Department of Mathematics, University of Utah
Analysis of Numerical Methods I
MTH6610 – Section 001 – Fall 2017

Lecture notes: Modified Gram-Schmidt
Wednesday September 13, 2017

These notes are not a substitute for class attendance. Their main purpose is to provide a lecture overview summarizing the topics covered.

Reading: Trefethen & Bau III, Lecture 8

We recall the Gram-Schmidt procedure from the previous lecture: Let $a_1, \ldots, a_n$ be any basis for $\mathbb{C}^n$. Our goal is to *orthogonalize* these vectors. The following inductive procedure generates an orthonormal set $q_1, \ldots, q_n$ via the scalars r_{ij}:

$$
\begin{aligned}
u_1 &= a_1, & r_{11} &= \|u_1\|, & q_1 &= \frac{u_1}{r_{11}} \\
u_2 &= a_2 - P_1 a_2, & r_{22} &= \|u_2\|, & q_2 &= \frac{u_2}{r_{22}} \\
&\vdots \\
u_{k+1} &= a_{k+1} - P_k a_{k+1}, & r_{k+1,k+1} &= \|u_{k+1}\|, & q_{k+1} &= \frac{u_{k+1}}{r_{k+1,k+1}}
\end{aligned}
$$

The projection matrix P_k is the orthogonal projection onto span $\{q_1, \ldots, q_k\}$.
It turns out that using the Gram-Schmidt procedure to compute QR factorizations is quite numerically unstable when implemented on a computer. A relatively straightforward methodology to fix this problem is to perform the "modified" Gram-Schmidt procedure. The standard Gram-Schmidt procedure orthogonalizes a_{k+1} against $a_1, \ldots, a_k$ in one step. The modified version performs this orthogonalization step-by-step. At iteration $k+1$:

$$
\begin{aligned}
r_{k,j} &= q_k^* a_j, & a_j &\leftarrow a_j - r_{k,j} q_k, & j &= k+1, \ldots, n \\
r_{k+1,k+1} &= \|a_{k+1}\|, & q_{k+1} &= \frac{a_{k+1}}{\|a_{k+1}\|}
\end{aligned}
$$

Note that the procedure operates on and updates all columns a_j at every iteration. The modified Gram-Schmidt operations are arithemtically equivalent to the standard Gram-Schmidt operations, but the modified version is more stable due to effects of finite-precision on computers polluting the standard Gram-Schmidt operations.
If A is an $m \times n$ matrix, how much work is required to compute a QR factorization? We can estimate this via the modified Gram-Schmidt computations above: at iteration $k+1$ the following operations are performed

- Compute $r_{j,k}$ (m multiplications, $m-1$ additions) for $j = k+1, \ldots, n$
- Update a_j (m multiplications, m additions) for $j = k+1, \ldots, n$
- Compute $r_{k+1,k+1}$ (m multiplications, $m-1$ additions, 1 square root operation)
- Compute q_{k+1} (m multiplications)

We count each addition, multiplication, and here square roots as well, as a single floating-point operation (flop). Then summing up the operation count above over $k = 1, \ldots, n-1$, yields

$$\sum_{k=1}^{n-1} (2m-1)(n-k) + 2m(n-k) + 2m + m \sim 2mn^2,$$

where the notation $\sim$ means

$$f(n,m) \sim g(n,m) \qquad \Longrightarrow \qquad \lim_{n,m\to\infty} \frac{f(n,m)}{g(n,m)} = 1.$$

Another way to communicate the computational *complexity* of this algorithm is to say that computing the QR factorization via modified Gram-Schmidt requires $\mathcal{O}(mn^2)$ work. The formal definition of big-$\mathcal{O}$ notation is

$$f(n) = \mathcal{O}(g(n)) \text{ for large } n \qquad \Longleftrightarrow \qquad \limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$