MIS

Homework 6

Xinhao Du

A11.2

First, we generate a matrix A and a vector b

```
julia> A = rand(20, 20)
20×20 Matrix{Float64}:
 0.261548    0.698089    0.647787    …  0.653351    0.0831231  0.491748
 0.365888    0.316649    0.978235       0.545412    0.511669   0.518374
 0.00802605  0.260996    0.0328441      0.49603     0.535621   0.201683
 0.274643    0.0486485   0.568807       0.270531    0.817884   0.595622
 0.868202    0.425347    0.632446       0.687194    0.73596    0.0844074
 0.641374    0.761704    0.556988    …  0.659173    0.266997   0.400281
 0.148168    0.0794909   0.00220659     0.842602    0.909896   0.948889
 0.241687    0.224677    0.911384       0.873799    0.941458   0.288177
 0.342297    0.164677    0.995533       0.417194    0.0445296  0.893832
 0.0405445   0.2617      0.783452       0.460405    0.558807   0.582523
 0.372603    0.530059    0.740915    …  0.644554    0.338327   0.952157
 0.647496    0.0432258   0.932285       0.0514869   0.509878   0.360923
 0.527476    0.323885    0.706926       0.511289    0.782281   0.18779
 0.491581    0.415911    0.637182       0.041532    0.607833   0.00104827
 0.659117    0.188736    0.124711       0.483397    0.972259   0.188886
 0.617969    0.13385     0.164547    …  0.0512322   0.69268    0.552819
 0.510083    0.315096    0.920266       0.822909    0.172881   0.667654
 0.122479    0.74159     0.740683       0.00105106  0.0330927  0.697696
 0.7612      0.747853    0.878419       0.41789     0.203592   0.506032
 0.58161     0.00211941  0.615975       0.608904    0.305964   0.662288

julia> b = rand(20)
20-element Vector{Float64}:
 0.801018192431523
 0.3507464386041045
 0.16862414353303257
 0.9180055537569789
 0.6625136571776602
 0.5230369362696002
 0.7568971719949673
 0.0794646000116358
 0.3979906123966287
 0.8058773659607892
 0.23501487361046747
 0.8701777690676733
 0.309147304251591
 0.5255382074690955
 0.06688497734441767
 0.8056599948528107
 0.5032350669566651
 0.9791288223521716
 0.9976210159966751
 0.6701368735942557
```

a. Using the backslash operator:

```
julia> x = A \ b
20-element Vector{Float64}:
  0.91933493329059
  0.6012948825927186
 -0.12257585618280441
 -0.6936677940365202
  0.2973313223753984
  0.7799667639187193
 -0.25730830330097826
 -0.7213451720675471
  0.12379196343165935
 -0.6594544459959096
  1.3094401679804646
 -0.3097410688417569
  0.5120152507680551
 -0.7758837789429298
  0.5834728225735548
 -0.36867279859889873
  0.31221989036097614
 -0.48623208199986223
  0.0497233913806306
  0.1868628454022244

julia> residual_norm = norm(A*x - b)
1.3101837120924467e-15
```

x satisfies the equations because the norm of the residual is very small.

b. Computing the inverse of A explicitly:

```
julia> x = inv(A) * b
20-element Vector{Float64}:
  0.9193349332905889
  0.6012948825927189
 -0.12257585618280398
 -0.6936677940365207
  0.29733132237539817
  0.7799667639187186
 -0.25730830330097826
 -0.7213451720675468
  0.12379196343165907
 -0.6594544459959101
  1.309440167980466
 -0.30974106884175695
  0.5120152507680549
 -0.7758837789429293
  0.5834728225735545
 -0.3686727985988987
  0.3122198903609763
 -0.4862320819998621
  0.04972339138062985
  0.18686284540222514

julia> residual_norm = norm(A*x - b)
3.78436357389957e-15
```

Also, x satisfies the equations because the norm of the residual is very

small.

