VE472 Lecture 3

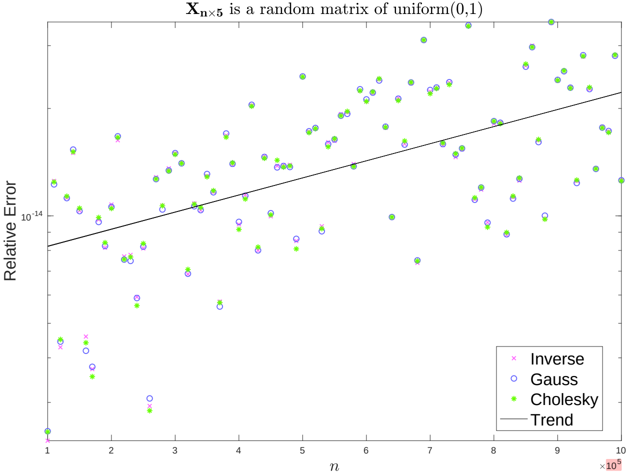
Jing Liu

UM-SJTU Joint Institute

Summer

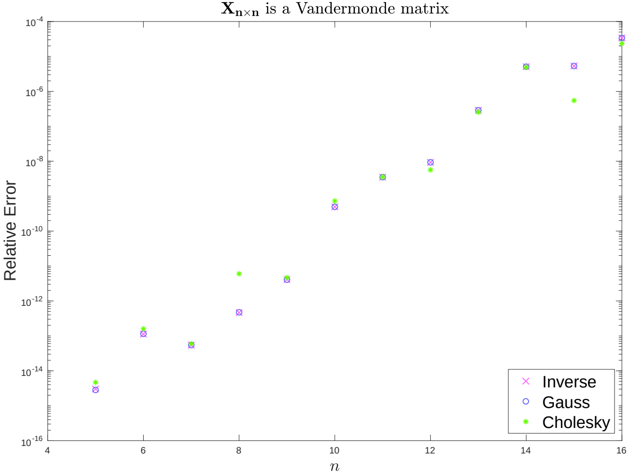
Random full rank matrix

linearly independent rows or/and linearlt independent columns



n很大时， 几种方法差不多, 且error不大。但是error都在逐渐增大

Vandermonde matrix

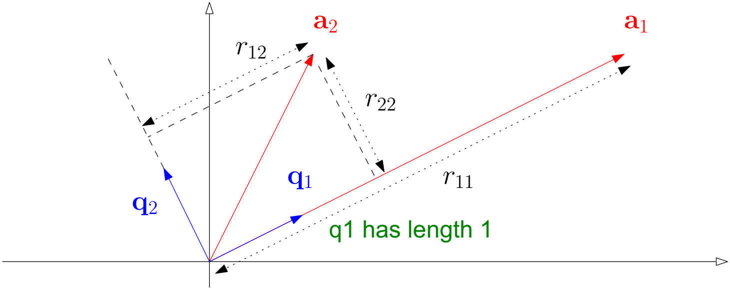


16x16 matrix

roughly share the same accuracy. But they loose precision very quickly.

Theorem 0.1 QR decomposition

Let $\mathbf{A}$ be a matrix of $m \times n$ with $m \geq n$. Suppose $\mathbf{A}$ is full rank. Then there is a matrix $\mathbf{Q}$ of $m \times n$ with orthonormal columns, $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$, and an upper triangular matrix $\mathbf{R}$ of $n \times n$ with positive diagonals $r_{ii} > 0$ such that $\mathbf{A} = \mathbf{QR}$.

