Applied Mathematics 205

Unit II: Numerical Linear Algebra

Lecturer: Dr. David Knezevic

Unit II: Numerical Linear Algebra

Chapter II.3: QR Factorization, SVD

QR Factorization

QR Factorization

A square matrix $Q \in \mathbb{R}^{n \times n}$ is called orthogonal if its columns and rows are orthonormal vectors

Equivalently, $Q^T Q = QQ^T = \mathrm{I}$

Orthogonal matrices preserve the Euclidean norm of a vector, i.e.

$$\|Qv\|_2^2 = v^T Q^T Qv = v^T v = \|v\|_2^2$$

Hence, geometrically, we picture orthogonal matrices as reflection or rotation operators

Orthogonal matrices are very important in scientific computing, since norm-preservation implies $\mathrm{cond}(Q) = 1$

QR Factorization

A matrix $A \in \mathbb{R}^{m \times n}$, $m \geq n$, can be factorized into

$$A = QR$$

where

- $Q \in \mathbb{R}^{m \times m}$ is orthogonal
- $R \equiv \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} \in \mathbb{R}^{m \times n}$
- $\hat{R} \in \mathbb{R}^{n \times n}$ is upper-triangular

As we indicated earlier, QR is very good for solving overdetermined linear least-squares problems, $Ax \simeq b$ [1]

[1] QR can also be used to solve a square system $Ax = b$, but requires $\sim 2\times$ as many operations as Gaussian elimination hence not the standard choice

QR Factorization

To see why, consider the 2-norm of the least-squares residual:

$$\begin{aligned} \|r(x)\|_2^2 &= \|b - Ax\|_2^2 = \|b - Q \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} x\|_2^2 \\ &= \|Q^T \left(b - Q \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} x \right) \|_2^2 \\ &= \|Q^T b - \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix} x\|_2^2 \end{aligned}$$

(We used the fact that $\|Q^T z\|_2 = \|z\|_2$ in the second line)

QR Factorization

Then, let $Q^T b = [c_1, c_2]^T$ where $c_1 \in \mathbb{R}^n, c_2 \in \mathbb{R}^{m-n}$, so that

$$\|r(x)\|_2^2 = \|c_1 - \hat{R}x\|_2^2 + \|c_2\|_2^2$$

Question: Based on this expression, how do we minimize $\|r(x)\|_2$?

QR Factorization

Answer: We don't have any control over the second term, $\|c_2\|_2^2$, since it doesn't contain x

Hence we minimize $\|r(x)\|_2^2$ by making the first term zero

That is, we solve the $n \times n$ triangular system $\hat{R}x = c_1$ — this is what Matlab does when we use "backslash" for least squares

Also, this tells us that $\min_{x \in \mathbb{R}^n} \|r(x)\|_2 = \|c_2\|_2$

QR Factorization

Recall that solving linear least-squares via the normal equations requires solving a system with the matrix $A^T A$

But using the normal equations directly is problematic since[2] $\mathrm{cond}(A^T A) = \mathrm{cond}(A)^2$

The QR approach avoids this condition-number-squaring effect and is much more numerically stable!

[2] This can be shown via the SVD

QR Factorization

How do we compute the QR Factorization?

There are three main methods

- Gram-Schmidt Orthogonalization
- Householder Triangularization
- Givens Rotations

We will cover Gram-Schmidt and Householder in class

QR Factorization

Suppose $A \in \mathbb{R}^{m \times n}$, $m \geq n$

One way to picture the QR factorization is to construct a sequence of orthonormal vectors $q_1, q_2, \ldots$ such that

$$\operatorname{span}\{q_1, q_2, \ldots, q_j\} = \operatorname{span}\{a_{(:,1)}, a_{(:,2)}, \ldots, a_{(:,j)}\}, \quad j = 1, \ldots, n$$

We seek coefficients r_{ij} such that

$$\begin{aligned} a_{(:,1)} &= r_{11} q_1, \\ a_{(:,2)} &= r_{12} q_1 + r_{22} q_2, \\ &\vdots \\ a_{(:,n)} &= r_{1n} q_1 + r_{2n} q_2 + \cdots + r_{nn} q_n. \end{aligned}$$

This can be done via the Gram-Schmidt process, as we'll discuss shortly

QR Factorization

In matrix form we have:

$$\left[\begin{array}{c|c|c|c} & & & \\ a_{(:,1)} & a_{(:,2)} & \cdots & a_{(:,n)} \\ & & & \end{array}\right] = \left[\begin{array}{c|c|c|c} & & & \\ q_1 & q_2 & \cdots & q_n \\ & & & \end{array}\right] \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}$$

This gives $A = \hat{Q}\hat{R}$ for $\hat{Q} \in \mathbb{R}^{m \times n}$, $\hat{R} \in \mathbb{R}^{n \times n}$

This is called the reduced QR factorization of A, which is slightly different from the definition we gave earlier

Note that for $m > n$, $\hat{Q}^T\hat{Q} = \mathrm{I}$, but $\hat{Q}\hat{Q}^T \neq \mathrm{I}$ (the latter is why the full QR is sometimes nice)

Full vs Reduced QR Factorization

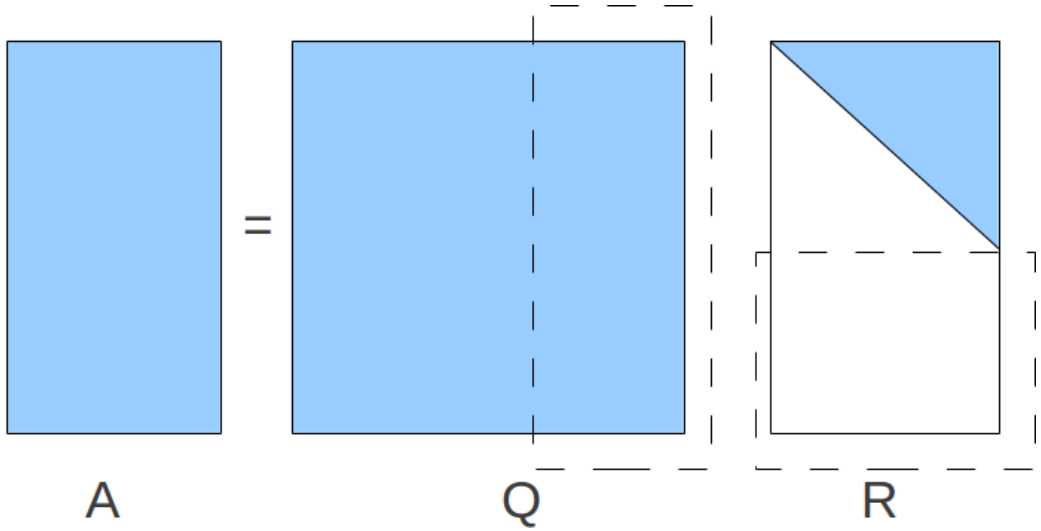
The full QR factorization (defined earlier)

