Pr. 1.
For a matrix $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, consider the following **matrix norms**:

- $\|\boldsymbol{A}\|_1 = \max\limits_{\boldsymbol{x}\neq 0} \frac{\|\boldsymbol{Ax}\|_1}{\|\boldsymbol{x}\|_1}$ where $\|\boldsymbol{Ax}\|_1$ and $\|\boldsymbol{x}\|_1$ are the vector 1-norms, *i.e.*, $\|\boldsymbol{x}\|_1 = \sum |x_i|$.
- $\|\boldsymbol{A}\|_2 = \max\limits_{\boldsymbol{x}\neq 0} \frac{\|\boldsymbol{Ax}\|_2}{\|\boldsymbol{x}\|_2}$ where $\|\boldsymbol{Ax}\|_2$ and $\|\boldsymbol{x}\|_2$ are the vector 2-norms, *i.e.*, $\|\boldsymbol{x}\|_2 = \left(\sum |x_i|^2\right)^{1/2}$.
- $\|\boldsymbol{A}\|_\infty = \max\limits_{\boldsymbol{x}\neq 0} \frac{\|\boldsymbol{Ax}\|_\infty}{\|\boldsymbol{x}\|_\infty}$ where $\|\boldsymbol{Ax}\|_\infty$ and $\|\boldsymbol{x}\|_\infty$ are the vector ∞-norms, *i.e.*, $\|\boldsymbol{x}\|_\infty = \max\limits_i |x_i|$.
- $\|\boldsymbol{A}\|_{\mathrm{F}} = \left(\sum\limits_i \sum\limits_j |a_{ij}|^2\right)^{1/2}$ is the Frobenius norm that we have seen before.
- $\|\boldsymbol{A}\|_* = \sum\limits_{i=1}^{r} \sigma_i$ is the nuclear norm, the sum of the singular values.

Show that the following inequalities hold when $\boldsymbol{A}$ has rank r:

(a) $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_F$ Hint: work with singular values for (a)-(d).

(b) $\|\boldsymbol{A}\|_F \leq \sqrt{r}\|\boldsymbol{A}\|_2$

(c) $\|\boldsymbol{A}\|_F \leq \|\boldsymbol{A}\|_*$ Hint: consider using the convexity of $f(x) = x^2$, like in **Jensen's inequality**.

(d) $\|\boldsymbol{A}\|_* \leq \sqrt{r}\|\boldsymbol{A}\|_F$

(e) $\|\boldsymbol{A}\|_\infty \leq \sqrt{n}\|\boldsymbol{A}\|_2$ Hint: first show that $\boldsymbol{x} \in \mathbb{R}^n \Rightarrow \frac{1}{\sqrt{n}}\|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_2$.

(f) $\|\boldsymbol{A}\|_2 \leq \sqrt{m}\|\boldsymbol{A}\|_\infty$ Hint: use the previous hint.

(g) $\frac{1}{\sqrt{m}}\|\boldsymbol{A}\|_1 \leq \|\boldsymbol{A}\|_2 \leq \sqrt{n}\|\boldsymbol{A}\|_1$ Hint: Assume the preceding inequalities all hold.

These inequalities can provide upper and lower bounds for the largest singular value via simple operations on the elements of the matrix. One concrete application where this is invaluable is in determining the step size for gradient descent algorithms; there the largest singular value (which equals $\|\boldsymbol{A}\|_2$) has a pivotal role.

Pr. 2.
(Additivity of nuclear norm)
Let $\boldsymbol{A}$ and $\boldsymbol{B}$ be two matrices of the same size. Because the **nuclear norm** is a matrix norm, we know (by the triangle inequality for norms) that $\|\boldsymbol{A}+\boldsymbol{B}\|_* \leq \|\boldsymbol{A}\|_* + \|\boldsymbol{B}\|_*$. This problem provides a sufficient condition for equality.
Show that if $\boldsymbol{AB}' = \boldsymbol{0}$ and $\boldsymbol{A}'\boldsymbol{B} = \boldsymbol{0}$ then $\|\boldsymbol{A}+\boldsymbol{B}\|_* = \|\boldsymbol{A}\|_* + \|\boldsymbol{B}\|_*$.
Hint: construct a valid SVD for $\boldsymbol{A}+\boldsymbol{B}$ from an SVD of $\boldsymbol{A}$ and an SVD of $\boldsymbol{B}$.

