Outline

1. Least Squares Data Fitting
2. Existence, Uniqueness, and Conditioning
3. Solving Linear Least Squares Problems
Measurement errors are inevitable in observational and experimental sciences.

Errors can be smoothed out by averaging over many cases, i.e., taking more measurements than are strictly necessary to determine parameters of system.

Resulting system is *overdetermined*, so usually there is no exact solution.

In effect, higher dimensional data are projected into lower dimensional space to suppress irrelevant detail.

Such projection is most conveniently accomplished by method of *least squares*.
For linear problems, we obtain *overdetermined* linear system $Ax = b$, with $m \times n$ matrix $A$, $m > n$.

System is better written $Ax \approx b$, since equality is usually not exactly satisfiable when $m > n$.

Least squares solution $x$ minimizes squared Euclidean norm of residual vector $r = b - Ax$,

$$\min_x \|r\|_2^2 = \min_x \|b - Ax\|_2^2$$
Least Squares Idea

Given \( b \in \mathbb{R}^m \), with \( m > n \), find:

\[
y := Ax = a_1x_1 + a_2x_2 + \cdots + a_nx_n \approx b
\]

\[
r := b - Ax = b - y
\]

Least squares:

\[
\text{Minimize } ||r||_2 = \left[ \sum_{i=1}^{m} (b_i - y_i)^2 \right]^{\frac{1}{2}}
\]

This system is overdetermined.

There are more equations than unknowns.
Least Squares Idea

With $m > n$, we have:

- Lots of data ($b \in \mathbb{R}^m$)
- A few model parameters ($x_1, x_2, \ldots, x_n$)
- A few candidate basis vectors ($a_1, a_2, \ldots, a_n$)
- Our estimate, $y = Ax$

The matrix $A$ is tall and thin.

$$y \approx Ax \approx b$$
The vector $\mathbf{y}$ is the *orthogonal projection* of $\mathbf{b}$ onto $\text{span}(\mathbf{A})$.

The projection results in minimization of $\| \mathbf{r} \|_2$, which, as we shall see, is equivalent to having $\mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x} \perp \text{span}(\mathbf{A})$.
1D Projection

- Consider the 1D subspace of $\mathbb{R}^2$ spanned by $a_1$:
  
  $$\alpha a_1 \in \text{span}\{a_1\}.$$  

- The projection of a point $b \in \mathbb{R}^2$ onto $\text{span}\{a_1\}$ is the point on the line $y = \alpha a_1$ that is closest to $b$.

- To find the projection, we look for the value $\alpha$ that minimizes $||r|| = ||\alpha a_1 - b||$ in the 2-norm. (Other norms are also possible.)
1D Projection

- Minimizing the square of the residual with respect to $\alpha$, we have

$$\frac{d}{d\alpha} ||r||^2 =$$

$$= \frac{d}{d\alpha} (b - \alpha a_1)^T (b - \alpha a_1)$$

$$= \frac{d}{d\alpha} [b^T b + \alpha^2 a_1^T a_1 - 2\alpha a_1^T b]$$

$$= 2\alpha a_1^T a_1 - 2 a_1^T b = 0$$

- For this to be a minimum, we require the last expression to be zero, which implies

$$\alpha = \frac{a_1^T b}{a_1^T a_1}, \quad \Rightarrow \quad y = \alpha a_1 = \frac{a_1^T b}{a_1^T a_1} a_1.$$

- We see that $y$ points in the direction of $a_1$ and has magnitude that scales as $b$ (but not with $a_1$).
Projection in Higher Dimensions

• Here, we have basis coefficients $x_i$ written as $\mathbf{x} = [x_1 \ldots x_n]^T$.

• As before, we minimize the square of the norm of the residual

$$||\mathbf{r}||^2 = ||\mathbf{Ax - b}||^2$$

$$= (\mathbf{Ax - b})^T (\mathbf{Ax - b})$$

$$= \mathbf{b}^T \mathbf{b} - \mathbf{b}^T \mathbf{Ax} - (\mathbf{Ax})^T \mathbf{b} + \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x}$$

$$= \mathbf{b}^T \mathbf{b} + \mathbf{x}^T \mathbf{A}^T \mathbf{Ax} - 2 \mathbf{x}^T \mathbf{A}^T \mathbf{b}.$$

• As in the 1D case, we require stationarity with respect to all coefficients

$$\frac{d}{dx_i} ||\mathbf{r}||^2 = 0$$

• The first term is constant.

• The second and third are more complex.
Projection in Higher Dimensions

- Define $\mathbf{c} = \mathbf{A}^T \mathbf{b}$ and $H = \mathbf{A}^T \mathbf{A}$ such that

$$\mathbf{x}^T \mathbf{A}^T \mathbf{b} = \mathbf{x}^T \mathbf{c} = x_1 c_1 + x_2 c_2 + \ldots + x_n c_n.$$  

$$\mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{H} \mathbf{x} = \sum_{j=1}^{n} \sum_{k=1}^{n} x_k H_{kj} x_j$$

- Differentiating with respect to $x_i$,

$$\frac{d}{dx_i} (\mathbf{x}^T \mathbf{A}^T \mathbf{b}) = c_i = (\mathbf{A}^T \mathbf{b})_i,$$  

and

$$\frac{d}{dx_i} (\mathbf{x}^T \mathbf{H} \mathbf{x}) = \sum_{j=1}^{n} H_{ij} x_j + \sum_{k=1}^{n} x_k H_{ki}$$

$$= 2 \sum_{j=1}^{n} H_{ij} x_j = 2 (\mathbf{H} \mathbf{x})_i.$$
Projection in Higher Dimensions

• From the preceding pages, the minimum is realized when
  
  $$0 = \frac{d}{dx_i} (x^T A^T A x - 2x^T A^T b) = 2 (A^T A x - A^T b)_i, \quad i = 1, \ldots, n$$

• Or, in matrix form:
  
  $$x = (A^T A)^{-1} A^T b.$$

• As in the 1D case, our projection is
  
  $$y = A x = A (A^T A)^{-1} A^T b.$$

• $y$ has units and length that scale with $b$, but it lies in the range of $A$.

• It is the projection of $b$ onto $R(A)$. 
Important Example: Weighted Least Squares

- Standard inner-product:

\[
(u, v)_2 := \sum_{i=1}^{m} u_i v_i = u^T v,
\]

\[
||r||_2^2 = \sum_{i=1}^{m} r_i^2 = r^T r,
\]

- Consider \textit{weighted} inner-product:

\[
(u, v)_W := \sum_{i=1}^{m} u_i w_i v_i = u^T W v, \text{ where}
\]

\[
W = \begin{bmatrix}
    w_1 \\
    w_2 \\
    \vdots \\
    w_m
\end{bmatrix}, \quad w_i > 0.
\]

\[
||r||_w^2 = \sum_{i=1}^{m} w_i r_i^2 = r^T W r.
\]
• If we want to minimize in a weighted norm:

\[ \text{Find } x \in \mathbb{R}^n \text{ such that } ||r||_W^2 \text{ is minimized.} \]

• Require

\[
\frac{d}{dx_i} \left[ (b - Ax)^T W (b - Ax) \right] \\
= \frac{d}{dx_i} \left[ b^T W b + x^T A^T W A x - x^T A^T W b - b^T W A x \right] \\
= \frac{d}{dx_i} \left[ x^T A^T W A x - 2x^T A^T W b \right] \\
= 0.
\]

• Thus,

\[
x = (A^T W A)^{-1} A^T W b, \\
y = Ax = A (A^T W A)^{-1} A^T W b, \approx b.
\]

• \( y \) is the \textbf{weighted} least-squares approximation to \( b \).

• Works for \textbf{any} SPD \( W \), not just (positive) diagonal ones.

• Can be used to solve linear systems.
Using Least Squares to Solve Linear Systems

• In particular, suppose $Wb = z$.

