Pr. 1.
Determine how **eigenvalues** and **eigenvectors** of

$$\boldsymbol{B} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{A} \\ \boldsymbol{A}' & \boldsymbol{0} \end{bmatrix},$$

are related to **singular values** and **singular vectors** of $\boldsymbol{A} \in \mathbb{F}^{N \times N}$. In other words, if $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$, then express eigenvalues and eigenvectors of $\boldsymbol{B}$ in terms of $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$.
Optional: write a unitary eigendecomposition of $\boldsymbol{B}$ in matrix form.
Hint: Start with some numerical experiments using self-generated $\boldsymbol{A}$ matrices and form a conjecture. The eigenvalues and singular values are intimately related.

Pr. 2.
Let $\boldsymbol{A} = \boldsymbol{x}\boldsymbol{y}^\top$, where neither $\boldsymbol{x}$ nor $\boldsymbol{y}$ is $\boldsymbol{0}$.

(a) How many **linearly independent** columns does $\boldsymbol{A}$ have? What is the **rank** of $\boldsymbol{A}$?

(b) With `Julia` open a Jupyter notebook and type in (cut and paste):

```
using LinearAlgebra: svdvals
using Plots; gr() # (choose your favorite back-end)
n = 100
x = randn(n); y = randn(n); A = x*y'
s = svdvals(A)
plot(s, marker = :circle, yscale = :log10, label= "",
    title = "singular values: log scale", xlabel="i",
    ylabel="σᵢ") # type: \sigma[tab]\_i[tab]
```

(Ideally the ylabel should look like σ_i, but if it does not, it is OK.)

How many numerically computed singular values of $\boldsymbol{A}$ are non-zero?
What answer do you get when you type `rank(A)` ?
Turn in your plot of the singular values and the answers.

You will have just seen that a theoretically rank-one matrix will have multiple non-zero singular values when expressed in machine precision arithmetic; this property is due to roundoff errors (finite numerical precision). (Round-off errors should not be thought of as "wrong answers," but just a fact of real-life computing to be understood and not feared.)

(c) To help locate the code for the `rank` function, you can type this into `Julia`: `which(rank, (Matrix,))`

From the output you can (with a little web search) find the source code here (because all of `Julia` is open source): `https://github.com/JuliaLang/julia/blob/master/stdlib/LinearAlgebra/src/generic.jl`

Examine the code for the `rank` function and write down the formula used to determine the threshold (aka tolerance) below which, due to roundoff errors, the code considers the singular value to be "zero."
Determine the numerical value of the threshold when $\boldsymbol{A} = [9\ 9]$.

Pr. 3.
Use an **SVD** of a square matrix $\boldsymbol{A}$ to describe a **polar factorization** $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{S}$ where $\boldsymbol{Q}$ is orthogonal and $\boldsymbol{S} = \boldsymbol{S}' \succeq \boldsymbol{0}$, *i.e.*, $\boldsymbol{S}$ is **positive semi-definite**. Express $\boldsymbol{Q}$ and $\boldsymbol{S}$ in terms of an SVD $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$ of $\boldsymbol{A}$. (This matrix decomposition is analogous to the polar form $z = re^{i\theta}$ of a complex scalar z, where r is like $\boldsymbol{S}$ and $e^{i\theta}$ is like $\boldsymbol{Q}$.)
Hint: $\boldsymbol{V}'\boldsymbol{V} = \boldsymbol{I}$.

