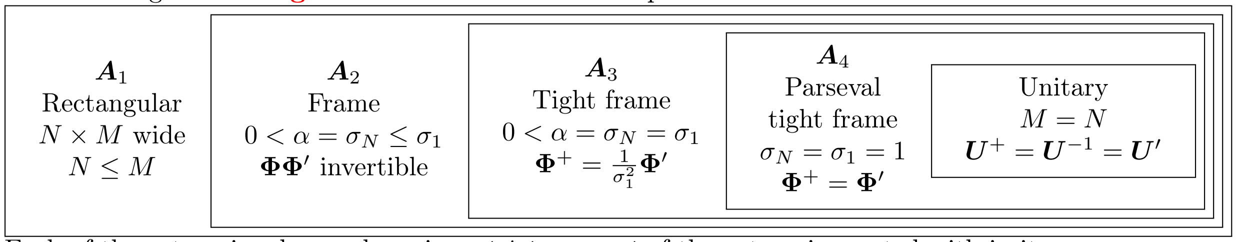
Pr. 1.
If $\boldsymbol{Q}$ is a $M \times K$ matrix having **orthonormal** columns, then $\boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}_K$ so $\|\boldsymbol{Qx}\|_2 = \|\boldsymbol{x}\|_2$ for all $\boldsymbol{x} \in \mathbb{C}^K$. Show that the converse is true, *i.e.*, if $\boldsymbol{Q}$ is a $M \times K$ matrix for which $\|\boldsymbol{Qx}\|_2 = \|\boldsymbol{x}\|_2$ for all $\boldsymbol{x} \in \mathbb{C}^K$, then $\boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}_K$. Your proof should be general enough to cover the case where $\boldsymbol{Q}$ has complex elements.
Hint. Examine products with standard unit vectors $\boldsymbol{e}_j$ and combinations like $\boldsymbol{e}_j + \boldsymbol{e}_k$.

Pr. 2.
The following **Venn diagram** describes the relationship between **frames** and other matrices.


$\boldsymbol{A}_1$
Rectangular
$N \times M$ wide
$N \le M$
$\boldsymbol{A}_2$
Frame
$0 < \alpha = \sigma_N \le \sigma_1$
$\boldsymbol{\Phi\Phi}'$ invertible
$\boldsymbol{A}_3$
Tight frame
$0 < \alpha = \sigma_N = \sigma_1$
$\boldsymbol{\Phi}^+ = \frac{1}{\sigma_1^2}\boldsymbol{\Phi}'$
$\boldsymbol{A}_4$
Parseval
tight frame
$\sigma_N = \sigma_1 = 1$
$\boldsymbol{\Phi}^+ = \boldsymbol{\Phi}'$
Unitary
$M = N$
$\boldsymbol{U}^+ = \boldsymbol{U}^{-1} = \boldsymbol{U}'$

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

(a) Provide example matrices $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_4$ 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.

(b) A subset of unitary matrices is certain **diagonal** matrices. Describe the necessary and sufficient conditions for a diagonal matrix to be unitary.

Pr. 3.
Consider the following **regularized low-rank approximation** method:

$$\hat{\boldsymbol{X}} = \arg\min_{\boldsymbol{X}} \frac{1}{2} \|\boldsymbol{Y} - \boldsymbol{X}\|_{\mathrm{F}}^2 + \beta \|\boldsymbol{X}\|_* .$$

Determine an expression for the approximation error $\left\|\hat{\boldsymbol{X}} - \boldsymbol{Y}\right\|_{\mathrm{F}}$. Simplify as much as possible.

