**Pr. 1.**
**(Gradient projection 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:

$$\boldsymbol{x}_{\text{NNLS}} = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \text{ subject to } \boldsymbol{x} \geq \boldsymbol{0},$$

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 unconstrained 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

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2 \text{ subject to } \boldsymbol{x} \in \mathcal{C},$$

where $\mathcal{C}$ denotes a **convex set**, via a **gradient projection** or **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

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

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 $\boldsymbol{y}$. Optional inputs are the step size $\mu$, and the initial guess $\boldsymbol{x}_0$. The function *must* compute a practical value for the step size $\mu$ if none is provided. This value should be practical to compute even for very a large matrix $\boldsymbol{A}$. Hint. See the solution to a previous autograder problem.

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

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

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

# In:
- `A` `M × N` matrix
- `y` vector of length `M`

# Option:
- `mu` step size to use, and must satisfy ``0 < μ < 2 / σ_1(A)^2``
  to guarantee convergence,
  where ``σ_1(A)`` is the first (largest) singular value.
  Ch.6 explains 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, y ; mu::Real=0, x0=zeros(size(A,2)), nIters::Int=200)
```

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

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

```
using Random: seed!
using Statistics: mean
seed!(0) # be sure to use this seed to get correct answers!
M = 100; N = 50; sigma = 2.0
A = randn(M, N)
xtrue = rand(N) # note that xtrue is non-negative
y = A * xtrue + sigma * randn(M)
x0 = A \ y; @show count(<(0), x0), minimum(x0) # negative values
x0[x0 .<= 0] .= mean(x0[x0 .> 0]) # reasonable initial nonnegative 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 1.48.

(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 - y)^2 # cost function
function grad!(g, x) # its gradient
    g[:] = A' * (A * x - y)
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 the `Optim.jl` documentation.
`https://julianlsolvers.github.io/Optim.jl/stable/#user/minimization`

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

Hint. `xnnls[8]` is about 0.99

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

Specifically, plot $\log_{10}(\|\boldsymbol{x}_{\text{NNLS}} - \boldsymbol{x}_k\|)$ as a function of $k \in [0, 100]$ using $\boldsymbol{A}$ and $\boldsymbol{y}$ generated as above. Submit one plot with four curves on it for the four $\mu$ values, with an appropriate legend.
Hint. The curve for the 1.9 case should decrease the fastest.

---

**Pr. 2.**

(a) Let $\boldsymbol{Z}_1$, $\boldsymbol{Z}_2$, …, $\boldsymbol{Z}_K$ and $\boldsymbol{Y}$ denote $M \times N$ matrices. Determine a solution to the "matrix fitting" least-squares optimization problem:

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}=(x_1,\ldots,x_K)\in\mathbb{F}^K}{\arg\min} \left\| \left( \sum\nolimits_{k=1}^{K} x_k \boldsymbol{Z}_k \right) - \boldsymbol{Y} \right\|_{\text{F}}.$$

(b) Specify conditions on the given matrices that ensure $\hat{\boldsymbol{x}}$ is unique.

(c) Write (by hand) a simple `Julia` expression for computing $\hat{\boldsymbol{x}}$ given $\boldsymbol{Y}$ and the $\boldsymbol{Z}_k$ matrices when $K = 3$.

(d) Optional challenge. How would you handle the general $K$ case in `Julia`?

---

**Pr. 3.**
**Matrix norm inequalities** (All norms are **equivalent** in finite dimensions.)
For a matrix $\boldsymbol{A} \in \mathbb{F}^{m \times n}$, consider the following **matrix norms**:

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

Show that the following matrix norm inequalities hold when $\boldsymbol{A}$ has rank $r$. For any part you may use previous parts if helpful.

(a) $\|\boldsymbol{A}\|_2 \leq \|\boldsymbol{A}\|_{\mathrm{F}}$ Hint. Work with singular values for (a)-(d).

(b) $\|\boldsymbol{A}\|_{\mathrm{F}} \leq \sqrt{r}\, \|\boldsymbol{A}\|_2$

(c) $\|\boldsymbol{A}\|_{\mathrm{F}} \leq \|\boldsymbol{A}\|_*$ Hint. Start with $\|\boldsymbol{A}\|_{\mathrm{F}}^2$

(d) Optional. $\|\boldsymbol{A}\|_* \leq \sqrt{r}\, \|\boldsymbol{A}\|_{\mathrm{F}}$ Hint. Consider using convexity of $f(x) = x^2$, as in **Jensen's inequality**.

