Numerical Linear Algebra : Homework 2

Aakriti Gupta
Mtech/SERC
09237

I.1: Prove that Eq. 3 is also equivalent to solving for {x} in Eq. 1.

Given [A] is a 'm' x 'n' matrix having full rank.
Equation 3: $\{x\} = A^T [AA^T]^{-1}\{b\}$
Equation 1: $[A]\{x\} = \{b\}$

Multiply both sides of equation 1 by $[AA^T]^{-1}$
(This is **possible only when m<n** otherwise this inverse is not defined)

$$[AA^T]^{-1} [A]\{x\} = [AA^T]^{-1}\{b\}$$

Multiply both sides by A^T

$$\underline{A^T [AA^T]^{-1} [A]}\{x\} = A^T [AA^T]^{-1}\{b\}$$

Now, R.H.S. is equal to what we want i.e. R.H.S. of Eq. 3.
Proving $\underline{A^T [AA^T]^{-1} [A]}$ is equal to Identity matrix will suffice.

Let $Q = A^T [AA^T]^{-1} [A]$

We see that AQ = A.
Which can be written as A(Q-I) = 0

Given A is of full rank, for A(Q-I) to be zero, Q-I = 0. This implies Q = I.

I.2: Assuming you are doing Cholesky decomposition for the matrix inversion, write down the exact number of operations (include only multiplications/divisions) for getting [x] using Eqs. 2 and 3. Which will offer computationally efficient scheme for conditions m > n, m = n, and m < n?

Order of computation for Cholesky decomposition for an order 'r' matrix is $r^3/6$. Computation of the inverse further takes $r(r^2+r)$ because we do two backward substitutions r times. Total computation for matrix inversion using Cholesky decomposition is $7r^3/6 + r^2$.

Equation 2:
Computation of A^TA: n*m*n
For inverse of n x n matrix: $7n^3/6 + n^2$
For the rest matrix multiplications: n*n*m + n*m

Total = $7n^3/6 + n^2 + 2n^2m + nm$ (Linear in 'm' and cubic in 'n')

Equation 3:
Computation of AA^T: m*n*m
For inverse of m x m matrix: $7m^3/6 + m^2$
For the rest matrix multiplications: n*m*m + n*m

Total = $7m^3/6 + m^2 + 2m^2n + nm$ (Linear in 'n' and cubic in 'm')

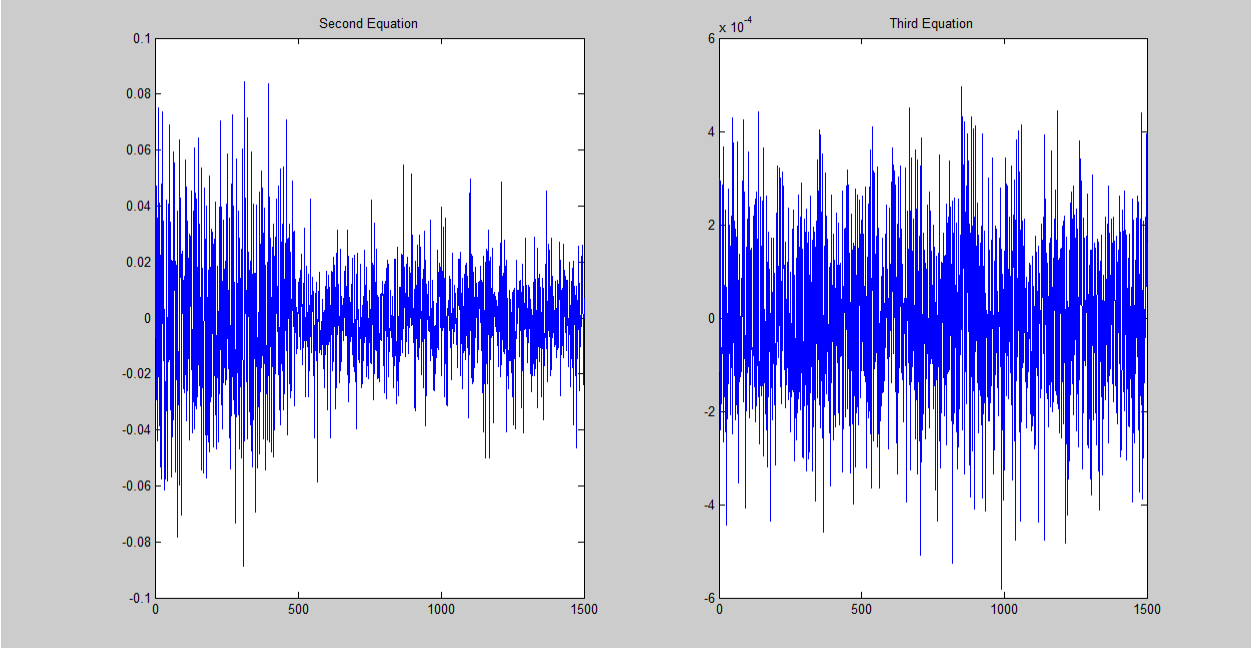
m > n
Equation 2 is computationally efficient.