• Linear system — $z$ is right-hand side, \textit{known}.
  
  — $b$ is \textit{unknown}.

• Want to find weighted least-squares fit, $y \approx b$, minimizing $\|y - b\|_W^2$ with $y \in \mathcal{R}(A)$.

• Answer:

$$y = A (A^T W A)^{-1} A^T W b$$
$$= A (A^T W A)^{-1} A^T z$$
$$= Ax$$
Using Least Squares to Solve Linear Systems

- Suppose $W$ is a sparse $m \times m$ matrix with (say) $m > 10^6$.
- Factor cost is likely very large (superlinear in $m$).
- If $A = (a_1 \ a_2 \cdots \ a_n)$, $n \ll m$, can form $n$ vectors,

$$WA = (Wa_1 \ Wa_2 \cdots \ Wa_n),$$

and the Gram matrix, $\tilde{W} = A^TWA = [a_i^T Wa_j]$, and solve

$$\tilde{W}x = A^Tz = \begin{pmatrix} a_1^Tz \\ a_2^Tz \\ \vdots \\ a_n^Tz \end{pmatrix},$$

which requires solution of a small $n \times n$ system, $\tilde{W}$. 
Using Least Squares to Solve Linear Systems

- Once we have $x$,
  \[ y = Ax = \sum_{j=1}^{n} a_j x_j \approx b := W^{-1}z. \]

- So, *weighted inner-product* allows us to approximate $b$, the solution to $Wb = z$, without knowing $b$!

- Approximate solution $y \in \mathcal{R}(A) = \text{span}\{a_1, a_2, \ldots, a_n\}$:
  \[ y = A(A^TWA)^{-1} A^Tz \]

- $y$ is the *projection* of $b$ onto $\mathcal{R}(A)$,
  - the *closest approximation* or *best fit* in $\mathcal{R}(A)$ in the $W$-norm.

- $r$ is $W$-orthogonal to $\mathcal{R}(A)$. 
Using Least Squares to Solve Linear Systems

- Very often can have accurate approximations with $n \ll m$.
- In particular, if $\kappa := \text{cond}(W)$, and
  \[
  \mathcal{R}(A) = \text{span}\{Wb, W^2b, \cdots, W^k b\} \\
  = \text{span}\{z, Wz, \cdots, W^{k-1}z\},
  \]
  then can have an accurate answer with $k \approx \sqrt{\kappa}$.
- Can keep increasing $\mathcal{R}(A)$ with additional matrix-vector products.
- This method corresponds to *conjugate gradient iteration* applied to the SPD system $Wb = z$. 
Back to Standard Least Squares

- Suppose we have observational data, \{ b_i \} at some independent times \{ t_i \} (red circles).
- The \( t_i \) s do not need to be sorted and can in fact be repeated.

- We wish to fit a smooth model (blue curve) to the data so we can compactly describe (and perhaps integrate or differentiate) the functional relationship between \( b(t) \) and \( t \).

A common model is of the form:

\[ y(t) = \phi_1(t)x_1 + \phi_2(t)x_2 + \ldots + \phi_n(t)x_n \]

The \( \phi_j(t) \)'s are the basis functions and \( x_j \)s the unknown basis coefficients.

The system is linear with respect to the unknowns, hence, these are linear least squares problems.
Example

- To proceed, we assume $b_i$ represents a function at time points $t_i$, which we are trying to model.

- We select basis functions, e.g., $\phi_j(t) = t^{j-1}$ would span the space of polynomials of up to degree $n-1$. (This might not be the best basis for the polynomials...)

- We then set $\{a_j\}_i = \phi_j(t_i)$ for each column $j = 1, \ldots, n$.

- We then solve the linear least squares problem: $\min ||b - Ax||^2$

- Once we have the $x_j$s, we can reconstruct the smooth function:

$$y(t) = \sum_{j=1}^{n} \phi_j(t)x_j$$
% Linear Least Squares Demo

degree=3; m=20; n=degree+1;

t=3*(rand(m,1)-0.5);
b = t.^3 - t; b=b+0.2*rand(m,1); %% Expect: x =~ [ 0 -1  0 1 ]

plot(t,b,'ro'), pause

%%% DEFINE a_ij = phi_j(t_i)

A=zeros(m,n); for j=1:n; A(:,j) = t.^(j-1); end;

A0=A; b0=b; % Save A & b.

%%% SOLVE LEAST SQUARES PROBLEM via Normal Equations &&&

x = (A'*A) \ A'*b

plot(t,b0,'ro',t,A0*x,'bo',t,1*(b0-A0*x),'kx'), pause
plot(t,A0*x,'bo'), pause

%%% CONSTRUCT SMOOTH APPROXIMATION

tt=(0:100)/100; tt=min(t) + (max(t)-min(t))*tt;
S=zeros(101,n); for k=1:n; S(:,k) = tt.^(k-1); end;
s=S*x;

plot(t,b0,'ro',tt,s,'b-')
title('Least Squares Model Fitting to Cubic')
xlabel('Independent Variable, t')
ylabel('Dependent Variable b_i and y(t)')
Python Least Squares Example

```python
# % Linear Least Squares Demo
import numpy as np
import scipy as sp
import matplotlib
matplotlib.use('MacOSX')
import matplotlib.pyplot as plt
##import pylab

degree=3; m=20; n=degree+1;

t=3*(np.random.rand(m,1)-.5);

b = t**3 - t;
b = b+0.2*np.random.rand(m,1);  # Expect: x = [ 0  -1  0  1 ]

plt.plot(t,b,'ro')
plt.show()

# % DEFINE a_ij = phi_j(t_i)
A=np.zeros((m,n))
for j in range(n):
    A[:,j] = (t**j).T
A0=A
b0=b;  # Save A & b.

# % SOLVE LEAST SQUARES PROBLEM via Normal Equations
x = np.linalg.solve(np.dot(A.T, A), np.dot(A.T,b))

plt.figure()
plt.plot(t,b0,'ro')
plt.plot(t,np.dot(A0,x),'bo')
plt.plot(t,b0-np.dot(A0,x),'kx')
plt.show()

plt.figure()
plt.plot(t,np.dot(A0,x),'bo')
plt.show()

# % CONSTRUCT SMOOTH APPROXIMATION
tt=np.linspace(0,100,101)/100
tt=min(t) + (max(t)-min(t))*tt;

S=np.zeros((101,n))
for k in range(n):
    S[:,k] = tt**k
s= np.dot(S, x)

plt.figure()
plt.plot(t,b0,'ro')
plt.plot(t,s,'b-')
plt.title('Least Squares Model Fitting to Cubic')
plt.xlabel('Independent Variable, t')
plt.ylabel('Dependent Variable b_i and y(t)')
plt.show()
```
Note on the text examples

- Note, the text uses similar examples.