$$A = QR$$

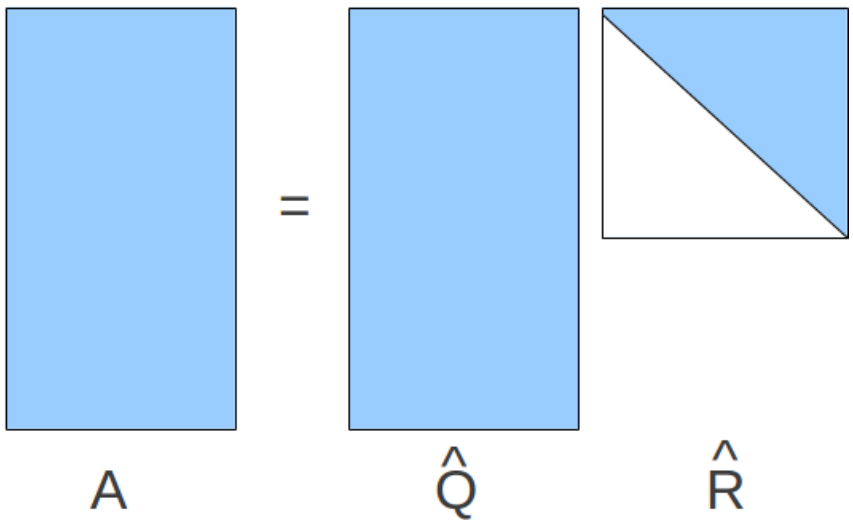
is obtained by appending $m - n$ arbitrary orthonormal columns to $\hat{Q}$ to make it an $m \times m$ orthogonal matrix

We also need to append rows of zeros to $\hat{R}$ to "silence" the last $m - n$ columns of Q, to obtain $R = \begin{bmatrix} \hat{R} \\ 0 \end{bmatrix}$

Full vs Reduced QR Factorization



Full QR



Reduced QR

Full vs Reduced QR Factorization

Exercise: Show that the linear least-squares solution is given by $\hat{R}x = \hat{Q}^T b$ by plugging $A = \hat{Q}\hat{R}$ into the Normal Equations

This is equivalent to the least-squares result we showed earlier using the full QR factorization, since $c_1 = \hat{Q}^T b$

Full vs Reduced QR Factorization

In Matlab, `qr` gives the full QR factorization by default

```
>> A = rand(5,3);
>> [Q,R] = qr(A)

Q =

    0.4927   -0.4806   -0.1780   -0.6413    0.2886
    0.5478   -0.3583    0.5777    0.4690   -0.1336
    0.0768    0.4754    0.6343   -0.5674   -0.2091
    0.5523    0.3391   -0.4808    0.0385   -0.5893
    0.3824    0.5473   -0.0311    0.2129    0.7127


R =

    1.6536    1.1405    1.2569
         0    0.9661    0.6341
         0         0    0.8816
         0         0         0
         0         0         0
```

Full vs Reduced QR Factorization

In Matlab, `qr(A,0)` gives the reduced QR factorization

```
>> [Q,R] = qr(A,0)

Q =

    0.4927   -0.4806   -0.1780
    0.5478   -0.3583    0.5777
    0.0768    0.4754    0.6343
    0.5523    0.3391   -0.4808
    0.3824    0.5473   -0.0311


R =

    1.6536    1.1405    1.2569
         0    0.9661    0.6341
         0         0    0.8816
```

Gram-Schmidt Orthogonalization

Question: How do we use the Gram-Schmidt process to compute the q_i, $i = 1, \ldots, n$?

Answer: In step j, find unit vector $q_j \in \operatorname{span}\{a_{(:,1)}, a_{(:,2)}, \ldots, a_{(:,j)}\}$ orthogonal to $\operatorname{span}\{q_1, q_n, \ldots, q_{j-1}\}$

To do this, we set $v_j \equiv a_{(:,j)} - (q_1^T a_{(:,j)})q_1 - \cdots - (q_{j-1}^T a_{(:,j)})q_{j-1}$, and then $q_j \equiv v_j / \|v_j\|_2$ satisfies our requirements

This gives the matrix $\hat{Q}$, and next we can determine the values of r_{ij} such that $A = \hat{Q}\hat{R}$

Gram-Schmidt Orthogonalization

We write the set of equations for the q_i as

$$\begin{aligned} q_1 &= \frac{a_{(:,1)}}{r_{11}}, \\ q_2 &= \frac{a_{(:,2)} - r_{12} q_1}{r_{22}}, \\ &\vdots \\ q_n &= \frac{a_{(:,n)} - \sum_{i=1}^{n-1} r_{in} q_i}{r_{nn}}. \end{aligned}$$

Then from the definition of q_j, we see that

$$\begin{aligned} r_{ij} &= q_i^T a_{(:,j)}, \qquad\qquad i \neq j \\ |r_{jj}| &= \|a_{(:,j)} - \sum_{i=1}^{j-1} r_{ij} q_i\|_2 \end{aligned}$$

The sign of r_{jj} is not determined uniquely, e.g. we could choose $r_{jj} > 0$ for each j