Pr. 3.

The final layer in many artificial neural networks is often "dense" or "fully connected" and often is affine or linear. This problem focuses on the linear case.

We are given training data $(\boldsymbol{x}_n, y_n), n = 1, \ldots, N$ consisting of pairs of features $\boldsymbol{x}_n \in \mathbb{R}^M$ and responses $y_n \in \mathbb{R}$. A linear artificial neuron makes a prediction simply by computing the inner product of an input feature $\boldsymbol{x} \in \mathbb{R}^M$ with a weight vector $\boldsymbol{w} \in \mathbb{R}^M$, *i.e.*, $\hat{y}_n = \boldsymbol{w}'\boldsymbol{x}_n$. We want to train the weight vector $\boldsymbol{w}$ to minimize the average loss over the training data by solving the following optimization problem:

$$\hat{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}), \quad L(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N} \ell(y_n, \hat{y}_n), \quad \hat{y}_n = \boldsymbol{w}'\boldsymbol{x}_n.$$

Determine analytically the optimal weight vector $\hat{\boldsymbol{w}}$ in the case where the loss function is the squared error

$$\ell(y_n, \hat{y}_n) = \frac{1}{2}(y_n - \hat{y}_n)^2.$$

State any conditions needed on N for your answer to be valid.

Hint. You should be able to set this up as a **linear least-squares** problem and express your answer in terms of the training data feature correlation matrix $\boldsymbol{K}_x = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n\boldsymbol{x}_n'$ and the cross-correlation between the training data features and responses $\boldsymbol{K}_{yx} = \frac{1}{N}\sum_{n=1}^{N} y_n\boldsymbol{x}_n'$.

Pr. 4.

This problem examines important properties of **positive semidefinite** and **positive definite** matrices.

Recall that the lecture notes show that $\boldsymbol{B}'\boldsymbol{B} \succeq \boldsymbol{0}$ for any matrix $\boldsymbol{B}$.

(a) (Optional) Show that $\boldsymbol{B}'\boldsymbol{B} \succ \boldsymbol{0}$ if $\boldsymbol{B}$ has full column rank.

(b) (Optional) Show that $\boldsymbol{A} \succ \boldsymbol{0}$ implies $\boldsymbol{A}$ is invertible.

(c) (Optional) Show $\boldsymbol{A} \succeq \boldsymbol{0}$ and $\boldsymbol{B} \succeq \boldsymbol{0} \Rightarrow \boldsymbol{A} + \boldsymbol{B} \succeq \boldsymbol{0}$.

(d) $\boldsymbol{A} \succ \boldsymbol{0}$ and $\boldsymbol{B} \succeq \boldsymbol{0} \Rightarrow \boldsymbol{A} + \boldsymbol{B} \succ \boldsymbol{0}$.

(e) Show $\boldsymbol{A}'\boldsymbol{A} + \boldsymbol{B}'\boldsymbol{B}$ is invertible if $\boldsymbol{B}$ has full column rank
(This property is relevant to **regularized least-squares** problems.)

(f) Show $\boldsymbol{A}'\boldsymbol{A} + \boldsymbol{B}'\boldsymbol{B}$ is invertible if $\mathcal{N}(\boldsymbol{A}) \cap \mathcal{N}(\boldsymbol{B}) = \{\boldsymbol{0}\}$, i.e., if $\boldsymbol{A}$ and $\boldsymbol{B}$ have disjoint null spaces other than $\boldsymbol{0}$.

(g) (Optional) Show that $\boldsymbol{A} \succ \boldsymbol{0},\ \boldsymbol{B} \succ \boldsymbol{0} \Rightarrow \boldsymbol{BAB} \succ \boldsymbol{0}$.

Pr. 5.

(Iterative method for solving non-negative least squares problems)

In many applications, we want to fit a system of equations and we know that the variables being fit are non-negative. For example if we are trying to fit $F = ma$ and we know that the mass m is positive. Then we might want to solve the **non-negative least squares** (NNLS) problem:

$$\boldsymbol{x}_{\mathsf{NNLS}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 \text{ subject to } \boldsymbol{x} \geq \boldsymbol{0},$$

where we have explicitly imposed the additional constraint that the solution must be non-negative. (Here the shorthand $\boldsymbol{x} \geq \boldsymbol{0}$ means each element of $\boldsymbol{x}$ is non-negative.) If we solve the least-squares problem without the non-negativity constraint in the usual way and happen to obtain a non-negative solution then, by definition, that solution is also the optimal NNLS solution. However, typically the unconstained solution will have negative values. Setting the negatives to zero after the fact usually does *not* solve the stated problem.