- The notation in the examples is a bit different from the rest of the derivation… so be sure to pay attention.
Data Fitting

- Given \( m \) data points \((t_i, y_i)\), find \( n \)-vector \( x \) of parameters that gives “best fit” to model function \( f(t, x) \),

\[
\min_x \sum_{i=1}^{m} (y_i - f(t_i, x))^2
\]

- Problem is \textit{linear} if function \( f \) is linear in components of \( x \),

\[
f(t, x) = x_1 \phi_1(t) + x_2 \phi_2(t) + \cdots + x_n \phi_n(t)
\]

where functions \( \phi_j \) depend only on \( t \)

- Problem can be written in matrix form as \( Ax \approx b \), with

\[
a_{ij} = \phi_j(t_i) \text{ and } b_i = y_i
\]
Data Fitting

- **Polynomial fitting**

\[
f(t, x) = x_1 + x_2 t + x_3 t^2 + \cdots + x_n t^{n-1}
\]

is linear, since polynomial linear in coefficients, though nonlinear in independent variable \( t \)

- **Fitting sum of exponentials**

\[
f(t, x) = x_1 e^{x_2 t} + \cdots + x_{n-1} e^{x_n t}
\]

is example of nonlinear problem

- **For now, we will consider only linear least squares problems**
Example: Data Fitting

- Fitting quadratic polynomial to five data points gives linear least squares problem

\[
Ax = \begin{bmatrix}
1 & t_1 & t_1^2 \\
1 & t_2 & t_2^2 \\
1 & t_3 & t_3^2 \\
1 & t_4 & t_4^2 \\
1 & t_5 & t_5^2 \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\end{bmatrix} \approx \begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
\end{bmatrix} = b
\]

- Matrix whose columns (or rows) are successive powers of independent variable is called **Vandermonde matrix**
Example, continued

- For data
  
  \[ t \begin{array}{cccccc} 
  -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\
  1.0 & 0.5 & 0.0 & 0.5 & 2.0 
  \end{array} \]

  overdetermined \( 5 \times 3 \) linear system is

  \[ A x = \begin{bmatrix} 
  1 & -1.0 & 1.0 \\
  1 & -0.5 & 0.25 \\
  1 & 0.0 & 0.0 \\
  1 & 0.5 & 0.25 \\
  1 & 1.0 & 1.0 
  \end{bmatrix} \begin{bmatrix} x_1 \\
  x_2 \\
  x_3 \end{bmatrix} \equiv \begin{bmatrix} 1.0 \\
  0.5 \\
  0.0 \\
  0.5 \\
  2.0 \end{bmatrix} = b \]

- Solution, which we will see later how to compute, is

  \[ x = \begin{bmatrix} 0.086 & 0.40 & 1.4 \end{bmatrix}^T \]

  so approximating polynomial is

  \[ p(t) = 0.086 + 0.4t + 1.4t^2 \]
Example, continued

- Resulting curve and original data points are shown in graph
Existence and Uniqueness

- Linear least squares problem $Ax \approx b$ always has solution

- Solution is unique if, and only if, columns of $A$ are linearly independent, i.e., $\text{rank}(A) = n$, where $A$ is $m \times n$

- If $\text{rank}(A) < n$, then $A$ is rank-deficient, and solution of linear least squares problem is not unique

- For now, we assume $A$ has full column rank $n$
Normal Equations

To minimize squared Euclidean norm of residual vector

\[ \| \mathbf{r} \|_2^2 = \mathbf{r}^T \mathbf{r} = (\mathbf{b} - \mathbf{Ax})^T (\mathbf{b} - \mathbf{Ax}) = \mathbf{b}^T \mathbf{b} - 2 \mathbf{x}^T \mathbf{A}^T \mathbf{b} + \mathbf{x}^T \mathbf{A}^T \mathbf{Ax} \]

take derivative with respect to \( \mathbf{x} \) and set it to 0,

\[ 2 \mathbf{A}^T \mathbf{Ax} - 2 \mathbf{A}^T \mathbf{b} = 0 \]

which reduces to \( n \times n \) linear system of normal equations

\[ \mathbf{A}^T \mathbf{Ax} = \mathbf{A}^T \mathbf{b} \]
Vectors $v_1$ and $v_2$ are **orthogonal** if their inner product is zero, $v_1^T v_2 = 0$.

Space spanned by columns of $m \times n$ matrix $A$, $\text{span}(A) = \{Ax : x \in \mathbb{R}^n\}$, is of dimension at most $n$.

If $m > n$, $b$ generally does not lie in $\text{span}(A)$, so there is no exact solution to $Ax = b$.

Vector $y = Ax$ in $\text{span}(A)$ closest to $b$ in 2-norm occurs when residual $r = b - Ax$ is **orthogonal** to $\text{span}(A)$,

$$0 = A^T r = A^T (b - Ax)$$

again giving system of **normal equations**

$$A^T Ax = A^T b$$
Geometric relationships among $b$, $r$, and $\text{span}(A)$ are shown in diagram.

\[ r = b - Ax \]

\[ \theta \]

\[ y = Ax \]

\[ \text{span}(A) \]
Orthogonal Projectors

- Matrix $P$ is **orthogonal projector** if it is idempotent ($P^2 = P$) and symmetric ($P^T = P$)
- Orthogonal projector onto orthogonal complement span$(P)^\perp$ is given by $P_\perp = I - P$
- For any vector $v$,
  \[ v = (P + (I - P))v = Pv + P_\perp v \]
- For least squares problem $Ax \approx b$, if $\text{rank}(A) = n$, then
  \[ P = A(A^T A)^{-1} A^T \]
  is orthogonal projector onto span$(A)$, and
  \[ b = Pb + P_\perp b = Ax + (b - Ax) = y + r \]
Pseudoinverse and Condition Number

- Nonsquare \( m \times n \) matrix \( A \) has no inverse in usual sense
- If \( \text{rank}(A) = n \), pseudoinverse is defined by
  \[
  A^+ = (A^T A)^{-1} A^T
  \]
  and condition number by
  \[
  \text{cond}(A) = \|A\|_2 \cdot \|A^+\|_2
  \]
- By convention, \( \text{cond}(A) = \infty \) if \( \text{rank}(A) < n \)
- Just as condition number of square matrix measures closeness to singularity, condition number of rectangular matrix measures closeness to rank deficiency
- Least squares solution of \( Ax \approx b \) is given by \( x = A^+ b \)
Sensitivity and Conditioning

- Sensitivity of least squares solution to $Ax \approx b$ depends on $b$ as well as $A$

- Define angle $\theta$ between $b$ and $y = Ax$ by

  $$
  \cos(\theta) = \frac{\|y\|_2}{\|b\|_2} = \frac{\|Ax\|_2}{\|b\|_2}
  $$

- Bound on perturbation $\Delta x$ in solution $x$ due to perturbation $\Delta b$ in $b$ is given by

  $$
  \frac{\|\Delta x\|_2}{\|x\|_2} \leq \text{cond}(A) \frac{1}{\cos(\theta)} \frac{\|\Delta b\|_2}{\|b\|_2}
  $$
Similarly, for perturbation $E$ in matrix $A$,

$$\frac{\|\Delta x\|^2}{\|x\|^2} \lesssim ([\text{cond}(A)]^2 \tan(\theta) + \text{cond}(A)) \frac{\|E\|^2}{\|A\|^2}$$

Condition number of least squares solution is about $\text{cond}(A)$ if residual is small, but can be squared or arbitrarily worse for large residual.
Normal Equations Method

- If $m \times n$ matrix $A$ has rank $n$, then symmetric $n \times n$ matrix $A^T A$ is positive definite, so its Cholesky factorization
  \[ A^T A = LL^T \]
  can be used to obtain solution $x$ to system of normal equations
  \[ A^T A x = A^T b \]
  which has same solution as linear least squares problem $Ax \approx b$

- Normal equations method involves transformations
  \[ \text{rectangular} \quad \longrightarrow \quad \text{square} \quad \longrightarrow \quad \text{triangular} \]
So far, our examples have used normal equations approach, as do the next examples.

After the introduction, most of this chapter is devoted to better methods in which columns of A are first *orthogonalized*.