Classical Gram-Schmidt Process

The Gram-Schmidt algorithm we have described is provided in the pseudocode below

1: **for** $j = 1 : n$ **do**
2: $\quad v_j = a_{(:,j)}$
3: $\quad$ **for** $i = 1 : j - 1$ **do**
4: $\quad\quad r_{ij} = q_i^T a_{(:,j)}$
5: $\quad\quad v_j = v_j - r_{ij} q_i$
6: $\quad$ **end for**
7: $\quad r_{jj} = \|v_j\|_2$
8: $\quad q_j = v_j / r_{jj}$
9: **end for**

This is referred to the classical Gram-Schmidt (CGS) method

Gram-Schmidt Orthogonalization

The only way the Gram-Schmidt process can "fail" is if $|r_{jj}| = \|v_j\|_2 = 0$ for some j

This can only happen if $a_{(:,j)} = \sum_{i=1}^{j-1} r_{ij} q_i$ for some j, i.e. if $a_{(:,j)} \in \operatorname{span}\{q_1, q_n, \ldots, q_{j-1}\} = \operatorname{span}\{a_{(:,1)}, a_{(:,2)}, \ldots, a_{(:,j-1)}\}$

This means that columns of A are linearly dependent

Therefore, Gram-Schmidt fails $\implies$ cols. of A linearly dependent

Gram-Schmidt Orthogonalization

Equivalently, by contrapositive: cols. of A linearly independent $\implies$ Gram-Schmidt succeeds

Theorem: Every $A \in \mathbb{R}^{m \times n} (m \geq n)$ of full rank has a unique reduced QR factorization $A = \hat{Q}\hat{R}$ with $r_{ii} > 0$

The only non-uniqueness in the Gram-Schmidt process was in the sign of r_{ii}, hence $\hat{Q}\hat{R}$ is unique if $r_{ii} > 0$

Gram-Schmidt Orthogonalization

Theorem: Every $A \in \mathbb{R}^{m \times n} (m \geq n)$ has a full QR factorization.

Case 1: A has full rank

- We compute the reduced QR factorization from above
- To make Q square we pad $\hat{Q}$ with $m - n$ arbitrary orthonormal columns
- We also pad $\hat{R}$ with $m - n$ rows of zeros to get R

Case 2: A doesn't have full rank

- At some point in computing the reduced QR factorization, we encounter $\|v_j\|_2 = 0$
- At this point we pick an arbitrary q_j orthogonal to $\text{span}\{q_1, q_2, \ldots, q_{j-1}\}$ and then proceed as in Case 1

Modified Gram-Schmidt Process

The classical Gram-Schmidt process is numerically unstable!
(sensitive to rounding error, orthogonality of the q_j degrades)

The algorithm can be reformulated to give the modified Gram-Schmidt process, which is numerically more robust

Key idea: when each new q_j is computed, orthogonalize each remaining columns of A against it

Modified Gram-Schmidt Process

Modified Gram-Schmidt (MGS):

1: **for** $i = 1 : n$ **do**
2: $\quad v_i = a_{(:,i)}$
3: **end for**
4: **for** $i = 1 : n$ **do**
5: $\quad r_{ii} = \|v_i\|_2$
6: $\quad q_i = v_i / r_{ii}$
7: $\quad$ **for** $j = i + 1 : n$ **do**
8: $\quad\quad r_{ij} = q_i^T v_j$
9: $\quad\quad v_j = v_j - r_{ij} q_i$
10: $\quad$ **end for**
11: **end for**

Modified Gram-Schmidt Process

MGS is just a rearrangement of CGS: In MGS outer loop is over rows, whereas in CGS outer loop is over columns

In exact arithmetic, MGS and CGS are identical

MGS is better numerically since when we compute $r_{ij} = q_i^T v_j$, components of $q_1, \ldots, q_{i-1}$ have already been removed from v_j

Whereas in CGS we compute $r_{ij} = q_i^T a_{(:,j)}$, which involves some extra "contamination" from components of $q_1, \ldots, q_{i-1}$ in $a_{(:,j)}$[3]

[3] The q_i's are not exactly orthogonal in finite precision arithmetic

Operation Count

Work in MGS is dominated by lines 8 and 9, the innermost loop:

$$\begin{aligned} r_{ij} &= q_i^T v_j \\ v_j &= v_j - r_{ij} q_i \end{aligned}$$

First line requires m multiplications, $m-1$ additions; second line requires m multiplications, m subtractions

Hence $\sim 4m$ operations per single inner iteration

Hence total number of operations is asymptotic to

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} 4m \sim 4m \sum_{i=1}^{n} i \sim 2mn^2$$

Householder Triangularization

The other main method for computing QR factorizations is Householder[4] triangularization

Householder algorithm is more numerically stable and more efficient than Gram-Schmidt

But Gram-Schmidt allows us to build up orthogonal basis for successive spaces spanned by columns of A

$$\mathrm{span}\{a_{(:,1)}\}, \mathrm{span}\{a_{(:,1)}, a_{(:,2)}\}, \ldots$$

which can be important in some cases

[4] Alston Householder, 1904–1993, American numerical analyst

Householder Triangularization

Householder idea: Apply a succession of orthogonal matrices $Q_k \in \mathbb{R}^{m \times m}$ to A to compute upper triangular matrix R

$$Q_n \cdots Q_2 Q_1 A = R$$

Hence we obtain the full QR factorization $A = QR$, where $Q \equiv Q_1^T Q_2^T \ldots Q_n^T$

Householder Triangularization

In 1958, Householder proposed a way to choose Q_k to introduce zeros below diagonal in col. k while preserving previous zeros

$$\begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{Q_2} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{Q_3} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$A \qquad Q_1 A \qquad Q_2 Q_1 A \qquad Q_3 Q_2 Q_1 A$$

This is achieved by Householder reflectors

Householder Reflectors

We choose

$$Q_k = \begin{bmatrix} \mathrm{I} & 0 \\ 0 & F \end{bmatrix}$$

where

- $\mathrm{I} \in \mathbb{R}^{(k-1)\times(k-1)}$
- $F \in \mathbb{R}^{(m-k+1)\times(m-k+1)}$ is a Householder reflector

The I block ensures the first $k-1$ rows are unchanged

F is an orthogonal matrix that operates on the bottom $m-k+1$ rows

(Note that F orthogonal $\implies$ Q_k orthogonal)