We can solve optimization problems of the form

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \|\boldsymbol{Ax} - \boldsymbol{b}\|_2 \text{ subject to } \boldsymbol{x} \in \mathcal{C},$$

where $\mathcal{C}$ denotes a **convex set**, via a **projected gradient descent** method that combines the usual **gradient descent** update for solving an unconstrained least squares problem with an additional **projection** (onto the convex set) step. For the NNLS problem, this projection step enforces (or ensures) non-negativity. The projected gradient descent method has an iteration given by

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

and it will converge to a minimizer of the cost function if $0 < \mu < 2/\sigma_1^2(\boldsymbol{A})$ and $P_{\mathcal{C}}$ is the projection onto the convex set. Here $\mathcal{C}$ is the convex set of non-negative real valued vectors, so using

$$P_{\mathcal{C}}(\boldsymbol{z}) = \max(0, z),$$

implemented via `max.(0,z)` in `Julia`, yields the desired iterative algorithm.

(a) Implement this algorithm as a function named `nnlsgd`. The required inputs are the matrix $\boldsymbol{A}$ and the vector $\boldsymbol{b}$. Optional inputs are the step size μ, and the initial guess $\boldsymbol{x}_0$. The function should compute a practical value for the step size μ if none is provided. This value should be practical to compute even for very a large matrix $\boldsymbol{A}$.

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

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

Performs projected gradient descent to solve the least squares problem:
``\\argmin_{x \\geq 0} 0.5 \\| b - A x \\|_2`` with nonnegativity constraint.

In:
- `A` `m x 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 nnlsgd(A, b ; mu::Real=0, x0=zeros(size(A,2)), nIters::Int=200)
```

Submit your solution to the autograder by emailing it as an attachment to `eecs551@autograder.eecs.umich.edu`.

(b) Apply your algorithm to the setup described next, with the $\boldsymbol{A}$, $\boldsymbol{b}$ and $\boldsymbol{x}_0$ therein.

```
using Random: seed!
using Statistics: mean
seed!(0)
m = 100; n = 50; sigma = 0.3
A = randn(m,n)
xtrue = rand(n) # note that xtrue is non-negative
b = A * xtrue + sigma * randn(m)
x0 = A \ b; x0[x0 .<= 0] .= mean(x0[x0 .> 0]) # reasonable initial guess
```

Run 100 iterations of projected gradient descent with $\mu = 1/\sigma_1^2$ to approximately solve the NNLS problem. Submit to gradescope the first three values of your $\boldsymbol{x}_{100}$, *i.e.*, `x[1:3]`

Hint. `x[4]` is about 0.782

(c) Use the `Julia` package `Optim` to compute the NNLS solution for the setup above.

```
using Optim # you will likely need to add this package
using LinearAlgebra: norm
lower = zeros(n)
```

