In this assignment you will write efficient MATLAB codes to solve least squares problems involving block structured matrices known as Kronecker products. Some information about Kronecker products and their properties are attached to this assignment.

Specifically, given $A_1$ and $A_2 \in \mathbb{R}^{n \times n}$, $b_1$ and $b_2 \in \mathbb{R}^{n^2}$ we consider solving the LS problem:

$$\min_x \|Ax - b\|_2^2 = \min_x \left\| \begin{bmatrix} A_1 \otimes A_2 \\ A_2 \otimes A_1 \end{bmatrix} x - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \right\|_2^2$$

For simplicity we'll assume $A_1$ and $A_2$ are nonsingular. Note since $A_1$ and $A_2$ are $n \times n$, the matrices $A_1 \otimes A_2$ and $A_2 \otimes A_1$ are $n^2 \times n^2$. There is a built-in MATLAB function kron that can be used to explicitly form Kronecker products, but only if $n$ is small. Thus we need to find more efficient approaches that do not require explicitly forming Kronecker products.

1. Given two $n \times n$ matrices $A_1$ and $A_2$, the generalized singular value decomposition (GSVD) computes the decompositions:

$$A_1 = U_1 \Sigma Y^T \quad \text{and} \quad A_2 = V \Delta Y^T$$

where $U$ and $V$ are $n \times n$ orthogonal matrices, $\Sigma$ and $\Delta$ are diagonal matrices with nonnegative diagonal elements, and $Y$ is a nonsingular matrix.

(a) Show that the SVD of $A_1 A_2^{-1}$ can be obtained from the GSVD of $A_1$ and $A_2$. Thus, the GSVD is sometimes called the quotient SVD.

(b) Using the GSVD of $A_1$ and $A_2$ show that we can obtain a factorization of the form:

$$A = \begin{bmatrix} A_1 \otimes A_2 \\ A_2 \otimes A_1 \end{bmatrix} = WDZ^T$$

where $W$ is an orthogonal matrix involving Kronecker products of $U$ and $V$, $Z$ is a nonsingular matrix involving Kronecker products of $Y$, and $D$ is a block diagonal matrix of the form:

$$D = \begin{bmatrix} \Sigma \otimes \Delta \\ \Delta \otimes \Sigma \end{bmatrix}$$

2. Show that

$$\|Ax - b\|_2^2 = \|D\hat{x} - \hat{b}\|_2^2$$

where $\hat{x}$ is defined in terms of $Z$ and $x$, and

$$\hat{b} = \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \end{bmatrix}$$

is defined in terms of $W$ and $b$. 
3. Now we look at solving the LS problem:

$$\min_\hat{x} \| D\hat{x} - \hat{b} \|_2^2$$

Since $D$ is a block diagonal matrix, we should be able to do this efficiently using Givens rotations. In particular, we want to use Givens rotations to (implicitly) construct an orthogonal matrix $Q$ such that

$$\| Q^T D\hat{x} - Q^T \hat{b} \|_2^2 = \| \hat{D}\hat{x} - \hat{b} \|_2^2$$

where $\hat{D}$ is diagonal; that is,

$$\hat{D} = Q^T D = Q^T \begin{bmatrix} \times & \cdots & \times \\ \cdots & \times \\ \times & \cdots & \times \end{bmatrix} = \begin{bmatrix} \hat{x} \\ \vdots \\ \hat{x} \end{bmatrix}$$

and $\hat{b} = Q^T \hat{b}$.

But we need to do this without explicitly forming $\Sigma \otimes \Delta$, $\Delta \otimes \Sigma$, $D$ or $\hat{D}$. Instead, we want to work only with vectors containing the diagonal entries of these matrices.

(a) Show that:

$$\text{diag}(\Sigma \otimes \Delta) = \text{diag}(\Sigma) \otimes \text{diag}(\Delta),$$

where $\text{diag}(C)$ forms a column vector containing the diagonal entries of the given matrix $C$. Note that a similar relation holds for $\text{diag}(\Delta \otimes \Sigma)$.

