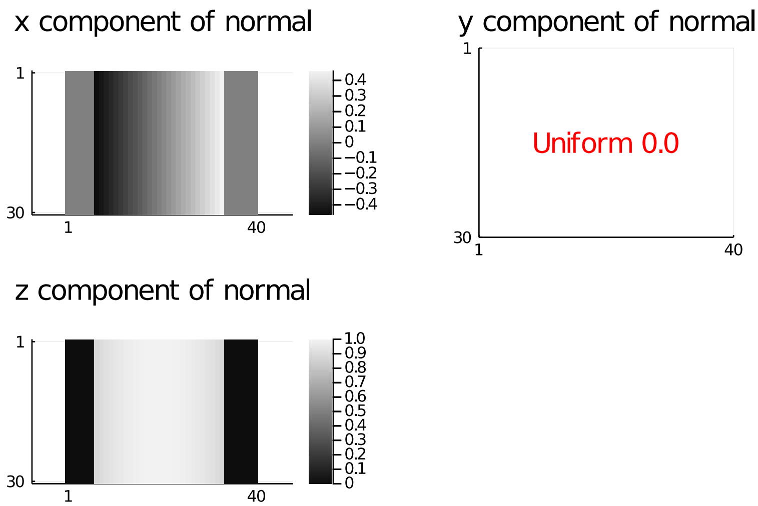
```
include("../auto/test/compute_normals.jl") # use your own path
using MIRTjim: jim

d = 8 # number of test images
phi = LinRange(-1,1,8)*pi/3 # light angles
L = [sin.(phi) 0*cos.(phi) cos.(phi)]' # light unit vectors
m,n = 40,30 # image size
x0 = m/2 # cylinder center coordinate
r0 = 0.7m # cylinder radius
afun = (x,y) -> abs(x-x0) < r0/2 # albedo alpha(x,y)
zfun = x -> abs(x) < r0 ? sqrt(r0^2 - x^2) : 0 # cylinder surface
normal = (x,y) -> [(x-x0)/r0, 0, zfun(x-x0)/r0] # surface normal
ntrue = [normal(x,y)*afun(x,y) for x in 1:m, y in 1:n]
ntrue = cat([map(x -> x[i], ntrue) for i=1:3]..., dims=3) # m × n × 3

data = zeros(Float32, m, n, d)
for l=1:d
    data[:,:,l] .= [L[:,l]'normal(x,y)*afun(x,y) for x in 1:m, y in 1:n]
end
jim(data, "$d images for different light positions")
nhat = compute_normals(data, L)
nhat[isnan.(nhat)] .= 0
jim(cat(ntrue, nhat, nhat-ntrue, dims=4),
    "true and estimated normals and error")

plot(
jim(nhat[:,:,1], "x component of normal"),
jim(nhat[:,:,2], "y component of normal"),
jim(nhat[:,:,3], "z component of normal"),
) # savefig("hp073.pdf")
```

This code simulates imaging part of the surface of a cylinder with different lighting positions and then applies `compute_normals` to the simulated data. If your code is working properly, then the three estimated normals should look like those in the following figure.



Pr. 9.

(Photometric stereo continued)

(This problem statement looks long, but it is actually quite easy because it uses tools you developed previously. You need not write any new code for it.)

For $f : \mathbb{R}^2 \mapsto \mathbb{R}$, from vector calculus, we can express the **surface normal vector** of f at (x, y) as

$$\boldsymbol{n}(x,y) = \frac{1}{\sqrt{1 + \left(\frac{\partial}{\partial x} f(x,y)\right)^2 + \left(\frac{\partial}{\partial y} f(x,y)\right)^2}} \begin{bmatrix} -\frac{\partial}{\partial x} f(x,y) \\ -\frac{\partial}{\partial y} f(x,y) \\ 1 \end{bmatrix} \triangleq \begin{bmatrix} n_1(x,y) \\ n_2(x,y) \\ n_3(x,y) \end{bmatrix}, \tag{5}$$

where $\frac{\partial}{\partial x} f$ and $\frac{\partial}{\partial y} f$ denote the partial derivatives of depth $f(x, y)$ with respect to x and y, respectively.
From (5), we can compute the partial derivatives as

$$\frac{\partial f(x,y)}{\partial x} = -\frac{n_1(x,y)}{n_3(x,y)}, \quad \frac{\partial f(x,y)}{\partial y} = -\frac{n_2(x,y)}{n_3(x,y)}. \tag{6}$$