Pr. 4.
The GD iteration for the LS problem $\arg\min_{\boldsymbol{x}\in\mathbb{R}^N} f(\boldsymbol{x}), \quad f(\boldsymbol{x}) \triangleq \frac{1}{2}\|\boldsymbol{Ax} - \boldsymbol{y}\|_2^2$ for $M \times N$ matrix $\boldsymbol{A}$ and length-M vector $\boldsymbol{y}$ is $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha\boldsymbol{A}'(\boldsymbol{Ax}_k - \boldsymbol{y})$. Instead of using a fixed step size α, an alternative is to do a **line search** to find the best step size *at each iteration*. This variation is called **steepest descent** (or GD with a line search). For a given preconditioning matrix $\boldsymbol{P} \in \mathbb{F}^{N\times N}$, here is how preconditioned steepest descent works:

$$\begin{aligned}
\boldsymbol{d}_k &= -\boldsymbol{P}\nabla f(\boldsymbol{x}_k) = -\boldsymbol{PA}'(\boldsymbol{Ax}_k - \boldsymbol{y}) \\
\alpha_k &= \arg\min_{\alpha} f(\boldsymbol{x}_k + \alpha\boldsymbol{d}_k) \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha_k\boldsymbol{d}_k.
\end{aligned}$$

Find an expression for the step size α_k that is easily implemented. It will depend on $\boldsymbol{x}_k$, $\boldsymbol{d}_k$, $\boldsymbol{A}$, and $\boldsymbol{y}$.
Optional: strive to find an expression that requires as little computation as possible.

Pr. 5.
(Projection onto orthogonal complement of a 1D subspace)

(a) Determine an orthonormal basis for the orthogonal complement of the span of the vector $\boldsymbol{v} = \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$.

(b) Determine an orthonormal basis for the orthogonal complement of the span of the vector $\boldsymbol{z} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}$.

(c) Determine (by hand, no `Julia`) the projection of the vector $\boldsymbol{y} = \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix}$ onto the orthogonal complement of $\text{span}(\{\boldsymbol{z}\})$ *without* using the orthonormal basis you found in the previous part.

(d) Write a function called `orthcomp1` that projects an input vector $\boldsymbol{y}$ onto the orthogonal complement of $\text{span}(\{\boldsymbol{x}\})$ for an (nonzero) input vector $\boldsymbol{x}$ of the same length as $\boldsymbol{y}$.

For full credit, your final version of the code should by computationally efficient, and it should be able to handle input vectors of length 10 million without running out of memory. Hint: your code must *not* call `svd` or `eig` or `I` and the like. This problem can be solved in one line with elementary vector operations.

In `Julia`, your file should be named `orthcomp1.jl` and should contain the following function:

```julia
"""
    z = orthcomp1(y, x)

Project `y` onto the orthogonal complement of `Span({x})`

In:
* `y` vector
* `x` nonzero vector of same length, both possibly very long

Out:
* `z` vector of same length

For full credit, your solution should be computationally efficient.
"""
function orthcomp1(y, x)
```

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

Test your code yourself using the example above (and others) *before* submitting to the autograder. Be sure to test it for very long input vectors.

(e) Submit your code (a screen capture is fine) to gradescope so that the grader can verify that your code is computationally efficient. (The autograder checks only correctness, not efficiency.)

Pr. 6.
(OptShrink for low-rank matrix denoising: small data size)
This problem implements and explores the **OptShrink** method of Prof. Nadakuditi as discussed in the course notes.

(a) Write a function called `optshrink1` that has two inputs. The first input $\boldsymbol{Y}$ is a noisy data array modeled to be of the form $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where $\boldsymbol{X}$ is a low-rank matrix and $\boldsymbol{\varepsilon}$ is a noise matrix of the same size. (This is a very typical situation in practice.) The second argument is guess of the rank of $\boldsymbol{X}$. (In practice one might guess this rank by examining the scree plot for $\boldsymbol{Y}$.) The function must return the rank-r estimate $\hat{\boldsymbol{X}}$ computed by the OptShrink method.

For this problem, you are permitted to use Matlab code for OptShrink that you find on the web as inspiration, as long as you cite it in your solution code.

For this problem it is acceptable if your function works only for matrices where the dimensions are not too large.

