Pr. 1.
Low-rank matrix denoising using non-convex Schatten p-"norm"

The **Schatten p-norm**: $\|\boldsymbol{A}\|_{S,p} = \left(\sum_{k=1}^{r} \sigma_k^p\right)^{1/p}$, where σ_k denotes the singular values of $\boldsymbol{A}$, is a proper matrix norm for $p \geq 1$. Values of $p \in (0,1)$ are also useful as regularizers for **low-rank matrix denoising** problems even though it is not a proper norm for $p < 1$.

(a) For $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon} \in \mathbb{C}^{M \times N}$, where we think $\boldsymbol{X}$ is low rank, find an expression for the solution of the regularized low-rank matrix denoising method that uses the following **Schatten-norm regularizer** for $p = 1/2$:

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathbb{C}^{M \times N}}{\arg\min} \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{X}\|_{\mathrm{F}}^2 + \beta R(\boldsymbol{X}, 1/2), \quad R(\boldsymbol{X}, p) = \|\boldsymbol{X}\|_{S,p}^p = \sum_{k=1}^{r} \sigma_k^p(\boldsymbol{X}).$$

Hint: a previous HW problem will be helpful. Submit your written answer to this part to gradescope.

(b) Write a `Julia` function that performs regularized low-rank matrix denoising using the above cost function.

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

```
"""
    lr_schatten(Y, reg::Real)

Compute the regularized low-rank matrix approximation as the minimizer over `X`
of `1/2 \\|Y - X\\|^2 + reg R(x)`
where `R(X)` is the Schatten p-norm of `X` raised to the pth power, for `p=1/2`,
i.e., `R(X) = \\sum_k (\\sigma_k(X))^{1/2}`

In:
- `Y`    `M by N` matrix
- `reg` regularization parameter

Out:
- `Xh`   `M by N `solution to above minimization problem
"""
function lr_schatten(Y, reg::Real)
```

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

Think about your solution and consider whether it seems to be a good method for denoising.

(c) Apply your denoising method to the 100×30 Block M image in the demo notebook `06_optshrink1.ipynb` at `http://web.eecs.umich.edu/~fessler/course/551/julia/demo/` using $\beta = 1000$.

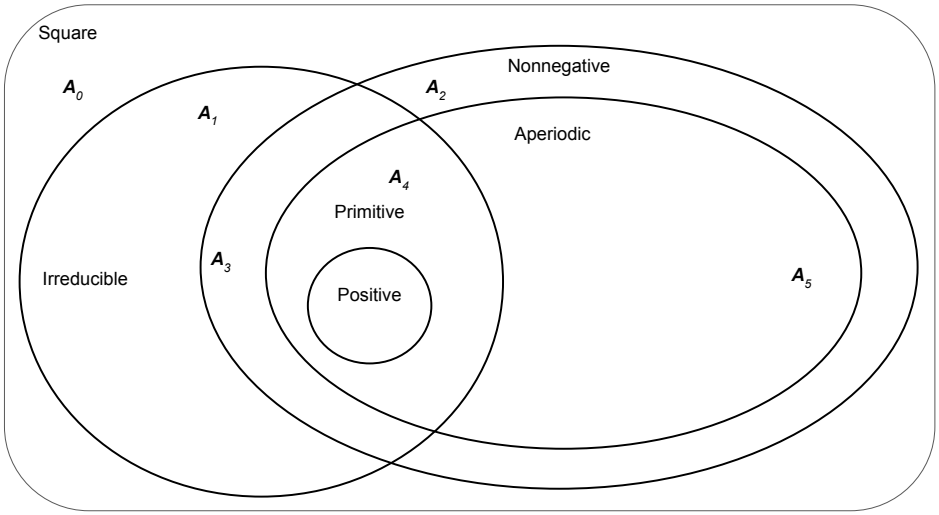
Report the NRMSE of your estimate $\hat{\boldsymbol{X}}$ and submit to gradescope a picture of $\hat{\boldsymbol{X}}$ and a plot of the singular values of $\hat{\boldsymbol{X}}$, $\boldsymbol{X}$ and $\boldsymbol{Y}$. All of the plotting commands are in the notebook already; you simply need to `include` your `lr_schatten.jl` solution.

