MIS8

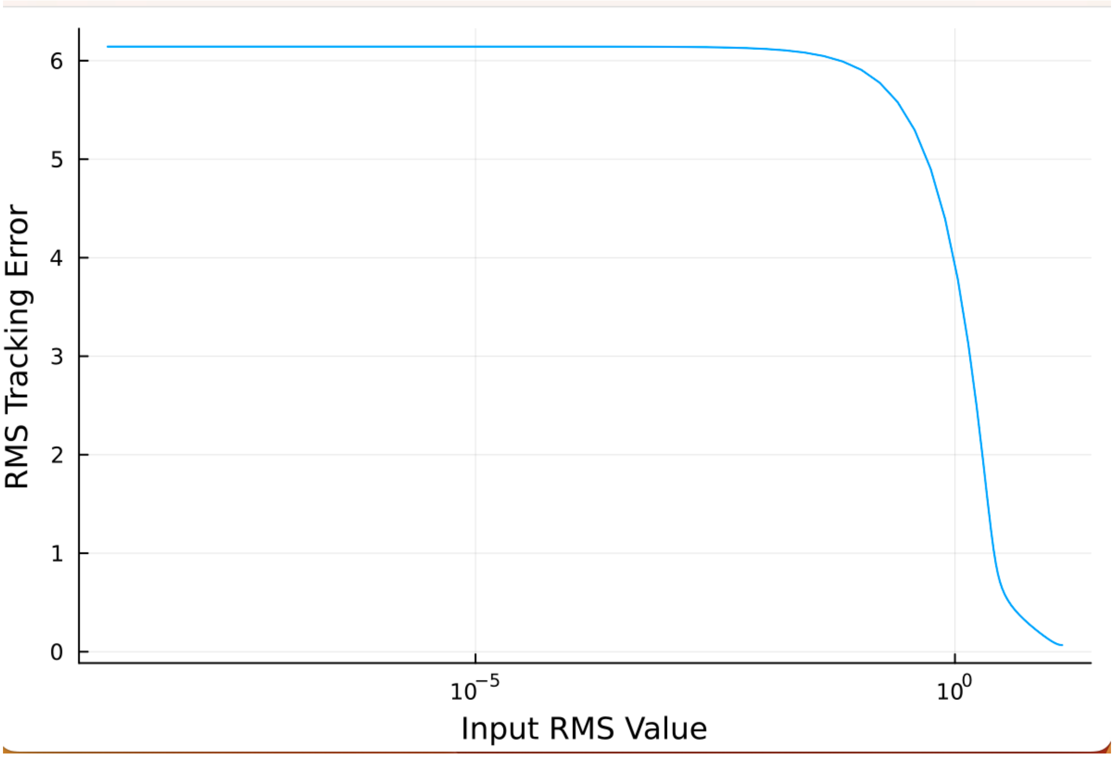
Xinhao Du

A15.1

(a)

For lambdas, we generate 100 values of lambdas $\in 10^{-10}$, 10^{10}

```
using LinearAlgebra
using Plots
using VMLS
using Statistics
n = 100
u = rand(n)
m = 7
h = [0.3, 0.5, 0.6, 0.4, 0.3, 0.2, 0.1]
y_des = zeros(n + m - 1)
y_des[10:39] .= 10
y_des[40:79] .= -5
A = toeplitz(h, n)
λ_range = 10 .^(range(-10, stop=10, length=100))
rms_error = zeros(length(λ_range))
rms_input = zeros(length(λ_range))
for (i, λ) in enumerate(λ_range)
    u_optimal = inv(A' * A + λ * I) * A' * y_des


    rms_error[i] = sqrt(mean((A * u_optimal - y_des).^2))
    rms_input[i] = sqrt(mean(u_optimal.^2))
end
plot(rms_input, rms_error, xaxis=:log10, xlabel="Input RMS Value", ylabel="RMS Tracking Error", label="", legend=:bottomright)
```