(e) Optional. $\|\boldsymbol{A}\|_\infty \leq \sqrt{n}\, \|\boldsymbol{A}\|_2$ Hint. First show that $\boldsymbol{x} \in \mathbb{F}^n \Rightarrow \frac{1}{\sqrt{n}} \|\boldsymbol{x}\|_2 \leq \|\boldsymbol{x}\|_\infty \leq \|\boldsymbol{x}\|_2$.

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

(g) Optional. $\frac{1}{\sqrt{m}} \|\boldsymbol{A}\|_1 \leq \|\boldsymbol{A}\|_2 \leq \sqrt{n}\, \|\boldsymbol{A}\|_1$ Hint. Use the preceding inequalities.

These inequalities can provide upper and lower bounds for the largest singular value via simple operations on the elements of the matrix. One application where this is useful is in determining the step size for gradient descent algorithms; there the largest singular value (which equals $\|\boldsymbol{A}\|_2$) has a pivotal role.
Optional challenge. For each of the above inequalities, find a small and simple (but nonzero) matrix $\boldsymbol{A}$ that shows that the inequality is **tight**, i.e., achieved for some $\boldsymbol{A}$. All of the inequalities hold as equalities when $\boldsymbol{A} = [1]$, so to make this more interesting, your matrix must be bigger than a $1 \times 1$ matrix.

---

**Pr. 4.**
**(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) that $\|\boldsymbol{A} + \boldsymbol{B}\|_* \leq \|\boldsymbol{A}\|_* + \|\boldsymbol{B}\|_*$. This problem provides a sufficient condition for equality.
Show that if $\boldsymbol{A}\boldsymbol{B}' = \boldsymbol{0}$ and $\boldsymbol{A}'\boldsymbol{B} = \boldsymbol{0}$ then $\|\boldsymbol{A} + \boldsymbol{B}\|_* = \|\boldsymbol{A}\|_* + \|\boldsymbol{B}\|_*$.
Hint. Construct a valid **compact SVD** for $\boldsymbol{A} + \boldsymbol{B}$ from compact SVDs $\boldsymbol{A} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r'$ and $\boldsymbol{B} = \boldsymbol{X}_s \boldsymbol{\Omega}_s \boldsymbol{Y}_s'$ where $r = \mathrm{rank}(\boldsymbol{A})$ and $s = \mathrm{rank}(\boldsymbol{B})$, using the given properties. Also think about an upper bound for $\mathrm{rank}(\boldsymbol{A}) + \mathrm{rank}(\boldsymbol{B})$.

---

**Pr. 5.**
Prove or disprove. If $\boldsymbol{A}$ and $\boldsymbol{B}$ are $N \times N$ **orthogonal projection** matrices, then the **spectral radius** of their average, $\frac{1}{2}\boldsymbol{A} + \frac{1}{2}\boldsymbol{B}$, is at most 1. Hint. Do not "invent" an incorrect triangle inequality.

---

**Pr. 6.**
Consider two **vector spaces** $\mathcal{V}_\alpha$ and $\mathcal{V}_\beta$ with corresponding **norms** $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$, respectively. A function $f : \mathcal{V}_\alpha \mapsto \mathcal{V}_\beta$ is called an $\mathcal{L}$-**Lipschitz function** with respect to those norms if for every $\boldsymbol{x}, \boldsymbol{z}$ in the domain of $f$, there exists a real constant $\mathcal{L} > 0$ such that

$$\|f(\boldsymbol{x}) - f(\boldsymbol{z})\|_\beta \leq \mathcal{L} \|\boldsymbol{x} - \boldsymbol{z}\|_\alpha.$$

The constant $\mathcal{L}$ is called a **Lipschitz constant** of $f$ and the smallest such constant is called the **best Lipschitz constant**.
We say that $f$ is **Lipschitz continuous** with Lipschitz constant $\mathcal{L}$. In general, the norm on the left hand side of the inequality can differ from the norm on the right hand side. When the norm $\|\cdot\|$ is not specified, it is assumed that $f$ is a Lipschitz function with respect to the Euclidean norm.

(a) Show that $g(\boldsymbol{x}) = \nabla\psi(\boldsymbol{x}) = \boldsymbol{A}'(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})$ is a Lipschitz function of $\boldsymbol{x}$ when $\boldsymbol{A}$ is an $M \times N$ matrix.
Express its best Lipschitz constant concisely.
Note that $g(\boldsymbol{x})$ is the gradient of the **least-squares** cost function $\psi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|_2^2$.