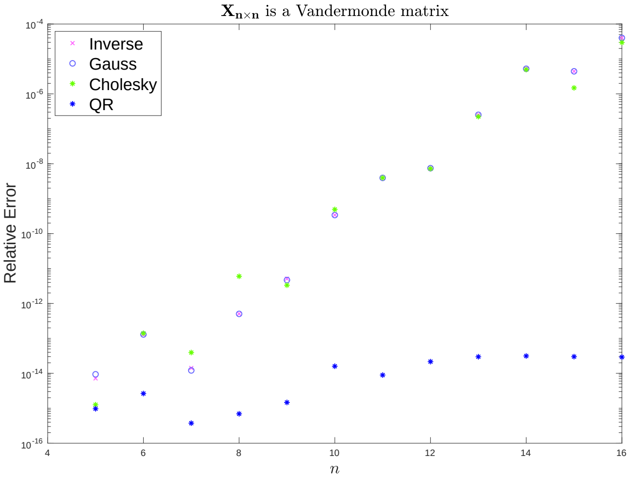


In general, the theorem replies on the idea of projection.

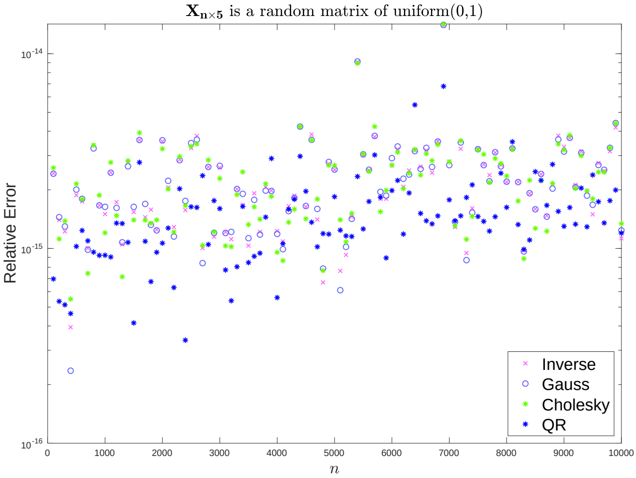
Q: Why is this theorem useful in terms of dealing with a big data matrix $\mathbf{X}$?

see notes

Vandermonde matrix



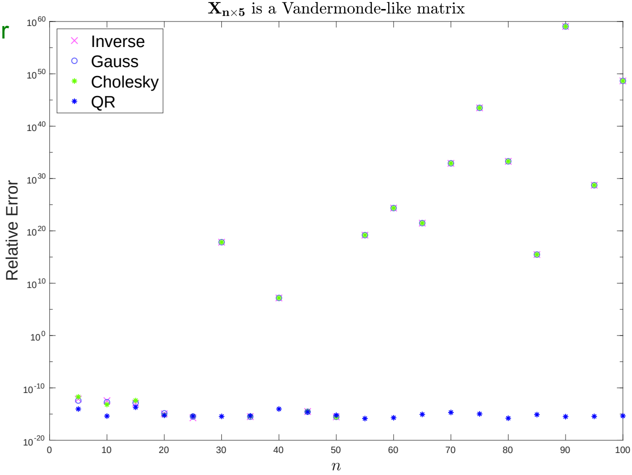
Random full rank matrix



In most of the times, QR is more stable

Vandermonde-like matrix

too big error



- QR is no question the slowest!

```
>> clear all
>> n = 1000; X = rand(n, 100); y = randn(n, 1);
tic; for i = 1:10000
    XX = transpose(X)*X; yy = transpose(X)*y;
    [L, U] = lu(XX); bhat = U\(L\yy);
end; toc;
tic; for i = 1:10000
    XX = transpose(X)*X; yy = transpose(X)*y;
    C = chol(XX, 'lower');
    bhat =  transpose(C)\(C\yy);
end; toc;
tic ;for i = 1:10000
    [Q, R] = qr(X); bhat =  R\(transpose(Q)*y);
end; toc;
```

```
Elapsed time is 3.266948 seconds.
Elapsed time is 2.457167 seconds.
Elapsed time is 44.321296 seconds.
```

stable but slow

Theorem 0.2 (Singular Value Decomposition)