(b) Show that from just the vectors

$$d_1 = \text{diag}(\Sigma) \otimes \text{diag}(\Delta) \quad \text{and} \quad d_2 = \text{diag}(\Delta) \otimes \text{diag}(\Sigma)$$

you can find the necessary Givens rotations to reduce the block diagonal matrix $D$ to the diagonal matrix $\hat{D}$. Moreover, show that you can obtain the vector $\text{diag}(\hat{D})$ without explicitly forming the matrix $\hat{D}$.

(c) Show that these Givens rotations can be applied to $\hat{b}$, obtaining $\hat{b} = Q^T b$, without explicitly forming $Q$. 
4. Now it’s time to put everything together and write some MATLAB code to solve our Kronecker product structured least squares problem.

(a) First we will need a function \texttt{Givens.m} that computes Givens rotations. Specifically, write a MATLAB function \texttt{Givens.m}, with the following specification:

\begin{verbatim}
function \[c, s\] = Givens(a, b)
  
  % Compute a Givens rotation to zero an element:
  
  % / c -s \ / a \_ / r \ 
  % \ s c / \ b / - \ 0 / 
  
  % Input: scalars a and b
  % Output: scalars c and s, satisfying c^2 + s^2 = 1
\end{verbatim}

(b) Write a function \texttt{KronLS.m} with the following specification:

\begin{verbatim}
function x = KronLS(A1, A2, b1, b2)
  
  % This function solves the LS problem:
  
  % \min || \begin{bmatrix} kron(A1,A2) & \ x - b1 \\ kron(A2,A1) & \ b2 \end{bmatrix} \||

  % without explicitly forming the matrices kron(A1,A2) and kron(A2,A1).

  % Input: A1 - n-by-n nonsingular matrix
  %        A2 - n-by-n nonsingular matrix
  %        b1 - vector of length n^2
  %        b2 - vector of length n^2

  % Output: x - solution of LS problem
\end{verbatim}

The steps in your code should follow what you did in the previous problems. That is,

- Use the built-in MATLAB function \texttt{gsvd} to compute the GSVD of $A_1$ and $A_2$.
- Use properties of Kronecker products to compute $\hat{b}_1$ and $\hat{b}_2$ without explicitly forming any Kronecker products. You should make use of the built-in MATLAB function \texttt{reshape}.
- Compute the vectors $d_1 = \text{diag}(\Sigma) \otimes \text{diag}(\Delta)$ and $d_2 = \text{diag}(\Delta) \otimes \text{diag}(\Sigma)$ here you should make use of the built-in MATLAB functions \texttt{diag} and \texttt{kron}.
- Next use your function \texttt{Givens} to reduce the block diagonal elements of $\hat{D}$ to the diagonal elements of $\tilde{D}$, and also to transform $\hat{b}$ to $\tilde{b}$. You should only need to access the vectors $d_1$ and $d_2$, and not create the diagonal matrices explicitly. You will need one \texttt{for} loop to do this.
Now you should have $\tilde{d} = \text{diag}(\tilde{D})$ and $\tilde{b}$, from which it should be a trivial matter, using the MATLAB ./ operator, to compute the solution of

$$\min_x \| \tilde{D}x - \tilde{b} \|_2^2$$

Now the last step is to compute $x$ from $\hat{x}$. You need to recall how these two vectors are related. This should suggest that you again need to use properties of Kronecker products here. You should also make use of \ (the MATLAB back-slash operator) and / (the MATLAB forward-slash operator), as well as reshape.

5. Put the following MATLAB statements into a script m-file, and use it to test your codes. What do these results tell you?

```matlab
k = 1;
for n = 5:30
    A1 = rand(n);
    A2 = rand(n);
    b1 = rand(n*n,1);
    b2 = rand(n*n,1);
    A = [kron(A1, A2); kron(A2, A1)];
    b = [b1;b2];
    tic;
    x_backslash = A\b;
    t1(k) = toc;
    tic;
    x_KronLS = KronLS(A1, A2, b1, b2);
    t2(k) = toc;
    error_nrm(k) = norm(x_backslash - x_KronLS)/norm(x_backslash);
    k = k+1;
end
figure(1), clf
semilogy(5:30, t1, 'b-o', 'LineWidth', 2)
hold on
semilogy(5:30, t2, 'r-s', 'LineWidth', 2)
xlabel('Dimension of A1 and A2')
ylabel('Time to solve (seconds)')
legend('Naive approach','Efficient approach','Location', 'NW')
figure(2), clf
semilogy(5:30,error_nrm, 'b-o', 'LineWidth', 2)
xlabel('Dimension of A1 and A2')
ylabel('norm of error in using KronLS')
```
6. Finally, download the data:

