Math 4355
Matlab Homework #B

You may work either solo or in a group of two. If you work in a group of two, each student must upload their own report into eLearning and you must list both names at the top of the report and briefly state who did what.

Your task is to write MATLAB functions that implement the standard Gram-Schmidt orthogonalization algorithm, show that it breaks down when the vectors in the initial basis are nearly linearly dependent, and to verify that better results can be obtained using a modified version of the algorithm.

Turn in a *single pdf file* containing the answers to the conceptual questions posed below, a print out of your code, and of the numerical results you obtained by running the code to answer the questions below. Make sure your code is commented well enough so that you will understand it in two month's time.

Background reading from Meyer Section 5.5: grey boxes on pages 309 and 316 and discussion on page 315.

1. Write a Matlab function `U=StdGS(X)` that implements the standard Gram-Schmidt process summarized by equation (5.5.4) on page 309 of Meyer. The input to this function is an $m \times n$ matrix `X` whose columns form a basis for an n-dimensional subspace $\mathcal{S}$ of R^m. The output is a matrix `U` whose columns form an orthonormal basis for $\mathcal{S}$.

2. Make sure your code is debugged by reproducing the results in Meyer page 309 Example 5.5.1.

3. To see that the standard Gram-Schmidt orthogonalization algorithm breaks down when the vectors in the initial basis are nearly linearly dependent, we need to quantify how close to being linearly dependent the columns of $\mathbf{X}$ are, and how far from being orthonormal the basis in $\mathbf{U}$ is.

4. To quantify the how close to being linearly dependent the columns of $\mathbf{X}$ are we use the condition number defined by

$$\operatorname{cond}(\mathbf{X}) := \frac{\max\limits_{\|\mathbf{v}\|=1} \|\mathbf{X}\mathbf{v}\|}{\min\limits_{\|\mathbf{v}\|=1} \|\mathbf{X}\mathbf{v}\|}. \tag{1}$$

   The idea is that $\operatorname{cond}(\mathbf{X}) \to \infty$ as the basis becomes more and more linearly dependent. Intuitively, if the columns of $\mathbf{X}$ were linearly dependent, then there would be a vector $\mathbf{v} \neq \mathbf{0}$ so that $\mathbf{X}\mathbf{v} = \mathbf{0}$, and by linearity we can choose $\|\mathbf{v}\| = 1$. So the denominator in (1) will approach zero as the vectors become more linearly dependent. Why do you think we need the numerator in (1)?

5. We can determine how far from orthonormal the new basis is as follows. In the situation that $n = m$ we know that $\mathbf{U}$ should be an orthogonal matrix: $\mathbf{U}^T\mathbf{U} = \mathbf{I} = \mathbf{U}\mathbf{U}^T$. However when $n < m$, we only get that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$. Why isn't $\mathbf{U}\mathbf{U}^T = \mathbf{I}$ too? Therefore is makes sense to define an error matrix $\mathbf{E} = \mathbf{U}^T\mathbf{U} - \mathbf{I}$. If the basis was truely orthogonal then all the entries of $\mathbf{E}$ should be zero. In reality they will not be, since real numbers (and operations on them) cannot be represented exactly on a computer. For small matrices, you can just print out and inspect at the entries of $\mathbf{E}$. However for large matrices this is not practical. Instead, we can use a matrix norm. An easy choice to work with is the Frobenius norm (see Meyer page 279), which takes the square root of the sum of the squares of all the entries of $\mathbf{E}$:

$$\|\mathbf{E}\|_F^2 := \sum_{i,j=1}^{n} |E_{ij}|^2. \tag{2}$$

In Matlab the Frobenius norm is given by $\|\mathbf{E}\|_F =$ `norm(E,'fro')`. (See `help norm`).

6. To show that the standard algorithm breaks down when the vectors in the original basis are nearly linearly dependent we first consider the following input matrices:

(a) $\mathbf{A}_\epsilon = \begin{bmatrix} 1 & 1 & 1 \\ \epsilon & \epsilon & 0 \\ \epsilon & 0 & \epsilon \end{bmatrix}$ with $\epsilon = 10^{-k}$ for $k = 0, 2, 4, 6, 8$.

(b) $\mathbf{B}_\epsilon = \begin{bmatrix} 1 & 1 & 1 \\ 1+\epsilon & 1 & 1 \\ 1 & 1+\epsilon & 1 \\ 1 & 1 & 1+\epsilon \end{bmatrix}$ with $\epsilon = 10^{-k}$ for $k = 0, 2, 4, 6, 8$.