In a previous HW you constructed a matrix $\boldsymbol{A}$ satisfying $\begin{bmatrix} \texttt{dfdx} \\ \texttt{dfdy} \end{bmatrix} = \boldsymbol{A}\, \texttt{fxy}$, where `dfdx` and `dfdy` denote the vectorized approximations of $\frac{\partial f(x,y)}{\partial x}$ and $\frac{\partial f(x,y)}{\partial y}$ and `fxy` denotes the vectorized approximation of $f(x, y)$. Using (6), we can compute `dfdx` and `dfdy` from our normal vectors, and, using our $\boldsymbol{A}$ matrix, we can obtain the surface corresponding to our normal vectors by solving the following **least squares** problem:

$$\texttt{fxy} \;=\; \underset{\boldsymbol{f} \in \mathbb{R}^{mn}}{\arg\min} \left\| \begin{bmatrix} \texttt{dfdx} \\ \texttt{dfdy} \end{bmatrix} - \boldsymbol{A}\boldsymbol{f} \right\|_2^2.$$

Download the `photometric_stereo_xy` notebook from the hw05 directory on Canvas, and copy your previous `compute_normals.jl` and `first_diffs_2d_matrix.jl` solution files into the same directory. Now use `Julia` to run that Jupyter notebook. If you have installed the necessary packages mentioned in the notebook, and if you have working versions of the `compute_normals` and `first_diffs_2d_matrix` functions, then the notebook will run properly and generate a surface view of the object.
After $f(x, y)$ is estimated, one can separately generate a stereolithography file (consisting of a collection of tessellated triangles) that can be rendered on a 3D display or printed by a 3D printer. Figure 2 depicts an actual 3D-print made from a solution to this problem generated with a Cube 3 printer.
Submit a screenshot of the surface plot produced by your Jupyter notebook.
It should look something like Fig. 2 but in color.

Figure 2: 3D printed reconstruction of the surface you will reconstruct in Problem 9. The 3D printing quality was intentionally set coarse so you can see how the printer constructed the shape from its level curves.

Pr. 10.

Hand-written digit classification using nearest subspace (discussion task)

This task illustrates how to use **subspace-based classification** with image features for classifying handwritten digits. For this machine learning problem, we focus on just the two digits "4" and "9," although the principles generalize to all digits. Previously we focused on "0" and "1" because the previous classification methods were less powerful. With the subspace approach, you can now tackle the harder problem of distinguishing "4" from "9."

Download the `task-2-classify-subspace.ipynb` jupyter notebook file from Canvas under this week's discussion folder and follow all instructions to complete the task. You may work individually, but we recommend that you work in pairs or groups of three.

When you are finished, upload your solutions to gradescope. Note that the submission for the task is separate from the rest of the homework because the task allows you to submit as a group. Only upload one submission per group! Whoever uploads the group submission should add all group members in gradescope, using the "View or edit group" option on the right-hand sidebar after uploading a PDF and matching pages. Make sure to add all group members, because this is how they will receive credit.

Non-graded problem(s) below

(Solutions will be provided for self check; do not submit to gradescope.)

Pr. 11.

Let $\mathcal{S}$ and $\mathcal{T}$ denote subspaces of a vector space $\mathcal{V}$. Prove or disprove the following.

(a) If $\mathcal{S}$ is a subset of $\mathcal{T}$ or vice versa, then the **union of subspaces** $\mathcal{S} \cup \mathcal{T}$ is a **subspace** of $\mathcal{V}$.

(b) If the **union of subspaces** $\mathcal{S} \cup \mathcal{T}$ is a **subspace** of $\mathcal{V}$, then $\mathcal{S}$ is a subset of $\mathcal{T}$ or vice versa.

Pr. 12.

Suppose $\boldsymbol{b} = \boldsymbol{\Phi z}$ for some vector $\boldsymbol{z}$, where $\boldsymbol{\Phi}$ is a **Parseval tight frame**.

