Monotonicity of loss and regularizer as the regularization parameter changes. In regularized empirical risk minimization, we choose the parameter $\theta \in \mathbb{R}^d$ to minimize the regularized empirical risk, $L(\theta) + \lambda r(\theta)$, where $L(\theta)$ is the empirical risk, $r(\theta)$ is the regularizer, and $\lambda > 0$ is the regularization hyper-parameter. (The exact form of the functions $L$ and $r$ is irrelevant in this problem.) The hyper-parameter $\lambda$ is used to trade off the two objectives, $L(\theta)$ and $r(\theta)$. Intuition suggests that as $\lambda$ increases, $r(\theta)$ decreases while $L(\theta)$ increases. In this exercise we verify that this is the case.

Suppose $0 < \lambda \leq \tilde{\lambda}$. Let $\theta^*$ minimize $L(\theta) + \lambda r(\theta)$, and $\tilde{\theta}^*$ minimize $L(\theta) + \tilde{\lambda} r(\theta)$.

(a) Show that $r(\theta^*) \geq r(\tilde{\theta}^*)$. In other words, increasing $\lambda$ will never make our regularization error larger.

(b) Show that $L(\theta^*) \leq L(\tilde{\theta}^*)$. That is, increasing $\lambda$ will never decrease our training error.

Hint. Use the fact that $\theta^*$ is the minimizer of $L(\theta) + \lambda r(\theta)$ and similarly for $\tilde{\theta}^*$. This means that for any $\theta$, we have $L(\theta) + \lambda r(\theta) \geq L(\theta^*) + \lambda r(\theta^*)$, and similarly for $\tilde{\theta}^*$.

Solution.

(a) By definition of a minimizer we have

$$L(\tilde{\theta}^*) + \tilde{\lambda} r(\tilde{\theta}^*) \geq L(\theta^*) + \lambda r(\theta^*),$$

and similarly for $\tilde{\theta}$,

$$L(\theta^*) + \tilde{\lambda} r(\theta^*) \geq L(\tilde{\theta}^*) + \tilde{\lambda} r(\tilde{\theta}^*).$$

We can subtract inequalities of opposite directions. Subtracting the two inequalities yields

$$L(\tilde{\theta}^*) + \lambda r(\tilde{\theta}^*) - L(\tilde{\theta}^*) - \tilde{\lambda} r(\tilde{\theta}^*) \geq L(\theta^*) + \lambda r(\theta^*) - L(\theta^*) - \tilde{\lambda} r(\theta^*)$$

$$(\lambda - \tilde{\lambda}) r(\tilde{\theta}^*) \geq (\lambda - \tilde{\lambda}) r(\theta^*),$$

and moving $r(\theta^*)$ to the left hand side yields

$$(\lambda - \tilde{\lambda}) (r(\tilde{\theta}^*) - r(\theta^*)) \geq 0.$$

But, since $\tilde{\lambda} \geq \lambda$, then

$$r(\tilde{\theta}^*) - r(\theta^*) \leq 0.$$

Thus,

$$r(\tilde{\theta}^*) \leq r(\theta^*).$$
(b) We can use the same trick as before, except that we multiply both sides of the inequalities by $1/\lambda$, $1/\tilde{\lambda}$ respectively, then,

$$(1/\lambda)\mathcal{L}(\tilde{\theta}^*) + r(\tilde{\theta}^*) \geq (1/\lambda)\mathcal{L}(\theta^*) + r(\theta^*),$$

and as before

$$(1/\tilde{\lambda})\mathcal{L}(\theta^*) + r(\theta^*) \geq (1/\tilde{\lambda})\mathcal{L}(\tilde{\theta}^*) + r(\tilde{\theta}^*).$$

Subtracting both inequalities yields

$$
(1/\lambda - 1/\tilde{\lambda}) \mathcal{L}(\tilde{\theta}^*) \geq (1/\lambda - 1/\tilde{\lambda}) \mathcal{L}(\theta^*),
$$

and rearranging yields,

$$
\left(1/\lambda - 1/\tilde{\lambda}\right) \left(\mathcal{L}(\tilde{\theta}^*) - \mathcal{L}(\theta^*)\right) \geq 0.
$$