http://www.mathcs.emory.edu/~nagy/courses/fall08/hw2_data.mat

and then put the following MATLAB statements into a script m-file. What results are computed?

```
load hw2_data
n = size(A1,1);
figure(3), clf
subplot(2,2,1), imshow(reshape(b1,n,n),[]) % b_1 Data
subplot(2,2,2), imshow(reshape(b2,n,n),[]) % b_2 Data
x_KronLS = KronLS(A1, A2, b1, b2);
subplot(2,1,2), imshow(reshape(x_KronLS, n, n), []) % LS Solution
```

Some Remarks:

- Your solutions should include any plots and tables generated from your numerical experiments, and used to explain your conclusions.
- All MATLAB codes you used to generate your results should be sent to me by email. If there are multiple files, it would help if you archived them in one `zip` or `tgz` file.
- All MATLAB codes should be clear, concise, and well documented. When I look at the codes, I do not want to guess what they are supposed to compute.
- You do not have to type up your solutions, but if you do, then I would prefer if you use LaTeX (if you want some guidance on how to include figures and tables in your LaTeX document, please let me know). In this case, you can email to me, along with your codes, a PDF copy of your write up.
Introduction to Kronecker Products

If $A$ is an $m \times n$ matrix and $B$ is a $p \times q$ matrix, then the Kronecker product of $A$ and $B$ is the $mp \times nq$ matrix

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix}$$

Note that if $A$ and $B$ are large matrices, then the Kronecker product $A \otimes B$ will be huge. MATLAB has a built-in function `kron` that can be used as

$$K = \text{kron}(A, B);$$

However, you will quickly run out of memory if you try this for matrices that are $50 \times 50$ or larger. Fortunately we can exploit the block structure of Kronecker products to do many computations involving $A \otimes B$ without actually forming the Kronecker product. Instead we need only do computations with $A$ and $B$ individually.

To begin, we state some very simple properties of Kronecker products, which should not be difficult to verify:

- $(A \otimes B)^T = A^T \otimes B^T$
- If $A$ and $B$ are square and nonsingular, then $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$
- $(A \otimes B)(C \otimes D) = AC \otimes BD$

The next property we want to consider involves the matrix-vector multiplication

$$y = (A \otimes B)x,$$

where $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{p \times q}$. Thus $A \otimes B \in \mathbb{R}^{mp \times nq}$, $x \in \mathbb{R}^{nq}$, and $y \in \mathbb{R}^{mp}$. Our goal is to exploit the block structure of the Kronecker product matrix to compute $y$ without explicitly forming $(A \otimes B)$. To see how this can be done, first partition the vectors $x$ and $y$ as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \mathbb{R}^{q}, \text{ and } y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad y_i \in \mathbb{R}^{p}$$

Then

$$y = (A \otimes B)x = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$
which can be written as

\[
\begin{bmatrix}
 y_1 \\
 y_2 \\
 \vdots \\
 y_m
\end{bmatrix} =
\begin{bmatrix}
 a_{11}Bx_1 + a_{12}Bx_2 + \cdots + a_{1n}Bx_n \\
 a_{21}Bx_1 + a_{22}Bx_2 + \cdots + a_{2n}Bx_n \\
 \vdots \\
 a_{m1}Bx_1 + a_{m2}Bx_2 + \cdots + a_{mn}Bx_n
\end{bmatrix}
\]

Now notice that each \( y_i \) has the form

\[
y_i = a_{i1}Bx_1 + a_{i2}Bx_2 + \cdots + a_{in}Bx_n
\]

Now define \( a_i^T = \begin{bmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{bmatrix} \) = \( i \)th row of \( A \), and define \( X \) to be a matrix with \( i \)th column \( x_i \); that is,

\[
X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}
\]

Then

\[
y_i = BXa_i, \quad i = 1, 2, \ldots, m
\]

Analogous to the matrix \( X \), define a matrix \( Y \) with columns \( y_i \):

\[
Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix}
\]

Then we have

\[
Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix} = \begin{bmatrix} BXa_1 & BXa_2 & \cdots & BXa_m \end{bmatrix} = BX \begin{bmatrix} a_1 & a_2 & \cdots & a_m \end{bmatrix} = BXA^T
\]

In the last step above, recall that \( a_i^T \) is the \( i \)th row of \( A \), so \( a_i \) is the \( i \)th column of \( A^T \).