Householder Reflectors

Let $x \in \mathbb{R}^{m-k+1}$ denote entries $k, \ldots, m$ of of the k^{th} column

We have two requirements for F:

1. F is orthogonal, so must have $\|Fx\|_2 = \|x\|_2$
2. Only the first entry of Fx should be non-zero

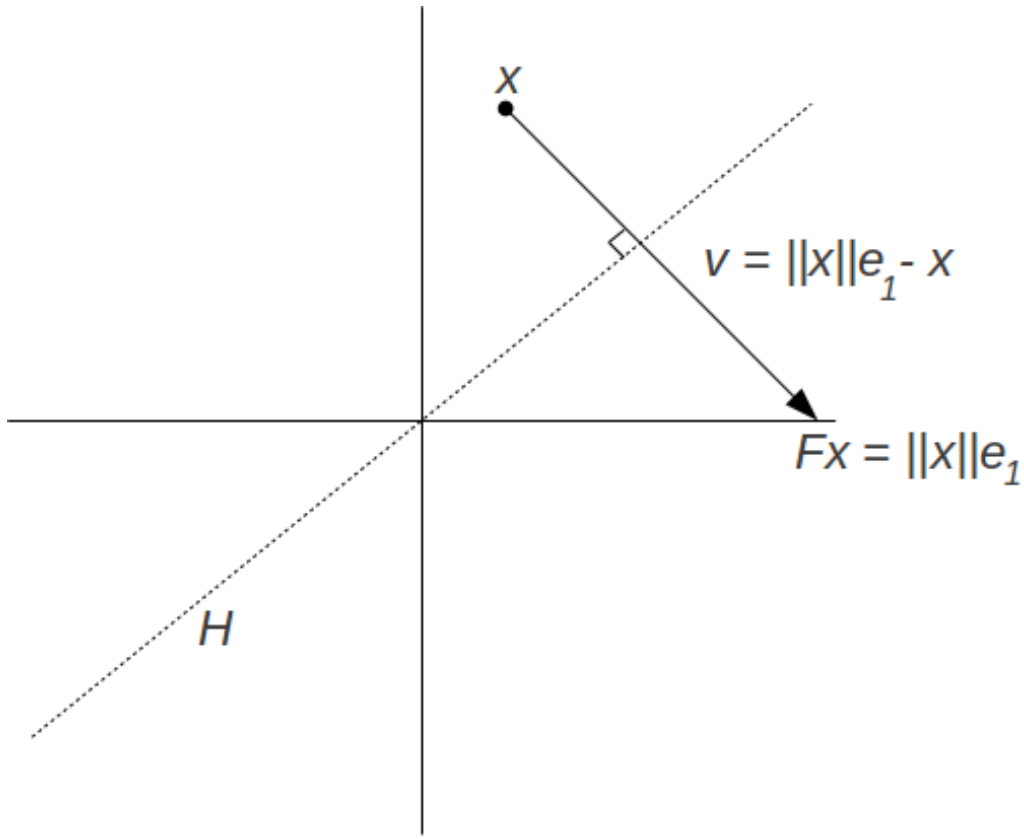
Hence we must have

$$x = \begin{bmatrix} \times \\ \times \\ \vdots \\ \times \end{bmatrix} \longrightarrow Fx = \begin{bmatrix} \|x\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|x\|_2 e_1$$

Question: How can we achieve this?

Householder Reflectors

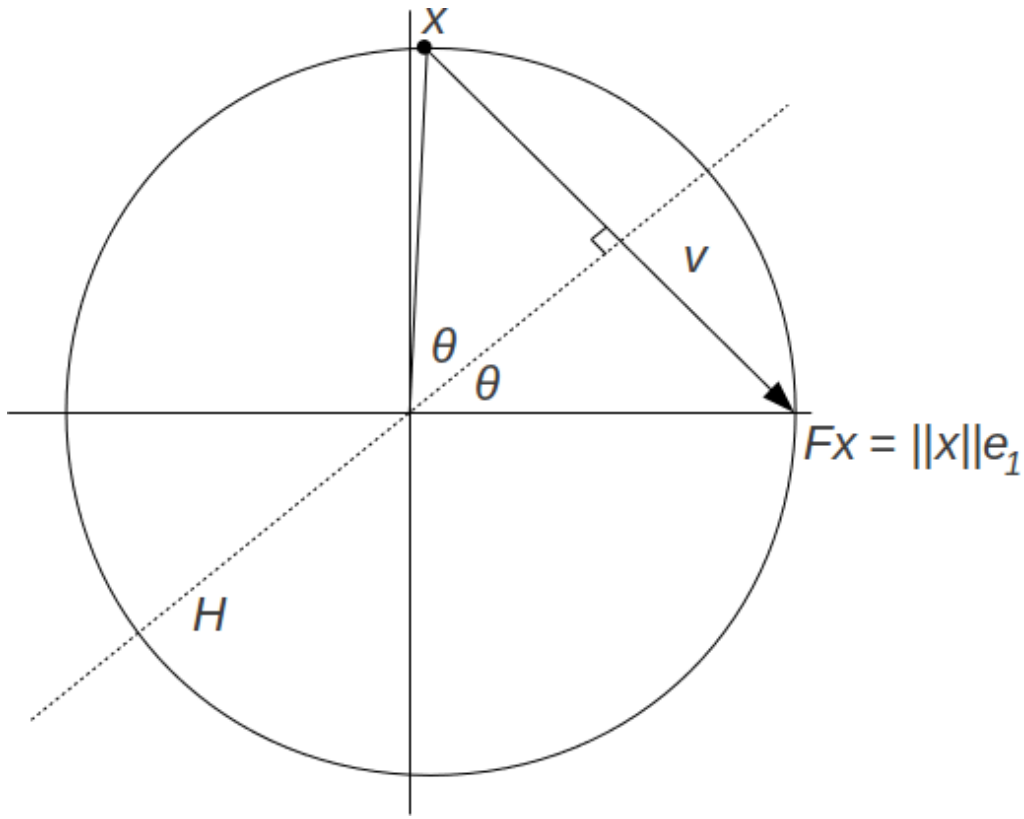
We can see geometrically that this can be achieved via reflection across a subspace H