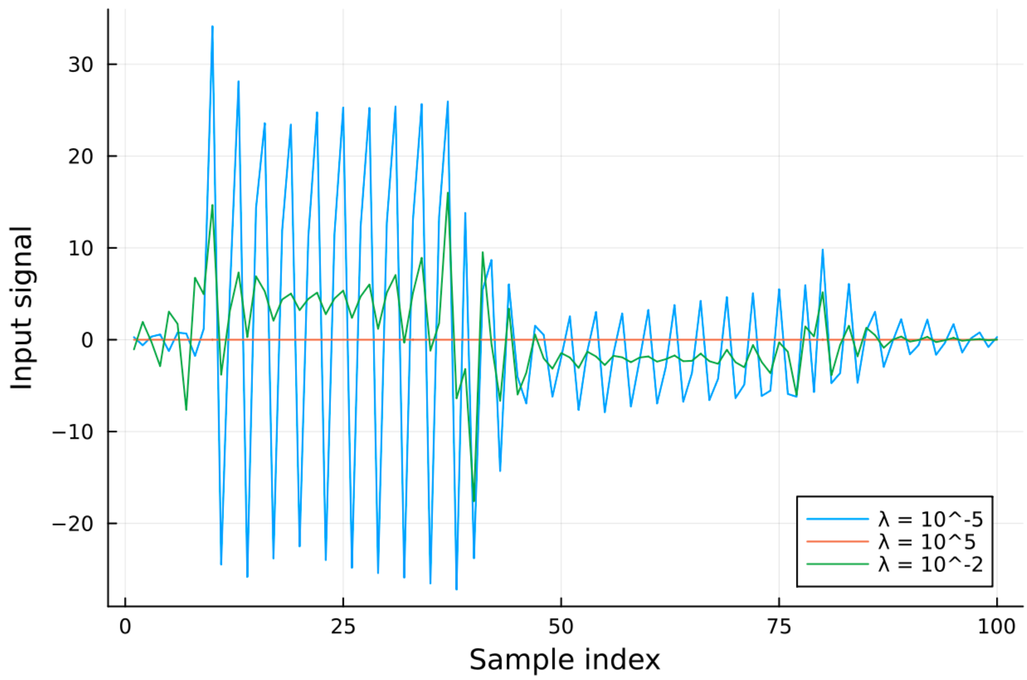


(b)

