Pr. 1.
Power iteration for extreme polynomial roots

(a) Given a polynomial $p(x)$ that has roots such that the magnitudes of its smallest and largest roots are unique. Describe an **iterative algorithm** based on the **power iteration** for computing the largest (in magnitude) root *without* computing all the roots.

(b) Now describe an algorithm for computing the smallest (in magnitude) root *without* computing all the roots and *without* inverting any matrix. Your method may use *at most one* run of the power iteration! You may not use the maximum value found in the previous part either.
Hint. Think about an earlier HW problem.
Hint: Be sure to consider the case(s) where certain coefficients are zero.

(c) Write a function called `outlying_roots` that implements your algorithm. The function should accept as input a coefficient vector `a` $\in \mathbb{F}^{n+1}$ defining the polynomial $p(x) = \sum_{i=0}^{n}$ `a[i+1]` x^i, where `a[n+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 roots, respectively, of $p(x)$.

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

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

```
"""
    zmax, zmin = outlying_roots(a, v0, nIters)

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

# In:
- `a` coefficient vector of length `n + 1` defining the degree-`n` polynomial
  `p(x) = a[n+1] x^n + a[n] x^{n-1} + ... + a[2] x + a[1]` with `a[n+1] != 0`

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

# Out:
- `zmax` root of `p(x)` with largest magnitude
- `zmin` root of `p(x)` with smallest magnitude
"""
function outlying_roots(a ; v0::AbstractVector{<:Real} = randn(length(a)-1), nIters::Int=100)
```

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

Be sure to test your routine with a variety of different polynomial degrees and coefficients before submitting. Examples: $x^2 - 3x$, $2x^2 - 6x$, $x^2 + 2\imath x + 15 = (x + 5\imath)(x - 3\imath)$, etc.

Pr. 2.

(a) Give an example of a rank-1 **transition matrix** $\boldsymbol{P} \in \mathbb{R}^{4\times 4}$ having equilibrium distribution $\boldsymbol{\pi} = (0, 1/6, 2/6, 3/6)$.

Or equivalently, give an example of a rank-1 **stochastic matrix** that has the (unit-sum) eigenvector $\boldsymbol{q} = (0, 1/6, 2/6, 3/6)$. with eigenvalue 1.

(b) Is your example the unique solution? Explain why or why not.

Pr. 3.

Show that when $\boldsymbol{P}$ is symmetric positive definite:
$\text{eig}\left(\boldsymbol{P}^{1/2}\boldsymbol{A}'\boldsymbol{A}\boldsymbol{P}^{1/2}\right) < 2 \iff 2\boldsymbol{P}^{-1} \succ \boldsymbol{A}'\boldsymbol{A}$. This property is important for the **preconditioned gradient descent** algorithm for solving **least-squares** and related problems.

Pr. 4.

This problem constructs a simple and practical **diagonal majorizer** for use in iterative algorithms.
A $N \times N$ square matrix $\boldsymbol{A}$ is called **diagonally dominant** iff $|a_{ii}| \geq \sum_{j\neq i} |a_{ij}|$ for $i = 1, \dots, N$.

(a) (Optional) Use the **Geršgorin disk** theorem to show that if $\boldsymbol{H}$ is a (Hermitian) symmetric matrix that is diagonally dominant with $h_{ii} \geq 0$ for all i, then $\boldsymbol{H} \succeq \boldsymbol{0}$.

(b) Show that if $\boldsymbol{B}$ is an $N \times N$ Hermitian symmetric matrix, and $|\boldsymbol{B}|$ denotes the matrix consisting of the absolute values of the elements of $\boldsymbol{B}$, *i.e.*, `abs.(B)` and $\boldsymbol{1}_N$ is `ones(N)` in `Julia`, then $\boldsymbol{D} \triangleq \text{Diag}(|\boldsymbol{B}|\boldsymbol{1}_N) \succeq \boldsymbol{B}$.

Pr. 5.

Consider the **regularized LS** cost function $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 + \beta^2 \frac{1}{2}\|\boldsymbol{x}\|_2^2$.

(a) (0 points) Determine the **gradient** of this cost function.

(b) Determine an easily computed upper bound on the **Lipschitz constant** of the gradient of this cost function. Your final expression must not involve any singular values, because the SVD is too expensive to compute for large-scale problems. Your bound should be one that is reasonably tight (see below); if your bound is too loose (too large), then using its reciprocal as the step size in gradient descent (or related algorithms) would lead to undesirably slow convergence.