Pr. 4.
Prove that the orthogonal complement of the range of a matrix $\boldsymbol{A}$ equals the nullspace of $\boldsymbol{A}'$:

$$\mathcal{R}^\perp(\boldsymbol{A}) = \mathcal{N}(\boldsymbol{A}').$$

Pr. 5.
When $\boldsymbol{D}$ is an $m \times n$ (rectangular) diagonal matrix, its **pseudo-inverse** $\boldsymbol{D}^+$ is an $n \times m$ (rectangular) diagonal matrix whose non-zero entries are the reciprocals $1/d_k$ of the non-zero diagonal entries of $\boldsymbol{D}$. Thus a matrix $\boldsymbol{A}$ having SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}'$ has $\boldsymbol{A}^+ = \boldsymbol{V}\boldsymbol{\Sigma}^+\boldsymbol{U}'$. Working with just this definition, determine by hand (experimenting / checking with code is OK) the pseudo-inverse of

(a) $\boldsymbol{A} = \boldsymbol{x}\boldsymbol{y}'$, where neither $\boldsymbol{x}$ nor $\boldsymbol{y}$ is $\boldsymbol{0}$,

(b) $\boldsymbol{A} = \boldsymbol{x}\boldsymbol{x}'$, where $\boldsymbol{x} \neq \boldsymbol{0}$.

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

(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 $||\boldsymbol{x}^* - \boldsymbol{z}^*||_2$ 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 $||\boldsymbol{x}^* - \boldsymbol{z}^*||_2$ here as $\delta \to 0$?

Pr. 7.
Let $f(t) = 0.5\, e^{0.8t}, t \in [0, 2]$.

(a) Suppose 16 exact measurements of $f(t)$ are available to you, taken at the times t in the following array T:

$$T = [2 \cdot 10^{-3}, \quad 0.136, \quad 0.268, \quad 0.402, \quad 0.536, \quad 0.668, \quad 0.802, \quad 0.936,$$
$$1.068, \quad 1.202, \quad 1.336, \quad 1.468, \quad 1.602, \quad 1.736, \quad 1.868, \quad 2.000].$$

Use `Julia` to generate 16 exact measurements: $y_i = f(t_i),\ i = 1, \ldots, 16, \quad t_i \in T$.

Now determine the coefficients of the **least square** error polynomial approximation of the data for

(a) a polynomial of degree 15: $p_{15}(t)$;

(b) a polynomial of degree 2: $p_2(t)$.

Compare the quality of the two approximations by plotting $y(t), p_{15}(t)$ and $p_2(t)$ for all $t \in T$. To see how well we are approximating the function on the interval, also plot $f(t), p_{15}(t)$ and $p_2(t)$ in the interval $[0, 2]$. (Pick a very fine grid for the interval, *e.g.*, `t = [0:1000]/500`.) Make the y-axis range equal [-1,4] by using the `ylim=(-1,4)` option. Submit your plot and also summarize the results qualitatively in one or two sentences.

(b) Now suppose the measurements are affected by some noise. Generate the measurements using $y_i = f(t_i) + e_i,\ i = 1, \ldots, 16, \quad t_i \in T$, where you generate the vector of noise values as follows:

```
using Random
Random.seed!(0); e = randn(length(T))
```

Determine the coefficients of the least square error polynomial approximation of the (noisy) measurements for

(a) a polynomial of degree 15: $p_{15}(t)$;

(b) a polynomial of degree 2: $p_2(t)$.

Compare the two approximations as in part (a). Again make the y-axis range equal [-1,4] by using the `ylim=(-1,4)` option. Submit your plot and also summarize the results qualitatively in a couple of sentences, including comparing the behavior in (a) and (b).

(c) Report the values of the **residual** norm $\|\boldsymbol{A}\hat{\boldsymbol{x}} - \boldsymbol{y}\|_2$ for the polynomial fits of degree 2 and degree 15 for both the noiseless and noisy cases above.

Explain why the residual norm for degree 2 is smaller or larger than that of degree 15.

Pr. 8.
(Photometric stereo)
In this problem you implement another tool needed for an algorithm from computer vision 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 sneak peek at what you'll be able to do after completing all the pieces.