c. Using QR factorization

```
julia> Q, R = qr(A)
LinearAlgebra.QRCompactWY{Float64, Matrix{Float64}, Matrix{Float64}}
Q factor:
20×20 LinearAlgebra.QRCompactWYQ{Float64, Matrix{Float64}, Matrix{Float64}}:
 -0.12027     -0.427164     …  -0.0294269    0.311818   -0.132146
 -0.168249    -0.071433        -0.205069    -0.0414041  -0.304879
 -0.00369067  -0.204112        -0.147595    -0.114356   -0.189136
 -0.126291     0.0970918       -0.00373855  -0.368074   -0.301541
 -0.399231     0.0904216       -0.346953    -0.333767    0.206676
 -0.294927    -0.289966     …  -0.077958    -0.11412    -0.201483
 -0.0681331    0.00992955       0.112274     0.237484   -0.0295335
 -0.111137    -0.0595547        0.0820995   -0.0301776  -0.113286
 -0.157401     0.0380576       -0.190653     0.244004   -0.0021159
 -0.0186438   -0.188586         0.0130066   -0.168025    0.455271
 -0.171337    -0.238256     …   0.105904     0.0278801   0.211031
 -0.297742     0.285881        -0.201857     0.20006    -0.317442
 -0.242553     0.00274246       0.250012    -0.478672    0.140123
 -0.226047    -0.0883855       -0.375028     0.204839    0.280484
 -0.303086     0.17562          0.338314     0.111098   -0.107948
 -0.284165     0.199022     …  -0.286661     0.144488    0.274951
 -0.234555     0.00114416       0.400358     0.3218      0.277534
 -0.0563204   -0.530649         0.00604726  -0.122392    0.0340257
 -0.350028    -0.21964          0.271021     0.0795422  -0.216632
 -0.267445     0.286057         0.259702    -0.114138    0.0512122
R factor:
20×20 Matrix{Float64}:
 -2.17469  -1.3495   -2.53832   -1.80464   …  -1.98191    -1.70318
  0.0      -1.25429  -0.884958  -0.921784       0.0840587  -0.693655
  0.0       0.0       1.59177    0.298759       0.307627    0.87627
  0.0       0.0       0.0       -1.52838       -0.816232   -0.510477
  0.0       0.0       0.0        0.0           -1.01122    -0.153355
  0.0       0.0       0.0        0.0        …  -0.0417305  -0.404645
  0.0       0.0       0.0        0.0           0.436512     0.000522688
  0.0       0.0       0.0        0.0          -0.177007     0.329258
  0.0       0.0       0.0        0.0          -0.138736     0.399092
  0.0       0.0       0.0        0.0           0.190082     0.94939
  0.0       0.0       0.0        0.0        …  0.462463     0.345649
  0.0       0.0       0.0        0.0          -0.0160572   -0.131817
  0.0       0.0       0.0        0.0          -0.536525     0.0125834
  0.0       0.0       0.0        0.0           0.284207    -0.168066
  0.0       0.0       0.0        0.0          -0.1052      -0.195769
  0.0       0.0       0.0        0.0        …  -0.180986   -0.182437
  0.0       0.0       0.0        0.0          -0.0647796    0.00207901
  0.0       0.0       0.0        0.0          -0.0524656    0.314734
  0.0       0.0       0.0        0.0          -0.426575     0.356649
  0.0       0.0       0.0        0.0           0.0          0.0765746
```

```
julia> x = R \ (transpose(Q)*b)
20-element Vector{Float64}:
   0.9193349332905879
   0.6012948825927206
  -0.12257585618280346
  -0.6936677940365205
   0.297331322375396
   0.779966763918719
  -0.2573083033009791
  -0.7213451720675476
   0.12379196343165971
  -0.6594544459959103
   1.3094401679804653
  -0.3097410688417556
   0.512015250768057
  -0.7758837789429305
   0.5834728225735547
  -0.3686727985988996
   0.3122198903609761
  -0.4862320819998611
   0.049723391380632156
   0.1868628454022231

julia> residual_norm = norm(A*x - b)
2.2707046587157715e-15

julia> println("Q is nearly orthogonal: ", isapprox(Q'*Q, Matrix{Float64}(I, 20, 20), atol=1e-8))
Q is nearly orthogonal: true

julia> println("R is upper triangular: ", isapprox(triu(R), R, atol=1e-8))
R is upper triangular: true

julia> println("A is very near QR: ", isapprox(A, Q*R, atol=1e-8))
A is very near QR: true
```

x satisfies the equations because the norm of the residual is very small, and the matrix Q is nearly orthogonal, R is an upper triangular matrix, and A is very near QR.

A11.3

a.

After we run the code

```
A = ones(2000,2000) + rand(2000, 2000)
b = ones(2000)
@time A\b;
We can have the time is 0.033287 seconds
```

```
julia> @time A\b;
  0.033287 seconds (4 allocations: 30.548 MiB)
```

b.