(b) Show that the largest **singular value** of a matrix is a Lipschitz function (of its $MN$ entries), with $\mathcal{L} = 1$, with respect to the **spectral norm** for $\mathcal{V}_\alpha$.
Hint. Use the reverse triangle inequality.

(c) Optional. Repeat (b) but show it with respect to the **Frobenius norm**.
Hint. Use a previous problem.

(d) Optional. Repeat (b) but show it with respect to $\|\cdot\|_1$.

Functions with Lipschitz continuous gradients are important because if the function has a Lipschitz constant $L$, then for any $\boldsymbol{x}, \boldsymbol{z} \in \mathbb{R}^N$,

$$f(\boldsymbol{x}) \le f(\boldsymbol{z}) + \langle \nabla f(\boldsymbol{z}), \boldsymbol{x} - \boldsymbol{z} \rangle + \frac{\mathcal{L}}{2}\|\boldsymbol{x} - \boldsymbol{z}\|_2^2.$$

This inequality is very helpful in analyzing iterative algorithms, such as **gradient descent**, because substituting $\boldsymbol{x} = \boldsymbol{x}_{k+1}$ and $\boldsymbol{z} = \boldsymbol{x}_k$ yields a bound on the value of the cost function after $k$ iterations that can be used to select the step size to ensure that the value of the cost function will decrease after every iteration (unless we are already at a minimizer where the gradient is zero).

---

**Pr. 7.**
Prove or disprove (with the simplest and smallest possible concrete counterexamples) the following statements about the **generalized inverses** of a matrix. (See §6.6 of book.)

(a) Every matrix $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ with rank $r > 0$ and singular values $\sigma_1, \sigma_2, \ldots$ has a generalized inverse matrix $\boldsymbol{G}$ whose spectral norm is $1/\sigma_r$.

(b) Every matrix $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ with rank $0 < r < \min(M, N)$ and singular values $\sigma_1, \sigma_2, \ldots$ for which $\sigma_r = 1$ has a generalized inverse matrix $\boldsymbol{G}$ whose spectral norm is 7.

---

**Pr. 8.**
**(Procrustes method: analysis)** This problem provides the analytical background for aligning shapes using the **Procrustes** method. 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

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

where $\boldsymbol{Q}$ is a $d \times d$ orthogonal matrix (rotation or generalization thereof), $\boldsymbol{\lambda}, \boldsymbol{\mu} \in \mathbb{R}^d$ 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 $\alpha$, and then translating again by $\boldsymbol{\mu}$. However, optimizing *both* $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ would be redundant, because we can replace $\boldsymbol{\lambda} \to \boldsymbol{\lambda} + \boldsymbol{\Delta}$ and $\boldsymbol{\mu} \to \boldsymbol{\mu} + \alpha \boldsymbol{Q} \boldsymbol{\Delta}$ for any $\boldsymbol{\Delta} \in \mathbb{R}^d$ without changing $\hat{\boldsymbol{A}}$. Thus, without loss of generality, we choose

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

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

$$(\alpha_*, \boldsymbol{\mu}_*, \boldsymbol{Q}_*) = \underset{\alpha \geq 0, \ \boldsymbol{\mu} \in \mathbb{R}^d, \ \boldsymbol{Q} : \boldsymbol{Q}'\boldsymbol{Q} = \boldsymbol{I}_d}{\arg\min} \left\| \boldsymbol{B} - \alpha \boldsymbol{Q} \left( \boldsymbol{A} - \boldsymbol{\mu}_A \boldsymbol{1}_n' \right) - \boldsymbol{\mu} \boldsymbol{1}_n' \right\|_{\mathrm{F}}. \tag{2}$$

Show that the solution is

$$\boldsymbol{\mu}_* = \boldsymbol{\mu}_B, \quad \boldsymbol{Q}_* = \boldsymbol{U} \boldsymbol{V}', \quad \alpha_* = \frac{\mathrm{Tr}(\boldsymbol{B}_0 \boldsymbol{A}_0' \boldsymbol{Q}_*')}{\mathrm{Tr}(\boldsymbol{A}_0 \boldsymbol{A}_0')},$$

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

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

Hint. First show that $\boldsymbol{\mu} = \boldsymbol{\mu}_*$ minimizes (2) for any fixed $\alpha$ and $\boldsymbol{Q}$ (it is a **least-squares** problem). Then show that $\boldsymbol{Q} = \boldsymbol{Q}_*$ minimizes (2) for any fixed $\alpha$ when $\boldsymbol{\mu} = \boldsymbol{\mu}_*$. Finally, show that $\alpha = \alpha_*$ minimizes (2) when $\boldsymbol{\mu} = \boldsymbol{\mu}_*$ and $\boldsymbol{Q} = \boldsymbol{Q}_*$.
Hint. With $\boldsymbol{\mu} = \boldsymbol{\mu}_*$ and fixed $\alpha$, (2) is a Procrustes problem. Along the way, it will be useful to remember that $\boldsymbol{Z}$ and $\alpha \boldsymbol{Z}$ have the same sets of singular vectors for any matrix $\boldsymbol{Z}$ when $\alpha > 0$.