```
upper = fill(Inf, (n,))
inner_optimizer = GradientDescent()
f = x -> 1/2 * norm(A * x - b)^2 # cost function
function grad!(g, x) # its gradient
    g[:] = A' * (A * x - b)
end
results = optimize(f, grad!, lower, upper, x0,
    Fminbox(inner_optimizer), Optim.Options(g_tol=1e-12))
xnnls = results.minimizer
```

For documentation about this solver, see
`http://julianlsolvers.github.io/Optim.jl/stable/#user/minimization/#box-constrained-optimization`

Submit these 3 values to gradescope: `xnnls[5:7]`

Hint. `xnnls[8]` is about 0.796

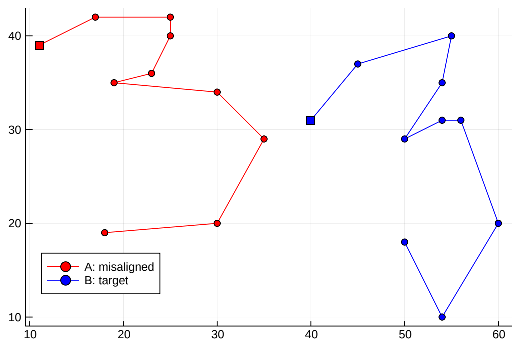
(d) For the above setup, compare the iterates $\{\boldsymbol{x}_k\}$ produced by your algorithm to NNLS "solution" for four different values of μ: $0.1/\sigma_1^2, 0.5/\sigma_1^2, 1.0/\sigma_1^2, 1.9/\sigma_1^2$.

Specifically, plot $\log(\|\boldsymbol{x}_{\mathsf{NNLS}} - \boldsymbol{x}_k\|)$ as a function of $k \in [0, 100]$ using $\boldsymbol{A}$ and $\boldsymbol{b}$ generated as above. Submit one plot with four curves on it for the four μ values, with an appropriate legend. Hint. The curve for the 1.9 case should decrease the fastest.

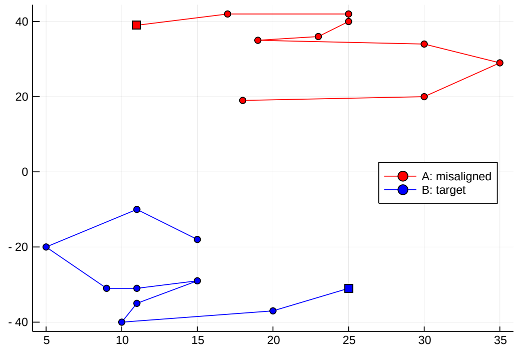
Pr. 6.
(Procrustes analysis)
In this problem you will align shapes using **Procrustes analysis**.
You can visualize the two datasets in `Julia` via

```
using Plots

# load data if needed (you must have web access to run this)
if !@isdefined(url)
    url = "http://web.eecs.umich.edu/~fessler/course/551/data/digits/"
    url1a = url * "digit1a2x10int16.dat"
    url1b = url * "digit1b2x10int16.dat"
    url2a = url * "digit2a2x10int16.dat"
    url2b = url * "digit2b2x10int16.dat"
    npoint = 10
    A = Array{Int16}(undef, (2,npoint))
    B = Array{Int16}(undef, (2,npoint))
    read!(download(url1a), A)
    read!(download(url1b), B)
end