Orthogonalization methods of choice:

- Householder transformations (very stable)
- Givens rotations (stable, and often cheaper than Householder)
- Gram-Schmidt (better than normal eqns, but not great)
- Modified Gram-Schmidt (better than “classical” Gram-Schmidt)
Example: Normal Equations Method

For polynomial data-fitting example given previously, normal equations method gives

\[ A^T A = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
-1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\
1.0 & 0.25 & 0.0 & 0.25 & 1.0
\end{bmatrix} \begin{bmatrix}
1 & -1.0 & 1.0 \\
1 & -0.5 & 0.25 \\
1 & 0.0 & 0.0 \\
1 & 0.5 & 0.25 \\
1 & 1.0 & 1.0
\end{bmatrix} \]

\[ = \begin{bmatrix}
5.0 & 0.0 & 2.5 \\
0.0 & 2.5 & 0.0 \\
2.5 & 0.0 & 2.125
\end{bmatrix}, \]

\[ A^T b = \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
-1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\
1.0 & 0.25 & 0.0 & 0.25 & 1.0
\end{bmatrix} \begin{bmatrix}
1.0 \\
0.5 \\
0.0 \\
0.5 \\
2.0
\end{bmatrix} = \begin{bmatrix}
4.0 \\
1.0 \\
3.25
\end{bmatrix} \]
Example, continued

- Cholesky factorization of symmetric positive definite matrix $A^T A$ gives

$$A^T A = \begin{bmatrix} 5.0 & 0.0 & 2.5 \\ 0.0 & 2.5 & 0.0 \\ 2.5 & 0.0 & 2.125 \end{bmatrix}$$

$$= \begin{bmatrix} 2.236 & 0 & 0 \\ 0 & 1.581 & 0 \\ 1.118 & 0 & 0.935 \end{bmatrix} \begin{bmatrix} 2.236 & 0 & 1.118 \\ 0 & 1.581 & 0 \\ 0 & 0 & 0.935 \end{bmatrix} = LL^T$$

- Solving lower triangular system $L z = A^T b$ by forward-substitution gives $z = \begin{bmatrix} 1.789 \ 0.632 \ 1.336 \end{bmatrix}^T$

- Solving upper triangular system $L^T x = z$ by back-substitution gives $x = \begin{bmatrix} 0.086 \ 0.400 \ 1.429 \end{bmatrix}^T$
Information can be lost in forming $A^T A$ and $A^T b$.

For example, take

$$A = \begin{bmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{bmatrix}$$

where $\epsilon$ is a positive number smaller than $\sqrt{\epsilon_{\text{mach}}}$.

Then in floating-point arithmetic

$$A^T A = \begin{bmatrix} 1 + \epsilon^2 & 1 \\ 1 & 1 + \epsilon^2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

which is singular.

Sensitivity of solution is also worsened, since

$$\text{cond}(A^T A) = [\text{cond}(A)]^2$$
Avoid normal equations:

\[ A^T A x = A^T b \]

Instead, orthogonalize columns of \( A \)

\[ A x = QRx = b \]

Columns of \( Q \) are orthonormal

\( R \) is upper triangular
Projection, \( QR \) Factorization, Gram-Schmidt

- Recall our linear least squares problem:

\[
\mathbf{y} = A \mathbf{x} \approx \mathbf{b},
\]

which is equivalent to minimization / orthogonal projection:

\[
\mathbf{r} := \mathbf{b} - A \mathbf{x} \perp \mathcal{R}(A)
\]
\[
||\mathbf{r}||_2 = ||\mathbf{b} - \mathbf{y}||_2 \leq ||\mathbf{b} - \mathbf{v}||_2 \ \forall \mathbf{v} \in \mathcal{R}(A).
\]

- This problem has solutions

\[
\mathbf{x} = (A^T A)^{-1} A^T \mathbf{b}
\]
\[
\mathbf{y} = A (A^T A)^{-1} A^T \mathbf{b} = P \mathbf{b},
\]

where \( P := A (A^T A)^{-1} A^T \) is the orthogonal projector onto \( \mathcal{R}(A) \).
Observations

\[(A^T A) \mathbf{x} = A^T \mathbf{b} = \begin{pmatrix}
   a_1^T b \\
   a_2^T b \\
   \vdots \\
   a_n^T b
\end{pmatrix}\]

\[(A^T A) = \begin{pmatrix}
   a_1^T a_1 & a_1^T a_2 & \cdots & a_1^T a_n \\
   a_2^T a_1 & a_2^T a_2 & \cdots & a_2^T a_n \\
   \vdots & \vdots & \ddots & \vdots \\
   a_n^T a_1 & a_n^T a_2 & \cdots & a_n^T a_n
\end{pmatrix} \]
Orthogonal Bases

• If the columns of $A$ were orthogonal, such that $a_{ij} = a_i^T a_j = 0$ for $i \neq j$, then $A^T A$ is a diagonal matrix,

$$(A^T A) = \begin{pmatrix}
    a_1^T a_1 \\
    \vdots \\
    a_n^T a_n
\end{pmatrix},$$

and the system is easily solved,

$$x = (A^T A)^{-1} A^T b = \begin{pmatrix}
    \frac{1}{a_1^T a_1} \\
    \vdots \\
    \frac{1}{a_n^T a_n}
\end{pmatrix} \begin{pmatrix}
    a_1^T b \\
    a_2^T b \\
    \vdots \\
    a_n^T b
\end{pmatrix}.$$

• In this case, we can write the projection in closed form:

$$y = \sum_{j=1}^{n} x_j a_j = \sum_{j=1}^{n} \frac{a_j^T b}{a_j^T a_j} a_j.$$

• For orthogonal bases, (1) is the projection of $b$ onto $\text{span}\{a_1, a_2, \ldots, a_n\}$. 

$$(1)$$
Orthonormal Bases

- If the columns are orthogonal and *normalized* such that $\|a_j\| = 1$, we then have $a_j^T a_j = 1$, or more generally

  $$a_i^T a_j = \delta_{ij}, \text{ with } \delta_{ij} := \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \text{ the Kronecker delta},$$

- In this case, $A^T A = I$ and the orthogonal projection is given by

  $$y = A A^T b = \sum_{j=1}^{n} a_j (a_j^T b).$$

**Example:** Suppose our model fit is based on sine functions, sampled uniformly on $[0, \pi]$:

$$\phi_j(t) = \sin j t_i, \quad t_i = \pi i / m, \quad i = 1, \ldots, m.$$ 

In this case,

$$A = \left( \phi_1(t_i) \ \phi_2(t_i) \ \cdots \ \phi_n(t_i) \right),$$

$$A^T A = \frac{n}{2} I.$$
**QR Factorization**

- Generally, we don’t *a priori* have orthonormal bases.

- We can construct them, however. The process is referred to as QR factorization.

- We seek factors $Q$ and $R$ such that $QR = A$ with $Q$ orthogonal (or, *unitary*, in the complex case).

- There are two cases of interest:

\[
\begin{align*}
\text{Reduced QR} & : Q_1 R = A \\
\text{Full QR} & : Q R_O = A
\end{align*}
\]

- Note that

\[
A = Q \begin{bmatrix} R \\ O \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ O \end{bmatrix} = Q_1 R.
\]

- The columns of $Q_1$ form an orthonormal basis for $\mathcal{R}(A)$.

- The columns of $Q_2$ form an orthonormal basis for $\mathcal{R}(A)\perp$. 

**QR Factorization: Gram-Schmidt**

- We’ll look at three approaches to \( QR \):
  - Gram-Schmidt Orthogonalization,
  - Householder Transformations, and
  - Givens Rotations
- We start with Gram-Schmidt - which is most intuitive.
- We are interested in generating orthogonal subspaces that match the nested column spaces of \( A \),

\[
\begin{align*}
\text{span}\{a_1\} &= \text{span}\{q_1\} \\
\text{span}\{a_1, a_2\} &= \text{span}\{q_1, q_2\} \\
\text{span}\{a_1, a_2, a_3\} &= \text{span}\{q_1, q_2, q_3\} \\
\text{span}\{a_1, a_2, \ldots, a_n\} &= \text{span}\{q_1, q_2, \ldots, q_n\}
\end{align*}
\]
**QR Factorization: Gram-Schmidt**

- It’s clear that the conditions
  
  \[
  \text{span}\{ a_1 \} = \text{span}\{ q_1 \} \\
  \text{span}\{ a_1, a_2 \} = \text{span}\{ q_1, q_2 \} \\
  \text{span}\{ a_1, a_2, a_3 \} = \text{span}\{ q_1, q_2, q_3 \} \\
  \text{span}\{ a_1, a_2, \ldots, a_n \} = \text{span}\{ q_1, q_2, \ldots, q_n \}
  \]

  are equivalent to the equations
  
  \[
  a_1 = q_1 r_{11} \\
  a_2 = q_1 r_{12} + q_2 r_{22} \\
  a_3 = q_1 r_{13} + q_2 r_{23} + q_3 r_{33} \\
  \vdots = \vdots + \cdots \\
  a_n = q_1 r_{1n} + q_2 r_{2n} + \cdots + q_n r_{nn}
  \]

  i.e., \( A = QR \)

  (For now, we drop the distinction between \( Q \) and \( Q_1 \), and focus only on the reduced \( QR \) problem.)
Gram-Schmidt Orthogonalization

- The preceding relationship suggests the first algorithm.