Pr. 2.
Type in the following commands that make some **companion matrix** forms.

```
using LinearAlgebra
n = 5
u = randn(n) # an n by one random vector of polynomial coefficients
uflip = reverse(u,dims=1) # flips the vector upside down
# the next line makes a companion matrix slightly different from course notes:
compan = (c) -> [-c'; [I zeros(length(c)-1)]]
A = compan(u[2:end] / u[1])
B = compan(uflip[2:end] / uflip[1])
@show [eigvals(A) 1 ./ eigvals(B)]
```

What pattern do you see? Repeat multiple times to form a conjecture on how the **eigenvalues** of `A` and `B` are related. Prove your conjecture theoretically.

Pr. 3.
The following **Venn diagram** summarizes relationships of various special square matrices.



Provide example matrices $\boldsymbol{A}_0, \ldots, \boldsymbol{A}_5$ that belong to the each of the categories above but *not* to *any* of the categories nested within it. Try to provide the simplest possible example in each case.
Hint. Some (but perhaps not all) of the following matrices are useful.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad [-1] \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad [e^t] \quad \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$$

Pr. 4.

(a) Show that all **circulant matrices** (of the same size) **commute**.

(b) Show that all **circulant matrices** are **normal**.

(c) Determine the eigenvalues of the $N \times N$ circulant matrix that does finite differences with periodic end conditions:

$$\boldsymbol{C} = \begin{bmatrix} -1 & 1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & \ldots & 0 & 0 \\ \vdots & & \ddots & \ddots & & & \vdots \\ 0 & 0 & 0 & \ldots & 0 & -1 & 1 \\ 1 & 0 & 0 & \ldots & 0 & 0 & -1 \end{bmatrix}.$$

Give an expression that holds for any $N \in \mathbb{N}$.

(d) Check your expression for the previous part numerically for the case $N = 4$.
Report the eigenvalues from your expression and from the `eigvals` command in `Julia`.

Pr. 5.
Make a clearly stated problem that could be used on an EECS 551 exam based the topics covered so far, and provide your own solution to the problem. Your problem could include something about `Julia`, but in that case should be about concepts, not mere syntax, and should not require submitting code to an autograder.
If you make a good problem (not too trivial, not too hard) then it might be used on an actual exam.
To earn full credit:

- Your problem and solution must fit on a single page with a horizontal line that clearly separates the question (above) from the answer (below).
- Your solution to the problem must be correct.
- Your problem should not be excessively trivial (nor beyond the scope of the course).
- Your problem must not be identical to any quiz or homework or clicker or sample exam problem.

- Your problem should not be of the form "prove that ..." but it is fine to pose a True/False problem with an answer that needs some proving to explain.
- You must upload your problem/solution to gradescope as usual for grading.
- You must submit your problem/solution in pdf format to Canvas for distribution by the deadline for this HW. (We can export from Canvas easily but not from gradescope.)
- Do not submit any other parts of your HW assignment to Canvas; only this 1 page pdf file.
- Do not put your name on the problem/solution that you upload to Canvas because we plan to export all problems/solutions to distribute as practice problems.
- We may make an announcement on Canvas about some additional formatting requirements, so be sure to check the announcements before submitting.

Pr. 6.

Apply the **multi-dimensional scaling** (MDS) algorithm to the following distance matrix: $\boldsymbol{D} = \begin{bmatrix} 0 & a & 2a \\ a & 0 & a \\ 2a & a & 0 \end{bmatrix}$.
Do the work by hand - no `Julia`. (You may use `Julia` to check your answer.) Hint: look for a rank-1 outer product.
Plot (by hand) the resulting coordinates in 2D.
Verify that your answer makes sense in light of the original distance matrix $\boldsymbol{D}$.