We run the code

```
L = ones(2000,2000) + rand(2000, 2000)
for i = 1:2000
for j = i+1:2000
L[i, j] = 0
end
end
b = ones(2000)
```

We can know that L is lower triangular matrix with ones on the diagonal.

```
julia> L
2000×2000 Matrix{Float64}:
 1.51708  0.0      0.0      0.0      0.0      0.0      …  0.0      0.0      0.0      0.0      0.0      0.0
 1.41783  1.03043  0.0      0.0      0.0      0.0         0.0      0.0      0.0      0.0      0.0      0.0
 1.40746  1.07926  1.36941  0.0      0.0      0.0         0.0      0.0      0.0      0.0      0.0      0.0
 1.51781  1.59786  1.0366   1.05876  0.0      0.0         0.0      0.0      0.0      0.0      0.0      0.0
 1.47246  1.36865  1.98783  1.79939  1.85614  0.0         0.0      0.0      0.0      0.0      0.0      0.0
 1.39791  1.54233  1.43284  1.74476  1.58196  1.4219   …  0.0      0.0      0.0      0.0      0.0      0.0
 1.77743  1.99968  1.10322  1.88773  1.91729  1.22051     0.0      0.0      0.0      0.0      0.0      0.0
 1.98243  1.37774  1.12647  1.17614  1.65379  1.68244     0.0      0.0      0.0      0.0      0.0      0.0
 1.71173  1.18578  1.50613  1.3293   1.4462   1.10445     0.0      0.0      0.0      0.0      0.0      0.0
 1.49116  1.32414  1.68339  1.46065  1.0349   1.68868     0.0      0.0      0.0      0.0      0.0      0.0
 1.24023  1.51866  1.5417   1.50288  1.39268  1.9523   …  0.0      0.0      0.0      0.0      0.0      0.0
 1.8009   1.4873   1.99571  1.22376  1.98363  1.46931     0.0      0.0      0.0      0.0      0.0      0.0
 1.63317  1.08431  1.80414  1.52911  1.46445  1.95195     0.0      0.0      0.0      0.0      0.0      0.0
 1.05206  1.00632  1.12782  1.6935   1.18122  1.00858     0.0      0.0      0.0      0.0      0.0      0.0
 1.50996  1.34204  1.54386  1.52147  1.87989  1.57757     0.0      0.0      0.0      0.0      0.0      0.0
 1.75294  1.1027   1.18416  1.05495  1.26616  1.15718  …  0.0      0.0      0.0      0.0      0.0      0.0
 1.80361  1.68565  1.65161  1.37913  1.52404  1.14973     0.0      0.0      0.0      0.0      0.0      0.0
 1.2901   1.99143  1.10435  1.76397  1.6829   1.02054     0.0      0.0      0.0      0.0      0.0      0.0
 1.23296  1.65267  1.63802  1.21099  1.38135  1.40415     0.0      0.0      0.0      0.0      0.0      0.0
 1.12063  1.44917  1.76293  1.71236  1.12339  1.59327     0.0      0.0      0.0      0.0      0.0      0.0
 1.23124  1.0903   1.02705  1.02318  1.04265  1.58797  …  0.0      0.0      0.0      0.0      0.0      0.0
 1.46237  1.78581  1.1281   1.94127  1.97636  1.52471     0.0      0.0      0.0      0.0      0.0      0.0
 1.94396  1.15537  1.70055  1.85718  1.78055  1.62427     0.0      0.0      0.0      0.0      0.0      0.0
 1.41436  1.29501  1.11523  1.14261  1.17757  1.41261     0.0      0.0      0.0      0.0      0.0      0.0
 1.72216  1.40211  1.73715  1.73114  1.12976  1.17199     0.0      0.0      0.0      0.0      0.0      0.0
 1.83625  1.74229  1.71965  1.85385  1.29885  1.10061  …  0.0      0.0      0.0      0.0      0.0      0.0
 ⋮                                              ⋮  ⋱           ⋮
 1.02465  1.66973  1.74429  1.73386  1.65048  1.01295     0.0      0.0      0.0      0.0      0.0      0.0
 1.68304  1.90725  1.8767   1.05663  1.91791  1.00612  …  0.0      0.0      0.0      0.0      0.0      0.0
 1.5872   1.33443  1.51302  1.35201  1.71452  1.81672     0.0      0.0      0.0      0.0      0.0      0.0
 1.50737  1.76786  1.2149   1.44081  1.18595  1.11722     0.0      0.0      0.0      0.0      0.0      0.0
 1.33301  1.06178  1.81094  1.94667  1.05225  1.50214     0.0      0.0      0.0      0.0      0.0      0.0
 1.20574  1.95146  1.65595  1.63283  1.28614  1.4673      0.0      0.0      0.0      0.0      0.0      0.0
 1.13654  1.70588  1.06281  1.2251   1.08422  1.67875  …  0.0      0.0      0.0      0.0      0.0      0.0
 1.9825   1.7369   1.15063  1.3263   1.50507  1.90241     0.0      0.0      0.0      0.0      0.0      0.0
 1.63627  1.46421  1.53384  1.30911  1.13625  1.29546     0.0      0.0      0.0      0.0      0.0      0.0
 1.87183  1.80036  1.87126  1.45733  1.42326  1.35634     0.0      0.0      0.0      0.0      0.0      0.0
 1.13453  1.60884  1.68711  1.16379  1.12407  1.77149     0.0      0.0      0.0      0.0      0.0      0.0
 1.73865  1.16831  1.93736  1.55115  1.15165  1.83501  …  0.0      0.0      0.0      0.0      0.0      0.0
 1.54655  1.19493  1.54011  1.1774   1.88025  1.01086     0.0      0.0      0.0      0.0      0.0      0.0
 1.62915  1.46658  1.40291  1.43527  1.41765  1.03851     0.0      0.0      0.0      0.0      0.0      0.0
 1.57504  1.25584  1.41849  1.96984  1.9679   1.67475     0.0      0.0      0.0      0.0      0.0      0.0
 1.31027  1.24763  1.67209  1.52291  1.36728  1.21807     0.0      0.0      0.0      0.0      0.0      0.0
 1.18748  1.10012  1.3202   1.9733   1.0115   1.61677  …  0.0      0.0      0.0      0.0      0.0      0.0
 1.93199  1.5742   1.37163  1.21577  1.37011  1.24464     0.0      0.0      0.0      0.0      0.0      0.0
 1.30749  1.18027  1.24059  1.53878  1.59356  1.74774     0.0      0.0      0.0      0.0      0.0      0.0
 1.83951  1.73258  1.13905  1.27121  1.78735  1.5411      0.0      0.0      0.0      0.0      0.0      0.0
 1.95023  1.81854  1.46378  1.44882  1.02035  1.32645     1.00154  0.0      0.0      0.0      0.0      0.0
 1.76317  1.59483  1.16     1.8824   1.77941  1.76164  …  1.13476  1.58074  0.0      0.0      0.0      0.0
 1.87922  1.19696  1.92582  1.04371  1.37373  1.86183     1.32848  1.14051  1.23464  0.0      0.0      0.0
 1.72159  1.11619  1.4975   1.35587  1.5263   1.1236      1.7101   1.72341  1.45156  1.37039  0.0      0.0
 1.30806  1.74441  1.81128  1.85243  1.9318   1.89949     1.92728  1.53434  1.96473  1.3848   1.26319  0.0
 1.48657  1.37792  1.39541  1.77116  1.52497  1.46728     1.592    1.73527  1.24669  1.25201  1.77106  1.03626
```