Here H is the subspace orthogonal to $v \equiv \|x\|e_1 - x$, and the key point is that H "bisects" v

Householder Reflectors

We see that this bisection property is because x and Fx both live on the hypersphere centered at the origin with radius $\|x\|_2$



Householder Reflectors

Next, we need to determine the matrix F which maps x to $\|x\|_2 e_1$

F is closely related to the orthogonal projection of x onto H, since that projection takes us "half way" from x to $\|x\|_2 e_1$

Hence we first consider orthogonal projection onto H, and subsequently derive F

Recall that the orthogonal projection of a onto b is given by $\frac{(a \cdot b)}{\|b\|^2} b$, or equivalently $\frac{bb^T}{b^T b} a$

Hence we can see that $\frac{bb^T}{b^T b}$ is a rank 1 projection matrix

Householder Reflectors

Let $P_H \equiv \mathrm{I} - \frac{vv^T}{v^T v}$

Then $P_H x$ gives "x minus the orthogonal projection of x onto v"

We can show that $P_H x$ is in fact the orthogonal projection of x onto H, since it satisfies:

- $P_H x \in H$
  $(v^T P_H x = v^T x - v^T \frac{vv^T}{v^T v} x = v^T x - \frac{v^T v}{v^T v} v^T x = 0)$
- The projection error $x - P_H x$ is orthogonal to H
  $(x - P_H x = x - x + \frac{vv^T}{v^T v} x = \frac{v^T x}{v^T v} v$, parallel to $v)$

Householder Reflectors

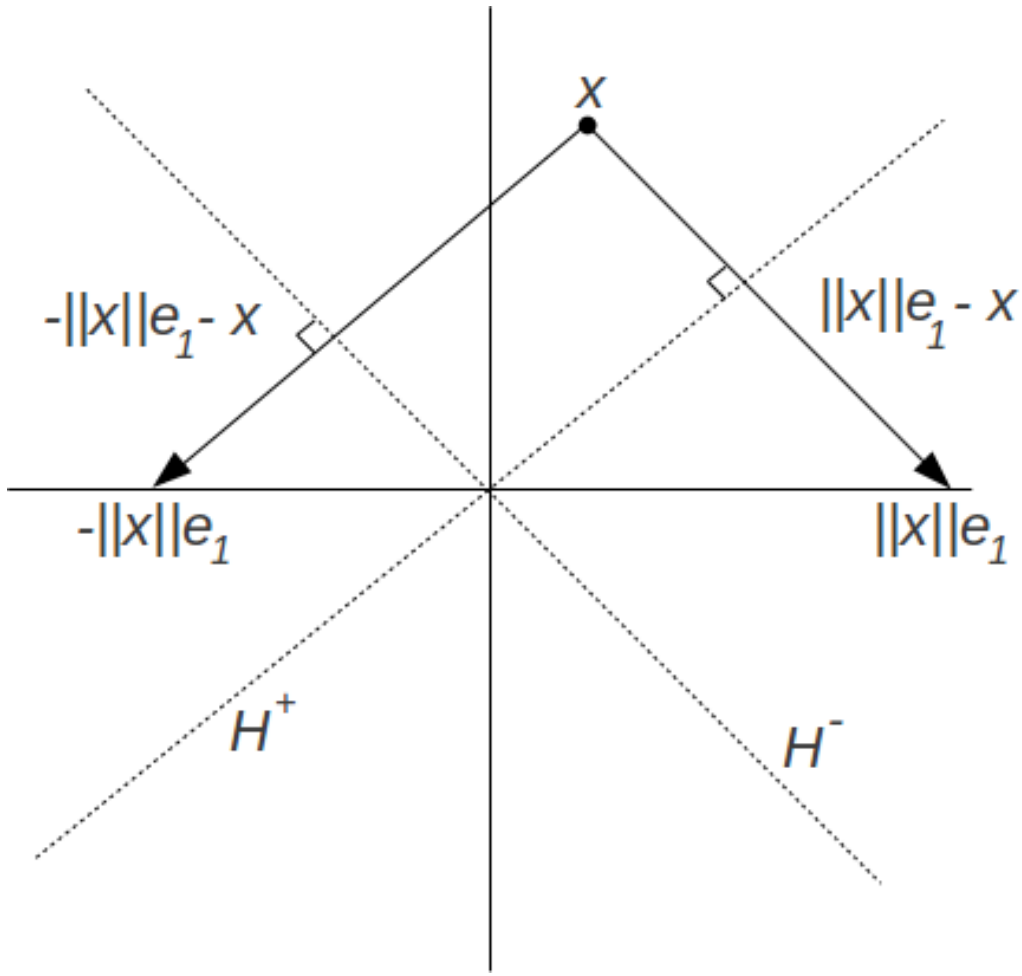
But recall that F should reflect across H rather than project onto H

Hence we obtain F by going "twice as far" in the direction of v compared to P_H, i.e. $F = \mathrm{I} - 2\frac{vv^T}{v^T v}$

Exercise: Show that F is an orthogonal matrix, i.e. that $F^T F = \mathrm{I}$

Householder Reflectors

But in fact, we can see that there are two Householder reflectors that we can choose from[5]



[5] This picture is "not to scale"; H^- should bisect $-\|x\|e_1 - x$

Householder Reflectors

In practice, it's important to choose the "better of the two reflectors"

If x and $\|x\|_2 e_1$ (or x and $-\|x\|_2 e_1$) are close, we could obtain loss of precision due to cancellation (cf. Unit 0) when computing v

To ensure x and its reflection are well separated we should choose the reflection to be $-\mathrm{sign}(x_1)\|x\|_2 e_1$ (more details on next slide)