# plot points with first point marked for clarity
plot(A[1,:], A[2,:], color=:red, marker=:circle, label="A: misaligned")
scatter!([A[1,1]], [A[2,1]], color=:red, marker=:square, label="")
plot!(B[1,:], B[2,:], color=:blue, marker=:circle, label="B: target")
scatter!([B[1,1]], [B[2,1]], color=:blue, marker=:square, label="")
plot!(legend=:bottomleft, aspect_ratio=:equal)
```

In the above, $\boldsymbol{A}$ and $\boldsymbol{B}$ are $2 \times n$ matrices whose columns contain the (2D) coordinates of n points on two shapes that we would like to align. (The figure shows clearly that they are not aligned initially.)



(a) Dataset 1



(b) Dataset 2

In general, suppose $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d \times n}$ and that we would like to rotate, translate and scale $\boldsymbol{A}$ to best align with $\boldsymbol{B}$. One way to express this mathematically is by computing the matrix $\hat{\boldsymbol{A}} \in \mathbb{R}^{d \times n}$ defined as

$$\hat{\boldsymbol{A}} = \alpha \boldsymbol{Q} \left(\boldsymbol{A} - \boldsymbol{\lambda} \mathbf{1}_n^T \right) + \boldsymbol{\mu} \mathbf{1}_n^T, \tag{1}$$

where $\boldsymbol{Q}$ is a $d \times d$ orthogonal (rotation) matrix, $\boldsymbol{\lambda}, \boldsymbol{\mu} \in \mathbb{R}^d$ are translation vectors, and $\alpha \in \mathbb{R}$ is a scalar. Geometrically, (1) corresponds to translating each column of $\boldsymbol{A}$ by $-\boldsymbol{\lambda}$, rotating by $\boldsymbol{Q}$, scaling by α, and then translating again by $\boldsymbol{\mu}$. However, note that having *both* $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ is redundant, because we can replace $\boldsymbol{\lambda} \to \boldsymbol{\lambda} + \boldsymbol{Q}' \boldsymbol{\Delta} / \alpha$ and $\boldsymbol{\mu} \to \boldsymbol{\mu} - \boldsymbol{\Delta}$ for any $\boldsymbol{\Delta}$ without changing $\hat{\boldsymbol{A}}$. Therefore, without loss of generality, set

$$\boldsymbol{\lambda} \triangleq \boldsymbol{\mu}_A \triangleq \frac{1}{n} \boldsymbol{A} \mathbf{1}_n = \frac{1}{n} \sum_{k=1}^{n} A[:, k],$$

the centroid of (the columns of) $\boldsymbol{A}$. Then choose the remaining parameters by solving

$$\alpha^\star, \ \boldsymbol{\mu}^\star, \ \boldsymbol{Q}^\star = \underset{\alpha \in \mathbb{R}, \ \boldsymbol{\mu} \in \mathbb{R}^d, \ \boldsymbol{Q} : \ \boldsymbol{Q}^T \boldsymbol{Q} = \boldsymbol{I}_d}{\arg\min} \left\| \boldsymbol{B} - \alpha \boldsymbol{Q} \left(\boldsymbol{A} - \boldsymbol{\mu}_A \mathbf{1}_n^T \right) - \boldsymbol{\mu} \mathbf{1}_n^T \right\|_{\mathrm{F}}. \tag{2}$$

(a) Show that the solution is

$$\begin{aligned}\boldsymbol{\mu}^\star &= \boldsymbol{\mu}_B \\ \boldsymbol{Q}^\star &= \boldsymbol{U}\boldsymbol{V}^T \\ \alpha^\star &= \frac{\operatorname{Tr}(\boldsymbol{B}_0 \boldsymbol{A}_0^T (\boldsymbol{Q}^\star)^T)}{\operatorname{Tr}(\boldsymbol{A}_0 \boldsymbol{A}_0^T)},\end{aligned}$$

where $\boldsymbol{\mu}_B \triangleq \frac{1}{n}\boldsymbol{B}\boldsymbol{1}_n$ is the centroid of (the columns of) $\boldsymbol{B}$, and $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ is an SVD of $\boldsymbol{B}_0\boldsymbol{A}_0^T$, where we have defined the "de-meaned" matrices

$$\boldsymbol{B}_0 \triangleq \boldsymbol{B} - \boldsymbol{\mu}_B \boldsymbol{1}_n^T, \quad \boldsymbol{A}_0 \triangleq \boldsymbol{A} - \boldsymbol{\mu}_A \boldsymbol{1}_n^T.$$

Hint: First show that $\boldsymbol{\mu} = \boldsymbol{\mu}^\star$ minimizes (2) for any fixed α and $\boldsymbol{Q}$ (it is a least squares problem). Then show that $\boldsymbol{Q} = \boldsymbol{Q}^\star$ minimizes (2) for any fixed α when $\boldsymbol{\mu} = \boldsymbol{\mu}^\star$. Finally, show that $\alpha = \alpha^\star$ minimizes (2) when $\boldsymbol{\mu} = \boldsymbol{\mu}^\star$ and $\boldsymbol{Q} = \boldsymbol{Q}^\star$.

Hint: With $\boldsymbol{\mu} = \boldsymbol{\mu}^\star$ and fixed α, (2) is a Procrustes problem. Along the way, it will be useful to remember that $\boldsymbol{Z}$ and $\alpha\boldsymbol{Z}$ have the same singular vectors for any matrix $\boldsymbol{Z}$.

(b) Write a function called `procrustes` that implements Procrustes alignment. Specifically, given $\boldsymbol{B}, \boldsymbol{A} \in \mathbb{R}^{d\times n}$, it should return the aligned coordinates $\hat{\boldsymbol{A}} \in \mathbb{R}^{d\times n}$.

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

```julia
"""
`Aa = procrustes(B, A ; center=true, scale=true)`

In:
* `B` and `A` are `d x n` matrices

Option:
* `center=true/false` : consider centroids?
* `scale=true/false` : optimize alpha or leave scale as 1?
Your solution needs only to consider the defaults for these.