Then we run the code:

```
@time L \ b;
```

and we will get the time

```
julia> @time L \ b;
  0.002185 seconds (1 allocation: 15.750 KiB)
```

c.

The times differ by so much between the two systems because the matrix L is lower triangular with ones on the diagonal, whereas the matrix A is a random matrix. Julia can solve a lower triangular linear system in O(n^2) time by using a specialized algorithm, whereas a general linear system requires O(n^3) time.

A11.10

By the description, we can substitute the NPV values into the NPV equation:

When r = 0, c1 + c2 + c3 = 1

When r = 0.05, c1 + c2/1.05 + c3/1.1025 = 0

When r = 0.10, c1 + c2/1.10 + c3/1.21 = -1

Then we rewrite the equation in matrix form:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1/1.05 & 1/1.1025 \\ 1 & 1/1.10 & 1/1.21 \end{bmatrix} \begin{bmatrix} c1 \\ c2 \\ c3 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

We use Julia to solve c1,c2 and c3.

```
julia> A = [1 1 1; 1 1/1.05 1/1.1025; 1 1/1.10 1/1.21]
3×3 Matrix{Float64}:
 1.0  1.0       1.0
 1.0  0.952381  0.907029
 1.0  0.909091  0.826446

julia> b = [1, 0, -1]
3-element Vector{Int64}:
  1
  0
 -1

julia> c = A \ b
3-element Vector{Float64}:
 -42.00000000000013
  66.10000000000029
 -23.10000000000017
```

Therefore, the cash flow c = (c1, c2, c3) is (-42, 66, -23).