(a) For $\delta \geq 0$, Express a solution of the **regularized least-squares** problem

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{\Phi x} - \boldsymbol{b}\|_2^2 + \delta^2 \|\boldsymbol{x}\|_2^2$$

in terms of $\boldsymbol{z}$ as simply as possibly.

(b) When is your solution unique?

(c) What value of δ would be the most appropriate in this special case where $\boldsymbol{\Phi}$ is a Parseval tight frame?

(d) What is the **condition number** of $\boldsymbol{\Phi}$ here?

Pr. 13.

(a) Show that if $\boldsymbol{\Phi}$ is a **tight frame** with frame bound α, then $\boldsymbol{\Phi}'\boldsymbol{\Phi} \preceq \alpha \boldsymbol{I}$, *i.e.*, $\alpha\boldsymbol{I} - \boldsymbol{\Phi}'\boldsymbol{\Phi}$ is **positive semidefinite**.

(b) What can you say about $\alpha\boldsymbol{I} - \boldsymbol{\Phi}\boldsymbol{\Phi}'$?

Pr. 14.

The following **Venn diagram** summarizes relationships related to **projection** operations.

$\boldsymbol{A}_1$ Square matrices	$\boldsymbol{A}_2$ Idempotent Projection matrix $\boldsymbol{P}^2 = \boldsymbol{P}$ diagonalizable eigenvalues $\{0, 1\}$	$\boldsymbol{A}_3$ Orthogonal projection matrix $\boldsymbol{P} = \boldsymbol{P}'$ normal	$\boldsymbol{A}_4$ Nonzero orthogonal projection matrix $\boldsymbol{P} = \boldsymbol{QQ}'$ where $\boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}$	$\boldsymbol{A}_5$ Diagonal with $\{0, 1\}$ elements	$\boldsymbol{I}$

Each of the categories shown above is a *strict superset* of the categories nested with in it.

Provide example matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_5$ that belong to the each of the categories above but *not* the next category nested within it. Try to provide the simplest possible example in each case.

Pr. 15.

Throughout the problems below you may reuse properties you derived in earlier parts as long as you refer back to the properties you use.

(a) Let $\boldsymbol{A} = \sum_{i=1}^r \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i'$ denote a **compact SVD** of $\boldsymbol{A}$ where r denotes the **rank** of $\boldsymbol{A}$.
Then in terms of the singular vectors $\{\boldsymbol{u}_i\}$ and $\{\boldsymbol{v}_i\}$:

(1) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} =$

(2) $\boldsymbol{P}_{\mathcal{R}^\perp(\boldsymbol{A})} =$

(3) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A}')} =$

(4) $\boldsymbol{P}_{\mathcal{R}^\perp(\boldsymbol{A}')} =$

(5) $\boldsymbol{P}_{\mathcal{N}(\boldsymbol{A})} =$

(6) $\boldsymbol{P}_{\mathcal{N}^\perp(\boldsymbol{A})} =$

(7) $\boldsymbol{P}_{\mathcal{N}(\boldsymbol{A}')} =$

(8) $\boldsymbol{P}_{\mathcal{N}^\perp(\boldsymbol{A}')} =$

(b) Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ denote a (full) SVD of $\boldsymbol{A}$ where r is the rank of $\boldsymbol{A}$.
Then in terms of the singular vectors $\{\boldsymbol{u}_i\}$ and $\{\boldsymbol{v}_i\}$:

(1) $\boldsymbol{P}^2_{\mathcal{R}(\boldsymbol{A})} = ?$

(2) $\boldsymbol{P}^3_{\mathcal{R}(\boldsymbol{A})} = ?$

(3) $\boldsymbol{P}^k_{\mathcal{R}(\boldsymbol{A})} =?$ for a positive integer k

(4) $(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})})\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} =?$

(5) $(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})})^k =?$ for a positive integer k

(6) $(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})})^k \boldsymbol{P}^j_{\mathcal{R}(\boldsymbol{A})} = ?$ for positive integers j and k

(7) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} - \boldsymbol{P}'_{\mathcal{R}(\boldsymbol{A})} = ?$

(8) $\operatorname{rank}(\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})}) = ?$

