CSE 8803 Homework 2

Jingyi Feng

October 24, 2023

1 Problem

Consider the least squares (LS) problem

$$\min_x \|Ax - b\|_2, \tag{1}$$

where $A \in \mathbf{R}^{m \times n}$ with $m >> n$, $rank(A) = n$, and $b \in \mathbf{R}^{m \times 1}$. We denote the full QR decomposition and the reduced QR decomposition of A as $A = Q \begin{pmatrix} R \\ 0 \end{pmatrix}$ and $A = Q_1 R$, respectively, where Q_1 is the first n columns of Q, $Q \in \mathbf{R}^{m \times m}$ with $Q^T Q = I_m$, and $R \in \mathbf{R}^{n \times n}$.

Also consider another LS problem

$$\min_x \|\tilde{A}x - \tilde{b}\|_2, \tag{2}$$

where $A = \begin{pmatrix} \tilde{A} \\ a^T \end{pmatrix}$ and $b = \begin{pmatrix} \tilde{b} \\ \beta \end{pmatrix}$, where $a \neq 0$, and A and b are the same as above. We denote the full QR decomposition and the reduced QR decomposition of $\tilde{A}$ as $\tilde{A} = \tilde{Q} \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix}$ and $\tilde{A} = \tilde{Q}_1 \tilde{R}$, respectively, where $\tilde{Q}_1$ is the first n columns of $\tilde{Q}$, $\tilde{Q} \in \mathbf{R}^{(m-1) \times (m-1)}$ with $\tilde{Q}^T \tilde{Q} = I_{m-1}$, and $\tilde{R} \in \mathbf{R}^{n \times n}$.

2 Answer

A is randomly generated to test the consistency of problem 1, 2, 3.

```
Random Matrix A:
 [[0.7362439069757161 0.0753600229910375 0.2369888878530965]
 [0.3294903673953531 0.7348264128850511 0.8121026488777162]
 [0.3721898730792679 0.2054806944205146 0.4894158721313325]
 [0.3280990612843724 0.4900078702528352 0.6402352908014283]
 [0.6775767571637153 0.5859311001711132 0.5843620931494613]]

Random Matrix A_:
 [[0.7362439069757161 0.0753600229910375 0.2369888878530965]
 [0.3294903673953531 0.7348264128850511 0.8121026488777162]
 [0.3721898730792679 0.2054806944205146 0.4894158721313325]
 [0.3280990612843724 0.4900078702528352 0.6402352908014283]]

Q Matrix of A_:
 [[-0.6322768991420242  0.5910432098646177  0.0169826000904935]
 [-0.28296213499363   -0.6977485996172593 -0.036438750971047 ]
 [-0.319631927154794   0.0690530542931523 -0.7640923237323034]
 [-0.2817672990088791 -0.3630486211680571 -0.3053537017865993]
 [-0.5818942973803286 -0.165052083191466   0.5668384658263994]]

R Matrix of A_:
 [[-1.1644327160691321 -0.8002728025880315 -1.0565041751209625]
 [ 0.                 -0.7285998314669077 -0.7216638725917706]
 [ 0.                  0.                 -0.2637855333780763]]
```

2.1 problem 1

Write a program *FullQRDown* that computes $\tilde{Q}$ and $\tilde{R}$ for the full QR decomposition of $\tilde{A}$ from downdating the full QR decomposition of A.

Please see function *full_qr_del* in attached code file. Here is the result of the randomly generated example A.

```
-------------------Full QR Downdating-------------------
Q Matrix of A_:
 [[-0.7774554977327814  0.509839442498171   0.3682459931784722  0.0046454737356173]
 [-0.3479337420037359 -0.7545406899300167  0.3042186107790545  0.4658985890415148]
 [-0.3930233721248128  0.0231199106244039 -0.865676720579557   0.3091955920497317]
 [-0.3464645568943413 -0.4125500695016402 -0.1498323757079435 -0.8290446370129375]]

R Matrix of A_:
 [[-0.9469917045062433 -0.5647890428115163 -0.8809769400502362]
 [-0.                 -0.713437002087801  -0.7447520730125929]
 [-0.                  0.                 -0.1852769537877857]
 [-0.                 -0.                  0.                ]]

Q @ R =
 [[0.7362439069757164 0.0753600229910377 0.2369888878530965]
 [0.3294903673953533 0.7348264128850511 0.8121026488777163]
 [0.372189873079268  0.2054806944205148 0.4894158721313326]
 [0.3280990612843726 0.4900078702528353 0.6402352908014285]]
```

2.2 problem 2

. Write a program *ReducedQRDown* that computes $\tilde{R}$ of the reduced QR decomposition of $\tilde{A}$ from downdating the reduced QR decomposition of A. Here we are assuming that only the first n columns Q_1 of the orthogonal factor Q of A are known, along with R.