In `Julia`, your file should be named `optshrink1.jl` and should contain the following function:

```
"""
`Xh = optshrink1(Y::AbstractMatrix, r::Int)

Perform rank-r denoising of data matrix Y using OptShrink
using the method by Prof. Raj Rao Nadakuditi in this May 2014
IEEE Transactions on Information paper:
http://doi.org/10.1109/TIT.2014.2311661

In:
- Y     2D array where Y = X + noise and goal is to estimate X
- r     estimated rank of X

Out:
- Xh    rank-r estimate of X using OptShrink weights for SVD components

This version works only if the size of Y is sufficiently small,
because it performs calculations involving arrays roughly of
size(Y'*Y) and size(Y*Y') so neither dimension of Y can be large.
"""
function optshrink1(Y::AbstractMatrix, r::Int)
```

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

The template above includes type declarations for the function arguments. You are not required to use such declarations, nor will such declarations be on exam problems, but those of you who want to really learn `Julia` will want to learn about them eventually so an example can help illustrate.

A benefit of such declarations is that if a naive user tries to call the function with a wrong argument type, then the error message will be more informative. Another benefit is that one can use the same function names for different argument types, a key feature of `Julia` called multiple dispatch.

(b) Test your code yourself using your own $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where $\boldsymbol{X}$ is a low-rank array and the noise level is small, *before* submitting to the autograder.

After your code passes the autograder, complete the following test code so that it computes both the OptShrink estimate $\hat{\boldsymbol{X}}$ and the standard rank-1 approximation to $\boldsymbol{Y}$ and see which is closer to the true $\boldsymbol{X}$.

```
# include("optshrink1.jl") # uncomment if you need this
using Random: seed!
using LinearAlgebra
seed!(0)
X = randn(30) * randn(20)'
Y = X + 40 * randn(size(X))
Xh_opt = optshrink1(Y, 1)
# Xh_lr = # you finish this to make the conventional rank-1 approximation

@show norm(Xh_opt - X)
@show norm(Xh_lr - X)
```

Submit (a screen shot of) your modified test code and the `norm` output values to gradescope.

Does OptShrink provide a better estimate of $\boldsymbol{X}$ than classical low-rank approximation?

Pr. 7.
(OptShrink for low-rank matrix denoising problems: big data)

(a) The OptShrink method needs to evaluate the function $D(z, \boldsymbol{S})$ and its derivative $D'(z, \boldsymbol{S})$ for a certain (rectangular) diagonal matrix $\boldsymbol{S}$. (See course notes.) If one of the two dimensions of this matrix is large, then either $\boldsymbol{S}\boldsymbol{S}'$ or $\boldsymbol{S}'\boldsymbol{S}$ may become impractical to store and manipulate.

The course notes show how to compute $D(z, \boldsymbol{S})$ without forming $\boldsymbol{S}\boldsymbol{S}'$ or $\boldsymbol{S}'\boldsymbol{S}$, thereby providing a step towards making OptShrink suitable for larger problems. Similarly, for this problem you can first derive an efficient expression for $D'(z, \boldsymbol{S})$. Alternatively you can simply think about how to modify the code directly by thinking about how to avoid dealing with large (but diagonal) matrices.

Write a function called `optshrink2` that has the same inputs and output as `optshrink1` but that is practical to use even when one of the two dimensions of $\boldsymbol{Y}$ is very large, *e.g.*, if $\boldsymbol{Y}$ is $10^6 \times 100$.

If you find any code online that solves this problem, you may use it for inspiration as long as you cite it. Please let me know by email if you do, because I am not aware of any!

Your solution should not use `tr` (trace) or `inv` or other such functions of large matrices.

