In [220]:
```
# 12.1 A)
using LinearAlgebra
A = rand(20,10);
```

In [221]:
```
b = rand(20);
```

In [222]:
```
xhat = A\b;
```

In [223]:
```
xhat2 = (inv(transpose(A)*A))*transpose(A)*b;
```

In [224]:
```

xhat3 = pinv(A)*b;
```

In [225]:
```
xhat - xhat2
```

Out[225]:
```
10-element Vector{Float64}:
  9.610368056911511e-16
 -2.5673907444456745e-16
  1.5543122344752192e-15
 -8.187894806610529e-16
  5.342948306008566e-16
 -2.220446049250313e-16
  5.828670879282072e-16
 -2.7755575615628914e-16
 -2.7755575615628914e-16
 -9.506284648352903e-16
```

In [226]:
```
xhat - xhat3
```

Out[226]:
```
10-element Vector{Float64}:
 -5.377642775528102e-16
  5.204170427930421e-16
 -5.551115123125783e-16
 -3.3306690738754696e-16
 -6.938893903907228e-16
  1.1102230246251565e-16
 -3.0531133177191805e-16
 -2.220446049250313e-16
 -2.220446049250313e-16
 -6.938893903907228e-18
```

In [227]:
```
xhat2-xhat3
```

Out[227]:
```
10-element Vector{Float64}:
 -1.4988010832439613e-15
  7.771561172376096e-16
 -2.1094237467877974e-15
  4.85722573273506e-16
 -1.2281842209915794e-15
  3.3306690738754696e-16
 -8.881784197001252e-16
  5.551115123125783e-17
  5.551115123125783e-17
  9.43689570931383e-16
```

**All the values above are in the range of of power -16 which is very small, hence can be neglected and value can be used by any formula for future purpose**

In [203]:
```
## 12.1B)
v = rand(10)
```

Out[203]:
```
10-element Vector{Float64}:
 0.20216306342383317
 0.49550672911091653
 0.1719612982233596
 0.9951033247126613
 0.36391229807187897
 0.5570279980097335
 0.8447617326711114
 0.5939197499515272
 0.7329641093730224
 0.6689700052830212
```

In [204]:
```
norm(A*(xhat+v) - b)*norm(A*(xhat+v) - b)
```

Out[204]:
```
160.03035114013008
```

In [205]:
```
norm(A*(xhat)-b)*norm(A*(xhat)-b)
```

Out[205]:
```
1.1563641867827565
```

In [206]:
```
v = rand(10) * 8
```

Out[206]:
```
10-element Vector{Float64}:
 1.200469906334794
 3.9184334161162475
 6.497882244702502
 6.333355564122142
 6.860619772478357
 3.185955410756627
 2.9099069879047104
 0.4583619421617513
 3.03637126866869
 2.7397817668127793
```

In [207]:
```
norm(A*(xhat+v) - b)*norm(A*(xhat+v) - b)
```
Out[207]:
```
6980.04890255244
```

In [208]:
```
norm(A*(xhat)-b)*norm(A*(xhat)-b)
```
Out[208]:
```
1.1563641867827565
```

In [209]:
```
v = rand(10) * 0.03
```
Out[209]:
```
10-element Vector{Float64}:
 0.0076721539207409
 0.004529097108400362
 0.003926902071743043
 0.005447408089930594
 0.023392973640088233
 0.012944877230835412
 0.003864945908715056
 0.026403805062364748
 0.007973425668180332
 0.005609527478197762
```

In [210]:
```
norm(A*(xhat+v) - b)*norm(A*(xhat+v) - b)
```
Out[210]:
```
1.2053048369484969
```

In [211]:
```
norm(A*(xhat)-b)*norm(A*(xhat)-b)
```
Out[211]:
```
1.1563641867827565
```

In [212]:

```
for i in 1 : 20
    v = rand(10)

    s=norm(A*(xhat+v) - b)*norm(A*(xhat+v) - b)
    s1=norm(A*(xhat)-b)*norm(A*(xhat)-b)
    if(s>s1)
        print("satisfied equality")
    end
    print(s)
    print("\n")
    print(s1)
    print("\n")
end
```

```
satisfied equality166.5315076835373
1.1563641867827565
satisfied equality168.86261937748455
1.1563641867827565
satisfied equality105.67210368800939
1.1563641867827565
satisfied equality167.79972410840102
1.1563641867827565
satisfied equality123.70281425155707
1.1563641867827565
satisfied equality116.11883873824989
1.1563641867827565
satisfied equality118.20905250841957
1.1563641867827565
satisfied equality79.72069127154414
1.1563641867827565
satisfied equality101.61269786172218
1.1563641867827565
satisfied equality138.1354329360261
1.1563641867827565
satisfied equality124.18844896614064
1.1563641867827565
satisfied equality218.11742874322954
1.1563641867827565
satisfied equality124.96018737045219
1.1563641867827565
satisfied equality87.78471689381513
1.1563641867827565
satisfied equality158.70115710652263
1.1563641867827565
satisfied equality184.7200418112652
1.1563641867827565
satisfied equality101.44283175980846
1.1563641867827565
satisfied equality91.23657402664412
1.1563641867827565
satisfied equality66.52381830424659
1.1563641867827565
satisfied equality209.73539101825912
1.1563641867827565
```