Out:
* `Aa` `d x n` matrix containing `A` Procrustes-aligned to `B`

Returns `Aa = alpha * Q * (A - muA) + muB`, where `muB` and `muA` are
the `d x n` matrices whose rows contain copies of the centroids of
`B` and `A`, and `alpha` (scalar) and `Q` (`d x d` orthogonal matrix) are
the solutions to the Procrustes + scaling problem

`\\argmin_{alpha, muA, muB, Q: Q^T Q = I} \\|(B - muB) - alpha * Q (A - muA) \\|_F`
"""
function procrustes(B, A ; center=true, scale=true)
```

Submit your solution to the autograder by emailing it as an attachment to `eecs551@autograder.eecs.umich.edu`.

(c) For each of the two datasets, submit a plot of $\boldsymbol{B}$ and $\hat{\boldsymbol{A}}$ (overlaid, so we can see the alignment in each case).

Pr. 7.
(Image compression with an SVD)
For any rank r matrix $\boldsymbol{A}$ with SVD $\boldsymbol{A} = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i'$, the solution to the **low-rank approximation** problem

$$\boldsymbol{A}_{\text{opt}} = \arg\min_{\boldsymbol{X}} \|\boldsymbol{A} - \boldsymbol{X}\|_F \text{ subject to } \operatorname{rank}(\boldsymbol{X}) \leq k,$$

is given by $\boldsymbol{A}_{\text{opt}} = \sum_{i=1}^{\min(k,r)} \sigma_i \boldsymbol{u}_i \boldsymbol{v}_i'$.
The approximation error between $\boldsymbol{A}_{\text{opt}}$ and $\boldsymbol{A}$ is given by $\|\boldsymbol{A} - \boldsymbol{A}_{\text{opt}}\|_F = \sqrt{\sum_{i=\min(k,r)+1}^{r} \sigma_i^2}$.
This expression reveals that the approximation error decrease monotonically to zero with increasing k. When $k \geq r$, the approximation error is identically zero (in theory, ignoring finite numerical precision effects) because we have just reconstituted the entire $\boldsymbol{A}$ matrix via the sum of the pertinent r outerproducts.
This problem examines the visual quality of the approximation and investigates the feasibility of an SVD-based **image compression** scheme whereby, instead of storing the mn numbers in the $\boldsymbol{A}$ matrix, we instead store $k(m + n + 1)$ numbers corresponding to the m, n and 1 number(s) for each of the k left and right singular vectors and the singular values, respectively.
In a `Julia` notebook, to load and view an image use:

```
using Images, FileIO
image_color = load("your_image.jpg")
image_matrix_rgb = convert(Array{Float32}, channelview(image_color)) # 3xMxN
image_matrix_gray = sum(image_matrix_rgb, dims=1)[1,:,:] # MxN
```

To install the `Images` package you might also need to install the `ImageMagick` package which might require "Developer Mode" on a PC, or perhaps running `Julia` as an administrator. (Nothing special like that was needed on a Mac.)
To convert it to an ordinary matrix (2D array) use `image_matrix = float32.(image_matrix_gray) .+ 0.`
As usual, to compute an SVD of $\boldsymbol{A}$ use: `U, s, V = svd(image_matrix_gray)`
Download the notebook `hw_svd_image_compression_demo.ipynb` from the hw07 directory on Canvas, along with the .jpg files therein.
Run the notebook (no coding necessary for this step) and examine how changing the rank of the approximation changes the error and the quality of the image representation.

(a) Replace the image in the notebook with a picture of your choosing. Use the SVD to find a compressed version (via the reduced rank procedure) of the image that uses 20% as many bits as the original image. For simplicity, assume that all real numbers are stored using the same number of bits, e.g., 32 bits for `Float32`.

Submit images of the original and compressed images and a plot of the error as a function of the rank. Is the quality of the compressed image "good enough"?

(b) Repeat this with the two MIT logo images in the hw07 directory on Canvas. Try it first for the basic "nameless" logo **without** any extra lettering.

Find the smallest rank that gives decent image quality.

Now try that same rank for compressing the logo with the extra lettering.

Which logo image is well compressible with a much lower rank? Why? Hint: What is the (approximate) rank of the MIT logo?

(c) Write a function called `compress_image` that takes as input an $m \times n$ matrix $\boldsymbol{A}$ (an image) and a scalar p in $(0, 1)$ and returns an $m \times n$ matrix `Ac` containing a compressed version of $\boldsymbol{A}$ that can be represented using at most $(100 \times p)\%$ as many bits as $\boldsymbol{A}$. Your function should also return the rank, r, of the compressed matrix. Use the largest rank you can while still satisfying the compression requirement.

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