A12.1

a.

After we run the commands, we can see the solutions are very close to each other.

```
julia> using LinearAlgebra

julia> A = rand(20,10);

julia> b = rand(20);

julia> x = A\b;

julia> x
10-element Vector{Float64}:
  0.17350439399252207
  0.6101276411775108
  0.40902473607687434
 -0.003950171107084964
 -0.32599688831224094
  0.06600417267207158
  0.02977528553457271
 -0.08299497430933958
  0.26786849937063023
 -0.4044008165308386

julia> x=(A'*A)\(A'*b)
10-element Vector{Float64}:
  0.17350439399252
  0.6101276411775132
  0.40902473607687245
 -0.003950171107084794
 -0.3259968883122428
  0.06600417267207374
  0.02977528553457447
 -0.08299497430934111
  0.2678684993706314
 -0.40440081653083876

julia> x=pinv(A)*b
10-element Vector{Float64}:
  0.17350439399252146
  0.6101276411775114
  0.4090247360768741
 -0.003950171107085454
 -0.3259968883122402
  0.06600417267207091
  0.029775285534572545
 -0.08299497430933979
  0.26786849937063006
 -0.40440081653083804
```

b.

We verify the equation is holding.

```
julia> using LinearAlgebra

julia> function verify(A, b, x, delta)
           return norm(A*(x+delta)-b).^2 > norm(A*x-b).^2
       end
verify (generic function with 1 method)

julia> delta1 = rand(10);

julia> println(verify(A, b, x, delta1));
true

julia> delta2 = rand(10);

julia> println(verify(A, b, x, delta2));
true

julia> delta3 = rand(10);

julia> println(verify(A, b, x, delta3));
true

julia> delta4 = rand(10);

julia> scaled_delta4 = (delta4.-minimum(delta4))./(maximum(delta4)-minimum(delta4));

julia> println(verify(A, b, x, scaled_delta4));
true
```

A12.2

```
julia> A = rand(100000,100);

julia> b = rand(100000);

julia> @time A\b
  0.178194 seconds (627 allocations: 77.346 MiB, 21.03% gc time)
```

```
julia> A = rand(100000,100);

julia> b = rand(100000);

julia> @time A\b
  0.141177 seconds (627 allocations: 77.346 MiB, 2.05% gc time)
```

```
julia> A = rand(100000,100);

julia> b = rand(100000);

julia> @time x_cap = A\b
  0.153912 seconds (627 allocations: 77.346 MiB, 1.68% gc time)
```

```
julia> A = rand(100000,100);

julia> b = rand(100000);

julia> @time A\b
  0.135977 seconds (627 allocations: 77.346 MiB, 0.24% gc time)
```

```
julia> A = rand(100000,100);

julia> b = rand(100000);

julia> @time A\b
  0.135738 seconds (627 allocations: 77.346 MiB, 0.22% gc time)
```

We can see that on 0.14 seconds to solve a least squares problem.

A12.4

For m-vector c, which means the total trip time for each link, so

$$c = f_{i-}s_i$$

For matrix R

$$R_{ij} = \begin{cases} 1 & passenger\ i\ went\ through\ link\ j \\ 0 & otherwise \end{cases}$$

i is row from 1 to m, j is column from 1 to n

A12.6

From hint we know that y has 6 entries, so y can be defined as

$$y = [y_{12}, y_{13}, y_{14}, y_{23}, y_{24}, y_{34}]$$

Then y can be expressed as

$$y_{i,j} = A(x_i + x_j)$$

So we can have a 6*4 matrix A and the equation:

$$\begin{bmatrix}1&1&0&0\\1&0&1&0\\1&0&0&1\\0&1&1&0\\0&1&0&1\\0&0&1&1\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}y_{12}\\y_{13}\\y_{14}\\y_{23}\\y_{24}\\y_{34}\end{bmatrix}$$

Therefore, the dimension of the matrix A is 6×4 and the vector b is 6×1, the entries of A are showed above and b is equal to y