Let $\mathbf{A}$ *be a rank* k *matrix of* $m \times n$ *with* $m \geq n$, *then we have* $\sigma_1 \geq \cdots \geq \sigma_k > 0$

k linearly independent columns

$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$

eigenvectors of X^TX

左上角最大

$$= \left[\begin{array}{ccc|ccc} \mathbf{u}_1 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} & \cdots & \mathbf{u}_m \end{array}\right] \left[\begin{array}{ccc|c} \sigma_1 & \cdots & 0 & \\ \vdots & \ddots & \vdots & \mathbf{0}_{k\times(n-k)} \\ 0 & \cdots & \sigma_k & \\ \hline & \mathbf{0}_{(m-k)\times k} & & \mathbf{0}_{(m-k)\times(n-k)} \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{v}_k^{\mathrm{T}} \\ \hline \mathbf{v}_{k+1}^{\mathrm{T}} \\ \vdots \\ \mathbf{v}_n^{\mathrm{T}} \end{array}\right]$$

where $\mathbf{U}$ *and* $\mathbf{V}$ *are orthogonal matrices of size* $m \times m$ *and* $n \times n$, *respectively.*

Q: Why is this theorem useful in terms of dealing with a big data matrix $\mathbf{X}$?

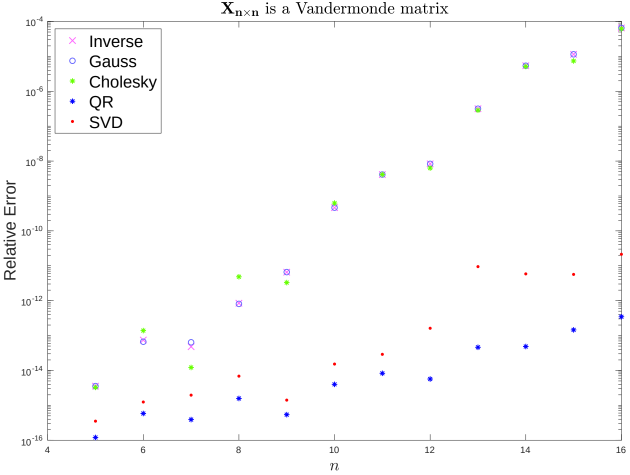
- SVD is faster than QR, but a magnitude slower than LU or Cholesky .

```
>> n = 1000; X = rand(n, 100); y = randn(n, 1);
tic;for i = 1:10000
    XX = transpose(X)*X; yy = transpose(X)*y;
    C = chol(XX, 'lower');
    bhat = transpose(C)\(C\yy);
end; toc;
tic;for i = 1:10000
    [Q, R] = qr(X); bhat = R\(transpose(Q)*y);
end; toc;
tic;for i = 1:10000                compact svd
    [U, S, V] = svd(X, 'econ');
    s = diag(S); s = 1./s; inverse of S
    bhat = V*diag(s)*transpose(U)*y;
end; toc;
```

```
Elapsed time is 2.521359 seconds.
Elapsed time is 45.891408 seconds.
Elapsed time is 19.551785 seconds.
```

faster than QR, slower than LU

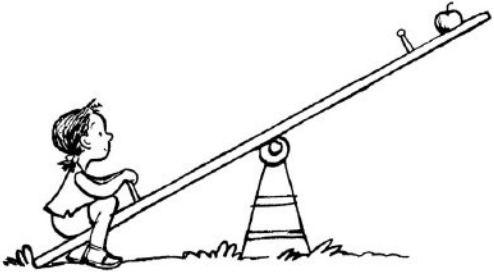
Vandermonde matrix



- Recall the stability of linear model can be studied using the eigenvalues of

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}$$

having small eigenvalues indicate columns are nearly linearly dependent.



- Using SVD gives us the stability of our model given the data as a byproduct.