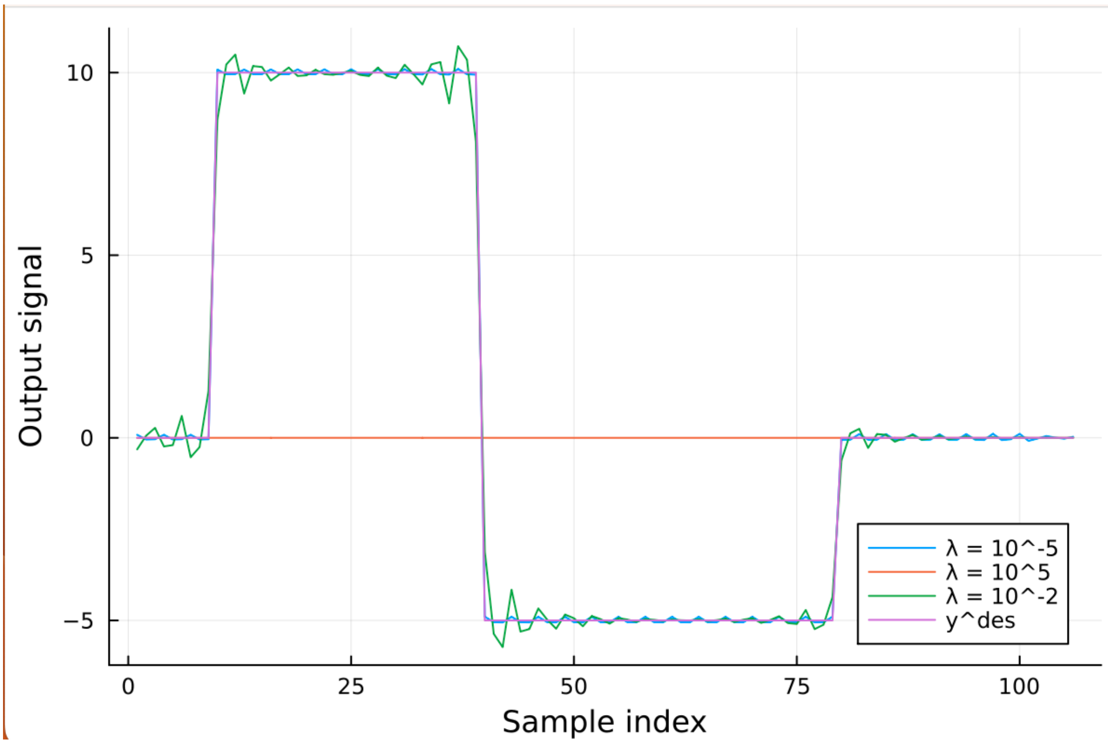
Let's set λ values of 10^-5, 10^5, and 10^-2 to represent too little, too much, and reasonable regularization, respectively.

```
using LinearAlgebra
using Plots
using VMLS
using Statistics
n = 100
u = rand(n)
m = 7
h = [0.3, 0.5, 0.6, 0.4, 0.3, 0.2, 0.1]
y_des = zeros(n + m - 1)
y_des[10:39] .= 10
y_des[40:79] .= -5
A = toeplitz(h, n)
λ_range = 10 .^(range(-10, stop=10, length=100))
u_values = Array{Vector{Float64}, 1}(undef, 3)
y_values = Array{Vector{Float64}, 1}(undef, 3)
for (i, λ) in enumerate([1e-5, 1e5, 1e-2])
    u_optimal = inv(A' * A + λ * I) * A' * y_des
    u_values[i] = u_optimal
    y_values[i] = A * u_optimal
end
plot(u_values[1], label="λ = 10^-5")
plot!(u_values[2], label="λ = 10^5")
plot!(u_values[3], label="λ = 10^-2", xlabel="Sample index", ylabel="Input signal", legend=:bottomright)

plot!(y_values[1], label="λ = 10^-5")
plot!(y_values[2], label="λ = 10^5")
plot!(y_values[3], label="λ = 10^-2")
plot!(y_des, label="y^des", xlabel="Sample index", ylabel="Output signal", legend=:bottomright)
```

Then we can see the input signals are shown on the top subplot, with the input signal for λ = 10^-5 in blue, λ = 10^5 in orange, and λ = 10^-2 in green.



Also, we can see the output signals are shown on the bottom subplot, with the output signal for λ = 10^-5 in blue, λ = 10^5 in orange, λ = 10^-2 in green, and the desired output signal y^des in purple.



A15.2

(a)

```
using LinearAlgebra
using Plots
using VMLS
using Statistics
import Random
Random.seed!(1);
n = 50
N = 300
w_true = randn(n)
v_true = 5
X = randn(n, N)
y = sign.(X'w_true .+ v_true + 10*randn(N))
N_test = 100
X_test = randn(n, N_test)
y_test = sign.(X_test'w_true .+ v_true + 10*randn(N_test))
A = [ones(size(X)[2]) X'];
Theta = A\y;
v = Theta[1,:]
beta = Theta[2:51];
y_hat = X' * beta .+ v;
test_y_hat = X_test' * beta .+ v;
println("The error rate on training data is ",(sum(((y_hat).>0) .!=(y.==1)) + sum(((y_hat).<0) .!=(y.==-1)))/size(y)[1]/2);
println("The error rate on testing data is ",(sum(((test_y_hat).>0) .!=(y_test.==1)) + sum(((test_y_hat).<0) .!=(y_test.==-1)))/size(y_test)[1]/2);
```