Let $Q_{j-1} := [q_1 q_2 \ldots q_{j-1}]$, $P_{j-1} := Q_j Q_{j-1}^T$, $P_{\perp,j-1} := I - P_{j-1}$.

for $j = 2, \ldots, n-1$

$v_j = a_j - P_{j-1} a_j = (I - P_{j-1}) a_j = P_{\perp,j-1} a_j$

$q_j = \frac{v_j}{||v_j||} = \frac{P_{\perp,j-1} a_j}{||P_{\perp,j-1} a_j||}$

end

- This is *Gram-Schmidt orthogonalization*.

- Each new vector $q_j$ starts with $a_j$ and subtracts off the projection onto $\mathcal{R}(Q_{j-1})$, followed by normalization.
Classical Gram-Schmidt Orthogonalization

\[ P_{2}a_{3} = Q_{2}Q_{2}^{T}a_{3} \]

\[ = q_{1} \frac{q_{1}^{T}a_{3}}{q_{1}^{T}q_{1}} + q_{2} \frac{q_{2}^{T}a_{3}}{q_{2}^{T}q_{2}} \]

\[ = q_{1}q_{1}^{T}a_{3} + q_{2}q_{2}^{T}a_{3} \]

In general, if \( Q_{k} \) is an orthogonal matrix, then
\( P_{k} = Q_{k}Q_{k}^{T} \) is an orthogonal projector onto \( R(Q_{k}) \)
Gram-Schmidt Orthogonalization

- The preceding relationship suggests the first algorithm.

  Let \( Q_{j-1} := [q_1 \ q_2 \ldots q_{j-1}] \), \( P_{j-1} := Q_j Q_{j-1}^T \), \( P_{\perp,j-1} := I - P_{j-1} \).

  for \( j = 2, \ldots, n - 1 \)

  \[
  v_j = a_j - P_{j-1} a_j = (I - P_{j-1}) a_j = P_{\perp,j-1} a_j
  \]

  \[
  q_j = \frac{v_j}{||v_j||} = \frac{P_{\perp,j-1} a_j}{||P_{\perp,j-1} a_j||}
  \]

  end

- This is *Gram-Schmidt orthogonalization*.

- Each new vector \( q_j \) starts with \( a_j \) and subtracts off the projection onto \( \mathcal{R}(Q_{j-1}) \), followed by normalization.
Gram-Schmidt: Classical vs. Modified

• We take a closer look at the projection step, $v_j = a_j - P_{j-1}a_j$.

• The classical (unstable) GS projection is executed as

\[
\begin{align*}
    v_j &= a_j \\
    &\text{for } k = 1, \ldots, j - 1, \\
    v_j &= v_j - q_k (q_k^T a_j) \\
    &\text{end}
\end{align*}
\]

• The modified GS projection is executed as

\[
\begin{align*}
    v_j &= a_j \\
    &\text{for } k = 1, \ldots, j - 1, \\
    v_j &= v_j - q_k (q_k^T v_j) \\
    &\text{end}
\end{align*}
\]
Mathematical Difference Between CGS and MGS

• Let \( \tilde{P}_{\perp,j} := I - q_j q_j^T \)

• The CGS projection step amounts to

\[
v_j = \left( \tilde{P}_{\perp,j-1} \tilde{P}_{\perp,j-2} \cdots \tilde{P}_{\perp,1} \right) a_j
= \left( I - \tilde{P}_1 - \tilde{P}_2 - \cdots - \tilde{P}_{j-1} \right) a_j
= a_j - \tilde{P}_1 a_j - \tilde{P}_2 a_j - \cdots - \tilde{P}_{j-1} a_j
= a_j - \sum_{k=1}^{j-1} \tilde{P}_k a_j.
\]

• The MGS projection step is equivalent to

\[
v_j = \tilde{P}_{\perp,j-1} \left( \tilde{P}_{\perp,j-2} \left( \cdots \left( \tilde{P}_{\perp,1} a_j \right) \cdots \right) \right)
= \left( I - \tilde{P}_{j-1} \right) \left( I - \tilde{P}_{j-2} \right) \cdots \left( I - \tilde{P}_1 \right) a_j
= \prod_{k=1}^{j-1} \left( I - \tilde{P}_k \right) a_j
\]
Mathematical Difference Between CGS and MGS

- Lack of associativity in floating point arithmetic drives the difference between CGS and MGS.
- Conceptually, MGS projects the residual, $r_j := a_j - P_{j-1}a_j$.
- As we shall see, neither GS nor MGS are as robust as Householder transformations.
- Both, however, can be cleaned up with a second-pass through the orthogonalization process. (Just set $A = Q$ and repeat, once.)

MGS is an example of the idea that “small corrections are preferred to large ones:

Better to update $v$ by subtracting off the projection of $v$, rather than the projection of $a$. 
Given vectors $a_1$ and $a_2$, we seek orthonormal vectors $q_1$ and $q_2$ having same span.

This can be accomplished by subtracting from second vector its projection onto first vector and normalizing both resulting vectors, as shown in diagram.

$$a_2 - (q_1^T a_2)q_1$$
Gram-Schmidt Orthogonalization

- Process can be extended to any number of vectors $a_1, \ldots, a_k$, orthogonalizing each successive vector against all preceding ones, giving classical Gram-Schmidt procedure

  \[
  \text{for } k = 1 \text{ to } n \\
  q_k = a_k \\
  \text{for } j = 1 \text{ to } k - 1 \\
  r_{jk} = q_j^T a_k \\
  q_k = q_k - r_{jk} q_j \\
  \text{end} \\
  r_{kk} = \|q_k\|_2 \\
  q_k = q_k / r_{kk} \\
  \text{end}
  \]

- Resulting $q_k$ and $r_{jk}$ form reduced QR factorization of $A$
Modified Gram-Schmidt

- Classical Gram-Schmidt procedure often suffers loss of orthogonality in finite-precision

- Also, separate storage is required for $A$, $Q$, and $R$, since original $a_k$ are needed in inner loop, so $q_k$ cannot overwrite columns of $A$

- Both deficiencies are improved by modified Gram-Schmidt procedure, with each vector orthogonalized in turn against all subsequent vectors, so $q_k$ can overwrite $a_k$
Modified Gram-Schmidt algorithm

\[
\text{for } k = 1 \text{ to } n \\
r_{kk} = \|a_k\|_2 \\
q_k = a_k / r_{kk} \\
\text{for } j = k + 1 \text{ to } n \\
r_{kj} = q_k^T a_j \\
a_j = a_j - r_{kj} q_k
\]

\[\leftarrow \text{Coefficient involves modified } a_j\]

Matlab Demo: house.m
n=20;