Clearly as $\epsilon \to 0$, the columns of $\mathbf{A}_\epsilon$ and $\mathbf{B}_\epsilon$ become linearly dependent. For both $\mathbf{A}_\epsilon$ and $\mathbf{B}_\epsilon$, make a table showing the condition number of the input matrix, and the Frobenius norm of the error matrix, $\mathbf{E}$, as functions of k. You should also check directly that as $\epsilon \to 0$ the angles between the vectors in the output basis can be quite far from 90°. **Hint:** Use the formula relating the dot product of two vectors to the angle between them.

7. The reason that the standard Gram-Schmidt algorithm breaks down so badly for $\mathbf{A}_\epsilon$ when $\epsilon = 10^{-8}$ is that the lengths of the column vectors of $\mathbf{A}_\epsilon$ and the inner products between different column vectors are not all computed correctly. This is because with the 16-digit arithmetic Matlab uses, when $\epsilon = 10^{-8}$, the computer thinks that $1+\epsilon^2 = 1$. Try it for yourself! *Here, be sure to use the command* `format long` *to print out numbers to all 16 decimal places. (You can restore Matlab's default output format using the command* `format`*.)* Even with larger values of ϵ, you should find that the output basis may not be quite orthonormal. Basically the reason is that for some value of k, $\mathbf{x}_k$ is almost a linear combination of $\{\mathbf{x}_1, \cdots, \mathbf{x}_{k-1}\}$ and hence of $\{\mathbf{u}_1, \cdots, \mathbf{u}_{k-1}\}$. Therefore

$$\mathbf{w}_k = \mathbf{x}_k - \sum_{i=1}^{k-1} \langle \mathbf{u}_i | \mathbf{x}_k \rangle \mathbf{u}_i \tag{3}$$

will be very small and hence will not be computed accurately. This error is then amplified when we compute $\mathbf{u}_k = \mathbf{w}_k/\|\mathbf{w}_k\|$.

8. The modified Gram-Schmidt algorithm does better than the standard algorithm. See Meyer pages 315-316. The standard algorithm produces intermediate bases such as $\{\mathbf{u}_1, \mathbf{u}_2, \mathbf{x}_3, \cdots, \mathbf{x}_n\}$. The modified algorithm instead produces intermediate bases such as $\{\mathbf{u}_1, \mathbf{u}_2, \mathbf{E}_3\mathbf{E}_2\mathbf{x}_3, \cdots, \mathbf{E}_3\mathbf{E}_2\mathbf{x}_n\}$, where $\mathbf{E}_3\mathbf{E}_2$ projects $\mathbf{x}_k$ onto the subspace orthogonal to the span of $\{\mathbf{u}_1, \mathbf{u}_2\}$. This enables angles between vectors in the intermediate basis to be computed more accurately than with the standard algorithm.

9. Write a Matlab function U=ModGS(X) that implements the modified Gram-Schmidt process summarized in the algorithm at the top of page 316 of Meyer.

10. Make sure your code is debugged by reproducing the results in Meyer page 309 Example 5.5.1.

11. Generate tables for the modified algorithm as in item 6 above. Comment on your results.

12. Finally test your code on the $m \times n$ Vandermonde matrix generated using the matlab code

```
 function V=Vandermonde(m,n)

p=([1:m]/m)';

for i=1:m
for j=1:n
V(i,j) = p(i)∧(j-1);
end
end

end
```

Generate tables of condition numbers and error norms for the standard and modified algorithms with $(n, m) = (4, 6)$, $(6, 9)$, $(10, 15)$, and $(12, 18)$. Comment on your results.

Matlab Hints

In addition to the Matlab commands in Project #A you may find the following useful.

- You can write the code in such a way that you just manipulate the columns of X and U. For example here is a column operation in which column 3 of U is updated to be column 3 of U plus 2 times column 4 of X: U(:,3) = U(:,3) + 2 * X(:,4).
- If $\mathbf{v}$ is a vector then $\|\mathbf{v}\|$ =norm($\mathbf{v}$).

- Here is a good way to output data in a table. Suppose m, n and row vectors of integers, and C, ErrorS, ErrorM are row vectors of floating point numbers, where all vectors have the same number of elements. To output this data to a nicely formatted table in which the row vectors become columns in the table, use:

  ```
   output = [m; n; C; ErrorS; ErrorM];

  sprintf(' %2d \t %2d \t %.1e \t %.1e \t %.1e \n',output)
  ```

  Here, the %2d displays an integer using 2 slots and \t is a tab. With these two commands the entries in the second column will all start at the same spot, lining up nicely. The command %.1e displays floating point numbers using exponential notation with one after the decimal place.