In [213]:

```
for i in 1 : 20
    v = rand(10)*0.2

    s=norm(A*(xhat+v) - b)*norm(A*(xhat+v) - b)
    s1=norm(A*(xhat)-b)*norm(A*(xhat)-b)
    if(s>s1)
        print("satisfied equality")
    end
    print(s)
    print("\n")
    print(s1)
    print("\n")
end
```

```
satisfied equality4.402776020228088
1.1563641867827565
satisfied equality3.338688797487605
1.1563641867827565
satisfied equality7.477090585311672
1.1563641867827565
satisfied equality6.490380378074782
1.1563641867827565
satisfied equality6.317840380563545
1.1563641867827565
satisfied equality7.2406897955792555
1.1563641867827565
satisfied equality5.341374205125416
1.1563641867827565
satisfied equality8.17203385525359
1.1563641867827565
satisfied equality5.70327175718082
1.1563641867827565
satisfied equality4.326902146040469
1.1563641867827565
satisfied equality5.345681545012422
1.1563641867827565
satisfied equality6.022707110672516
1.1563641867827565
satisfied equality3.622796216801072
1.1563641867827565
satisfied equality3.7769758194380247
1.1563641867827565
satisfied equality6.519846671879815
1.1563641867827565
satisfied equality4.510645525239529
1.1563641867827565
satisfied equality5.987306160926755
1.1563641867827565
satisfied equality11.02245141637949
1.1563641867827565
satisfied equality6.702481159699659
1.1563641867827565
satisfied equality6.597950188632389
1.1563641867827565
```

**We have tried with different cases without scaling and with scaling by 8 and scaling by 0.03 and 0.2 for various iteration and clearly our comparison stll holds true as tha values of the LHS are clearly greater.**

In [217]:
```
#12.2)
A = rand(100000,100);
```

In [218]:
```
b = rand(100000);
```

In [219]:
```
time = rand(20)
for i in 1:20
    time[i] = @elapsed  A\b


end
print("On an average time is ",sum(time)/20)
```

```
On an average time is 1.0194320850000003
```

**Approximately, the time taken by my computer to compute is 1.05 seconds.**

## 13.3 b

In [178]:
```
using LinearAlgebra
```

In [179]:
```
function squaring(N)
    return N*N
end
```

Out[179]:
```
squaring (generic function with 1 method)
```

In [180]:
```
include("time_series_data.jl");
```

```
function toeplitz(vect,n) ==>
        forms the toeplitz matrix as described in section 7.4 in textbook
        n is the size of the vector to convolve with
        select the rows you need from this

data given is x_train and x_test
```

In [181]:

```
function todetermineA(x_vec,M)
    return toeplitz(x_vec,M)[M:N-1,1:M]
end
function determiningJloss(N,M,A,B,b)
    return (1/(N-M))*squaring(norm(A*B - b))
end
function determiningB(b,A)
    return A\b


end
```

Out[181]:

```
determiningB (generic function with 1 method)
```

In [215]:

```
J_train = rand(11);
J_test = rand(11)
element = 1;
for i in 2:12
    # finding B for x_train
    M = i
    N = 100   #SIZE OF XTRAIN
    A = todetermineA(x_train,M)
    b = x_train[M+1:N]
    B = determiningB(b,A)
    J_train[element] = determiningJloss(N,M,A,B,b)

    # for x_test with x_train's B
    N = 100     #size OF XTEST
    A = todetermineA(x_test,M)
    b = x_test[M+1:N]
    J_test[element] = determiningJloss(N,M,A,B,b)
    element = element+1
end
J_train
```

Out[215]:

```
11-element Vector{Float64}:
 0.030494624025126194
 0.030292696561256324
 0.024496551272111776
 0.024394903527141285
 0.02299631429887345
 0.019842912056501266
 0.019137191183089056
 0.019081546282888478
 0.018999488352156632
 0.019200812978895986
 0.01891584139384952
```

**GOOD CHOICE OF M FROM ABOVE RSULTS SEEMS TO BE M = 12 AND IT HAS MINIMUM LOSS OF 0.018915**

In [216]:
```
J_test
```

Out[216]:
```
11-element Vector{Float64}:
 0.03874425534739427
 0.03743353941193069
 0.03385594443279199
 0.031685969813553636
 0.03040027620370789
 0.03524397169474642
 0.03459784623483322
 0.037050154240589904
 0.03768621129443151
 0.03815483757921519
 0.04030160698971953
```

In [192]:
```
function simplePredictor1J(xdash,xvector)
    return squaring(norm(xdash- xvector))/size(xdash)[1]
end
```

Out[192]:
```
simplePredictor1J (generic function with 1 method)
```

In [193]:
```
# for simple predictor 1
M = 1
N = size(x_train)[1]
xdash = toeplitz(x_train,M)[M:N-1]
xvector = x_train[M+1:N]
simplePredictor1J(xdash ,xvector)
```

Out[193]:
```
2.6512574393840063
```

In [176]:
```
function calculateJLoss(A,B,b)
    # since the dimention of matrix is (N-M) = 100 -2
    return (1/(100-2))*squaring(norm(A*B - b))
end
```

Out[176]:
```
calculateJLoss (generic function with 1 method)
```

In [177]:
```
# for simple predictor second


M = 2
N = size(x_train)[1]
xdash = toeplitz(x_train,M)[M:N-1,1:2]
xvector = x_train[M+1:N]
B = [2,-1]
calculateJLoss(xdash,B,xvector)
```

Out[177]:
```
5.020605189805685
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: