Lab Session 8

MA-571 : Numerical Linear Algebra Lab 2022 R. Alam

1. **Comment:** The numerical rank of $A \in \mathbb{R}^{n\times m}$ is obtained in MATLAB by the command `rank(A)`. MATLAB uses the `SVD` of A to obtain this value. More specifically, it is obtained by calculating a tolerance level **tol** $= \max\{n, m\}\|A\|_2$ **eps**, where **eps** is the machine precision, and then setting `rank(A)` to be the number of singular values of A which are greater than this value of **tol**. It is possible for the user to change this **tol** value to something else. Type `help rank` for details. The purpose of the following experiment is to illustrate that the above method computes the rank of a matrix quite efficiently in the presence of rounding.

   Consider `A = randn(7,4)` and examine its rank by typing `rank(A)`. Since random matrices are usually full rank, its rank will be 4 in all probability. Add two more columns to A as follows: `A(:,5:6) = A(:,2:3) + A(:,3:4)`; This will create a fifth column in A which is the sum of the 2nd and 3rd columns and a sixth column which is the sum of the 3rd and 4th columns. Theoretically, the rank of A should remain unchanged. This is correctly detected by numerical rank in MATLAB. Type `rank(A)` to find out.

2. **Comment:** The purpose of the following experiment is to illustrate that in the presence of rounding, the `SVD` is generally more efficient in determining the rank of a matrix than the rank revealing `QR` factorization (i.e., QR factorization with column pivoting).

   The *Kahan matrix* $R_n(\theta)$ is an $n \times n$ upper triangular matrix depending on a parameter θ. Let $c = \cos(\theta)$ and $s = \sin(\theta)$. Then

$$R_n(\theta) := \begin{pmatrix} 1 & & & & \\ & s & & & \\ & & s^2 & & \\ & & & \ddots & \\ & & & & s^{n-1} \end{pmatrix} \begin{pmatrix} 1 & -c & -c & \dots & -c \\ & 1 & -c & \dots & -c \\ & & 1 & & -c \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix}.$$

   If θ and n are chosen so that s is close to 1 and n is modestly large, then none of the main diagonal entries is extremely small. It appears that the matrix is far from rank deficient, which is actually not the case. Consider $R_n(\theta)$ when $n = 90$ and $\theta = 1.2$ radians. Verify that the largest main diagonal entry of $R_n(\theta)$ is 1 and the smallest is .001.

   (a) To generate $R_n(\theta)$ in MATLAB and find its singular value use the following commands:

```
A = gallery('kahan', 90, 1.2, 0);   % generates Kahan matrix
sig = svd(A);  % computes singular values.
```

   Now use `format short e` and examine σ_1, σ_{89} and σ_{90}. Type `rank(A)` to get MATLAB's opinion of the numerical rank of A. Check that the numerical rank is 89.
   Next, use the command `[Q,R,E] = qr(A)` to compute a QR decomposition of A with column pivoting. Verify that no pivoting was done in this case by examining the value of `dif` = `norm(eye(90) - E)`. Examine $R(90, 90)$ and the smallest diagonal entry `min(diag(R))` to check the numerical rank of A.

   (b) Next, use the command `A = gallery('kahan',90,1.2,25)` to get a slightly perturbed version of the Kahan matrix. (This produces the Kahan matrix with very small perturbations to the diagonal entries. Type `help private/kahan` for more details.) Repeat part (a) for the perturbed matrix.
   Next, use the command `[Q,R,E] = qr(A)` to compute a QR decomposition of A with column pivoting. Verify that no pivoting was done in this case by examining the value of `dif` = `norm(eye(90) - E)`. Examine $R(90, 90)$ and the smallest diagonal entry `min(diag(R))` and infer that the rank revealing QR decomposition failed to detect the numerical rank deficiency of A.

3. The purpose of this example is to illustrate that Householder QR factorization is backward stable, that is, computed QR factors are the exact QR factors of a slightly perturbed matrix. More precisely, if Q and R are computed QR factors of A using reflectors then $\|Q^TQ - I\|_2 = \mathcal{O}(\mathbf{u})$ and $A+E = QR$ for some matrix E such that $\|E\|_2/\|A\|_2 = \mathcal{O}(\mathbf{u})$. This, however, does not necessarily mean that Q and R will be close to the exact QR factors of A. This can be test as follows:

```
>>R = triu(randn(50)); % compute a 50-by-50 random upper triangular matrix.
>>[Q, X] = qr(randn(50)); % compute a 50-by-50 random unitary matrix Q.
>>A = Q*R; % A is a matrix with known QR factors.
>>[S, T] = qr(A); % Computes Householder QR factorization of A.
```

Now test the backward stability of the Householder QR factorization. Setting `E = S*T - A`, we have `A+E = S*T`. Hence if $\|E\|_2/\|A\|_2 = \mathcal{O}(\mathbf{u})$ then clearly the algorithm `qr` is backward stable. Compute `>> norm(E)`. What is your conclusion?