m < n
Equation 3 is computationally efficient.

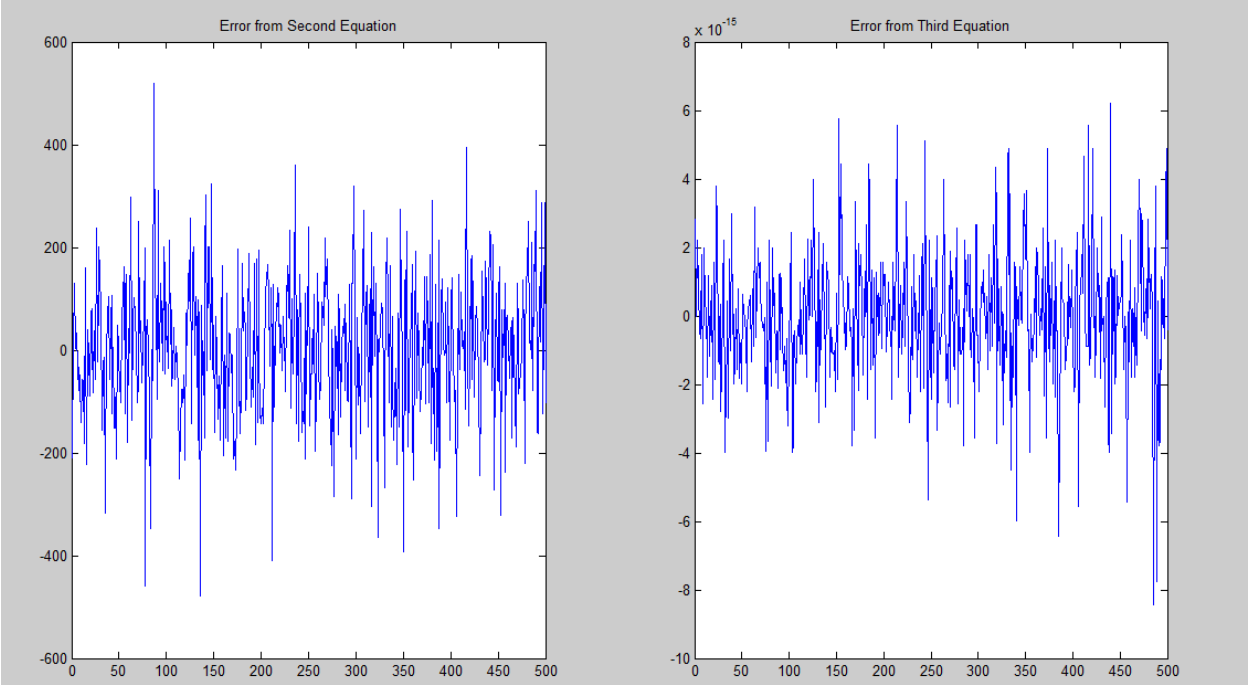
m = n
Both equations have exactly equal number of flops. In this case the system could be solved by simply computing A^{-1} rather than using these 2 equations.

