

Gram-Schmidt Orthogonalization

Objective: Classical Gram-Schmidt is unstable, so we present modified Gram-Schmidt, which is stable. We also interpret it as right-multiplication by upper-triangular matrices.

Gram-Schmidt projections

We have just seen Gram-Schmidt in its classical form. Now we describe the same algorithm but in another way: using orthogonal projectors.

Let $A \in \mathbb{R}^{m \times n}$, ($m \geq n$) have full rank. Consider now the sequence of formulas

$$q_1 = \frac{P_1 a_1}{\|P_1 a_1\|}, \quad q_2 = \frac{P_2 a_2}{\|P_2 a_2\|}, \quad \dots, \quad q_n = \frac{P_n a_n}{\|P_n a_n\|}$$

where each P_j is an orthogonal projector.

Specifically, $P_j \in \mathbb{R}^{m \times m}$ of rank $m - (j - 1)$ that projects \mathbb{R}^m orthogonally onto the space orthogonal to

$$\langle q_1, q_2, \dots, q_{j-1} \rangle.$$

Note

$$P_1 = I.$$

★

Also note that as defined, for each j

- q_j is orthogonal to q_1, q_2, \dots, q_{j-1} ,
- q_j lies in $\langle a_1, a_2, \dots, a_j \rangle$,
- $\|q_j\| = 1$.

This algorithm is equivalent to the classical Gram Schmidt Algorithm 7.1.

But, each P_j can be represented explicitly:

Let \hat{Q}_{j-1} ($m \times (j-1)$) contain the first $j-1$ columns of \hat{Q}

$$\hat{Q}_{j-1} = [q_1 | q_2 | \dots | q_{j-1}].$$

Then P_j is given by

$$P_j = I - \hat{Q}_{j-1} \hat{Q}_{j-1}^T.$$

Note

$$\begin{aligned} P_j a_j &= a_j - (q_1^T a_j) q_1 - \dots - (q_{j-1}^T a_j) q_{j-1} \\ &= a_j - (q_1 q_1^T) a_j - \dots - (q_{j-1} q_{j-1}^T) a_j \\ &= (I - q_1 q_1^T - \dots - q_{j-1} q_{j-1}^T) a_j \end{aligned}$$

Modified Gram-Schmidt

Classical Gram-Schmidt is unstable numerically and so is never implemented in practice.

(We will treat numerical stability systematically in Lecture 14 \rightarrow for now it is sufficient to consider a stable algorithm as one which does not suffer drastically from perturbations due to roundoff errors)

Fortunately, there is a simple modification that makes Gram-Schmidt stable.

For each j , Algorithm 7.1 computes a single orthogonal projector of rank $m - (j - 1)$

$$v_j = P_j a_j.$$

In contrast, the modified Gram Schmidt algorithm achieves the same result but by means of $(j - 1)$ projections of rank $(m - 1)$.

Recall from Lecture 6 that $P_{\perp q}$ denotes the rank $m - 1$ orthogonal projector onto the space orthogonal to some nonzero vector $q \in \mathbb{R}^m$.

From the definition of P_j , it is easy to see that

$$P_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1}$$

with $P_1 = I$.

Thus, we can write

$$v_j = P_{\perp q_{j-1}} \cdots P_{\perp q_2} P_{\perp q_1} a_j.$$

Mathematically, the two formulas for v_j are equivalent, but modified Gram-Schmidt is based on the second formula rather than the first.

★

The modified Gram-Schmidt goes as follows:

$$\begin{aligned} v_j^{(1)} &= a_j \\ v_j^{(2)} &= P_{\perp q_1} v_j^{(1)} = v_j^{(1)} - q_1 q_1^T v_j^{(1)} \\ v_j^{(3)} &= P_{\perp q_2} v_j^{(2)} = v_j^{(2)} - q_2 q_2^T v_j^{(2)} \\ &\vdots \\ v_j^{(j)} &= v_j = P_{\perp q_{j-1}} v_j^{(j-1)} \\ &= v_j^{(j-1)} - q_{j-1} q_{j-1}^T v_j^{(j-1)} \end{aligned}$$

In finite precision computer arithmetic, we will see the formulation introduces smaller errors than the previous one.

For the implementation, $P_{\perp q_i}$ can be conveniently applied to $v_j^{(i)}$ for each $j > i$ as soon as q_i is known.