Pr. 7.
Testing for common roots of polynomials

(a) Given two polynomials $p_1(x) = x^n + \sum_{i=0}^{n-1} \alpha_i x^i$ and $p_2(x) = x^m + \sum_{j=0}^{m-1} \beta_j x^j$ for n not necessarily equal to m, and the definition of **Kronecker product**, how would you check if the equations $p_1(x) = 0$ and $p_2(x) = 0$ share a common (possibly complex valued) solution? In other words, we are checking if the algebraic curves $p_1(x) = 0$ and $p_2(x) = 0$ intersect in the complex plane.

You are only allowed to invoke the determinant and the trace to ascertain the desired property. You are **not** allowed to explicitly compute the eigenvalues of the respective companion matrices to see if there are common roots. Other matrix decompositions (QR, LU) are not allowed either.

Hint: Use the relationship between the eigenvalues of the **Kronecker sum** of matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ with the eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{B}$.

(b) Write a function called `common_root` that takes as input two vectors (not necessarily of equal dimension) whose elements represent the coefficients of the polynomials $p_1(x)$ and $p_2(x)$ and returns `true` if the two polynomials share a common root and `false` otherwise. Use the convention that the first element of each vector contains the leading polynomial coefficient as shown in the function comments below.

Caution: the leading coefficient is not necessary 1 here, but you can assume it is nonzero.

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

```
"""
    haveCommonRoot = common_root(p1, p2)

Determine whether the input polynomials share a common root

In:
- `p1` is a vector of length `m + 1` with `p1[1] != 0`
that defines an `m`th degree polynomial of the form:
`P1(x) = p1[1] x^m + p1[2] x^(m - 1) + ... + p1[m] x + p1[m + 1]`

- `p2` is a vector of length `n + 1` with `p2[1] != 0`
that defines an `n`th degree polynomial of the form:
`P2(x) = p2[1] x^n + p2[2] x^(n - 1) + ... + p2[n] x + p2[n + 1]`

Out:
- `haveCommonRoot` = `true` when `P1` and `P2` share a common root, else `false`
"""
function common_root(p1, p2)
```

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

Hint 1: Use the `Julia` function `kron`.

Hint 2: Due to finite numerical precision, you may need to set an appropriately chosen threshold in your code for deciding whether two roots are close enough to be "equal."

Test your code with some polynomials of your own design before submitting to the autograder.

Pr. 8.
Power iteration for extreme polynomial roots

(a) Suppose we are given a polynomial $p(x)$. Assume that $p(x)$ has zeros that can be ordered in magnitude and that the smallest and largest zero (in magnitude) are unique. Describe an **iterative algorithm** based on the **power iteration** for computing the largest (in magnitude) zero that does not compute all the zeros first.

(b) Now describe an algorithm for computing the smallest (in magnitude) zero that does not compute all the zeros and does not involve inverting any matrix. Your method may use *at most one* use of the power iteration! You may not use the maximum value found in the previous part either.
Hint. Think about an earlier problem on this HW.
Hint: Be sure to consider the case(s) where certain coefficients are zero.

(c) Write a function called `outlying_zeros` that implements your algorithm. The function should accept as input a coefficient vector `p` $\in \mathbb{R}^{n+1}$ defining the polynomial $P(x) = \sum_{i=1}^{n+1} \mathtt{p}_i x^{n+1-i}$, where `p[1]` $\neq 0$ and $n \in \mathbb{N}$. Optional arguments are a vector `v0` $\in \mathbb{R}^n$ to initialize the power iteration(s), and a positive integer `nIters` specifying the number of power iterations to perform. It should return (approximations of) the largest and smallest magnitude zeros, respectively, of $P(x)$.

Do not use expensive decomposition functions like `inv` `eig` `svd` backslash etc.