I.3: Take random matrices [A] and [b] with m = 500 and n = 1500, if you implement Eqs. 2 and 3 are you getting the same solution? Explain your findings? (pre-multiply [A] with an 100*eye(500) to make it diagonally dominant).

```
function Generate

A = randn(500,1500);
b = randn(500,1);

I = 100*eye(500);
A = I * A;
AT = A';

%Second equation
in1 = inv(AT*A);
x1 = in1 * AT * b;

%Third equation
in2 = inv(A*AT);
x2 = AT * in2 * b;
```



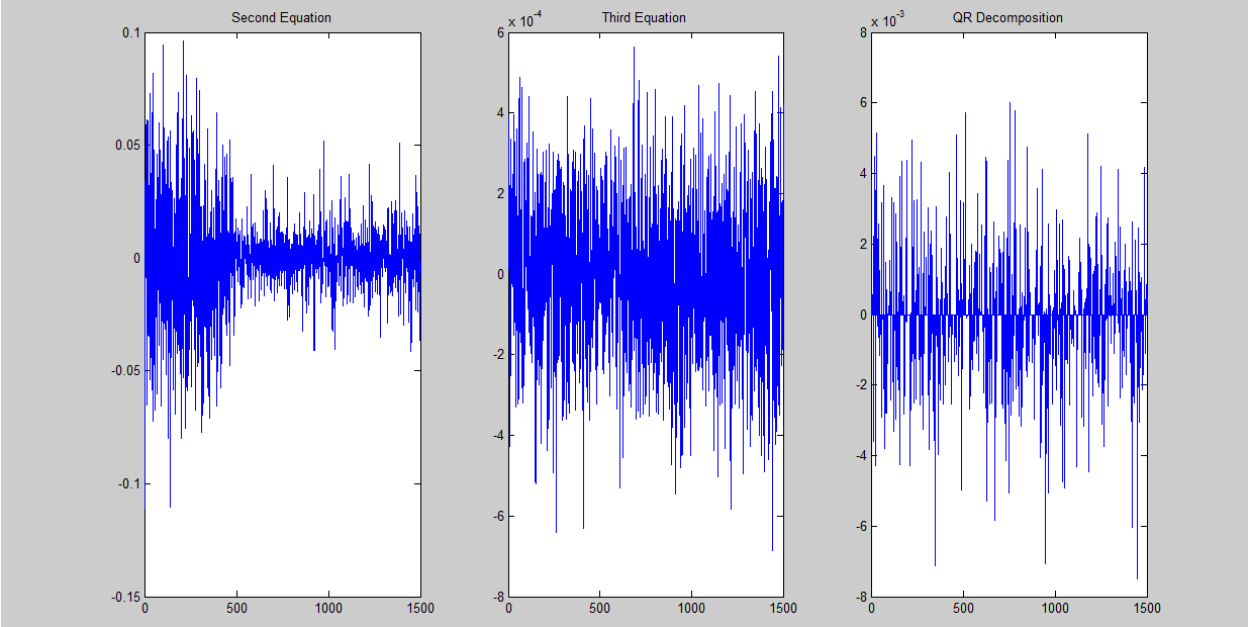
The solution is very different for the two equations. This is because equation 2 is producing a singular matrix by computing A^TA when m<n. Since rank of A is at most 500 here but A^TA is of order 1500. Computing inverse of this matrix increases error. To check the error in the two equations the plot of residual analysis is:



Error is calculated by putting the computed value of x back in first equation (Ax=b) and calculating the difference Ax – b. The error in 3rd equation is close to zero whereas the error blows up for second equation. When m<n, equation 3 should be used.

