Question 5

```
% compare and constrast the various ways of computing the matrix
% exponential by evaluating the methods on the following matrix, A:

A = [[0 0 0 0 0]; [0 -2 -1 1 0]; [0 2 1 -1 0]; [0 0 0 -1 1]; [0 0 0 0 -1]];

% MATLAB's preferred matrix exponential method
tic
scale_square = expmdemo1(A)
toc

% breaks down when norm(A) is large
tic
power_series = expmdemo2(A/1000)
toc

% breaks down when cond(A) is large
tic
eigendecomposition = expmdemo3(A)
toc


% explain results
```

```
scale_square =

    1.0000         0         0         0         0
         0   -0.2642   -0.6321    0.3679    0.1839
         0    1.2642    1.6321   -0.3679   -0.1839
         0         0         0    0.3679    0.3679
         0         0         0         0    0.3679

Elapsed time is 0.544801 seconds.

power_series =

    1.0000         0         0         0         0
         0    0.9980   -0.0010    0.0010    0.0000
         0    0.0020    1.0010   -0.0010   -0.0000
         0         0         0    0.9990    0.0010
         0         0         0         0    0.9990

Elapsed time is 0.006021 seconds.
Warning: Matrix is close to singular or badly scaled. Results may be
inaccurate. RCOND =  1.156034e-32.

eigendecomposition =

    1.0000         0         0         0         0
         0   -0.2642   -0.6321    0.4250         0
```

```
         0    1.2642    1.6321   -0.9036         0
         0         0         0    0.3679         0
         0         0         0         0    0.3679
```

Elapsed time is 0.032261 seconds.

Explanation

```
% method 1, the square scaling method, is the one implemented by the
% baseline expm command in Matlab (with an additional pade approximation),
% it is therefore probably the most generally performant/accurate of the
% methods discussed.

% for the matrix provided, the first two methods show good agreement,
% indicating veracity in the results they provide. note that the norm of A

norm = norm(A)

% is not particularly small, which would otherwise cause computational
% intensity increases in the power_series approximation.

% method 2, the power series approximation is neither efficient nor
% particularly accurate for matrices with large norms. As the norm of the
% matrix increases, the time it takes to converge to a stable value is
% likely to increase as well. In our case, the norm of A is quite low, so
% this method is quite accurate. Note that this method actually ran the
% fastest on our A matrix, but this pattern probably will not hold as the
% norm of A increased.

% method 3, the eigendecomposition method, has a more difficult time
% matching the perceived accuracy of the first two methods for this
% particular matrix. This method works best for matrices that are
% symmetric, orthogonal, etc. The A matrix in this case is certainly not
% symmetric and it admits repeated eigenvalues and therefore encodes
% generalized eigenvectors. In this case, with repeated eigenvalues, the
% expression for x(t) will likely involve polynomials of t rather than
% constant terms.

[V, D] = eig(A);
V
D

% in the case of our a matrix, the repeated eigenvalues at -1 lead to a
% jordan form that consists of a Jordan block of size 3:

J = jordan(A)

% this characterizes A as defective, and implies that it does not have a
% full set of linearly independent eigenvectors. The sensitivity of the
% matrix exponential is highly dependent on the size of the Jordan block(s)
% of the A matrix. the upper bound for the perturbation function of the
% matrix exponential is dependent on a relation that involves the (size of
% max jordan block)^2 * e^(size of max jordan block). So for this A, small
```

```
% perturbations can have extremelly large effects on the matrix
% exponential. In real world terms, this means that a matrix is extremelly
% sensitive to modelling errors, which are a constant presence in system
% definition.

% additionally, the condition of A is infinite, which implies that this
% eigendecomposition method may fail completely, producing inaccurate
% results.

cond = cond(A)
```

```
norm =

    3.4925


V =

         0         0    1.0000         0         0
   -0.4472    0.7071         0   -0.7071    0.7071
    0.8944   -0.7071         0    0.7071   -0.7071
         0         0         0    0.0000   -0.0000
         0         0         0         0    0.0000


D =

     0     0     0     0     0
     0    -1     0     0     0
     0     0     0     0     0
     0     0     0    -1     0
     0     0     0     0    -1


J =

     0     0     0     0     0
     0     0     0     0     0
     0     0    -1     1     0
     0     0     0    -1     1
     0     0     0     0    -1


cond =

   Inf
```

Published with MATLAB® R2023a