A = rand(n,n); [Q,R]=qr(A);
for i=1:n; R(i,i)=R(i,i)/(1.2^i); end;
A=Q*R; [Q,R]=qr(A);

v=A; q=Q; a=A; % Classical GS
for j=1:n;
    for k=1:(j-1);
        v(:,j)=v(:,j)-q(:,k)*(q(:,k)'*a(:,j)); end;
        q(:,j)=v(:,j)/norm(v(:,j));
    end;
    qc=q;

v=A; q=Q; a=A; % Modified GS
for j=1:n;
    for k=1:(j-1);
        v(:,j)=v(:,j)-q(:,k)*(q(:,k)'*v(:,j)); end;
        q(:,j)=v(:,j)/norm(v(:,j));
    end;
    qm=q;
v=A; q=Q; a=A;  % Classical GS, text
for k=1:n;
    q(:,k)=a(:,k);
    for j=1:k-1; r(j,k)=q(:,j)'*a(:,k);
        q(:,k)=q(:,k)-r(j,k)*q(:,j); end;
    r(k,k)=norm(q(:,k));
    q(:,k)=q(:,k) / r(k,k);
end;
qct=q;

v=A; q=Q; a=A;  % Modified GS, text
for k=1:n;
    r(k,k)=norm(a(:,k));
    q(:,k)=a(:,k) / r(k,k);
    for j=k+1:n; r(k,j)=q(:,k)'*a(:,j);
        a(:,j)=a(:,j)-r(k,j)*q(:,k); end;
end;
qmt=q;
Householder Transformations: Notes

```matlab
a=A;  \% Householder, per textbook
I=eye(n);  QH=I;
for k=1:n;
    v=a(:,k);  v(1:k-1)=0;
    alphak=-sign(a(k,k))*norm(v);
    v(k)=v(k)-alphak;
    betak=v'*v;
    for j=k:n;  gammaj=v'*a(:,j);
        a(:,j)=a(:,j)-(2*gammaj/betak)*v;  end;
    QH=QH-(2/betak)*v*(v'*QH);
end;
QH=QH';  qht=QH;

nq =norm(Q'*Q-eye(n));
nc =norm(qc'*qc-eye(n));
nm =norm(qm'*qm-eye(n));
nct=norm(qct'*qct-eye(n));
nmt=norm(qmt'*qmt-eye(n));
nht=norm(qht'*qht-eye(n));

[nc nct nm nmt nht nq]

>> house
ans =
    1.6971e-03  1.6971e-03  4.5031e-07  4.5031e-07  1.4232e-15  1.0825e-15
```
Orthogonal Transformations

- We seek alternative method that avoids numerical difficulties of normal equations.
- We need numerically robust transformation that produces easier problem without changing solution.
- What kind of transformation leaves least squares solution unchanged?

Square matrix $Q$ is **orthogonal** if $Q^T Q = I$.

Multiplication of vector by orthogonal matrix preserves Euclidean norm:

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

Thus, multiplying both sides of least squares problem by orthogonal matrix does not change its solution.
As with square linear systems, suitable target in simplifying least squares problems is triangular form.

Upper triangular overdetermined \((m > n)\) least squares problem has form

\[
\begin{bmatrix}
R \\
O
\end{bmatrix} x \simeq \begin{bmatrix} b_1 \\
b_2
\end{bmatrix}
\]

where \(R\) is \(n \times n\) upper triangular and \(b\) is partitioned similarly.

Residual is

\[
\| r \|_2^2 = \| b_1 - Rx \|_2^2 + \| b_2 \|_2^2
\]
Triangular Least Squares Problems, continued

- We have no control over second term, \( \|b_2\|^2 \), but first term becomes zero if \( x \) satisfies \( n \times n \) triangular system

\[
Rx = b_1
\]

which can be solved by back-substitution

- Resulting \( x \) is least squares solution, and minimum sum of squares is

\[
\|r\|^2 = \|b_2\|^2
\]

- So our strategy is to transform general least squares problem to triangular form using orthogonal transformation so that least squares solution is preserved
Given $m \times n$ matrix $A$, with $m > n$, we seek $m \times m$ orthogonal matrix $Q$ such that

$$A = Q \begin{bmatrix} R \\ O \end{bmatrix}$$

where $R$ is $n \times n$ and upper triangular.

Linear least squares problem $Ax \approx b$ is then transformed into triangular least squares problem

$$Q^T Ax = \begin{bmatrix} R \\ O \end{bmatrix} x \approx \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = Q^T b$$

which has same solution, since

$$\|r\|_2^2 = \|b - Ax\|_2^2 = \|b - Q \begin{bmatrix} R \\ O \end{bmatrix} x\|_2^2 = \|Q^T b - \begin{bmatrix} R \\ O \end{bmatrix} x\|_2^2$$
If we partition \( m \times m \) orthogonal matrix \( Q = [Q_1 \, Q_2] \), where \( Q_1 \) is \( m \times n \), then

\[
A = Q \begin{bmatrix} R \\ O \end{bmatrix} = [Q_1 \, Q_2] \begin{bmatrix} R \\ O \end{bmatrix} = Q_1 R
\]

is called \textit{reduced} QR factorization of \( A \).

- Columns of \( Q_1 \) are orthonormal basis for \( \text{span}(A) \), and columns of \( Q_2 \) are orthonormal basis for \( \text{span}(A)^\perp \).
- \( Q_1 Q_1^T \) is orthogonal projector onto \( \text{span}(A) \).
- Solution to least squares problem \( Ax \approx b \) is given by solution to square system

\[
Q_1^T Ax = Rx = c_1 = Q_1^T b
\]
QR for Solving Least Squares

- Start with \( Ax \approx b \)
  \[
  Q \begin{bmatrix} R \\ O \end{bmatrix} x \approx b
  \]
  \[
  Q^T Q \begin{bmatrix} R \\ O \end{bmatrix} x = \begin{bmatrix} R \\ O \end{bmatrix} x \approx Q^T b = [Q_1 \ Q_2] \ b = \begin{bmatrix} \ c_1 \\ c_2 \end{bmatrix}.
  \]

- Define the residual, \( r := b - y = b - Ax \)
  \[
  ||r|| = ||b - Ax||
  = ||Q^T (b - Ax)||
  = \left\| \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} - \begin{pmatrix} Rx \\ O \end{pmatrix} \right\|
  = \left\| \begin{pmatrix} c_1 - Rx \\ c_2 \end{pmatrix} \right\|
  \]
  \[
  ||r||^2 = ||c_1 - Rx||^2 + ||c_2||^2
  \]

- Norm of residual is minimized when \( Rx = c_1 = Q_1^T b \), and takes on value \( ||r|| = ||c_2|| \).
To compute QR factorization of $m \times n$ matrix $A$, with $m > n$, we annihilate subdiagonal entries of successive columns of $A$, eventually reaching upper triangular form.

Similar to LU factorization by Gaussian elimination, but use orthogonal transformations instead of elementary elimination matrices.

Possible methods include:
- Householder transformations
- Givens rotations
- Gram-Schmidt orthogonalization
Method 2: Householder Transformations
Householder Transformsations

- **Householder transformation** has form

\[
H = I - 2 \frac{vv^T}{v^Tv}
\]

for nonzero vector \(v\)

- \(H\) is orthogonal and symmetric: \(H = H^T = H^{-1}\)

- Given vector \(a\), we want to choose \(v\) so that

\[
H \cdot a = \begin{bmatrix}
\alpha \\
0 \\
\vdots \\
0
\end{bmatrix} = \alpha \begin{bmatrix}
1 \\
0 \\
\vdots \\
0
\end{bmatrix} = \alpha e_1
\]

- Substituting into formula for \(H\), we can take

\[
v = a - \alpha e_1
\]

and \(\alpha = \pm \|a\|_2\), with sign chosen to avoid cancellation
Recall, $I - \overline{v}(v^T v)^{-1}v^T$ is a projector onto $R^\perp(v)$.

Therefore, $I - 2\overline{v}(v^T v)^{-1}v^T$ will reflect the transformed vector past $R^\perp(v)$.

With Householder, choose $\overline{v}$ such that the reflected vector has all entries below the $k$th one set to zero.