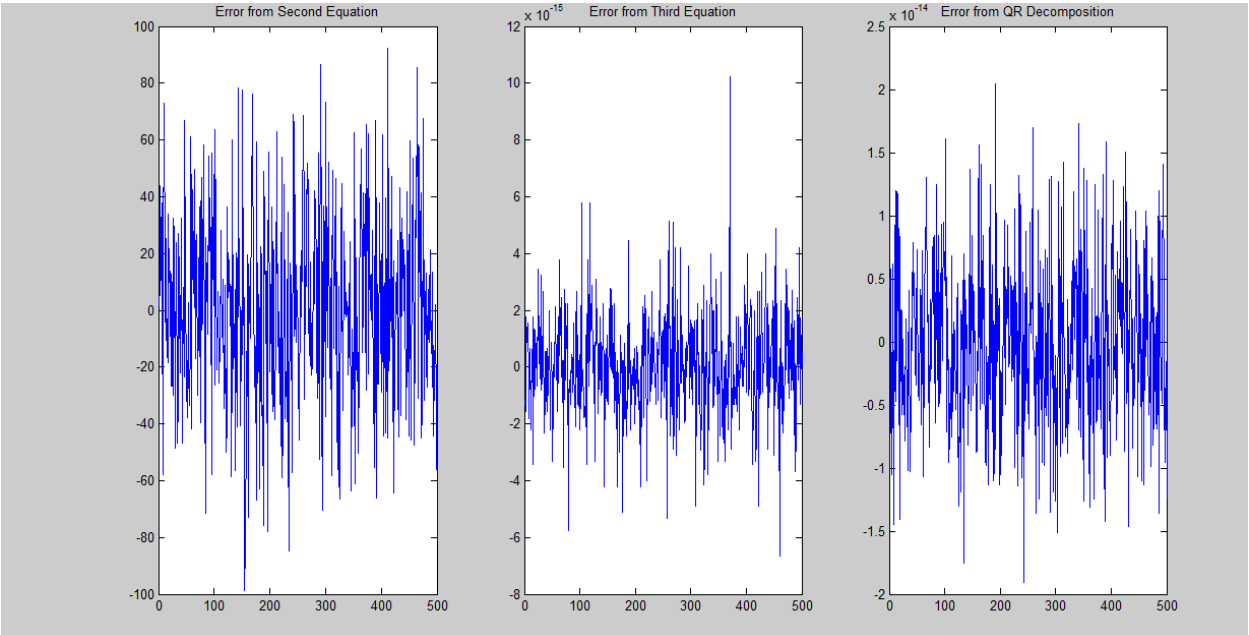
I.4: Take the same matrices you used in part-3 and try to solve Eq. 1 using QR decomposition, is your solution matching with the one (Eq. 2) obtained in part-3? Explain your answer.

```
function x = QR(A,b)

[Q R] = qr(A);
c = Q' * b;
x = R\c;
```



The solution is different than what is obtained by using the 2nd equation from part 3. Further, by looking at the residue error analysis we see how close each method gets to computing x.
QR decomposition is closer to Equation 3 and better than equation 2 for solving this system of linear equations.

QR does better than equation 2 since there is no production of a rank deficient matrix here unlike equation 2.



II.1: Write down the exact number of operations (include only multiplications/divisions) required for getting [b] in Eq. 7 (assume you have used Gaussian-elimination for getting the inverse of a matrix).

*Since matrix multiplication is associative, there could be many ways of multiplying a sequence of matrices. Some of these ways could be computationally more advantageous than others. Here I multiply matrices from left to right sequentially.

Computation of SK^T = r*n*m
Multiplying $[SK^T]^T * W_{delta}$ = m*r*r

Multiplying the result with SK^T = m*r*m

Now we calculate inverse of an 'm' x 'm' matrix using Gaussian Elimination. While reducing A to triangular form, we Augment Identity matrix to make the order 'm' x '2m'.