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

```
"""
    zmax, zmin = outlying_zeros(p, v0, nIters)

Use power iteration to compute the largest and smallest magnitude zeros,
respectively, of the polynomial defined by the input coefficients

In:
- `p` is a vector of length `n + 1` defining the polynomial
`P(x) = p[1] x^n + p[2] x^n-1 + ... + p[n] x + p[n + 1]` with `p[1] != 0`

Option:
- `v0` is a vector of length `n` with initial guess of an eigenvector; default randn
- `nIters` is the number of power iterations to perform; default 100

Out:
- `zmax` is the zero of `P(x)` with largest magnitude
- `zmin` is the zero of `P(x)` with smallest magnitude
"""
function outlying_zeros(p ; v0::Vector{<:Real}=randn(length(p)-1), nIters::Int=100)
```

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

Pr. 9.
Handwritten digit classification via nearest subspace: all 10 digits (discussion task)

(a) This problem explores handwritten digit classification using projection on to the nearest subspace where the subspaces are learned via the SVD. Suppose

- `test` is an $n \times p$ matrix whose columns contain p vectorized test images to be classified
- `train` is an $n \times m \times 10$ matrix containing (up to) m vectorized training images for each digit 0-9 (in ascending order along the third dimension of `train`). If some digit has fewer than m training images, then the remaining columns are padded with zeros.
- K is an integer between 1 and $\min(m, n)$ specifying the subspace dimension

Write a function called `classify_image` that does the following, in order:

(i) Compute the ten $n \times n$ rank-K (orthogonal) projection matrices for the span the first K left singular vectors associated with the training images of each digit, respectively. For digit $d = 0, \ldots, 9$ these are the K left singular vectors associated with the K largest singular values of the dth "slice" of `train`.

(ii) Store the projection matrices in a $n \times n \times 10$ array—call it `P`

(iii) Using *only* these projection matrices, compute the squared-norm of the error vector between each test image and the projection of each test image onto the K-dimensional subspaces spanned by each digit. Store the squared error in a $10 \times p$ matrix—call it `err`.

Hint: You can express this computation succinctly for digit $d = 0, \ldots, 9$ via the command

```
err[i,:] = sum((test - P[:,:,i] * test).^2, dims=1)
```

(iv) Generate a vector—call it `labels`—containing the digit (for each test image) with the smallest squared error.

Hint: You can compute this for all digits at once using the commands

```
idx = findmin(err, dims=1)[2] # These are linear indices!!
idx = [CartesianIndices(size(err))[i][1] for i in vec(idx)] # think about this!
labels = idx .- 1 # Convert to digits in (0-9)
```

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

```
"""
    labels = classify_image(test, train, K::Int)

Classify `test` signals using `K`-dimensional subspaces
found from `train`ing data via SVD

In:
* `test` `n x p` matrix whose columns are vectorized test images to be classified
* `train` `n x m x 10` array containing `m` training images for each digit 0-9 (in ascending order)
* `K` in `[1, min(n, m)]` is the number of singular vectors to use during classification

Out:
`labels` vector of length `p` containing the classified digits (0-9) for each test image
"""
function classify_image(test, train, K::Int)
```

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

(b) Use your `classify_image` function to classify the MNIST test data from class (task sheet). Because we are now classifying all 10 digits, more training data may be helpful, so here use 256 images for training and the remaining 744 for testing, for each digit.

For efficiency, you should call your `classify_image` function only once per K, not 100's of times! Otherwise you will be waiting a long time.

Turn in a single plot with 10 (labeled!) curves on it showing the percent of the test digits correctly classified versus K for each of the ten digits. Examine K from 1 to at least 50.

Previously you probably downloaded only the 0 and 1 digits, so you will need download to get the remaining digits to complete this step.

(c) Plot the singular values associated with the training images for each digit. (Submit 1 or 2 of these plots, not all 10, or put several on one plot if it is still interpretable.) Discuss how close to the optimal performance do you get when you select the value of k associated with the "gap" in the singular value spectrum.