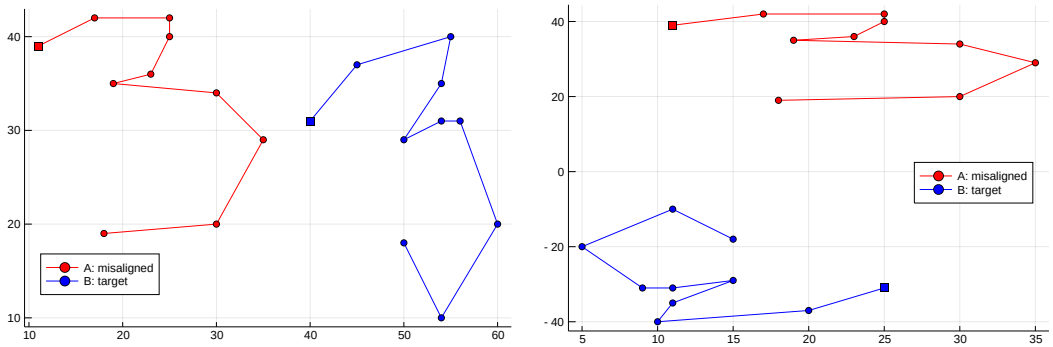
---

**Pr. 9.**
**(Procrustes method: practical implementation)**
In this problem you will align shapes using **Procrustes analysis**.
You can visualize the two datasets in `Julia` via the following code.

```
using Plots

function reader(url::String, T::DataType, dims::Dims)
    data = Array{T}(undef, dims)
    read!(download(url), data)
    return data
end

if !@isdefined(data) # load data if needed (needs web access)
    url = "http://web.eecs.umich.edu/~fessler/course/551/data/digits/"
    files = ["digit1a2x10int16.dat", "digit1b2x10int16.dat",
            "digit2a2x10int16.dat", "digit2b2x10int16.dat"]
    data = [reader(url * file, Int16, (2,10)) for file in files]
end
    A = data[1]; B = data[2] # dataset 1
#    A = data[3]; B = data[4] # dataset 2

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