Therefore, we want v to be $v = -\mathrm{sign}(x_1)\|x\|_2 e_1 - x$

But note that the sign of v does not affect F, hence we scale v by -1 to get an equivalent formula without "minus signs":

$$v = \mathrm{sign}(x_1)\|x\|_2 e_1 + x$$

Householder Reflectors

Let's compare the two options for v in the potentially problematic case when $x \approx \pm\|x\|_2 e_1$, i.e. when $|x_1| \approx \|x\|_2$:

$$v_{\text{bad}} \equiv -\text{sign}(x_1)\|x\|_2 e_1 + x$$

$$v_{\text{good}} \equiv \text{sign}(x_1)\|x\|_2 e_1 + x$$

$$\begin{aligned}
\|v_{\text{bad}}\|_2^2 &= \|-\text{sign}(x_1)\|x\|_2 e_1 + x\|_2^2 \\
&= (-\text{sign}(x_1)\|x\|_2 + x_1)^2 + \|x_{2:(m-k+1)}\|_2^2 \\
&= (-\text{sign}(x_1)\|x\|_2 + \text{sign}(x_1)|x_1|)^2 + \|x_{2:(m-k+1)}\|_2^2 \\
&\approx 0
\end{aligned}$$

$$\begin{aligned}
\|v_{\text{good}}\|_2^2 &= \|\text{sign}(x_1)\|x\|_2 e_1 + x\|_2^2 \\
&= (\text{sign}(x_1)\|x\|_2 + x_1)^2 + \|x_{2:(m-k+1)}\|_2^2 \\
&= (\text{sign}(x_1)\|x\|_2 + \text{sign}(x_1)|x_1|)^2 + \|x_{2:(m-k+1)}\|_2^2 \\
&\approx (2\text{sign}(x_1)\|x\|_2)^2
\end{aligned}$$

Householder Reflectors

Recall that v is computed from two vectors of magnitude $\|x\|_2$

The argument above shows that with v_{bad} we can get $\|v\|_2 \ll \|x\|_2$
$\Longrightarrow$ "loss of precision due to cancellation" is possible

In contrast, with v_{good} we always have $\|v_{\text{good}}\|_2 \geq \|x\|_2$, which rules out loss of precision due to cancellation

Householder Triangularization

We can now write out the Householder algorithm:

1: **for** $k = 1 : n$ **do**
2: $\quad x = a_{(k:m,k)}$
3: $\quad v_k = \operatorname{sign}(x_1)\|x\|_2 e_1 + x$
4: $\quad v_k = v_k / \|v_k\|_2$
5: $\quad a_{(k:m,k:n)} = a_{(k:m,k:n)} - 2v_k(v_k^T a_{(k:m,k:n)})$
6: **end for**

This replaces A with R and stores $v_1, \ldots, v_n$

Note that we don't divide by $v_k^T v_k$ in line 5 (as in the formula for F) since we normalize v_k in line 4

Householder algorithm requires $\sim 2mn^2 - \frac{2}{3}n^3$ operations[6]

[6] Compared to $2mn^2$ for Gram-Schmidt

Householder Triangularization

Note that we don't explicitly form Q

We can use the vectors $v_1, \ldots, v_n$ to compute Q in a post-processing step

Recall that

$$Q_k = \begin{bmatrix} \mathrm{I} & 0 \\ 0 & F \end{bmatrix}$$

and $Q \equiv (Q_n \cdots Q_2 Q_1)^T = Q_1^T Q_2^T \cdots Q_n^T$

Also, the Householder reflectors are symmetric (refer to the definition of F), so $Q = Q_1^T Q_2^T \cdots Q_n^T = Q_1 Q_2 \cdots Q_n$

Householder Triangularization

Hence, we can evaluate $Qx = Q_1 Q_2 \cdots Q_n x$ using the v_k:

1: **for** $k = n : -1 : 1$ **do**
2: $\quad x_{(k:m)} = x_{(k:m)} - 2v_k(v_k^T x_{(k:m)})$
3: **end for**

Question: How can we use this to form the matrix Q?

Householder Triangularization

Answer: Compute Q via Qe_i, $i = 1, \ldots, m$

Similarly, compute $\hat{Q}$ (reduced QR factor) via Qe_i, $i = 1, \ldots, n$

However, often not necessary to form Q or $\hat{Q}$ explicitly, e.g. to solve $Ax \simeq b$ we only need the product $Q^T b$

Note the product $Q^T b = Q_n \cdots Q_2 Q_1 b$ can be evaluated as:

1: **for** $k = 1 : n$ **do**
2: $\quad b_{(k:m)} = b_{(k:m)} - 2v_k(v_k^T b_{(k:m)})$
3: **end for**

Singular Value Decomposition (SVD)

Singular Value Decomposition

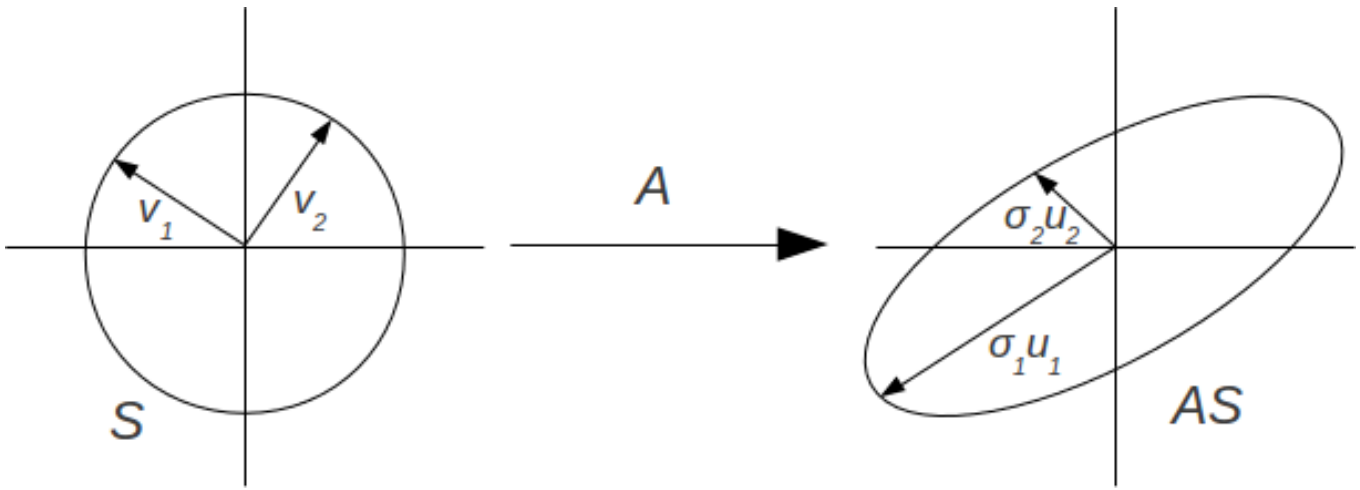
The Singular Value Decomposition (SVD) is a very useful matrix factorization