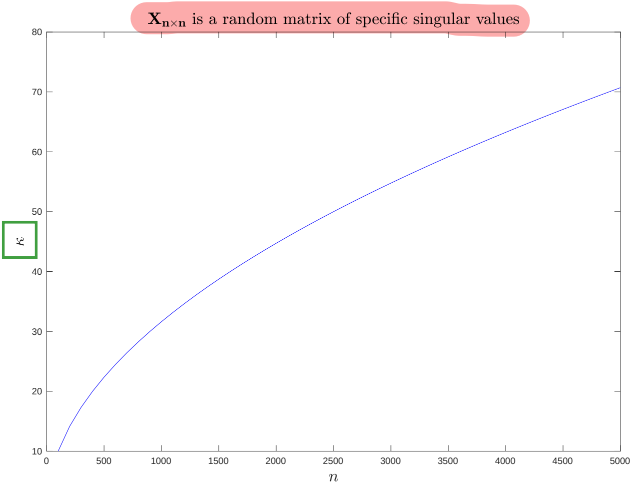
capper

$$\kappa = \frac{\max \sigma_i}{\min \sigma_i}$$

samllest non zero sv

Increasing κ as n increases by construction

true random

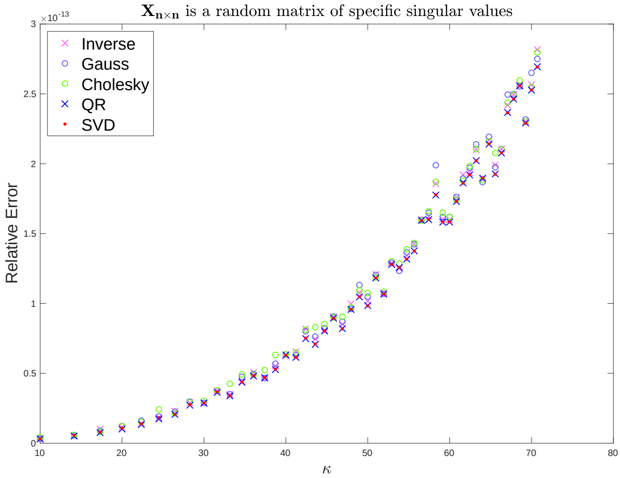


n变大，sv变多，k更容易变大

Increasing relative error

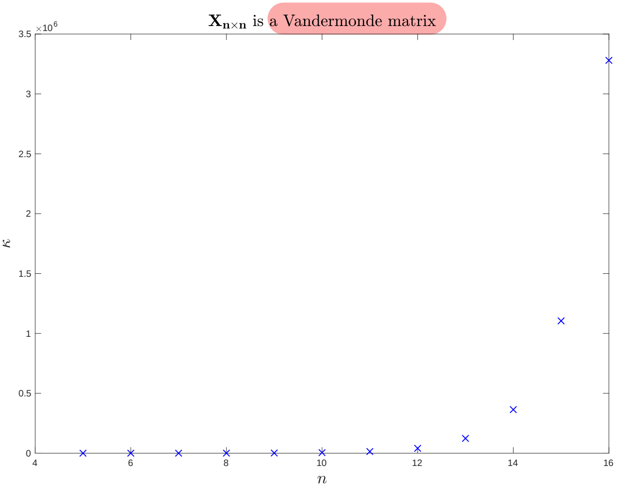
k变大， relative error变大（numerical error cumulate -> unstable）