Also, choose $\overline{v}$ to avoid cancellation in $k$th component.
Householder Derivation

\[ Ha = a - 2 \frac{v^T a}{v^T v} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix} \]

\[ v = a - \alpha e_1 \quad \text{Choose } \alpha \text{ to get desired cancellation.} \]

\[ v^T a = a^T a - \alpha a_1, \quad v^T v = a^T a - 2\alpha a_1 + \alpha^2 \]

\[ Ha = a - 2 \frac{(a^T a - \alpha a_1)}{a^T a - 2\alpha a_1 + \alpha^2} (a - \alpha e_1) \]

\[ = a - 2 \frac{||a||^2 \pm ||a||a_1}{2||a||^2 \pm 2||a||a_1} (a - \alpha e_1) \]

\[ = a - (a - \alpha e_1) = \alpha e_1. \]

Choose  \( \alpha = -\text{sign}(a_1)||a|| = -\left(\frac{a_1}{||a||}\right)||a||. \)
Example: Householder Transformation

- If \( \mathbf{a} = \begin{bmatrix} 2 & 1 & 2 \end{bmatrix}^T \), then we take

\[
\mathbf{v} = \mathbf{a} - \alpha \mathbf{e}_1 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix}
\]

where \( \alpha = \pm ||\mathbf{a}||_2 = \pm 3 \)

- Since \( a_1 \) is positive, we choose negative sign for \( \alpha \) to avoid cancellation, so

\[
\mathbf{v} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}
\]

- To confirm that transformation works,

\[
H\mathbf{a} = \mathbf{a} - 2 \frac{\mathbf{v}^T \mathbf{a}}{\mathbf{v}^T \mathbf{v}} \mathbf{v} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - 2 \frac{15}{30} \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}
\]
Householder QR Factorization

- To compute QR factorization of $A$, use Householder transformations to annihilate subdiagonal entries of each successive column.

- Each Householder transformation is applied to entire matrix, but does not affect prior columns, so zeros are preserved.

- In applying Householder transformation $H$ to arbitrary vector $u$,

\[
Hu = \left( I - 2\frac{vv^T}{v^Tv} \right) u = u - \left( 2\frac{v^T u}{v^Tv} \right) v
\]

which is much cheaper than general matrix-vector multiplication and requires only vector $v$, not full matrix $H$. 

- Householder QR Factorization
- Existence, Uniqueness, and Conditioning
- Normal Equations
- Orthogonal Methods
- SVD
Householder QR Factorization, continued

- Process just described produces factorization

\[ H_n \cdots H_1 A = \begin{bmatrix} R \\ O \end{bmatrix} \]

where \( R \) is \( n \times n \) and upper triangular

- If \( Q = H_1 \cdots H_n \), then \( A = Q \begin{bmatrix} R \\ O \end{bmatrix} \)

- To preserve solution of linear least squares problem, right-hand side \( b \) is transformed by same sequence of Householder transformations

- Then solve triangular least squares problem

\[ \begin{bmatrix} R \\ O \end{bmatrix} x \simeq Q^T b \]
For solving linear least squares problem, product $Q$ of Householder transformations need not be formed explicitly.

$R$ can be stored in upper triangle of array initially containing $A$.

Householder vectors $v$ can be stored in (now zero) lower triangular portion of $A$ (almost).

Householder transformations most easily applied in this form anyway.
Example: Householder QR Factorization

For polynomial data-fitting example given previously, with

\[ A = \begin{bmatrix}
1 & -1.0 & 1.0 \\
1 & -0.5 & 0.25 \\
1 & 0.0 & 0.0 \\
1 & 0.5 & 0.25 \\
1 & 1.0 & 1.0 \\
\end{bmatrix}, \quad b = \begin{bmatrix}
1.0 \\
0.5 \\
0.0 \\
0.5 \\
2.0 \\
\end{bmatrix} \]

Householder vector \( v_1 \) for annihilating subdiagonal entries of first column of \( A \) is

\[ v_1 = \begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix} - \begin{bmatrix}
-2.236 \\
0 \\
0 \\
0 \\
\end{bmatrix} = \begin{bmatrix}
3.236 \\
1 \\
1 \\
1 \\
\end{bmatrix} \]
Applying resulting Householder transformation $H_1$ yields transformed matrix and right-hand side

$$H_1 A = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & -0.191 & -0.405 \\ 0 & 0.309 & -0.655 \\ 0 & 0.809 & -0.405 \\ 0 & 1.309 & 0.345 \end{bmatrix}, \quad H_1 b = \begin{bmatrix} -1.789 \\ -0.362 \\ -0.862 \\ -0.362 \\ 1.138 \end{bmatrix}$$

Householder vector $v_2$ for annihilating subdiagonal entries of second column of $H_1 A$ is

$$v_2 = \begin{bmatrix} 0 \\ -0.191 \\ 0.309 \\ 0.809 \\ 1.309 \end{bmatrix} - \begin{bmatrix} 0 \\ 1.581 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.772 \\ 0.309 \\ 0.809 \\ 1.309 \end{bmatrix}$$
Applying resulting Householder transformation $H_2$ yields

$$H_2H_1A = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & 1.581 & 0 \\ 0 & 0 & -0.725 \\ 0 & 0 & -0.589 \\ 0 & 0 & 0.047 \end{bmatrix}, \quad H_2H_1b = \begin{bmatrix} -1.789 \\ 0.632 \\ -1.035 \\ -0.816 \\ 0.404 \end{bmatrix}$$

Householder vector $v_3$ for annihilating subdiagonal entries of third column of $H_2H_1A$ is

$$v_3 = \begin{bmatrix} 0 \\ -0.725 \\ -0.589 \\ 0.047 \end{bmatrix} - \begin{bmatrix} 0 \\ 0.935 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.660 \\ -0.589 \\ 0.047 \end{bmatrix}$$
Example, continued

- Applying resulting Householder transformation $H_3$ yields

\[
H_3H_2H_1A = \begin{bmatrix}
-2.236 & 0 & -1.118 \\
0 & 1.581 & 0 \\
0 & 0 & 0.935 \\
0 & 0 & 0
\end{bmatrix}, \quad H_3H_2H_1b = \begin{bmatrix}
-1.789 \\
0.632 \\
1.336 \\
0.026 \\
0.337
\end{bmatrix}
\]

- Now solve upper triangular system $Rx = c_1$ by back-substitution to obtain $x = \begin{bmatrix} 0.086 & 0.400 & 1.429 \end{bmatrix}^T$
$k$th Householder Transformation (Reflection)

$$A_k = \begin{pmatrix}
    x & x & x & x & x & x \\
    x & x & x & x & x & x \\
    x & x & x & x & x & x \\
    x & x & x & x & x & x \\
    x & x & x & x & x & x \\
    x & x & x & x & x & x \\
\end{pmatrix} \quad \leftarrow k\text{th row}
$$

Note: $H_k a_j = a_j$ for $j < k$. 
Householder Transformations

\[ H_1 A = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}, \quad H_1 b \rightarrow b^{(1)} = \begin{pmatrix} x \\ x \\ x \end{pmatrix} \]

\[ H_2 H_1 A = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}, \quad H_2 b^{(1)} \rightarrow b^{(2)} = \begin{pmatrix} x \\ x \\ x \end{pmatrix} \]

\[ H_3 H_2 H_1 A = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \end{pmatrix}, \quad H_3 b^{(2)} \rightarrow b^{(3)} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \]

Questions: How does \( H_3 H_2 H_1 \) relate to \( Q \) or \( Q_1 \)?