In `Julia`, your file should be named `optshrink2.jl` and should contain the following function:

```julia
"""
`Xh = optshrink2(Y::AbstractMatrix, r::Int)`

Perform rank-r denoising of data matrix Y using OptShrink
using the method by Prof. Raj Rao Nadakuditi in this May 2014
IEEE Transactions on Information paper:
http://doi.org/10.1109/TIT.2014.2311661

In:
- Y     2D array where Y = X + noise and goal is to estimate X
- r     estimated rank of X

Out:
- Xh    rank-r estimate of X using OptShrink weights for SVD components

This version works even if one of the dimensions of Y is large,
as long as the other is sufficiently small.
"""
function optshrink2(Y::AbstractMatrix, r::Int)
```

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

Test your code yourself using your own $\boldsymbol{Y} = \boldsymbol{X} + \boldsymbol{\varepsilon}$ where $\boldsymbol{X}$ is a low-rank array and the noise level is small, *before* submitting to the autograder.

Your solution should produce identical (or nearly identical) results as your `optshrink1` for small problems.

(b) After your code passes the autograder, complete the following test code so that it computes both the OptShrink estimate $\hat{\boldsymbol{X}}$ and the standard rank-1 approximation to $\boldsymbol{Y}$ and see which is closer to the true $\boldsymbol{X}$.

```julia
# include("optshrink2.jl") # uncomment if needed
using LinearAlgebra
using Random: seed!
seed!(0)
X = randn(10^5) * randn(100)' / 8 # test large case now
Y = X + randn(size(X))
Xh_opt = optshrink2(Y, 1)
# Xh_lr = # you finish this part

@show norm(Xh_opt - X)
@show norm(Xh_lr - X)
```

Submit (a screen shot of) your modified test code and the `norm` output values to gradescope.

Does OptShrink provide a better estimate of $\boldsymbol{X}$ than classical low-rank approximation?

(c) (Not graded) Will your solution work if both dimensions of $\boldsymbol{Y}$ are large? Why or why not?

Pr. 8.
(Shrinkage using $\|\cdot\|_p$ with $p = 1/2$ non-convex) When v is real and nonnegative, the course notes derive the following minimization problem solution:

$$\arg\min_{x\geq 0} \frac{1}{2}(v-x)^2 + \beta|x| = [v-\beta]_+.$$

A drawback of this **soft thresholding** formulation in the context of **low-rank matrix approximation** (and in other sparsity problems) is that even the large singular values are shrunk by β. An alternative is to replace the **regularizer** $|x|$ with a function that increases less rapidly, such as $|x|^{1/2}$.
So now consider this minimization problem:

$$\hat{x} = \arg\min_{x\geq 0} f(x; v, \beta), \quad f(x; v, \beta) \triangleq \frac{1}{2}(v-x)^2 + \beta|x|^{1/2}.$$

Letting $t = \sqrt{x}$, we want to minimize $\frac{1}{2}\left|v - t^2\right|^2 + \beta t$. Differentiating yields the (depressed) cubic equation $t^3 - vt + \beta/2 = 0$ that turns out to have 3 real roots when v is sufficiently large. One can solve it using a **trigonometric method** leading to the solution $\hat{t} = 2\sqrt{\frac{v}{3}}\cos\left(\frac{1}{3}\arccos\left(-\frac{3\beta}{4v}\sqrt{\frac{3}{v}}\right)\right)$. By considering the second derivative, one can (optionally) verify that this solution is correct when $v > \frac{3}{2}\beta^{2/3}$. Thus the final solution is

$$\hat{x} = \begin{cases} \frac{4}{3}v\cos^2\left(\frac{1}{3}\arccos\left(-\frac{3^{3/2}\beta}{4v^{3/2}}\right)\right), & v > \frac{3}{2}\beta^{2/3} \\ 0, & \text{otherwise.} \end{cases}$$

(a) Write a `Julia` function that evaluates this solution for a given regularization parameter β and for an array of input v values.

In `Julia`, your file should be named `shrink_p_1_2.jl` and should contain the following function:

```
"""
    out = shrink_p_1_2(v, reg::Real)

Compute minimizer of ``1/2 |v - x|^2 + reg |x|^p``
for `p=1/2` when `v` is real and nonnegative.

In:
* `v` scalar, vector, or array of (real, nonnegative) input values
* `reg` regularization parameter

Out:
* `xh`  solution to minimization problem for each element of `v`
(same size as `v`)

"""
function shrink_p_1_2(v, reg::Real)
```

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

You will use this function for a low-rank matrix completion problem in a later HW. Like $\|\cdot\|_0$, the $|x|^{1/2}$ function is **non-convex**, but at least it is continuous, whereas $\|\cdot\|_0$ is discontinuous.