Motivation for SVD: image of the unit sphere, S, from any $m \times n$ matrix is a hyperellipse

A hyperellipse is obtained by stretching the unit sphere in $\mathbb{R}^m$ by factors $\sigma_1, \ldots, \sigma_m$ in orthogonal directions $u_1, \ldots, u_m$

Singular Value Decomposition

For $A \in \mathbb{R}^{2\times 2}$, we have



Singular Value Decomposition

Based on this picture, we make some definitions:

- Singular values: $\sigma_1, \sigma_2, \ldots, \sigma_n \geq 0$ (we typically assume $\sigma_1 \geq \sigma_2 \geq \ldots$)
- Left singular vectors: $\{u_1, u_2, \ldots, u_n\}$, unit vectors in directions of principal semiaxes of AS
- Right singular vectors: $\{v_1, v_2, \ldots, v_n\}$, preimages of the u_i so that $Av_i = \sigma_i u_i$, $i = 1, \ldots, n$

(The names "left" and "right" come from the formula for the SVD below)

Singular Value Decomposition

The key equation above is that

$$Av_i = \sigma_i u_i, \quad i = 1, \ldots, n$$

Writing this out in matrix form we get

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} \left[\begin{array}{c|c|c|c} v_1 & v_2 & \cdots & v_n \end{array}\right] = \left[\begin{array}{c|c|c|c} u_1 & u_2 & \cdots & u_n \end{array}\right] \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}$$

Or more compactly:

$$AV = \widehat{U}\widehat{\Sigma}$$

Singular Value Decomposition

Here

- $\widehat{\Sigma} \in \mathbb{R}^{n \times n}$ is diagonal with non-negative, real entries
- $\widehat{U} \in \mathbb{R}^{m \times n}$ with orthonormal columns
- $V \in \mathbb{R}^{n \times n}$ with orthonormal columns

Therefore V is an orthogonal matrix ($V^T V = VV^T = \mathrm{I}$), so that we have the reduced SVD for $A \in \mathbb{R}^{m \times n}$:

$$A = \widehat{U}\widehat{\Sigma}V^T$$

Singular Value Decomposition

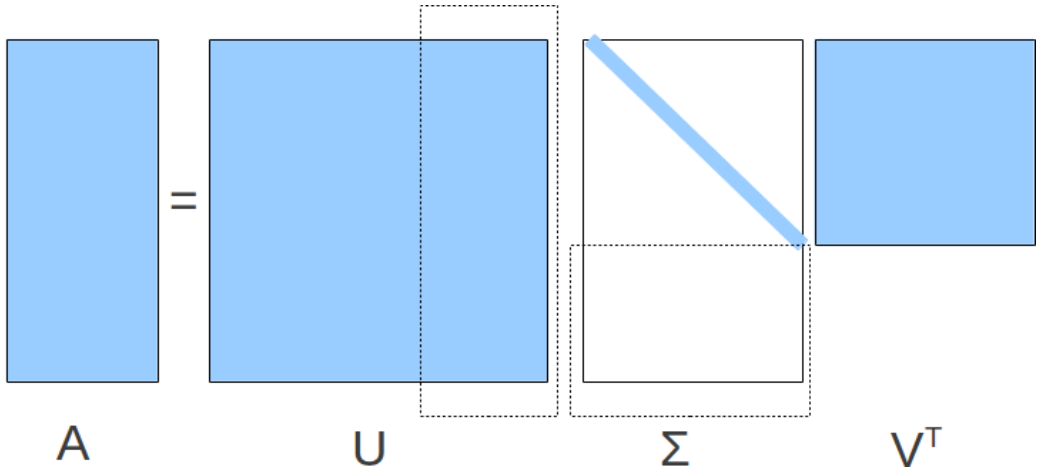
Just as with QR, we can pad the columns of $\widehat{U}$ with $m - n$ arbitrary orthogonal vectors to obtain $U \in \mathbb{R}^{m \times m}$

We then need to "silence" these arbitrary columns by adding rows of zeros to $\widehat{\Sigma}$ to obtain Σ

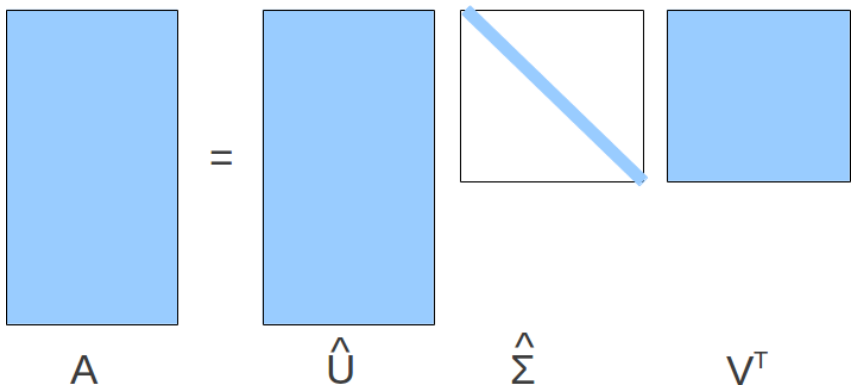
This gives the full SVD for $A \in \mathbb{R}^{m \times n}$:

$$A = U \Sigma V^T$$

Full vs Reduced SVD



Full SVD



Reduced SVD

Singular Value Decomposition

Theorem: Every matrix $A \in \mathbb{R}^{m \times n}$ has a full singular value decomposition. Furthermore:

- The σ_j are uniquely determined
- If A is square and the σ_j are distinct, the $\{u_j\}$ and $\{v_j\}$ are uniquely determined up to sign

Singular Value Decomposition

This theorem justifies the statement that the image of the unit sphere under any $m \times n$ matrix is a hyperellipse