One may ask: how accurate are `S` and `T`? Show that `S` and `T` are very far from the known QR factors Q and R of A. This illustrates that the Householder QR factrization algorithm is not **forward stable**, that is, the computed factors are not close to the exact factors. Compute the errors

```
>>[norm( Q-S ), norm( R-T )]
```

What do you observe?

4. The Least Squares Problem (LSP) $Ax = b$ has a solution where the fit is good if b is nearly in the range $R(A)$ of A or in other words the angle θ between b and Ax is very small. The purpose of the following exercise is to show that in such cases, the QR method of solving the LSP $Ax = b$ is better than Normal Equations method.

   (a) Use the linspace command to generate a *column vector* X consisting of 50 equally spaced points between 0 and 1. Generate the Vandermonde matrix which has columns $X.^{i-1}$ for $i = 1 : 7$. Choose $\mathtt{w} = \mathtt{randn}(7, 1)$ and $\mathtt{b} = \mathtt{A} * \mathtt{w}$. Then $b \approx p(X)$ where p is the polynomial $p(t) = w(1) + w(2)t + \cdots + w(7)t^6$. This ensures that $\theta \approx 0$ for the LSP $Ax = b$. Solve this problem via Normal Equations method and QR method via reflectors (this is the default procedure so that you just have to type `A\b` for this!) and denote the solutions as `xhat` and `xtilde`, respectively. Examine the relative errors $\dfrac{\|\mathtt{xhat} - \mathtt{w}\|_2}{\|\mathtt{w}\|_2}$, and $\dfrac{\|\mathtt{xtilde} - \mathtt{w}\|_2}{\|\mathtt{w}\|_2}$ in the solutions as well as those in the fits $\dfrac{\|\mathtt{rhat}\|_2}{\|\mathtt{b}\|_2}$ and $\dfrac{\|\mathtt{rtilde}\|_2}{\|\mathtt{b}\|_2}$ where $\mathtt{rhat} := \mathtt{b} - \mathtt{A} * \mathtt{xhat}$ and $\mathtt{rtilde} := \mathtt{b} - \mathtt{A} * \mathtt{xtilde}$. Which method fares better? Also find the condition number of A.

   (b) Repeat the above process for 10 Vandermonde matrices corresponding to polynomials of degrees 6 to 15. Store the relative errors in the solutions corresponding to each method in separate arrays and plot them on the same graph in $\log 10$ scale on the $y - axis$ for a comparative analysis. Do the same also for the relative errors in the fits (measured relative to b as above). Also store the condition numbers of the Vandermonde matrices at each step in a single array.

   What do you observe about the performance of the two methods as the polynomials increase in degree? Which is more sensitive to ill conditioning, the solutions or the fits?

5. **Planetary Orbit:** Consider the quadratic form $q(x, y) := ax^2 + bxy + cy^2 + dx + ey + f$. The set $\mathcal{C} := \{(x, y) \in \mathbb{R}^2 : q(x, y) = 0\}$ is a *conic section*. Cutting the surface $z = q(x, y)$ by the place $z = 0$, one obtains the conic $\mathcal{C}$. Write a MATLAB script using commands `meshgrid` and `contour` to generate $\mathcal{C}$.

   Suppose that a planet follows an elliptical orbit. Here are ten observations of its position in (x, y) plane:

$$
\begin{aligned}
x &= \begin{bmatrix} 1.02 & 0.95 & 0.87 & 0.77 & 0.67 & 0.56 & 0.44 & 0.30 & 0.16 & 0.01 \end{bmatrix} \\
y &= \begin{bmatrix} 0.39 & 0.32 & 0.27 & 0.22 & 0.18 & 0.15 & 0.13 & 0.12 & 0.13 & 0.15 \end{bmatrix}
\end{aligned}
$$

(a) Determine the conic section that fits this data (in the sense of least squares). This can be done by setting one of the coefficients, say, $f = 1$ and solving the resulting 10-by-5 over determined LSP using backslash command. Plot the orbit (that is the conic) with x on the x-axis and y on the y-axis. Superimpose the ten data points on the plot. Write a single MATLAB script that implements the above task.

(b) Conclude that the LSP is nearly rank deficient. Your next task is to illustrate the effect of small perturbation in the data to the orbit of the planet. Perturb the data x and y by adding to each component a random number uniformly distributed in the interval $[-0.005, 0.005]$:

```
x = x + .005*(2*rand(n,1)-1); y = y + .005*(2*rand(n,1)-1);
```

Compute the new coefficients resulting from the perturbed data. Plot the new orbit on the same plot of the old orbit. Write a MATLAB script that implements the job. Run the script a couple of times and comment on your results.

*** End ***