```
"""
    Ac, r = compress_image(A, p)

In:
* `A` `m x n` matrix
* `p` scalar in `(0, 1]`

Out:
* `Ac` a `m x n` matrix containing a compressed version of `A`
that can be represented using at most `(100 * p)%` as many bits
required to represent `A`
* `r` the rank of `Ac`
"""
function compress_image(A, p)
```

Submit your solution to the autograder by emailing it as an attachment to `eecs551@autograder.eecs.umich.edu`.

Optional problem(s) below
(not graded, but solutions will be provided for self check; do not submit to gradescope)

Pr. 8.
Suppose $\boldsymbol{A} \in \mathbb{R}_8^{19\times 48}$. Here the 8 in the subscript refers to the rank of the matrix $\boldsymbol{A}$. Determine how many linearly independent solutions can be found to the homogeneous linear system $\boldsymbol{Ax} = \boldsymbol{0}$. Hint: Use the rank plus nullity theorem.

Pr. 9.
Suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric. Show that $\operatorname{trace}(\boldsymbol{AB}) = \operatorname{vec}(\boldsymbol{A})^T \operatorname{vec}(\boldsymbol{B})$.
What is the corresponding property when neither are symmetric?

Pr. 10.
Show that the weighted Euclidean norm $\|\boldsymbol{x}\|_{\boldsymbol{W}}$ is a valid **norm** iff $\boldsymbol{W}$ is a **positive definite matrix**.
Assume $\boldsymbol{W}$ is (Hermitian) symmetric throughout.

Pr. 11.
Show how to express $\operatorname{trace}\big((\boldsymbol{AA}')^2\big)$ concisely in terms of a Schatten p-norm.

Pr. 12.

(a) Show that $\big\|\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\end{bmatrix}\big\|_2^2 \le \|\boldsymbol{A}\|_2^2 + \|\boldsymbol{B}\|_2^2$.

(b) We would like to verify that this is a "tight" upper bound. Simply showing equality for arbitrary $\boldsymbol{A}$ and $\boldsymbol{B}$ is uninteresting because $\boldsymbol{A} = \boldsymbol{B} = \boldsymbol{0}$ is a trivial example. To define tightness more meaningfully, consider the set $\mathcal{B}_b \triangleq \{\boldsymbol{B} \in \mathbb{F}^{M\times K} : \|\boldsymbol{B}\|_2 = b\}$. Show that for any $\boldsymbol{A} \in \mathbb{F}^{M\times N}$ and any $b \ge 0$, and any $M, N, K \in \mathbb{N}$, there exists $\boldsymbol{B} \in \mathcal{B}_b$ such that $\big\|\begin{bmatrix}\boldsymbol{A} & \boldsymbol{B}\end{bmatrix}\big\|_2^2 = \|\boldsymbol{A}\|_2^2 + \|\boldsymbol{B}\|_2^2$. Show existence constructively, *i.e.*, show how to define $\boldsymbol{B}$ in terms of $\boldsymbol{A}$.

(c) Find a tight upper bound on the spectral norm of the matrix $\begin{bmatrix}\boldsymbol{A} \\ \boldsymbol{B}\end{bmatrix}$ in terms of the spectral norms of $\boldsymbol{A}$ and $\boldsymbol{B}$.

Pr. 13.
The course notes discuss the (convex) regularized **low-rank approximation** problem

$$\hat{\boldsymbol{X}} = \operatorname*{arg\,min}_{\boldsymbol{X}\in\mathbb{C}^{M\times N}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_{\mathrm{F}}^2 + \beta\,\|\boldsymbol{X}\|_* .$$

Determine the solution when we replace the Frobenius norm with the **spectral norm**:

$$\hat{\boldsymbol{X}} = \operatorname*{arg\,min}_{\boldsymbol{X}\in\mathbb{C}^{M\times N}} \frac{1}{2}\|\boldsymbol{Y} - \boldsymbol{X}\|_2^2 + \beta\,\|\boldsymbol{X}\|_* .$$

It is sufficient to solve it for the case where $\operatorname{rank}(\boldsymbol{Y}) = 1$.
