Problem 5.1 *worth 6 points*
Linear independence of stacked vectors. Consider the stacked vectors

$$c_1 = \begin{bmatrix} a_1 \\ b_1 \end{bmatrix}, \quad \ldots, \quad c_k = \begin{bmatrix} a_k \\ b_k \end{bmatrix}$$

where $a_1, \ldots, a_k$ are n n-vectors and $b_1, \ldots, b_k$ are m-vectors.

(a) Suppose $a_1, \ldots, a_k$ are linearly independent. (We make no assumptions about the vectors $b_1, \ldots, b_k$.) Can we conclude that the stacked vectors $c_1, \ldots, c_k$ are linearly independent?

(b) Now suppose that $a_1, \ldots, a_k$ are linearly dependent. (Again, with no assumptions about $b_1, \ldots, b_k$.) Can we conclude that the stacked vectors $c_1, \ldots, c_k$ are linearly dependent?

Solution:

(a) If $a_1, \ldots, a_k$ are linearly independent then that means each vector in the set of a is unique and cannot be made from other vectors in the set. So if c_1 has a_1, no matter what scalar you multiply c_1 by, you cannot get any other element of a, meaning you cannot get any other element of c.

$$c_n = \begin{bmatrix} a_n \\ b_n \end{bmatrix} c_j = \begin{bmatrix} a_j \\ b_j \end{bmatrix}$$
$$\beta c_n = c_j$$
$$\begin{bmatrix} \beta a_n \\ \beta b_n \end{bmatrix} = \begin{bmatrix} a_j \\ b_j \end{bmatrix}$$

If $\beta a_n = a_j$ then they are not linearly independent, because linear independence implies there does not exist a scalar which can multiply a_n to become a_j meaning there is not a scalar that can multiply c_n to become c_j meaning the set of c vectors is linearly independent.

(b) If $a_1, \ldots, a_k$ are linearly dependent and we do not know anything about $b_1, \ldots, b_k$ then we cannot conclude anything about c. As shown in (a) if $b_1, \ldots, b_k$ is linearly independent then we will get another linearly independent set, but if $b_1, \ldots, b_k$ is linearly dependent then c could be linearly dependent as well.

Problem 5.5 *worth 4 points*
Orthogonalizing vectors. Suppose that a and b are any n-vectors. Show that we can always find a scalar γ so that $(a - \gamma b) \perp b$, and that is unique if $b = 0$. (Give a formula for the scalar γ.) In other words, we can always subtract a multiple of a vector from another one, so that the result is orthogonal to the original vector. The orthogonalization step in the Gram–Schmidt algorithm is an application of this.

Solution:

$$(a - \gamma b)^T b = 0$$
$$a^T b - \gamma b^T b = 0$$
$$a^T b = \gamma b^T b$$
$$\frac{a^T b}{b^T b} = \gamma$$

Problem 5.6 *worth 6 points*
Gram-Schmidt algorithm. Consider the list of n n-vectors

$$a_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad a_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots, \quad a_n = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}.$$

(The vector a_i has its first i entries equal to one, and the remaining entries zero.) Describe what happens when you run the Gram–Schmidt algorithm on this list of vectors, *i.e.*, say what $q_1, \ldots, q_n$ are. Is $a_1, \ldots, a_n$ a basis?

Solution: Gram-Schmidt:

$$\tilde{q}_i = a_i - (q_1^T a_i)q_1 - \cdots - (q_{i-1}^T a_i)q_{i-1}$$

Let $n = 4$

$$a_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad a_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad a_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \quad a_4 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

$$\tilde{q}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\tilde{q}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} - \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\tilde{q}_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} - (1) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\tilde{q}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\tilde{q}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} - \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \right) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} - \left(\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\tilde{q}_3 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} - (1) \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} - (1) \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

$$\tilde{q}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\tilde{q_4} = \begin{bmatrix}1\\1\\1\\1\end{bmatrix} - \left(\begin{bmatrix}1\\0\\0\\0\end{bmatrix}^T \begin{bmatrix}1\\1\\0\\0\end{bmatrix}\right)\begin{bmatrix}1\\0\\0\\0\end{bmatrix} - \left(\begin{bmatrix}0\\1\\0\\0\end{bmatrix}^T \begin{bmatrix}1\\1\\1\\0\end{bmatrix}\right)\begin{bmatrix}0\\1\\0\\0\end{bmatrix} - \left(\begin{bmatrix}0\\0\\1\\0\end{bmatrix}^T \begin{bmatrix}1\\1\\1\\1\end{bmatrix}\right)\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

$$\tilde{q_4} = \begin{bmatrix}1\\1\\1\\1\end{bmatrix} - (1)\begin{bmatrix}1\\0\\0\\0\end{bmatrix} - (1)\begin{bmatrix}0\\1\\0\\0\end{bmatrix} - (1)\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$$

$$\tilde{q_4} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

As we can see, as each iteration continues *Gram-Schmidt* produces e_i for every q_i. If we step inductively to $n+1$ we see *Gram-Schmidt* looks like

$$\begin{aligned} \tilde{q_{n+1}} &= a_{n+1} - (q_1^T a_i)q_1 - \cdots - (q_{n+1-1}^T a_i)q_{n+1-1} \\ &= a_{n+1} - (q_1^T a_i)q_1 - \cdots - (q_n^T a_i)q_n \end{aligned}$$

and from our $n = 4$ example, we see that $q_n = e_n$ and that $q_n^T a_i = 1$, so this would produce e_{n+1}. This fits because the vectors are n n-vectors so they grow as the iterations continue. The vectors are a basis because they are linearly independent.

Problem 5.9 *worth 4 points*
A particular computer can carry out the Gram–Schmidt algorithm on a list of $k = 1000$ n-vectors, with $n = 10000$, in around 2 seconds. About how long would you expect it to take to carry out the Gram–Schmidt algorithm with $\tilde{k} = 500$ n-vectors, with $\tilde{n} = 1000$?

Solution: Gram Schmidt complexity is $2nk^2$, so with our first numbers we see

$$\begin{aligned} &2(10,000)(1,000)^2 \\ =&2(10,000)(1,000,000) \\ =&2(10,000,000,000) \end{aligned}$$

So we have a flop count of 20 billion, dividing by 2 seconds we get 10 billion flops per second on this computer. Using our new $\tilde{k} = 500$ n-vectors and $\tilde{n} = 1000$ we get

$$\begin{aligned} &2(1,000)(500)^2 \\ =&2(1,000)(250,000) \\ =&2(250,000,000) \end{aligned}$$

which gives a flop count of 500 million, setting a ratio we see

$$\frac{500,000,000}{x} = \frac{20,000,000,000}{2}$$

$$20,000,000,000x = 1,000,000,000$$

$$\boxed{x = \frac{1}{20}}\text{seconds}$$