Classification error rate on the training: 0.213

Classification error rate on the test sets: 0.35

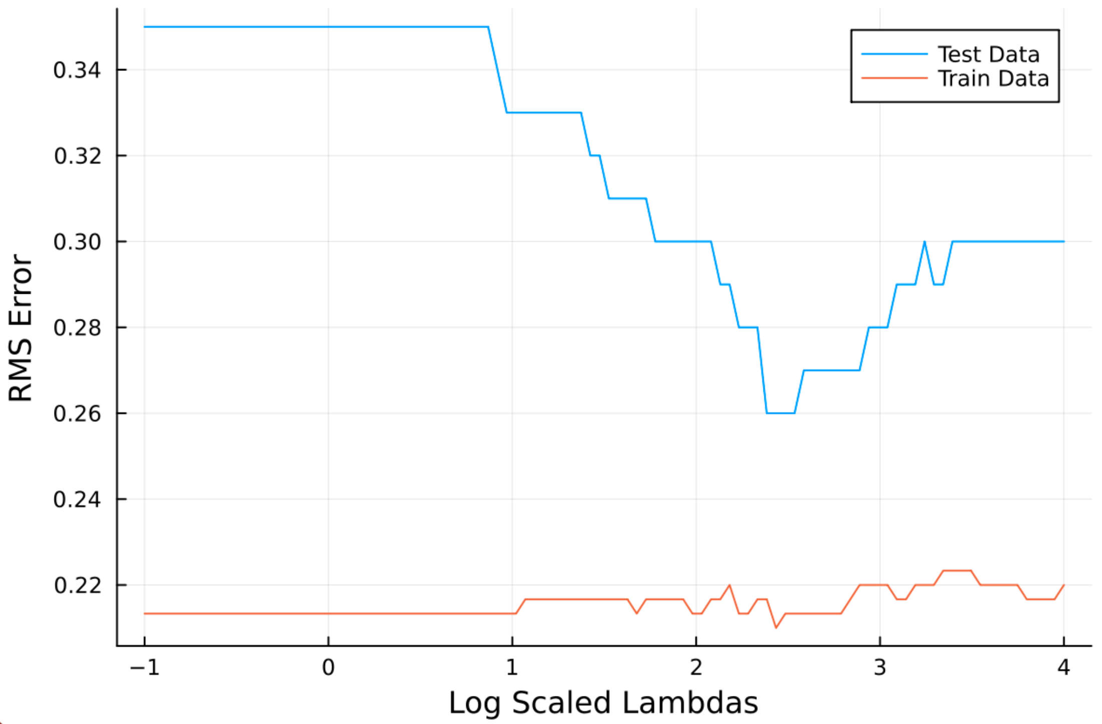
```
The error rate on training data is 0.21333333333333335

julia> println("The error rate on testing data is ",(sum(((test_y_hat).>0) .!=(y
_test.==1)) + sum(((test_y_hat).<0) .!=(y_test.==-1)))/size(y_test)[1]/2);
The error rate on testing data is 0.35
```

