Lab Session 12

MA-581 : Numerical Computations Lab R. Alam November 01, 2020

---

1. This example illustrates image compression techniques using SVD. The following commands will first load a built-in 320×200 matrix X that represents the pixel image of a clown, computes its SVD $X = U \begin{bmatrix} \text{diag}(\sigma_1, \ldots, \sigma_r) & 0 \\ 0 & 0 \end{bmatrix} V^T$, where $r = \text{rank}(A)$, and then displays the image when X is approximated by its best rank k approximation $X_k := U \begin{bmatrix} \text{diag}(\sigma_1, \ldots, \sigma_k) & 0 \\ 0 & 0 \end{bmatrix} V^T$ for a chosen value of $k \leq r$. Use the following commands:

```
load clown.mat; [U, S, V] = svd(X); colormap('gray');
image(U(:, 1:k)*S(1:k, 1:k)*V(:,1:k)')
```

The storage required for A_k is $k(m+n) = 520k$ words whereas the storage required for the full image is $n \times m = 6400$ words in this case. Therefore, $\frac{520k}{6400}$ gives the compression ratio for the compressed image. Also the error in the representation is $\frac{\sigma_{k+1}}{\sigma_1}$. Run the above commands for various choices of k and make a table that records the relative errors and compression ratios for each choice.

2. **Comment:** The numerical rank of $A \in \mathbb{R}^{n \times m}$ is obtained in MATLAB by typing `rank(A)`. MATLAB uses the `SVD` of A to obtain this value. More specifically, it is obtained by calculating a tolerance level `tol` $=$ `eps` $\max\{n, m\}\|A\|_2$ where `eps` is the machine epsilon 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 this example 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.

   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

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

   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 are 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 for yourself 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 values type

$$\texttt{A} = \texttt{gallery}(\texttt{'kahan'}, 90, 1.2, 0); \quad \texttt{sig} = \texttt{svd}(\texttt{A})$$

   Type `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.

   (b) Type `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. Perform a QR decomposition by column pivoting on A by typing `[Q, R, P] = qr(A)`. Verify that no pivoting was done in this case by examining the value of `dif = norm(eye(90)-P)`. Examine $R(90, 90)$ and infer that the rank revealing QR decomposition failed to detect the numerical rank deficiency of A.

3. 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 $\frac{\|\mathtt{xhat} - \mathtt{w}\|_2}{\|\mathtt{w}\|_2}$, and $\frac{\|\mathtt{xtilde} - \mathtt{w}\|_2}{\|\mathtt{w}\|_2}$ in the solutions as well as those in the fits $\frac{\|\mathtt{rhat}\|_2}{\|\mathtt{b}\|_2}$ and $\frac{\|\mathtt{rtilde}\|_2}{\|\mathtt{b}\|_2}$ 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?

4. If $A \in \mathbb{C}^{n\times n}$ is Hermitian then the spectral theorem says that there is a unitary matrix U such that $A = U\mathrm{diag}(\lambda_1, \ldots, \lambda_n)U^*$, where $\lambda_1, \ldots, \lambda_n$ are real eigenvalues of A. The MATLAB command `[U, D] = eig(A)` computes spectral decomposition of A when A is Hermitian. Singular value decomposition (SVD) of a nonsingular matrix A can be computed as follows.

   1. Compute spectral decomposition $A^*A = V\mathrm{diag}(\lambda_1, \cdots, \lambda_n)V^*$ such that $\lambda_j \geq \lambda_{j+1}$.
   2. Define $\Sigma := \mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$.
   3. Compute $U := AV\Sigma^{-1}$. Then $A = U\Sigma V^*$ is an SVD of A.

   Write a MATLAB function, say, `[U, S, V] = mysvd(A)` that implements the above method. Your function may look like this

```
function [U, S, V] = mysvd(A)
% [U, S, V] = mysvd(A) produces a diagonal matrix S of singular
% values of nonsingular matrix A and two unitary matrices U and V
% whose columns are the corresponding left and right singular
% vectors so that AV = US.
```

   Your task is to compare performance of `mysvd` with that of the MATLAB command `svd` by considering the test matrices given below. For this purpose, compute $\|V^*V - I\|_2$, $\|U^*U - I\|_2$ and $\|AV - US\|_2$ when U, S and V are obtained by your functions as well as by `svd(A)`. Prepare a table of these results and check which method is better and reliable. Let $\hat{\sigma}_\ell$ and σ_ℓ be singular values computed by `mysvd` and `svd`, respectively. Compute $|\sigma_\ell - \hat{\sigma}_\ell|/\sigma_\ell$ for 4 smallest singular values, prepare a table and comment on the results. The test matrices are given below.

   1. Consider the Hilbert matrix `hilb(n)` for $n = 9, 11, 13$.
   2. The frank matrix `F = gallery('frank',n)` for $n = 8, 10, 12$.

Finally, define a 7×7 matrix A with singular values $\sigma_j := 10^{-2j}$ for $j = 1, 2, \ldots, 7$. It can be defined as follows.

```
>> rand('state', 100); X = rand(7); [U, R] = qr(X);
>> rand('state', 50); Y = rand(7); [V, R] = qr(Y);
```

Define $A = U\text{diag}(\sigma_1, \cdots, \sigma_7)V^*$. Let $\hat{\sigma}_j, j = 1, 2, \ldots, 7$, be the computed singular values of A. For $\ell = 1, 3, 5, 7$, compute the relative errors $|\sigma_\ell - \hat{\sigma}_\ell|/\sigma_\ell$ for the methods `mysvd` and `svd`. Which method is better?.

*** End ***