虽然在增大但是k<100依然可以接受

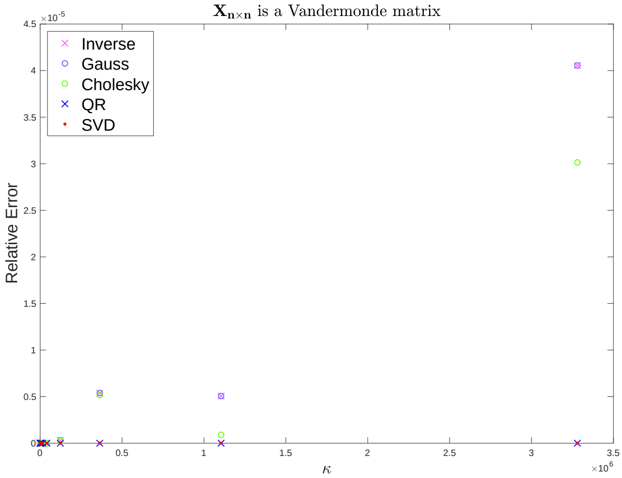


Really big κ

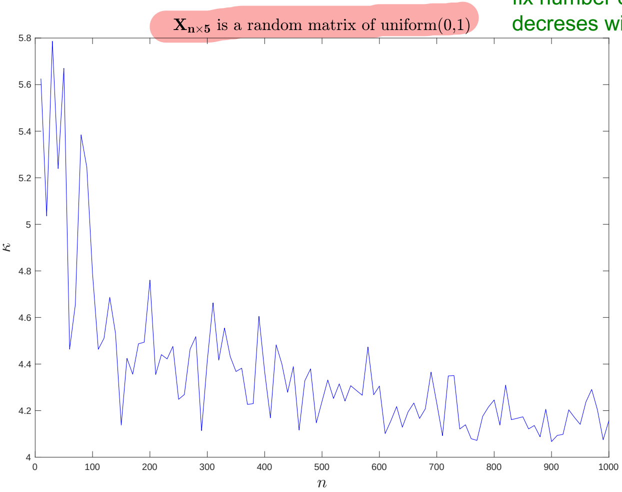
extreme case



Really big κ really big relative error



Decreasing κ as n increases for a fixed k

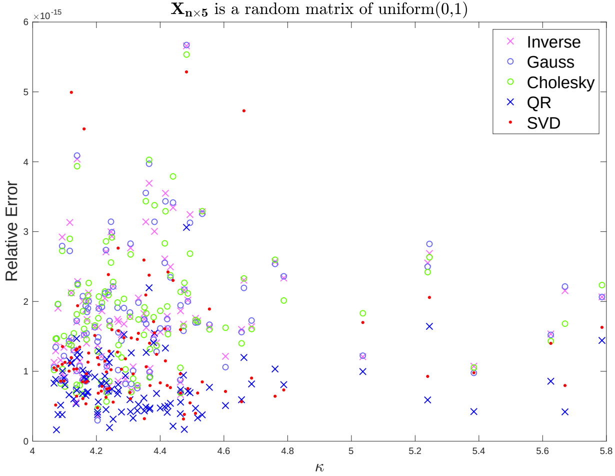


fix number of columns, k
decreses with incresing n

column不变，
data越多，
越stable

Only having really big κ is problematic

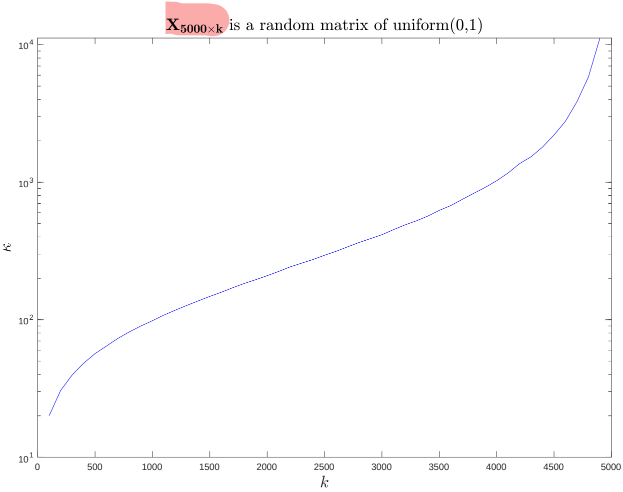
small error



small k

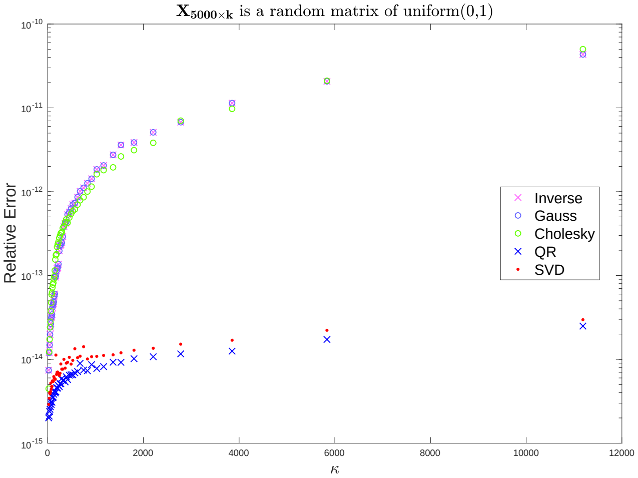
Increasing rapidly κ as k increases for a fixed n

big error



big column number

A more complex data is more problematic than a large data



large number of column may lead to large error

- So far we have considered the data matrix $\mathbf{X}$ that is full rank when solving

$$\arg\min_{\mathbf{b}\in\mathbb{R}^{k+1}} \|\mathbf{y}-\mathbf{X}\mathbf{b}\|^2$$

Q: What happens when $\mathbf{A}$ is rank deficient or “close" to being rank deficient?

- Such problems often arise with big data , e.g. extracting signals from noisy data, digital image restoration as well as big prediction or classification.

Theorem 0.3

Let $\sigma_{min} > 0$ denote the smallest singular value of $\mathbf{X}$ and $\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}$ be the singular value decomposition of $\mathbf{X}$. If $\mathbf{b}$ minimises $\|\mathbf{y}-\mathbf{X}\mathbf{b}\|$, then

can have large norm
change 'y' a bit, 'b' may change a lot

$$\|\mathbf{b}\| \geq \frac{\left|\mathbf{u}^{\mathrm{T}}\cancel{\mathbf{b}}\right|}{\sigma_{min}}$$