(c) Determine (analytically) the numerical value of your upper bound when $\boldsymbol{A} = \begin{bmatrix} \boldsymbol{I}_5 \\ \boldsymbol{1}_5\boldsymbol{1}_5' \end{bmatrix}$ and $\beta = 2$.

(d) Determine (analytically) the exact value of the Lipschitz constant of the gradient of the cost function for the $\boldsymbol{A}$ in the previous part. If your bound is more than 10% larger than the exact Lipschitz constant for this case, then you have not found a good enough bound and you should return to part (b) and improve your answer.

Pr. 6.

The following matrix is known to be some constant plus a rank-1 matrix, but some values are missing:

$$\boldsymbol{A} = \begin{bmatrix} 20 & 14 & w & 17 \\ 44 & x & 16 & y \\ 8 & 6 & z & 7 \end{bmatrix}.$$

Determine the values of w, x, y, z. This is a simple variation on the **matrix completion** problem.

Pr. 7.
Algebraic numbers and polynomials with integer coefficients

• Recall the roots of any **monic polynomial** are the **eigenvalues** of a corresponding **companion matrix**.

• The **Kronecker sum** of two square matrices $\boldsymbol{A} \in \mathbb{F}^{M \times M}$ and $\boldsymbol{B} \in \mathbb{F}^{N \times N}$ is defined as $\boldsymbol{A} \oplus \boldsymbol{B} = \boldsymbol{A} \otimes \boldsymbol{I}_N + \boldsymbol{I}_M \otimes \boldsymbol{B}$ or $\boldsymbol{I}_N \otimes \boldsymbol{A} + \boldsymbol{B} \otimes \boldsymbol{I}_M$. Regardless, the MN **eigenvalues** of $\boldsymbol{A} \oplus \boldsymbol{B}$ are $\lambda_i(\boldsymbol{A}) + \lambda_j(\boldsymbol{B})$ for $i = 1, \dots, M$ and $j = 1, \dots N$.

• An **algebraic number** is a value in $\mathbb{C}$ that is a root of some polynomial having *integer* coefficients.
For example, $x = \sqrt{5}$ is an algebraic number because it is a root of the polynomial $p(x) = x^2 - 5$.
Consider $x_1 = -2 + \imath\sqrt{3}$ and $x_2 = 5 - \sqrt{7}$. These are algebraic numbers because they are roots of corresponding polynomials $p_1(x) = (x - (-2 + \imath\sqrt{3})(x - (-2 - \imath\sqrt{3})) = x^2 + 4x + 7$ and $p_2(x) = (x - (5 - \sqrt{7}))(x - (5 + \sqrt{7}))$, respectively, having integer coefficients (when expanded). Now define $x_3 \triangleq x_1 + x_2$ and $x_4 \triangleq x_1 x_2$. A **theorem in algebra** says that x_3 and x_4 are also **algebraic** because algebraic numbers form a **field**. In other words, there is a polynomial $p_3(x)$ with integer coefficients that has one root equal to x_3, and a polynomial $p_4(x)$ with integer coefficients that has one root equal to x_4.
Use the **Kronecker product** and/or **Kronecker sum** to find those polynomials $p_3(x)$ and $p_4(x)$.
Do not use the `poly_coeff` function described below!
Hint. Start by finding the companion matrix with integer coefficients that has an eigenvalue equal to x_1 and another companion matrix with eigenvalue x_2.
You may use symbolic computation to find characteristic polynomials. Example:

```
using AbstractAlgebra: charpoly, matrix, ZZ, coefficients
charpol(A::Matrix) = charpoly(ZZ["x"][1], matrix(ZZ,A))
A = [1 2; 3 4] # Julia matrix
p = charpol(A) # x^2 - 5*x - 2 : characteristic polynomial of A
Int.(coefficients(p)) # [-2, -5, 1] coefficients of that polynomial
```

Pr. 8.
Let $a, b, c, d \in \mathbb{Z}$ and set $x_1 = a + \sqrt{b}$ and $x_2 = c - \sqrt{d}$. Write a function called `poly_coeff` that takes (a, b, c, d) as input and returns vectors $g, h \in \mathbb{Z}^5$ of integer-valued coefficients defining the monic ($g_4 = h_4 = 1$) quartic polynomials $p_3(x) = \sum_{i=0}^{4} g_i x^i$ and $p_4(x) = \sum_{i=0}^{4} h_i x^i$ with the properties that $p_3(x_3) = 0$ and $p_4(x_4) = 0$. Use your "hand" solution from a previous HW problem as a test of your function.
You may use the `Polynomials` package, but not `AbstractAlgebra` or `SymPy`.