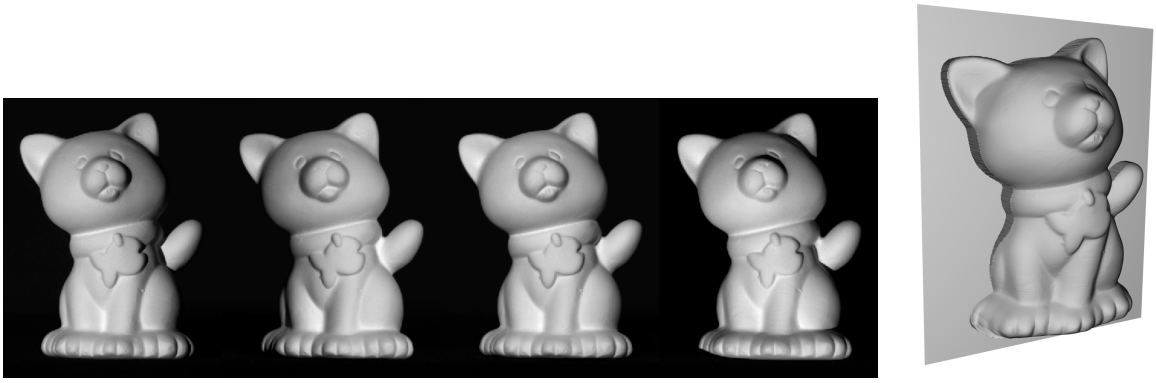

Figure 1: Photometric stereo. Left: 2D images of a common scene under different lighting conditions. Right: 3D surface reconstruction computed from the input images.

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 a surface $z = f(x, y)$ where (x, y) denotes coordinates on the table and z is 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 $\ell_1, \ldots, \ell_d$. For any (x, y), we can stack (1) into an **overdetermined** system of equations:

$$\underbrace{\begin{bmatrix} I_1(x,y) \\ \vdots \\ I_d(x,y) \end{bmatrix}}_{\triangleq \boldsymbol{g}_{xy}} \approx \underbrace{\begin{bmatrix} \ell_1 & \ldots & \ell_d \end{bmatrix}^T}_{\triangleq \boldsymbol{L}^T} \underbrace{(\alpha(x,y)\boldsymbol{n}(x,y))}_{\triangleq \boldsymbol{\rho}(x,y)}. \tag{2}$$

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

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

and approximate the surface normal $\boldsymbol{n}$ by $\boldsymbol{n}^*$

$$\boldsymbol{n}^*(x, y) = \frac{\boldsymbol{\rho}(x, y)}{\|\boldsymbol{\rho}(x, y)\|_2}. \tag{4}$$

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 by solving (3).
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:

```
function compute_normals(data, L)
#
# Syntax:       N = compute_normals(data, L)
#
# Inputs:       data is an m x n x d matrix whose d slices contain m x n images
#               of a common scene under different lighting conditions
#
#               L is a 3 x d matrix whose columns are the lighting direction
#               vectors for the images in data, with d >= 3
#
# Outputs:      N is an m x n x 3 matrix containing the unit-norm
#               surface normal vectors for each pixel in the scene
#
```

Submit your solution to the autograder by emailing it 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 (3) for each pixel into a single matrix-valued least-squares problem. This can be expressed very elegantly without using loops! (But it is OK to use loops if you prefer.)

Optional problem(s) below
(not graded, but solutions will be provided for self check)

Pr. 9.
Suppose $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ with $M \geq N$ has **full column rank**, *i.e.*, $\text{rank}(\boldsymbol{A}) = N$.
Show, using an SVD of $\boldsymbol{A}$, that $\boldsymbol{A}^+ = (\boldsymbol{A}'\boldsymbol{A})^{-1}\boldsymbol{A}'$. When $\boldsymbol{A}$ is known to be full column rank, one could use this direct formula to compute the **pseudo-inverse** instead of employing an SVD.

Pr. 10.
Consider the following set of three measurements (x_i, y_i): $(1, 2)$, $(2, 1)$, $(3, 3)$.

(a) Find the best (in the 2-norm sense) line of the form $y = \alpha x + \beta$ that fits this data.

(b) Find the best (in the 2-norm sense) line of the form $x = \gamma y + \delta$ that fits this data.
Hint: Re-use your answer from (a).