u^Ty

where $\mathbf{u}$ is the last column of $\mathbf{U}$. Furthermore changing $\mathbf{y}$ to $\mathbf{y}+\delta\mathbf{y}$ can induce a change of $\delta\mathbf{b}$ to $\mathbf{b}$, where $\|\delta\mathbf{b}\|$ is as large as $\|\delta\mathbf{y}\|/\sigma_{min}$.

Q: What does the last theorem tell us if $\mathbf{A}$ is nearly rank deficient?

Theorem 0.4

Let $\mathbf{X}$ be a rank r matrix of $n \times (k+1)$ with $n \geq k+1$. If $r < k+1$, then there is an $\cancel{n-r}$ dimensional set of vectors $\mathbf{b} \in \mathbb{R}^{k+1}$ that solve the following

k+1-r

$$\arg\min_{\mathbf{b}\in\mathbb{R}^{k+1}} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|^2 \qquad \textit{where} \quad \mathbf{y} \in \mathbb{R}^n$$

Furthermore, the singular value decomposition of $\mathbf{X}$ can be written as

$$\mathbf{A} = \begin{bmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{bmatrix} \begin{bmatrix}\mathbf{\Sigma}_{r\times r} & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix} \begin{bmatrix}\mathbf{V}_1 & \mathbf{V}_2\end{bmatrix}^{\mathrm{T}}, \quad \textit{where } \mathbf{U}_1 \textit{ and } \mathbf{V}_1 \textit{ have } r \textit{ columns},$$

and all the minimisers take the following form

$$\mathbf{b} = \mathbf{V}_1 \mathbf{\Sigma}_{r\times r}^{-1} \mathbf{U}_1^{\mathrm{T}} \mathbf{y} + \mathbf{V}_2 \mathbf{z}, \qquad \textit{for any } \mathbf{z} \in \mathbb{R}^r.$$

Q: Which of those minimisers shall we use as the best $\mathbf{b}$?