Please see function *reduced_qr_del* in attached code file. Here is the result of the randomly generated example A.

```
-------------------Reduced QR Downdating-------------------
Q Matrix of A_:
 [[ 0.7774554977327812 -0.509839442498171  -0.3682459931784722]
 [ 0.3479337420037357  0.7545406899300167 -0.3042186107790545]
 [ 0.3930233721248129 -0.0231199106244038  0.8656767205795568]
 [ 0.3464645568943413  0.4125500695016403  0.1498323757079433]]

R Matrix of A_:
 [[ 0.9469917045062431  0.5647890428115161  0.880976940050236 ]
 [ 0.                  0.713437002087801   0.7447520730125929]
 [ 0.                 -0.                  0.1852769537877857]]

Q @ R =
 [[0.7362439069757162 0.0753600229910374 0.2369888878530961]
 [0.3294903673953531 0.734826412885051  0.8121026488777161]
 [0.372189873079268  0.2054806944205148 0.4894158721313326]
 [0.3280990612843726 0.4900078702528353 0.6402352908014284]]
```

2.3 problem 3

Write a program *RDown* that computes $\tilde{R}$ for $\tilde{A}$, assuming that the orthogonal factor Q for A is not known, but R is known.

Please see function *r_qr_del* in attached code file. Here is the result of the randomly generated example A.

```
-------------------Cholesky Downdating-------------------
R Matrix of A_:
 [[ 0.9469917045062435  0.5647890428115164  0.8809769400502361]
 [ 0.                  0.7134370020878011  0.744752073012593 ]
 [ 0.                 -0.                  0.1852769537877857]
 [-0.                  0.                  0.                ]]

A_.T @ A_ =
 [[0.8967932884036395 0.5348505383385271 0.8342778540888673]
 [0.5348505383385271 0.8279790188279778 1.0288998089787706]
 [0.8342778540888673 1.0288998089787706 1.365103568761713 ]]

R.T @ R =
 [[0.8967932884036404 0.5348505383385277 0.8342778540888678]
 [0.5348505383385277 0.8279790188279781 1.028899808978771 ]
 [0.8342778540888678 1.028899808978771  1.3651035687617135]]
```

2.4 problem 4

The purpose of this problem is to explore that the above three methods can produce numerically different solutions. From understanding the properties of the methods, try to generate an artificial example for which the differences in the solution quality is as large as possible (trying random matrices here will not work).

Present an input data (i.e. A and b), 20×6 matrix A and a 20×1 vector b, so that the differences among the solutions produced by the following three downdating methods are as dramatic as possible.

First compute the full QR decomposition of A using any built in function. Then

- Applying FullQRDown, compute the downdated solution for Eqn 2.
- Applying ReducedQRDown, compute the downdated solution for Eqn 2.
- Applying RDown, compute the downdated solution for Eqn 2.

You will need to create the input A and b artificially, so that the certain properties of the problem or an algorithm work better over the others. Discuss how you generated the data, why you expected a better behavior of an algorithm over another, and what you observed.

Please see function *Full_QR_Down, Reduced_QR_Down, R_Down* in attached code file.