(9) $\operatorname{rank}(\boldsymbol{P}_{\mathcal{R}^\perp(\boldsymbol{A})}) = ?$

(10) $\operatorname{rank}(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})}) =?$

(c) Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ be an SVD of $\boldsymbol{A}$ and r be the rank of $\boldsymbol{A}$.
Then in terms of the singular vectors $\{\boldsymbol{u}_i\}$ and $\{\boldsymbol{v}_i\}$:

(1) $\|\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})}\|_F = ?$

(2) $\|\boldsymbol{P}^k_{\mathcal{R}(\boldsymbol{A})}\|_F = ?$ for a positive integer k

(3) $\|\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})}\|_F = ?$

(4) $\|(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})})^k\|_F = ?$ for a positive integer k

(5) $\|\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})}(\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})})\|_F = ?$

(6) $\prod_{i=1}^k (\boldsymbol{I} - \boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_i)}) - \boldsymbol{P}_{\mathcal{R}^\perp(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_k)} = ?$ for any positive integer k

(d) Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ be an SVD of $\boldsymbol{A}$ and r be the rank of $\boldsymbol{A}$. Find expressions for both a **compact SVD** and a **full SVD** for each of the following matrices.

(1) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})} = ?$

(2) $\boldsymbol{P}^\perp_{\mathcal{R}(\boldsymbol{A})} = ?$

(3) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1)}\boldsymbol{A} = ?$

(4) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_k)}\boldsymbol{A} = ?$ for $k = 1, \ldots, r$

(5) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_k)}\boldsymbol{A} = ?$ for $r + 1 \leq k \leq \min(m, n)$

(6) $\boldsymbol{A}\boldsymbol{P}_{\mathcal{R}(\boldsymbol{v}_1)} = ?$

(7) $\boldsymbol{A}\boldsymbol{P}_{\mathcal{R}(\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k)} = ?$ for $k = 1, \ldots, r$

(8) $\boldsymbol{A}\boldsymbol{P}_{\mathcal{R}(\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k)} = ?$ for $1 \leq k \leq \min(m, n)$ (Hint: think carefully about r.)

(9) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_k)}\boldsymbol{A}\boldsymbol{P}_{\mathcal{R}(\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k)} = ?$ for $k = 1, \ldots, r$

(10) $\boldsymbol{P}_{\mathcal{R}(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_j)}\boldsymbol{A}\boldsymbol{P}_{\mathcal{R}(\boldsymbol{v}_1,\ldots,\boldsymbol{v}_k)} = ?$ for $1 \leq j, k \leq \min(m, n)$

Pr. 16.
Let $\boldsymbol{A}$ be an $m \times n$ matrix with SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$. Let $\boldsymbol{u}_i$ denote the ith column of $\boldsymbol{U}$, and $\boldsymbol{v}_i$ denote the ith column of $\boldsymbol{V}$. Let σ_i is the ith diagonal element of $\boldsymbol{\Sigma}$, and let $\boldsymbol{A}$ have rank r. Let $\boldsymbol{I}_d$ denote the $d \times d$ identity matrix.
Simplify the following **pseudo-inverse** expressions as much as possible.

(a) (1) $(\boldsymbol{u}_i)^+ =$

(2) $(\boldsymbol{u}_i \boldsymbol{u}_i')^+ =$

(3) $(\boldsymbol{u}_i \boldsymbol{v}_i')^+ =$

(4) $(\sigma_i \boldsymbol{u}_i \boldsymbol{v}_i')^+ =$

(b) (1) $\boldsymbol{U}^+ =$

(2) $\boldsymbol{V}^+ =$

(3) $\boldsymbol{U}\boldsymbol{U}^+ =$

(4) $\boldsymbol{V}\boldsymbol{V}^+ =$

(5) $\boldsymbol{\Sigma}^+ =$

(6) $(\boldsymbol{U}\boldsymbol{\Sigma})^+ =$

(7) $(\boldsymbol{\Sigma}\boldsymbol{V}')^+ =$

(8) $\boldsymbol{A}^+ = (\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}')^+ =$