(d) Optional. Try nearest-mean-training-digit-based classification instead. Or, because it is simpler, you may re-use your same routine and simply project onto the one-dimensional subspace spanned by the mean of each set of training images.

Show graphically the classification accuracy. How much better is SVD-based classification?

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

Pr. 10.
Prove or disprove (by a counter-example) the following statement. If $\boldsymbol{T}$ is any matrix for which $\boldsymbol{T}^k = \boldsymbol{I}$ for some natural number $k > 1$, then $\boldsymbol{T}$ is **normal**.

Pr. 11.
Let $t_k = k\Delta$, for some integer k, and $\Delta \in \mathbb{R}$. Consider the sum of sinusoids signal

$$\boldsymbol{y}(t_k) = \sum_{i=1}^{r} \boldsymbol{b}_i e^{\imath w_i t_k},$$

where $\boldsymbol{y}_i \in \mathbb{C}^n$, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_r \in \mathbb{C}^n$ are linearly independent vectors, and $w_i \in [0, 1]$ are distinct, and $\imath = \sqrt{-1}$. Express the recursion in the following simple matrix form:

$$\boldsymbol{y}(t_{k+1}) = \boldsymbol{A}\boldsymbol{y}(t_k).$$

(a) Determine the matrix $\boldsymbol{A}$.

(b) Determine the eigenvalues and eigenvectors of $\boldsymbol{A}$.

Hint: For a matrix $\boldsymbol{B}$ of full column rank r, we have that $\boldsymbol{B}^\dagger \boldsymbol{B} = \boldsymbol{I}_r$, where $\boldsymbol{I}_r$ is an identity matrix of rank r.

Pr. 12.
Define $\mathbb{R}_+ \triangleq [0, \infty)$. A **symmetric gauge function** $\phi(\boldsymbol{x})$ is a mapping from $\mathbb{R}^n$ into $\mathbb{R}_+$ that satisfies the following four properties for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$.

- $\phi(\boldsymbol{x}) > 0$ if $\boldsymbol{x} \neq \boldsymbol{0}$ (positivity)
- $\phi(\alpha \boldsymbol{x}) = |\alpha| \, \phi(\boldsymbol{x})$ for all $\alpha \in \mathbb{R}$ (homogeneity)
- $\phi(\boldsymbol{x} + \boldsymbol{y}) \leq \phi(\boldsymbol{x}) + \phi(\boldsymbol{y})$ (triangle inequality)
- $\phi(s_1 x_{[1]}, \ldots, s_n x_{[n]}) = \phi(\boldsymbol{x})$ for all $s_k = \pm 1$ and for any permutation $\left(x_{[1]}, \ldots, x_{[n]}\right)$ of the elements of $\boldsymbol{x}$. (symmetry)

Some examples are:

- $\phi(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$
- $\phi(\boldsymbol{x}) = \max_i |x_i| = \|\boldsymbol{x}\|_\infty$
- $\phi(\boldsymbol{x}) = 7\left|x_{(1)}\right| + 5\left|x_{(2)}\right|$, where $x_{(1)}$ and $x_{(2)}$ denote the first and second largest elements of $\boldsymbol{x}$ in magnitude.

Such symmetric gauge functions are at the heart of many matrix norm properties.
Let $\phi(\cdot)$ be any symmetric gauge function and define a matrix norm by $\|\boldsymbol{A}\|_\phi = \phi(\sigma_1, \ldots, \sigma_{\min(M,N)})$ where $\{\sigma_k\}$ denote the singular values of a $M \times N$ matrix $\boldsymbol{A}$.

(a) Verify for yourself that this definition does indeed define a proper matrix norm.

(b) Now prove or disprove (by counter-example) that any such matrix norm $\|\boldsymbol{A}\|_\phi$ is **unitarily invariant**.