(b)

```
regularization_factor =  10 .^ range(-1,4,length = 100);
test_values = []
train_values = []
logvalues = []
for i in regularization_factor
    Theta = inv(transpose(A)*A + i*Matrix{Float64}(I,51,51)) * transpose(A) * y;
    v = Theta[1,:]
    beta = Theta[2:51];
    y_hat = X' * beta .+ v;
    test_y_hat = X_test' * beta .+ v;
    append!(train_values,(sum(((y_hat).>0) .!=(y.==1)) + sum(((y_hat).<0) .!=(y.==-1)))/size(y)[1]/2);
    append!(test_values,(sum(((test_y_hat).>0) .!=(y_test.==1)) + sum(((test_y_hat).<0) .!=(y_test.==-1)))/size(y_test)[1]/2);
    append!(logvalues,log10(i));
end
plot(logvalues, xlabel = "Log Scaled Lambdas", ylabel = "RMS Error", train_values, label = "Train Data")
plot(logvalues, xlabel = "Log Scaled Lambdas", ylabel = "RMS Error", test_values, label = "Test Data")
println("Reasonable lambda for train data is ", regularization_factor[findmin(train_values)[2]]);
```

```
julia> println("Reasonable lambda for train data is ", regularization_factor[fin
dmin(train_values)[2]]);
Reasonable lambda for train data is 271.85882427329403
```

As shown in the picture, reasonable lambda for train data is 271.85882427329403.



A15.6

15.6

I think Oscar is right. First, we consider Bob's approach.

$f(x) = \|Ax - b_1\|^2 + \cdots + \|Ax - b_K\|^2$

$= (Ax - b_1)^T (A_1 - b_1) + \cdots + (Ax - b_K)^T (Ax - b_K)$

$= x^T (A^T A) x - 2(b_1 + \cdots b_K)^T Ax + (b_1^T b_1 + \cdots + b_K^T b_K)$

The third term is just a constant, so we won't consider it.

$f(x) = x^T (A^T A) x - 2(b_1 + \cdots b_K)^T Ax$

$f'(x) = 2A^T (Ax - (\frac{1}{K})(b_1 + \cdots b_K)) = 0$

So, we have $x = (A^T A)^{-1} A^T (\frac{1}{K})(b_1 + \cdots + b_K)$, which is exactly the solution proposed by Alice. In conclusion, both ways will end up with the same choise of x.

A15.8

I would choose λ = 3.0, which gives the lowest test RMS error and therefore seems to provide a good balance between underfitting and overfitting.

A15.10

The correct answer is (a), the training RMS error will stay the same or increase.

A16.1

```
using LinearAlgebra
using LinearLeastSquares
A = randn(10, 100)
b = randn(10)

x1 = A' * inv(A * A') * b
x2 = pinv(A) * b
x3 = A \ b
x4 = llsq(A, b; trans=true, bias=false).coef
norm(x1 - x2)
norm(x1 - x3)
norm(x1-x4)
tolerance = 1e-10
if norm(x1 - x2) < tolerance && norm(x1 - x3) < tolerance && norm(x1 - x4) < tolerance
    println("All solutions are close to each other.")
else
    println("Solutions are not close to each other.")
end
```

Then we can get the value of the matrix A and the vector b.

```
julia> A = randn(10, 100)
10×100 Matrix{Float64}:
 -0.350426    0.0816251  -1.09602   …  -1.25247    0.615233  -0.757082
  0.453306    0.0436246  -0.246713     -0.687518  -0.302825  -1.29879
 -0.563746   -0.795709    0.122105     -1.32026   -0.733463  -0.162106
  1.28651     0.774981    1.23493      -0.348768  -0.867166   0.0461009
  1.39228     0.916479   -0.507263      0.136745   0.448621  -1.43863
 -0.0635091  -0.53471    -0.496855  …   0.198625   0.999072   0.0379664
 -0.621088   -0.610635    0.661928      1.49169   -1.0531     1.25613
 -1.46809     0.0493928   0.170964     -0.815303   1.57221    1.19227
 -0.633521   -0.184444   -0.632862     -0.178261   1.03528    0.39186
  0.85734    -0.149834    0.979223     -1.54891    0.679991   1.15426

julia> b = randn(10)
10-element Vector{Float64}:
 -0.15504781733320247
 -0.46522503891836453
 -0.7314607060472206
  1.1463952817418253
 -0.8633357815600599
  0.6212004387401958
 -0.28857635262190445
  1.5666775704210631
  0.3914056983881698
 -2.094606401710734
```