If matrix is 'm' x 'n' the flops required are:

$$(m-1)m(n+2) - (m+n+2)m(m-1)/2 + (m-1)m(2m-1)/6$$

For 'm' x '2m' matrix:

$$(m-1)m(2m+2) - (3m+2)m(m-1)/2 + (m-1)m(2m-1)/6 = 5m(m^2-1)/6$$

Then we do m back substitutions: $m*[m(m-1)/2] = m^3/2 - m^2/2$

Total for inverse computation: $4m^3/3 - m^2/2 + 5m/2$

Computation of $[SK^T]^T * W_{delta}$ is already done. Now multiplication of this inverse with this is of cost :

$$m * m * r$$

Total computation cost: $mnr + mr^2 + 2m^2r + mr + 4m^3/3 - m^2/2 + 5m/2$

II.2: Rewrite Eq. 8 by doing QR decomposition of [K], do you see any computational advantage in estimating [b]?

$$[B] = ([SK^T]^T [W_{delta}] [SK^T] + [W_b])^{-1} [SK^T]^T [W_{delta}]$$

$K = QR$
$K^T = R^TQ^T$

$$[B] = ([S\ R^TQ^T]^T [W_{delta}] [S\ R^TQ^T] + [W_b])^{-1} [S\ R^TQ^T]^T [W_{delta}]$$

We get:
$$[B] = (QRS^T [W_{delta}] [SR^TQ^T] + [W_b])^{-1} QRS^T [W_{delta}]$$

Here R in upper triangular with dimension 'm' x 'n. If m > n then R will be sparse with more than half entries equal to zero. In that case multiplication with R will be easier as no. of floating point operations with be halved. But for n>m QR will give no computational advantage.

II.3: If d is getting updated in real-time and we are interested in finding corresponding b, which methods among the ones discussed in class will have computationally more advantage? Do you see QR-decomposition fitting into one of those efficient techniques?

$$\{b\} = [B]\ \{d\}$$

where $[B] = ([SK^T]^T\ [W_{delta}]\ [SK^T] + [W_b])^{-1}\ [SK^T]^T\ [W_{delta}]$

Computation of {b} from {d} will take m*r*1 = mr number of operations. When {d} is updated in real time, which each new {d} this matrix multiplication will have to be done again for each case.
In the computation of [B], changing {d} doesn't affect its computation. We will compute it once and then use it each time. Since it is given that $[W_{delta}]$ is equal to some scalar times Identity hence computation with a scalar the equation reduces to:
$[B] = ([SK^T]^T\ a_d[I]\ [SK^T] + a_b[I])^{-1}\ [SK^T]^T\ a_d[I]$

Implies:

$$[B] = (a_d\ [SK^T]^T\ [SK^T] + a_b[I])^{-1}\ [SK^T]^T\ a_d$$

Now if we do QR decomposition of SK^T then we get some computational advantage. Say $M = SK^T$. By QR decomposition we get $M = QR$. Substituting in the equation we get,
$[B] = (a_d\ [QR]^T\ [QR] + a_b[I])^{-1}\ [QR]^T\ a_d$
This reduces to:
$[B] = (a_d\ R^T\ R + a_b[I])^{-1}\ [QR]^T\ a_d$

Further R is upper triangular. So computation of $R^T\ R$ is also reduced. If we say that $[W_{delta}]$ is dependent on {d} then computation of each B benefits as we decompose it once and use it each time.
We cannot apply Cholesky decomposition or LU decomposition on as $[SK^T]$ is not a square matrix.

III.4: Discuss part-3 in terms of m > n, m = n, and m < n.

After doing QR decomposition we get:
$[B] = (a_d\ R^T\ R + a_b[I])^{-1}\ [QR]^T\ a_d$

Dimension of R is 'r' x 'm'.
Since 'r' is at most 'n' we do the following analysis:

n > m
If r>m then R has more than half entries equal to zero so computation of $R^T\ R$ is reduced. QR is very beneficial for this case. If we have r<m then it is same as next case.

n < m
R has less than half entries equal to zero so computation of $R^T\ R$ is closer to matrix multiplication with not huge advantage.

m = n
Since r<n we have r<m. Same as case two.