Variance between Algorithm 1 and 2: We can see *Full_QR_Down, Reduced_QR_Down, R_Down* are differentiated from each other in this case. I generated this near rank data-set by adding really tiny difference from the first row to the rest of the rows of the matrix. We can see in this case even algorithm 1 and algorithm 2 can give different results.

```
==============================Problem 4==============================
Artificial Matrix A:
 [[0.7703768702300525 0.8484670467645299 0.0162296951694821 0.3676896289764187 0.6222194396014694 0.2085926612070041]
 [0.7703768712300525 0.8484670477645299 0.0162296961694821 0.3676896299764187 0.6222194406014694 0.2085926622070041]
 [0.7703768722300525 0.8484670487645298 0.0162296971694821 0.3676896309764187 0.6222194416014694 0.2085926632070041]
 [0.7703768732300524 0.8484670497645298 0.0162296981694821 0.3676896319764187 0.6222194426014693 0.2085926642070041]
 [0.7703768742300524 0.8484670507645298 0.0162296991694821 0.3676896329764188 0.6222194436014693 0.2085926652070041]
 [0.7703768752300524 0.8484670517645297 0.0162297001694821 0.3676896339764188 0.6222194446014693 0.2085926662070041]
 [0.7703768762300524 0.8484670527645297 0.0162297011694821 0.3676896349764188 0.6222194456014692 0.2085926672070041]
 [0.7703768772300523 0.8484670537645297 0.0162297021694821 0.3676896359764188 0.6222194466014692 0.2085926682070041]
 [0.7703768782300523 0.8484670547645297 0.0162297031694821 0.3676896369764189 0.6222194476014692 0.2085926692070041]
 [0.7703768792300523 0.8484670557645296 0.0162297041694821 0.3676896379764189 0.6222194486014692 0.2085926702070041]
 [0.7703768802300522 0.8484670567645296 0.0162297051694821 0.3676896389764189 0.6222194496014691 0.2085926712070041]
 [0.7703768812300522 0.8484670577645296 0.0162297061694821 0.367689639976419  0.6222194506014691 0.2085926722070041]
 [0.7703768822300522 0.8484670587645295 0.0162297071694821 0.367689640976419  0.6222194516014691 0.2085926732070041]
 [0.7703768832300522 0.8484670597645295 0.0162297081694821 0.367689641976419  0.622219452601469  0.2085926742070041]
 [0.7703768842300521 0.8484670607645295 0.0162297091694821 0.367689642976419  0.622219453601469  0.2085926752070041]
 [0.7703768852300521 0.8484670617645295 0.0162297101694821 0.3676896439764191 0.622219454601469  0.2085926762070041]
 [0.7703768862300521 0.8484670627645294 0.0162297111694821 0.3676896449764191 0.622219455601469  0.2085926772070041]
 [0.770376887230052  0.8484670637645294 0.0162297121694821 0.3676896459764191 0.6222194566014689 0.2085926782070041]
 [0.770376888230052  0.8484670647645294 0.0162297131694821 0.3676896469764191 0.6222194576014689 0.2085926792070041]
 [0.770376889230052  0.8484670657645293 0.0162297141694821 0.3676896479764192 0.6222194586014689 0.2085926802070041]]

Artificial Vector b:
 [0.4291453127425701 0.3455325423107544 0.3280314538377431 0.1912911724397027 0.121864084804158  0.4375962030293389 0.2163364211089526 0.0046418638556935 0.0781043774220403 0.2049057415003991
 0.8056873861721554 0.1413418756392562 0.2384887504198518 0.6140861119070448 0.1192697211555039 0.4546314492752777 0.4930456098234508 0.5442752656304267 0.9037825992467226 0.0841581126138828]

-------------------Solution Comparison-------------------
Numpy package:
 [ 1.6097443782960053e+14 -5.7723187858693734e+13  3.6902097723430453e+13 -2.7342504707436848e+13 -1.0165832694682073e+14 -1.1152492224688340e+13]

Full QR Downdating:
 [ 6.1899749549225125e+14  3.6365283330505231e+14  2.6615188265117625e+15 -2.3019769778425460e+15  7.1992446260521300e+14 -2.0621165208546390e+15]

Reduced QR Downdating:
 [ 7.0289883649315138e+14  2.3747103558220097e+14  2.4093365959641190e+15 -2.2878509088348365e+15  6.7845740659495100e+14 -1.7403128466420045e+15]

Cholesky R Downdating:
 [-5.2182507218360294e+14 -3.7160973526881325e+14 -3.5772348569506156e+14  4.8642869924029850e+15 -7.5374566690783262e+14 -2.8593831915183825e+15]
```

Why: Since we have one step of reorthogonalization of added column $[1, 0, 0, ...]^T$ and it uses Gram-Schmidt method. We have learned from numerical linear algebra class that Gram-Schmidt method is not a stable algorithm. Therefore, the generated column may not be numerically orthogonal to the original Q_1, which could lead to the instability.