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

```
"""
    g, h = poly_coeff(a, b, c, d)

Use Kronecker sums/products to construct polynomials with integer coefficients
with the given (algebraic) numbers as zeros.

# In:
* `a, b, c, d` are integers

# Out:
* `g` and `h` are Int vectors of length 5 with `p[5] = 1` and `q[5] = 1`
such that, in the notation defined below, `p3(x3) = 0` and `p4(x4) = 0`

# Notation:
* `x1 = a + √b`
* `x2 = c - √d`
* `x3 = x1 + x2`
* `x4 = x1 * x2`
* `p1(x)` quadratic monic polynomial with integer-valued coefficients with a zero at `x1`
* `p2(x)` quadratic monic polynomial with integer-valued coefficients with a zero at `x2`
* `p3(x) = g[5] x^4 + g[4] x^3 + g[3] x^2 + g[2] x + g[1]` where `g[5] = 1`
* `p4(x) = h[5] x^4 + h[4] x^3 + h[3] x^2 + h[2] x + h[1]` where `h[5] = 1`
"""
function poly_coeff(a::Int, b::Int, c::Int, d::Int)
```

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

Pr. 9.
(Steepest descent for least squares problems)
The gradient descent method for LS problems requires selecting a step size parameter. In contrast, the **steepest descent** method determines the **step size** automatically (for quadratic problems), as discussed in the course notes and analyzed in a previous HW problem.

(a) Write a function called `lssd` that implements the steepest descent method for iteratively finding the minimizer $\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x} \in \mathbb{F}^N} f(\boldsymbol{x}), \quad f(\boldsymbol{x}) = 1/2 \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$.

For full credit, your final version of the code should should use only one multiplication by $\boldsymbol{A}$ and one multiply by $\boldsymbol{A}'$ each iteration. (For debugging you might initially write a version that uses more multiplies.)

Your function should use `Julia`'s **function keyword arguments** capability so that the initial estimate $\boldsymbol{x}_0$ is $\mathbf{0}$ by default and so that the number of iterations is 10 by default.

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

```
"""
    x = lssd(A, b ; x0=zeros(size(A,2)), nIters::Int=10)

Perform steepest descent to solve the least-squares problem
`\\hat{x} = \\arg\\min_x f(x), f(x) = 1/2 ‖ A x - b ‖_2`

# In:
* `A` : `M × N` matrix
* `b` : vector of length `M`

# Option:
* `x0` : initial vector (of length `N`); default `zeros`
* `nIters` : number of iterations to perform; default `10`

# Out:
* `x` a vector of length `N` containing the approximate solution

Because this is a quadratic cost function, there is a closed-form solution
for the step size each iteration, so no "line search" procedure is needed.

A full-credit solution uses only *one* multiply by `A` and one by `A'` per iteration.
"""
function lssd(A, b ; x0=zeros(size(A,2)), nIters::Int=10)
```

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

(b) After your code passes, use it to generate a plot of $\log_{10}(\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|)$ as a function of iteration k for $\boldsymbol{A}$ and $\boldsymbol{b}$ generated as follows. Use the default $\boldsymbol{x}_0$, and show 100 iterations.

```
using Random: seed!
seed!(0) # seed random number generator
M = 100; N = 50; sigma = 0.1
A = randn(M, N); xtrue = rand(N)
b = A * xtrue + sigma * randn(M)
```

Does $\|\boldsymbol{x}_k - \hat{\boldsymbol{x}}\|$ **decrease monotonically** with k in the plots?

Optional: compare to GD on the same plot using your `lsgd` function. Does SD converge faster than GD here?

(c) Submit your code (a screen capture is fine) to gradescope so that the grader can verify that you used only one multiply by $\boldsymbol{A}$ and $\boldsymbol{A}'$ each iteration.

Hint: If $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{d}$ then $(\boldsymbol{A}\boldsymbol{x}_{k+1}) = (\boldsymbol{A}\boldsymbol{x}_k) + (\boldsymbol{A}\boldsymbol{d})$, so think about using a `+=` operation.