(9) $(\boldsymbol{U}\boldsymbol{V}')^+ =$

(c) (1) For $1 \leq k \leq m$, $\boldsymbol{U}[:, 1:k]^+ =$

(2) For $1 \leq k \leq n$, $\boldsymbol{V}[:, 1:k]^+ =$

(3) For $1 \leq k \leq \min\{m, n\}$. $\boldsymbol{\Sigma}[1:k, 1:k]^+ =$

(4) For $1 \leq k \leq m$, $\boldsymbol{U}[:, 1:k]\boldsymbol{U}[:, 1:k]^+ =$

(5) For $1 \leq k \leq n$, $\boldsymbol{V}[:, 1:k]\boldsymbol{V}[:, 1:k]^+ =$

(6) For $1 \leq k \leq m$, $\boldsymbol{U}[:, 1:k]^+\boldsymbol{U}[:, 1:k] =$

(7) For $1 \leq k \leq n$, $\boldsymbol{V}[:, 1:k]^+\boldsymbol{V}[:, 1:k] =$

(d) (1) For $1 \leq k \leq m$, $\boldsymbol{I}_m - \boldsymbol{U}[:, 1:k]\boldsymbol{U}[:, 1:k]^+ =$

(2) For $1 \leq k \leq n$, $\boldsymbol{I}_n - \boldsymbol{V}[:, 1:k]\boldsymbol{V}[:, 1:k]^+ =$

(3) For $1 \leq k \leq m$, $\boldsymbol{I}_k - \boldsymbol{U}[:, 1:k]^+\boldsymbol{U}[:, 1:k] =$

(4) For $1 \leq k \leq n$, $\boldsymbol{I}_k - \boldsymbol{V}[:, 1:k]^+\boldsymbol{V}[:, 1:k] =$

(e) (1) For $1 \leq k \leq r$, $(\boldsymbol{U}[:, 1:k]\boldsymbol{\Sigma}[1:k, 1:k]\boldsymbol{V}[:, 1:k]')^+ =$

(2) $(\boldsymbol{A}')^+ =$

(3) $(\boldsymbol{A}^+)' =$

(4) For $\alpha \neq 0$, $(\alpha\boldsymbol{A})^+ =$

(5) $(\boldsymbol{A}\boldsymbol{A}^+)' =$

(6) $(\boldsymbol{A}^+\boldsymbol{A})' =$

(f) (1) $\boldsymbol{A}\boldsymbol{A}^+ =$

(2) $\boldsymbol{A}^+\boldsymbol{A} =$

(3) $\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} =$

(4) $\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ =$

(5) $(\boldsymbol{P}_{\mathcal{R}(\boldsymbol{A})})^+ =$

(g) (1) $(\boldsymbol{A}'\boldsymbol{A})^+\boldsymbol{A}' =$

(2) $\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{A}')^+ =$

(3) If $r = n$, then $\boldsymbol{A}^+\boldsymbol{A} =$

(4) If $r = m$, then $\boldsymbol{A}\boldsymbol{A}^+ =$

(5) If $r = n$, then $(\boldsymbol{A}'\boldsymbol{A})^{-1}\boldsymbol{A}' =$

(6) If $r = m$, then $\boldsymbol{A}'(\boldsymbol{A}\boldsymbol{A}')^{-1} =$

Pr. 17.

(a) Show that if $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})$ are all **convex functions** of $\boldsymbol{x}$, then the following are also **convex functions**:

(1) $\sum_{i=1}^n f_i(\boldsymbol{x})$.

(2) $\sum_{i=1}^n w_i f_i(\boldsymbol{x})$ for scalars $w_i \geq 0$.

(3) Provide an example wherein w_i being negative breaks the convexity.

(4) $\max\{f_1, f_2, \ldots, f_n\}$.

(b) Which of the following cost functions are convex functions of $\boldsymbol{x}$? Assume that $\boldsymbol{A}$ and $\boldsymbol{D}$ are matrices and $\boldsymbol{b}$ is a vector with compatible dimensions. Justify your answer: using the results from a previous part may help.

(1) $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \|\boldsymbol{D}\boldsymbol{x}\|_2^2$.

(2) $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \|\boldsymbol{D}\boldsymbol{x}\|_1$.

(3) $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 - \|\boldsymbol{D}\boldsymbol{x}\|_2^2$.

(4) $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 - \|\boldsymbol{D}\boldsymbol{x}\|_1$.