(b) Plot the solution for `v = LinRange(0, 8*reg, 801)` when $\beta = 2$. Also plot the soft thresholding function on this same graph, and show the line of identity.

Discuss what advantages this shrinkage function appears to have over soft thresholding.

Submit your plot and comments to gradescope.

Pr. 9.
(Localization using inter-node distances)

(a) Write a function called `dist2locs` that implements the multidimensional scaling (MDS) algorithm. Given an $n \times n$ dissimilarity matrix $\boldsymbol{D}$ and an embedding dimension $d > 0$, your function should return the $n \times d$ matrix `Xr` whose rows contain the relative coordinates of the n vertices described by $\boldsymbol{D}$.

This problem uses the rows for coordinates, where as the course notes used the columns. Such tranpositions between conventions are common in practice, and here you must adapt the ideas in the course notes to this problem's specifications.

Because the coordinates returned by MDS are unique only up to rotation/translation, one needs to impose additional constraints on `Xr` to make it unique. In particular, for this problem, you should transform `Xr` so that

- The ith column of `Xr` is computed from the ith largest eigenpair of $\boldsymbol{C}\boldsymbol{C}^T$. (Reorder columns if needed.)
- The centroid of the coordinates is zero.
- The largest magnitude element of each column of `Xr` is positive (flip signs if needed).

In `Julia`, your file should be named `dist2locs.jl` and should contain the following function:

```julia
"""
    Xr = dist2locs(D, d)

In:
* `D` is an `n x n` matrix such that `D[i, j]` is the distance from object `i` to object `j`
* `d` is the desired embedding dimension.

Out:
* `Xr` is an `n x d` matrix whose rows contains the relative coordinates of the `n` objects

Note: MDS is only unique up to rotation and translation,
so we enforce the following conventions on Xr in this order:
* [ORDER] `Xr[:,i]` corresponds to ith largest eigenpair of `C * C'`
* [CENTER] The centroid of the coordinates is zero
* [SIGN] The largest magnitude element of `Xr[:, i]` is positive
"""
function dist2locs(D, d)
```

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

(b) Apply your function to the following distance matrix:

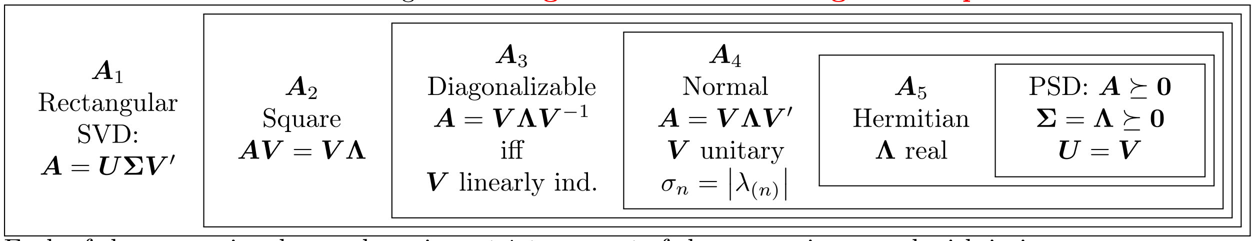
```
  0.0     8.0     8.0    11.314  12.649   4.0    8.944  8.944  4.0
  8.0     0.0    11.314   8.0    12.649   4.0    8.944  4.0    8.944
  8.0    11.314   0.0     8.0     5.657   8.944  4.0    8.944  4.0
 11.314   8.0     8.0     0.0     5.657   8.944  4.0    4.0    8.944
 12.649  12.649   5.657   5.657   0.0    12.0    4.0    8.944  8.944
  4.0     4.0     8.944   8.944  12.0     0.0    8.0    5.657  5.657
  8.944   8.944   4.0     4.0     4.0     8.0    0.0    5.657  5.657
  8.944   4.0     8.944   4.0     8.944   5.657  5.657  0.0    8.0
  4.0     8.944   4.0     8.944   8.944   5.657  5.657  8.0    0.0
```

Make a scatter plot of the estimated coordinates and submit that plot to gradescope.