The value of x1 generated by the formula is shown in the picture below:

```
100-element Vector{Float64}:
 -0.06366043724850312
 -0.002969343153621431
 -0.015146755362225262
  0.002769810135822208
 -0.0096036156972204
  0.007303531795551231
 -0.020250501807172673
 -0.0129195082965609
  0.033402561726179975
 -0.016223971264070543
  ⋮
 -0.025263287310766543
  0.0059662400984078715
 -0.07459239838183061
  0.007954706754422333
  0.09971337186667799
 -0.03084624300534448
  0.04916298672931276
  0.006300678093085072
  0.01945439035916537
```

The value of x2 generated by the pseudo inverse is shown in the picture below:

```
100-element Vector{Float64}:
 -0.06366043724850318
 -0.00296934315362151
 -0.015146755362225225
  0.002769810135822174
 -0.009603615697220482
  0.007303531795551216
 -0.020250501807172645
 -0.01291950829656089
  0.03340256172618
 -0.01622397126407054
  ⋮
 -0.025263287310766557
  0.005966240098407843
 -0.0745923983818306l
  0.007954706754422305
  0.09971337186667795
 -0.030846243005344498
  0.04916298672931275
  0.006300678093085078
  0.01945439035916541
```

The value of x3 generated by backslash operator is shown in the picture below:

```
100-element Vector{Float64}:
 -0.06366043724850316
 -0.002969343153621467
 -0.015146755362225274
  0.002769810135822177
 -0.009603615697220413
  0.007303531795551224
 -0.020250501807172683
 -0.012919508296560918
  0.03340256172618001
 -0.016223971264070568
  ⋮
 -0.02526328731076656
  0.005966240098407861
 -0.07459239838183065
  0.00795470675442233
  0.09971337186667797
 -0.030846243005344467
  0.04916298672931279
  0.006300678093085064
  0.01945439035916543
```

Then we calculate the norm between each x:

```
julia> norm(x1 - x2)
3.0194374160106687e-16

julia> norm(x1 - x3)
2.7069348209633995e-16
```

In conclusion, the value of each norm is very small, so all the solutions are close to each other.

A16.2

```
for i in 1:10
    C=rand(600,4000);
    d=rand(600);
    @time C\d
end
```

```
julia "/Users/xinhaodu/Desktop/julia/mis8.jl"
  0.176028 seconds (3.63 k allocations: 40.983 MiB, 4.25% gc time)
  0.209669 seconds (3.63 k allocations: 40.983 MiB, 4.69% gc time)
  0.226710 seconds (3.63 k allocations: 40.983 MiB, 16.42% gc time)
  0.199235 seconds (3.63 k allocations: 40.983 MiB, 14.51% gc time)
  0.219798 seconds (3.63 k allocations: 40.983 MiB, 0.47% gc time)
  0.168019 seconds (3.63 k allocations: 40.983 MiB, 0.65% gc time)
  0.169652 seconds (3.63 k allocations: 40.983 MiB, 0.58% gc time)
  0.165711 seconds (3.63 k allocations: 40.983 MiB, 0.64% gc time)
  0.162358 seconds (3.63 k allocations: 40.983 MiB, 0.62% gc time)
  0.179309 seconds (3.63 k allocations: 40.983 MiB, 0.59% gc time)
```

The complexity of solving the least squares problem with $m \times n$ matrix A is $2mn^2$ flops. So, the approximate flop rate is 1.13×10^{11} flop/sec.

A16.3

According to the optimality conditions for this constrained least squares problem, we have:

$$\begin{bmatrix} 2A^T A & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \hat{x} \\ \hat{z} \end{bmatrix} = \begin{bmatrix} 2A^T b \\ d \end{bmatrix}$$

Also, the two lines of Juila code is shown below:

```
xz = [2*A'A C';C zeros(p,p)]\[2*A'b;d];

x=xz[1:n];
```