To summarize, we have the following property of Kronecker products:

\[
y = (A \otimes B)x \iff Y = BXA^T
\]

where \( X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \in \mathbb{R}^{q \times n} \) and \( Y = \begin{bmatrix} y_1 & y_2 & \cdots & y_m \end{bmatrix} \in \mathbb{R}^{p \times m} \).
Remarks on notation and MATLAB computations:

- If $X = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix}$ is an array with columns $x_i$, then in many books you will see the notation $x = \text{vec}(X)$ used to denote a vector obtained by stacking the columns of $X$ on top of each other. That is, 

$$x = \text{vec}(X) = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- MATLAB does not have a $\text{vec}$ function, but it does have a built-in function called $\text{reshape}$ that can be used for this purpose. Specifically, if $X \in \mathbb{R}^{q \times n}$, then the vector $x = \text{vec}(X)$ can be obtained with the MATLAB statement:

$$x = \text{reshape}(X, q*n, 1);$$

A short cut to using the reshape command is to use the colon operator. That is,

$$x = X(:);$$

does the same thing as $x = \text{reshape}(X, q*n, 1)$. To go the other way, from $x$ to $X$, you can again use the $\text{reshape}$ function:

$$X = \text{reshape}(x, q, n);$$

For more information, see the MATLAB doc page for $\text{reshape}$.

- Thus, suppose we are given $A$, $B$, and $x$ in MATLAB, and we want to compute $y = (A \otimes B)x$. This can be done without explicitly forming $A \otimes B$ as follows:

$$[m, n] = \text{size}(A);$$
$$[p, q] = \text{size}(B);$$
$$X = \text{reshape}(x, q, n);$$
$$Y = B*X*A';$$
$$y = \text{reshape}(Y, m*p, 1);$$

Note that this computation requires $O(npq+qnm)$ arithmetic operations (FLOPS) to compute $y$. On the other hand, if we explicitly form $A \otimes B$, then the cost to compute $y$ is $O(mpnq)$. In addition to the computational savings, we save a lot on storage if we don’t explicitly construct $A \otimes B$, and instead just store the smaller matrices $A$ and $B$.

- As with matrix-vector multiplication, we can efficiently solve linear systems

$$(A \otimes B)x = y$$

using properties of Kronecker products. Specifically, assume $A$ and $B \in \mathbb{R}^{n \times n}$ are both nonsingular. Then using properties of Kronecker products we know

$$x = (A \otimes B)^{-1}y = (A^{-1} \otimes B^{-1})y$$
From the matrix-vector multiplication property this is equivalent to computing:

\[ X = B^{-1}Y A^{-T}, \quad x = \text{vec}(X), \quad y = \text{vec}(Y) \]

Generally we never explicitly form the inverse of a matrix, and instead should read “inverse” for meaning “solve”. That is:

- Computing \( Z = B^{-1}Y \) is equivalent to the problem: Given \( B \) and \( Y \), solve the matrix equation \( BZ = Y \) for \( Z \).
- Computing \( X = ZA^{-T} \) is equivalent to the problem: Given \( A \) and \( Z \), solve the matrix equation \( XA^T = Z \) for \( X \). (Note that this is also equivalent to solving the matrix equation \( AX^T = Z^T \) for \( X \)).

The cost of doing this with matrix factorization methods (e.g., \( PA = LU \)) is only \( O(n^3) \). On the other hand, if we explicitly form \( A \otimes B \) and then solve the system, the cost will be \( O(n^6) \).

In MATLAB, using the efficient method outlined above to solve \((A \otimes B)x = y\) can be implemented as:

```matlab
n = size(A,1);
Y = reshape(y, n, n);
X = B \ Y / A';
x = reshape(X, n*n, 1);
```

If you’re not sure how the back-slash \( \backslash \) and the forward-slash \( / \) operators work, then you should read the MATLAB doc pages: doc slash.