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

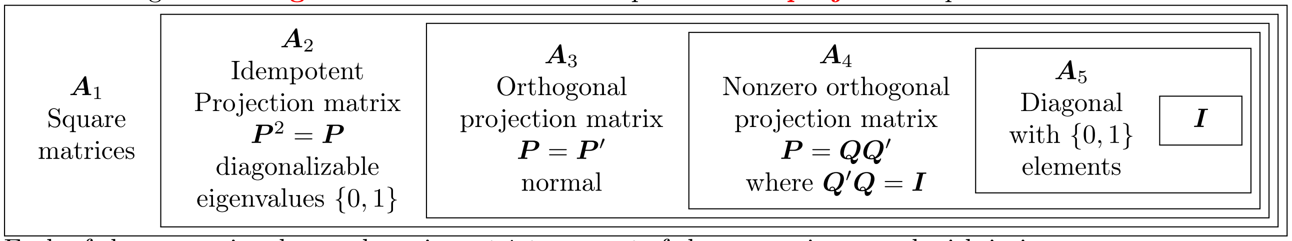
Pr. 10.
Prove that $\sigma_1^2(\boldsymbol{A})\boldsymbol{I} \succeq \boldsymbol{A}'\boldsymbol{A}$. This inequality is important for establishing convergence of certain iterative algorithms for **least-squares** and related problems.

Pr. 11.
The Ch. 2 notes show the following **Venn diagram** of matrices and **eigendecompositions**:


$\boldsymbol{A}_1$ Rectangular SVD: $\boldsymbol{A} = \boldsymbol{U\Sigma V}'$
$\boldsymbol{A}_2$ Square $\boldsymbol{AV} = \boldsymbol{V\Lambda}$
$\boldsymbol{A}_3$ Diagonalizable $\boldsymbol{A} = \boldsymbol{V\Lambda V}^{-1}$ iff $\boldsymbol{V}$ linearly ind.
$\boldsymbol{A}_4$ Normal $\boldsymbol{A} = \boldsymbol{V\Lambda V}'$ $\boldsymbol{V}$ unitary $\sigma_n = |\lambda_{(n)}|$
$\boldsymbol{A}_5$ Hermitian $\boldsymbol{\Lambda}$ real
PSD: $\boldsymbol{A} \succeq \boldsymbol{0}$ $\boldsymbol{\Sigma} = \boldsymbol{\Lambda} \succeq \boldsymbol{0}$ $\boldsymbol{U} = \boldsymbol{V}$

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.
For example, the matrix $\boldsymbol{A}_1 = [1\ 2]$ is rectangular, but not square. Now you do $\boldsymbol{A}_2, \ldots, \boldsymbol{A}_5$.

Pr. 12.
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. 13.
For $\boldsymbol{Y} \in \mathbb{C}^{M \times N}$, determine (analytically) the solution of the **regularized low-rank approximation** method:

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

(This problem was suggested by a student in f17 for Exam2.)
Think about your solution and consider whether it seems to be a good method for low-rank approximation.

Pr. 14.
Prove that if $\boldsymbol{P}$ is a projection matrix, and, for a square matrix $\boldsymbol{A}$, we define $e^{\boldsymbol{A}} = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{A}^k$, where $k!$ denotes factorial of k, then

$$e^{\boldsymbol{P}} = \boldsymbol{I} + (e - 1)\boldsymbol{P}.$$

Hint: what is $\boldsymbol{P}^k$ when $\boldsymbol{P}$ is a projection matrix?