Pr. 10.
Handwritten digit classification via nearest subspace: all 10 digits (discussion task)
Download the `task-5-classify10.ipynb` jupyter notebook file from Canvas under the discussion folder and follow all instructions to complete the task. Finally you will classify all 10 digits! 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.

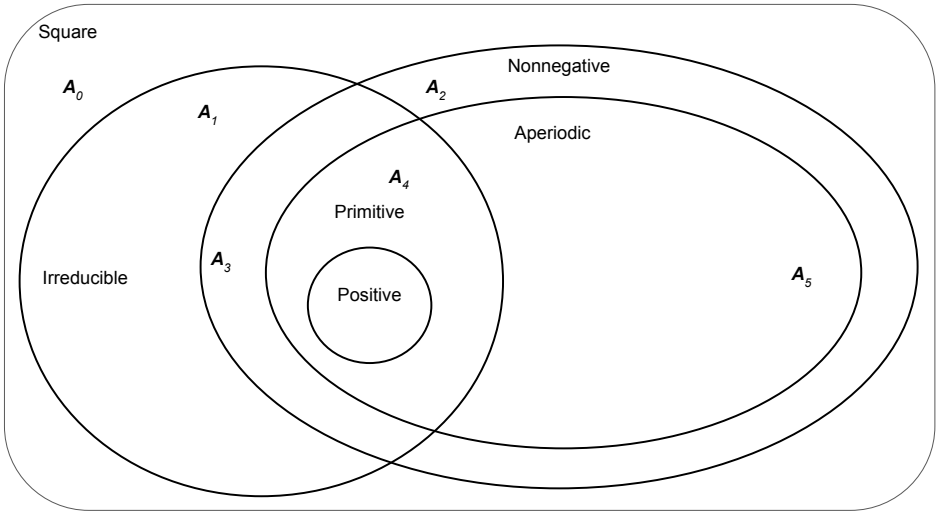
As part of that task you will (individually!) submit a solution to the following autograder problem.
In `Julia`, your file should be named `classify_image.jl` and should contain the following function:

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

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

# In:
* `test` `n × p` matrix whose columns are vectorized test images to be classified
* `train` `n × m × 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::Matrix, train::Array{<:Number,3}, K::Int)
```

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

Non-graded problem(s) below
(Solutions will be provided for self check; do not submit to gradescope.)

Pr. 11.
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.

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

Pr. 12.

(a) Derive the optimal **step size** α_* for classical **gradient descent** of the **least-squares** cost function $f(\boldsymbol{x}) = \frac{1}{2} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$, in terms of the singular values of the (full column rank) matrix $\boldsymbol{A}$. Here, optimal means $\rho(\boldsymbol{I} - \alpha \boldsymbol{A}'\boldsymbol{A})$ is as small as possible.

(b) For that optimal step size α_*, determine the **spectral radius** $\rho(\boldsymbol{I} - \alpha_* \boldsymbol{A}'\boldsymbol{A})$ that governs the **convergence rate** of the sequence $\{\boldsymbol{x}_k\}$ produced by the GD method.
Express your answer in terms of the **condition number** of $\boldsymbol{A}'\boldsymbol{A}$.

Pr. 13.
Practical preconditioners are often built on approximations of a cost function's Hessian inverse. The initial design of such preconditioners often does not ensure convergence of PGD, so a step size α is usually needed. Specifically, for a LS problem we might have $\boldsymbol{P}_0 \approx (\boldsymbol{A}'\boldsymbol{A})^{-1}$ and then let $\boldsymbol{P} = \alpha \boldsymbol{P}_0$, where $\boldsymbol{P}_0$ is positive definite.

(a) Determine the optimal (fixed) **step size** α_* that makes the **preconditioned gradient descent** (PGD) iteration $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \boldsymbol{P}_0 \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x}_k - \boldsymbol{y})$ have the fastest possible **convergence rate**.
Hint. The answer depends on both $\boldsymbol{A}$ and the preconditioner $\boldsymbol{P}_0$.

(b) Simplify your expression for α_* as much as possible in the special case where we use the ideal preconditioner $\boldsymbol{P}_0 = (\boldsymbol{A}'\boldsymbol{A})^{-1}$. Does your simple expression (a number) make sense?