Algorithm 3 weakly stable: We can see *RDown* is the least stable algorithm among those three algorithms in this case. The data set I artificially generated has the last row dominating the over-determinated system, which means that the weights *i.e.* norm, of the last row is much larger than the others.

```
==============================Problem 4==============================
Artificial Matrix A:
 [[      -1.2256090048201813        2.6796750236336346       -0.9918138486001298       -0.5972438422373372       -0.8754556270484132        0.8898190986197155]
 [       -1.1083329915292777        1.5052850888877567       -0.9380419962365394       -0.347047792154025        -0.9260318730651296        1.8163057380697767]
 [       -0.6796515467535           0.8517391054988546       -0.6948982793930084       -0.0381626676286464       -0.8792643720234077        1.4942546991048868]
 [       -1.901348203573293         2.963276983016401        -1.2820278809747823       -1.219549532937076        -1.0285773075273097        2.3384972650043805]
 [       -0.4198835537604921       -0.2705497554963166        0.023410976892244        -1.043387012254556        -0.2160759181249818        1.9721663261322762]
 [       -0.0377911225848219        1.180925912210357        -0.2906690932317102        0.9557356229200776        0.3927333777429088       -2.2804396942173732]
 [        2.0674851083850876       -3.0126579382237435        1.4219654874570362        1.5443261827490078        1.421350085053656        -3.2876832801983467]
 [        2.9895636323678274       -5.484426337147801         2.064604629744469         2.1279929594915257        2.270283682048207        -3.6510153521337854]
 [       -1.079170591352139         1.7857846502381816       -0.7149613135710913       -0.7351196638769724       -0.9069351804638115        1.5927880242715873]
 [       -2.9436036596603428        6.418956141640924        -2.2797513074205056       -1.1346309079740955       -2.082428242559175         1.6833613898848416]
 [        3.4336258479657387       -5.851904743260103         2.5044550977116455        2.1920712462727767        2.580474480552425        -4.562924542956752 ]
 [       -1.0071588969185619        1.7974802972947181       -0.887204402136553        -0.1835857158703746       -0.988006750834351         1.242332726830151 ]
 [       -0.2422918680455627        0.6129173063258906        0.0631882851544327       -0.5561808414404277       -0.1959740635447722        0.272662775357164 ]
 [        2.361295820817681        -4.581843745825213         2.128339327160523         0.1001266603533049        1.79321891687645         -1.591001779675183 ]
 [       -0.8998724103972974        1.702712107077482        -0.9220292891171774       -0.0102638658881524       -0.7649756054962958        0.8742891543375293]
 [       -0.9682187530507609        1.9904726189073705       -0.9298742488355419       -0.0400391011183391       -0.5782824469381028        0.4638066396646184]
 [        2.156139965544845        -4.284608997225855         1.9534655983544762       -0.1004006582361205        1.6690271697349843       -1.191286710385253 ]
 [       -2.8976907433324746        6.33314244544666         -2.4410225269170054       -0.5733225460416698       -1.9217045435690825        1.1882562607503202]
 [        2.03407309370649         -5.265709388399285         1.6851026554889705        0.3161255970104803        1.0099172950660602        0.5943580247426715]
 [ 10334957.048815183        -24993934.151600607         11216170.36242011         -3824701.9277026053         7707234.867456123          897135.2135883662      ]]

Artificial Vector b:
 [0.7247179013441419 0.5816610182912001 0.6061450417539109 0.2184949775512397 0.9458675580250088 0.4005368110346954 0.3000917118376405 0.953531168912676  0.8262992554244011 0.078144296230206
 0.4228060112809364 0.2851236614909038 0.5667764034404202 0.5112617791397809 0.5450368941482716 0.1692258852322578 0.2458278053701127 0.9489963603658356 0.2762393783721984 0.3048124542934613]

-------------------Solution Comparison-------------------
Numpy package:
 [17.007005658821186 20.742066648629546 22.348866800275097 19.276641888905758 19.985718924730953 18.90581264773564 ]

Full QR Downdating:
 [17.007005658821097 20.742066648629596 22.348866800275324 19.276641888905854 19.985718924731003 18.9058126477357  ]

Reduced QR Downdating:
 [17.007005658821097 20.7420666486296   22.34886680027534  19.27664188890586  19.98571892473101  18.905812647735708]

Cholesky R Downdating:
 [22.435808379829904 27.294775788835064 29.469115871144968 25.037628692985365 26.272490460756394 24.810727315574415]
```

Why: Because we are subtracting entry a_k from r_{kk}, we know if $r_{kk} > a_k$, r_{kk} should be close a_k since a_k is dominating. After we eliminate most of the information from an entry, we should get the entry close to 0. Hence, it is easy to cause instability.