```
plot!(B[1,:], B[2,:], color=:blue, marker=:circle, label="B: target")
scatter!([A[1,1]], [A[2,1]], color=:red,  marker=:square, label="")
scatter!([B[1,1]], [B[2,1]], color=:blue, marker=:square, label="")
```

In the above code, $\boldsymbol{A}$ 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

(a) Using the expressions from a previous problem, 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:

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

In:
* `B` and `A` are `d × 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 × n` matrix containing `A` Procrustes-aligned to `B`

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

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

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

(b) 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).

(c) Report the resulting error $\|\hat{\boldsymbol{A}} - \boldsymbol{B}\|_{\mathrm{F}}$ for both datasets.

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

**Pr. 10.**
Suppose $\boldsymbol{A}$ and $\boldsymbol{B}$ are symmetric. Show that $\text{trace}(\boldsymbol{AB}) = \text{vec}(\boldsymbol{A})^\top \text{vec}(\boldsymbol{B})$.
What is the corresponding property when neither are symmetric?

---

**Pr. 11.**
Suppose $\boldsymbol{A} \in \mathbb{F}^{19 \times 48}$ has rank 8. 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. 12.**
Show that the weighted Euclidean norm $\|\boldsymbol{x}\|_{\boldsymbol{W}}$ is a valid vector **norm** iff $\boldsymbol{W}$ is a **positive definite** matrix.
Assume $\boldsymbol{W}$ is (Hermitian) symmetric.

---

**Pr. 13.**

(a) Show that if $f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})$ are all **convex functions** of $\boldsymbol{x}$, then the following are also **convex functions**:

(1) $\sum_{i=1}^n f_i(\boldsymbol{x})$.
(2) $\sum_{i=1}^n w_i f_i(\boldsymbol{x})$ for scalars $w_i \geq 0$.
(3) Provide an example wherein $w_i$ being negative breaks the convexity.
(4) $\max\{f_1, f_2, \ldots, f_n\}$.

(b) Which of the following cost functions are convex functions of $\boldsymbol{x}$? Assume that $\boldsymbol{A}$ and $\boldsymbol{D}$ are matrices and $\boldsymbol{b}$ is a vector with compatible dimensions. Justify your answer: using the results from a previous part may help.

(1) $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 + \|\boldsymbol{Dx}\|_2^2$.
(2) $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 + \|\boldsymbol{Dx}\|_1$.
(3) $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 - \|\boldsymbol{Dx}\|_2^2$.
(4) $\|\boldsymbol{Ax} - \boldsymbol{b}\|_2^2 - \|\boldsymbol{Dx}\|_1$.

---

**Pr. 14.**
Let $\boldsymbol{A} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$ and $\boldsymbol{B} = \begin{bmatrix} 2 & 0 \\ 2 & 3 \\ 0 & 0 \end{bmatrix}$. Determine the cosine of the **angle between subspaces** $\mathcal{R}(\boldsymbol{A})$ and $\mathcal{R}(\boldsymbol{B})$.

---

**Pr. 15.**
Let $\boldsymbol{A}$ be an $M \times N$ matrix with full column rank, and let $\boldsymbol{B}$ be an $M \times K$ matrix.

(a) Find a concise expression for the solution to

$$\hat{\boldsymbol{X}} = \underset{\boldsymbol{X} \in \mathbb{F}^{N \times K}}{\arg\min} \left\| \boldsymbol{AX} - \boldsymbol{B} \right\|_{\text{F}}.$$

(b) Write (by hand) a short `Julia` expression for computing $\hat{\boldsymbol{X}}$ efficiently given $\boldsymbol{A}$ and $\boldsymbol{B}$ in the case that $\boldsymbol{B}$ is very wide, *i.e.*, $M \approx N \ll K$.

---

**Pr. 16.**
Express $\text{trace}\big((\boldsymbol{AA}')^2\big)$ concisely in terms of a Schatten $p$-norm.

---

**Pr. 17.**

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

(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 $\mathcal{B}_b \triangleq \left\{ \boldsymbol{B} \in \mathbb{F}^{M \times K} : \|\boldsymbol{B}\|_2 = b \right\}$. Show that for any $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ and any $b \geq 0$, and any $M, N, K \in \mathbb{N}$, there exists $\boldsymbol{B} \in \mathcal{B}_b$ such that $\left\| \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} \end{bmatrix} \right\|_2^2 = \|\boldsymbol{A}\|_2^2 + \|\boldsymbol{B}\|_2^2$. 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. 18.**

This problem explores extensions of **unitary invariance** of norms. In this problem, let $\boldsymbol{A} \in \mathbb{F}^{M \times N}$ and assume $\boldsymbol{X}$ is a $K \times M$ matrix with orthonormal columns and $\boldsymbol{Y}$ is an $N \times L$ matrix with orthonormal rows. (Do not assume $\boldsymbol{X}$ or $\boldsymbol{Y}$ are square.)

(a) Prove that $\|\boldsymbol{X}\boldsymbol{A}\boldsymbol{Y}\|_{\mathrm{F}} = \|\boldsymbol{A}\|_{\mathrm{F}}$.

This is a kind of generalization of unitary invariance. (Maybe it should be called "orthonormal invariance" or "semi-unitary invariance"?)

(b) For any matrix $\boldsymbol{B}$ with $J \in \mathbb{N}$ rows, define the function $f(\boldsymbol{B}) = J \|\boldsymbol{B}\|_*$. Show that $f(\boldsymbol{B})$ is a unitarily invariant norm.

(c) Prove or disprove: $f(\boldsymbol{X}\boldsymbol{A}\boldsymbol{Y}) = f(\boldsymbol{A})$.

(d) Consider the following "unified" matrix norm defined in Ch. 6: $\|\boldsymbol{A}\|_{K,p} \triangleq \left( \sum_{k=1}^{K} \sigma_k^p \right)^{1/p}$, where $K \in \mathbb{N}$ and $1 \leq p < \infty$.
It is easy to verify that this "unified" norm is unitarily invariant.
Prove or disprove: $\|\boldsymbol{A}\|_{K,p} = \|\boldsymbol{X}\boldsymbol{A}\boldsymbol{Y}\|_{K,p}$.

---