ALGORITHM 8.1:
MODIFIED GRAM-SCHMIDT (STABLE)

```
for  $i = 1$  to  $n$  do  
     $v_i = a_i$   
end for  
for  $i = 1$  to  $n$  do  
     $r_{ii} = \|v_i\|$   
     $q_i = \frac{v_i}{r_{ii}}$   
    for  $j = i + 1$  to  $n$  do  
         $r_{ij} = q_i^T v_j$   
         $v_j = v_j - r_{ij}q_i$   
    end for  
end for
```

Note

We let v_j overwrite a_i and q_i overwrite v_i to save storage.

Operation count

With any algorithm, it is important to assess its cost. We assess cost in the classical sense of counting floating-point operations (*flops*)

Each $+$, $-$, $*$, $/$, $\sqrt{\quad}$ all count as one flop.

This is a simplification in the sense that there is much more going on during program execution that will affect performance besides simple flop counts, e.g., how data stored and accessed in memory, competing jobs, message passing, etc.

For both variants of the Gram-Schmidt algorithm, the result is the following:

Theorem (8.1)

Algorithms 7.1 and 8.1 require $\sim 2mn^2$ flops to compute a (reduced) QR factorization of an $m \times n$ matrix.

Note

We only worry about the leading-order behaviors when counting flops
i.e., we assume m, n are large.

$$\sim mn^2 \quad \text{means} \quad \lim_{m,n \rightarrow 0} \frac{\text{number of flops}}{2mn^2} = 1$$

It is similar to $\Theta(\cdot)$ notation, except more precise.
We could restate Theorem 8.1 as taking $\Theta(mn^2)$ flops,

but then we would not know that the leading coefficient was 2.

We now derive the flop count specifically for modified Gram-Schmidt:

For large m, n , the work is dominated by the innermost loop:

$$\begin{aligned} r_{ij} &= q_i^T v_j \rightarrow m \text{ multiplications}/m-1 \text{ additions} \\ v_j &= v_j - r_{ij}q_i \rightarrow m \text{ multiplications}/m \text{ subtractions} \end{aligned}$$

\therefore Total work $\sim 4m$ flops (~ 4 flops per vector component).

$$\begin{aligned} \therefore \text{total flops} &\sim \sum_{i=1}^n \sum_{j=i+1}^n 4m \\ &= \sum_{i=1}^n (n-i)4m \\ &= \left(n^2 - \frac{n(n+1)}{2} \right) \cdot 4m \\ &\sim 2mn^2 \end{aligned}$$

Gram-Schmidt as triangular orthogonalization

Each outer step of modified Gram-Schmidt can be interpreted as right-multiplication by a square upper-triangular matrix.

Beginning with A , the first iteration multiplies a_1 by $\frac{1}{r_{11}}$, then subtracts r_{ij} times the result from each of the remaining a_j

This is equivalent to right-multiplication by a matrix R_1 :

$$\begin{array}{cccc}
 [a_1 & | & a_2 & | \dots & | & a_n] \\
 \uparrow & & \uparrow & & & \uparrow \\
 v_1^{(1)} & & v_2^{(1)} & & & v_n^{(1)}
 \end{array}
 \begin{bmatrix}
 \frac{1}{r_{11}} & \frac{-r_{12}}{r_{11}} & \frac{-r_{13}}{r_{11}} & \cdots \\
 & 1 & & \\
 & & 1 & \\
 & & & \ddots
 \end{bmatrix}
 = [q_1 | v_2^{(2)} | v_3^{(2)} | \dots | v_n^{(2)}]$$

In general, step i subtracts $\frac{r_{ij}}{r_{ii}} * \text{column } i$ of the “current A ” from columns $j > i$ and replaces column i by $\frac{1}{r_{ii}}$ times itself.

This corresponds to multiplication by an upper triangular matrix R_i .

e.g.,

$$\begin{aligned} R_2 &= \begin{bmatrix} 1 & & & & \\ & \frac{1}{r_{22}} & \frac{-r_{23}}{r_{22}} & \cdots & \\ & & 1 & & \\ & & & \ddots & \\ & & & & \ddots \end{bmatrix} \\ R_3 &= \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \frac{1}{r_{33}} & \cdots & \\ & & & \ddots & \\ & & & & \ddots \end{bmatrix} \\ &\vdots \end{aligned}$$

At the end, we have

$$AR_1R_2\cdots R_n = \hat{Q}$$

where $\hat{R}^{-1} = R_1R_2\cdots R_n$.

\Rightarrow Gram-Schmidt is a method of *triangular orthogonalization*.

It applies triangular operations on the right of a matrix to reduce it to a matrix with orthonormal columns.

Note

We never compute the R_i explicitly!

They are meant only to give insight.

We will see a similarity to Gaussian elimination.