Since $\lambda \leq \tilde{\lambda}$, we have $
(1/\lambda - 1/\tilde{\lambda}) \geq 0$,

so

$$\mathcal{L}(\tilde{\theta}^*) - \mathcal{L}(\theta^*) \geq 0.$$

Thus,

$$\mathcal{L}(\tilde{\theta}^*) \geq \mathcal{L}(\theta^*).$$

2. **Sequential outlier removal.** We consider the problem of fitting data corrupted with outliers, using a simple sequential outlier removal method. In outlier_rem.json, you will find a $250 \times 10$ matrix $U_{\text{train}}$ and a $250$-vector $v_{\text{train}}$ consisting of raw training input and output data, and a $250 \times 10$ matrix $U_{\text{test}}$ and a $250$-vector $v_{\text{test}}$ consisting of raw test input and output data, respectively. We will work with input and output embeddings $x = \phi(u) = u$ and $y = \psi(v) = v$, and you will use a simple linear predictor (without a constant feature) with square loss to fit the model. We will judge model performance using the RMS error on the test set.

A number of the output data entries in the training set have been corrupted (but in a non-obvious way). You do not know which data points have been corrupted, or how many, but you can assume no more than 50. You will explore a simple sequential method to remove the corrupted data points and form a prediction model.

Repeat the following for 50 iterations:

- Create a linear predictor from the training data set.
- Find the data point in the training data set with the largest prediction error.
- Remove the data point from the training data set.

This results in 50 predictors. Plot the test RMS error for them, versus the number of points removed. Give a guess as to how many of the data points were corrupted, with justification.
*Julia hint.* To remove row $i$ from a matrix $X$ and a vector $y$, use $X[\text{setdiff}(1:\text{end}, i), :]$ and $y[\text{setdiff}(1:\text{end}, i)]$, respectively.

**Solution.** A plot of test RMS error for each iteration is shown below. There were actually 25 outliers in this data set, but estimates between 15–30 were accepted, with proper justification. The justification is that the RMS test error flattens out between iterations 15–30; removing the outliers significantly reduces the test RMS error, helping the predictive power of the model, but removing data points that are not outliers hurts the model’s predictive power.

![Plot of test RMS error versus iteration.](image)

**Figure 1** Plot of test RMS error versus iteration.

```julia
include("readclassjson.jl")
include("writeclassjson.jl")

import PyCall
import PyPlot; const plt = PyPlot; plt.plt.style.use("seaborn")

using Random
using LinearAlgebra
using Statistics

using LaTeXStrings

#Random.seed!(0)

#############################
# Generate data

Data = readclassjson("outlier_rem.json")

X_train = Data["U_train"]
X_test = Data["U_test"]
y_train = Data["v_train"]
y_test = Data["v_test"]

n = 500
d = 10

# X = randn(n,d)
# y = [sum(X[i,:])+0.1*randn() for i=1:n]

# global X_train = X[1:250, :]
# X_test = X[251:500, :]

# global y_train = y[1:250]
# y_test = y[251:500]

# outlier_idxs = shuffle(1:250)[1:25]
# y_train[outlier_idxs] = -y_train[outlier_idxs]

# Data = Dict()
# Data["U_train"] = X_train
# Data["U_test"] = X_test
# Data["v_train"] = y_train
# Data["v_test"] = y_test

# writeclassjson(Data, "outlier_rem.json")

########################################################################

function RMSE(yhat, y)
    mse = sum((yhat .- y).^2)/length(y)
    return sqrt(mse)
end

########################################################################

RMSEs = []
for m=0:125
    theta = X_train\y_train
pred_err = (X_test*theta - y_test).^2
rem_idx = argmax((X_train*theta - y_train).^2)
println(m, " ", RMSE(X_test*theta, y_test), " ", rem_idx)
push!(RMSEs, RMSE(X_test*theta, y_test))

    global X_train = X_train[setdiff(1:end, rem_idx), :]
    global y_train = y_train[setdiff(1:end, rem_idx)]

end

plt.plot(0:125, RMSEs, color="red")
plt.ylabel("Test RMS error")
plt.xlabel("Number of data points removed from training set")
plt.savefig("outlier_rem.pdf", bbox_inches="tight")