Consider $A = U\Sigma V^T$ (full SVD) applied to the unit sphere, S, in $\mathbb{R}^n$:

1. The orthogonal map V^T preserves S
2. Σ stretches S into a hyperellipse aligned with the canonical axes e_j
3. U rotates or reflects the hyperellipse without changing its shape

SVD in Matlab

Matlab's `svd` function computes the full SVD of a matrix

```
>> A = rand(4,2);
>> [U,S,V] = svd(A)

 U =

      0.4775   0.5908   0.6085   -0.2293
      0.4463   0.1122   -0.1281   0.8785
      0.3541   -0.7954   0.4919   -0.0066
      0.6689   -0.0756   -0.6094   -0.4190

 S =
 1.7335   0
 0        0.8555
 0        0
 0        0

 V =
 0.6404   -0.7681
 0.7681   0.6404
```

SVD in Matlab

As with QR, `svd(A,0)` returns the reduced SVD

```
>> [U,S,V] = svd(A,0)

 U =
      0.4775   0.5908
      0.4463   0.1122
      0.3541   -0.7954
      0.6689   -0.0756

 S =
 1.7335   0
 0        0.8555

 V =
 0.6404   -0.7681
 0.7681   0.6404

>> norm(A - U*S*V')

ans =

    4.2123e-16
```

Matrix Properties via the SVD

- The rank of A is r, the number of nonzero singular values[7]

Proof: In the full SVD $A = U\Sigma V^T$, U and V^T have full rank, hence it follows from linear algebra that $\text{rank}(A) = \text{rank}(\Sigma)$

- $\text{image}(A) = \text{span}\{u_1, \ldots, u_r\}$ and $\text{null}(A) = \text{span}\{v_{r+1}, \ldots, v_n\}$

Proof: This follows from $A = U\Sigma V^T$ and

$$\begin{aligned} \text{image}(\Sigma) &= \text{span}\{e_1, \ldots, e_r\} \in \mathbb{R}^m \\ \text{null}(\Sigma) &= \text{span}\{e_{r+1}, \ldots, e_n\} \in \mathbb{R}^n \end{aligned}$$

[7] This also gives us a good way to define rank in finite precision: the number of singular values larger than some (small) tolerance

Matrix Properties via the SVD

- $\|A\|_2 = \sigma_1$

Proof: Recall that $\|A\|_2 \equiv \max_{\|v\|_2=1} \|Av\|_2$. Geometrically, we see that $\|Av\|_2$ is maximized if $v = v_1$ and $Av = \sigma_1 u_1$.

- The singular values of A are the square roots of the eigenvalues of $A^T A$ or AA^T

Proof: (Analogous for AA^T)

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) = V\Sigma U^T U\Sigma V^T = V(\Sigma^T \Sigma)V^T,$$

hence $(A^T A)V = V(\Sigma^T \Sigma)$, or $(A^T A)v_{(:,j)} = \sigma_j^2 v_{(:,j)}$

Matrix Properties via the SVD

The pseudoinverse, A^+, can be defined more generally in terms of the SVD

Define pseudoinverse of a scalar σ to be $1/\sigma$ if $\sigma \neq 0$ and zero otherwise

Define pseudoinverse of a (possibly rectangular) diagonal matrix as transpose of the matrix and taking pseudoinverse of each entry

Pseudoinverse of $A \in \mathbb{R}^{m \times n}$ is defined as

$$A^+ = V \Sigma^+ U^T$$

A^+ exists for any matrix A, and it captures our definitions of pseudoinverse from Unit I

Matrix Properties via the SVD

We generalize the condition number to rectangular matrices via the definition $\kappa(A) = \|A\|\|A^+\|$

We can use the SVD to compute the 2-norm condition number:

- $\|A\|_2 = \sigma_{\text{max}}$
- Largest singular value of A^+ is $1/\sigma_{\text{min}}$ so that $\|A^+\|_2 = 1/\sigma_{\text{min}}$

Hence $\kappa(A) = \sigma_{\text{max}}/\sigma_{\text{min}}$

Matrix Properties via the SVD

These results indicate the importance of the SVD, both theoretically and as a computational tool

Algorithms for calculating the SVD are an important topic in Numerical Linear Algebra, but outside scope of this course

Requires $\sim 4mn^2 - \frac{4}{3}n^3$ operations

For more details on algorithms, see Trefethen & Bau, or Golub & van Loan

Low-Rank Approximation via the SVD

One of the most useful properties of the SVD is that it allows us to obtain an optimal low-rank approximation to A

See Lecture: We can recast SVD as

$$A = \sum_{j=1}^{r} \sigma_j u_j v_j^T$$

Follows from writing Σ as sum of r matrices, Σ_j, where $\Sigma_j \equiv \mathrm{diag}(0, \ldots, 0, \sigma_j, 0, \ldots, 0)$

Each $u_j v_j^T$ is a rank one matrix: each column is a scaled version of u_j

Low-Rank Approximation via the SVD

Theorem: For any $0 \leq \nu \leq r$, let $A_\nu \equiv \sum_{j=1}^{\nu} \sigma_j u_j v_j^T$, then

$$\|A - A_\nu\|_2 = \min_{B \in \mathbb{R}^{m \times n},\ \mathrm{rank}(B) \leq \nu} \|A - B\|_2 = \sigma_{\nu+1}$$

That is:

- A_ν gives us the closest rank ν matrix to A, measured in the 2-norm
- The error in A_ν is given by the first *omitted* singular value

Low-Rank Approximation via the SVD

A similar result holds in the Frobenius norm:

$$\|A - A_\nu\|_F = \min_{B \in \mathbb{R}^{m \times n},\ \mathrm{rank}(B) \leq \nu} \|A - B\|_F = \sqrt{\sigma_{\nu+1}^2 + \cdots + \sigma_r^2}$$

Low-Rank Approximation via the SVD

These theorems indicate that the SVD is an effective way to *compress* data encapsulated by a matrix!

If singular values of A decay rapidly, can approximate A with few rank one matrices (only need to store σ_j, u_j, v_j for $j = 1, \ldots, \nu$)

Matlab example: Image compression via the SVD