What is \( Q \) in this case?
Note: Householder Procedure

\[ H_3 H_2 H_1 A = \begin{pmatrix} R \\ O \end{pmatrix}, \quad A = Q \begin{pmatrix} R \\ O \end{pmatrix}. \]

\[ H_3 H_2 H_1 A = Q^{-1} Q \begin{pmatrix} R \\ O \end{pmatrix} = Q^T Q \begin{pmatrix} R \\ O \end{pmatrix} = Q^T A. \]

\[ Q^T = H_3 H_2 H_1 \]

\[ Q = H_1^T H_2^T H_3^T = H_1 H_2 H_3. \]

- Technically, we usually don’t need \( Q \) nor the action of \( Q \).
- Just need the action of \( Q^T \) on a matrix or vector.
- Never form \( Q \) or \( H_k \) (large, \( m \times m \) matrices), just apply \( H_k \) to vectors:

\[ H_k u = u - 2 \left( \frac{u^T u}{u^T u} \right) v_k. \]
Method 3: Givens Rotations
Givens Rotations

- **Givens rotations** introduce zeros one at a time
- Given vector \([a_1 \ a_2]^T\), choose scalars \(c\) and \(s\) so that

\[
\begin{bmatrix}
  c & s \\
  -s & c
\end{bmatrix}
\begin{bmatrix}
  a_1 \\
  a_2
\end{bmatrix}
= \begin{bmatrix}
  \alpha \\
  0
\end{bmatrix}
\]

with \(c^2 + s^2 = 1\), or equivalently, \(\alpha = \sqrt{a_1^2 + a_2^2}\)

- Previous equation can be rewritten

\[
\begin{bmatrix}
  a_1 & a_2 \\
  a_2 & -a_1
\end{bmatrix}
\begin{bmatrix}
  c \\
  s
\end{bmatrix}
= \begin{bmatrix}
  \alpha \\
  0
\end{bmatrix}
\]

- Gaussian elimination yields triangular system

\[
\begin{bmatrix}
  a_1 & a_2 \\
  0 & -a_1 - a_2^2/a_1
\end{bmatrix}
\begin{bmatrix}
  c \\
  s
\end{bmatrix}
= \begin{bmatrix}
  \alpha \\
  -\alpha a_2/a_1
\end{bmatrix}
\]
Givens Rotations, continued

- Back-substitution then gives
  
  $$s = \frac{\alpha a_2}{a_1^2 + a_2^2} \quad \text{and} \quad c = \frac{\alpha a_1}{a_1^2 + a_2^2}$$

- Finally, $c^2 + s^2 = 1$, or $\alpha = \sqrt{a_1^2 + a_2^2}$, implies
  
  $$c = \frac{a_1}{\sqrt{a_1^2 + a_2^2}} \quad \text{and} \quad s = \frac{a_2}{\sqrt{a_1^2 + a_2^2}}$$
% Rotation Matrix Demo

X=[ 0 1 ; ... ] % [ x0 x1
    0 2 ]; % y0 y1 ]

hold off
X0=X;
for t=0:.2:3;
    c=cos(t); s=sin(t);
    R= [ c s ; -s c ];
    X=R*X0;
    x=X(1,:); y=X(2,:);
    plot(x,y,'r.-');
    axis equal; axis ([ -3 3 -3 3 ])
    hold on
    pause(.3)
end;
Example: Givens Rotation

- Let \( \mathbf{a} = \begin{bmatrix} 4 & 3 \end{bmatrix}^T \)

- To annihilate second entry we compute cosine and sine

\[
c = \frac{a_1}{\sqrt{a_1^2 + a_2^2}} = \frac{4}{5} = 0.8 \quad \text{and} \quad s = \frac{a_2}{\sqrt{a_1^2 + a_2^2}} = \frac{3}{5} = 0.6
\]

- Rotation is then given by

\[
\mathbf{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix}
\]

- To confirm that rotation works,

\[
\mathbf{G}\mathbf{a} = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}
\]
More generally, to annihilate selected component of vector in $n$ dimensions, rotate target component with another component

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & c & 0 & s & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & -s & 0 & c & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3 \\
a_4 \\
a_5
\end{bmatrix}
=
\begin{bmatrix}
a_1 \\
\alpha \\
a_3 \\
a_4 \\
a_5
\end{bmatrix}
$$

By systematically annihilating successive entries, we can reduce matrix to upper triangular form using sequence of Givens rotations.

Each rotation is orthogonal, so their product is orthogonal, producing QR factorization.
Straightforward implementation of Givens method requires about 50% more work than Householder method, and also requires more storage, since each rotation requires two numbers, $c$ and $s$, to define it.

These disadvantages can be overcome, but requires more complicated implementation.

Givens can be advantageous for computing QR factorization when many entries of matrix are already zero, since those annihilations can then be skipped.
A particularly attractive use of Givens QR is when A is upper Hessenberg – *A is upper triangular with one additional nonzero diagonal below the main one:* \( A_{ij} = 0 \) if \( i > j+1 \)

In this case, we require Givens row operations applied only \( n \) times, instead of \( O(n^2) \) times.

Work for Givens is thus \( O(n^2) \), vs. \( O(n^3) \) for Householder.

Upper Hessenberg matrices arise when computing eigenvalues.
Successive Givens Rotations

As with Householder transformations, we apply successive Givens rotations, $G_1$, $G_2$, etc.

\[
G_1 A = \begin{pmatrix}
    x & x & x \\
    x & x & x \\
    x & x & x \\
    x & x & x
\end{pmatrix}, \quad H_1 b \rightarrow b^{(1)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}
\]

\[
G_2 G_1 A = \begin{pmatrix}
    x & x & x \\
    x & x & x \\
    x & x & x \\
    x & x & x
\end{pmatrix}, \quad G_2 b^{(1)} \rightarrow b^{(2)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}
\]

\[
G_3 G_2 G_1 A = \begin{pmatrix}
    x & x & x \\
    x & x & x \\
    x & x & x \\
    x & x & x
\end{pmatrix}, \quad G_3 b^{(2)} \rightarrow b^{(3)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}
\]

- How many Givens rotations (total) are required for the $m \times n$ case?
- How does $\ldots G_3 G_2 G_1$ relate to $Q$ or $Q_1$?
- What is $Q$ in this case?
If \( \text{rank}(A) < n \), then QR factorization still exists, but yields singular upper triangular factor \( R \), and multiple vectors \( x \) give minimum residual norm.

- Common practice selects minimum residual solution \( x \) having smallest norm.
- Can be computed by QR factorization with column pivoting or by singular value decomposition (SVD).
- Rank of matrix is often not clear cut in practice, so relative tolerance is used to determine rank.
Example: Near Rank Deficiency

- Consider $3 \times 2$ matrix

$$A = \begin{bmatrix} 0.641 & 0.242 \\ 0.321 & 0.121 \\ 0.962 & 0.363 \end{bmatrix}$$

- Computing QR factorization,

$$R = \begin{bmatrix} 1.1997 & 0.4527 \\ 0 & 0.0002 \end{bmatrix}$$

- $R$ is extremely close to singular (exactly singular to 3-digit accuracy of problem statement)

- If $R$ is used to solve linear least squares problem, result is highly sensitive to perturbations in right-hand side

- For practical purposes, $\text{rank}(A) = 1$ rather than 2, because columns are nearly linearly dependent
Instead of processing columns in natural order, select for reduction at each stage column of remaining unreduced submatrix having maximum Euclidean norm

If \( \text{rank}(A) = k < n \), then after \( k \) steps, norms of remaining unreduced columns will be zero (or “negligible” in finite-precision arithmetic) below row \( k \).

Yields orthogonal factorization of form

\[
Q^T A P = \begin{bmatrix} R & S \\ O & O \end{bmatrix}
\]

where \( R \) is \( k \times k \), upper triangular, and nonsingular, and permutation matrix \( P \) performs column interchanges
Basic solution to least squares problem $Ax \approx b$ can now be computed by solving triangular system $Rz = c_1$, where $c_1$ contains first $k$ components of $Q^Tb$, and then taking

$$x = P \begin{bmatrix} z \\ 0 \end{bmatrix}$$

Minimum-norm solution can be computed, if desired, at expense of additional processing to annihilate $S$

$\text{rank}(A)$ is usually unknown, so rank is determined by monitoring norms of remaining unreduced columns and terminating factorization when maximum value falls below chosen tolerance
Comparison of Methods

- Forming normal equations matrix $A^T A$ requires about $n^2 m/2$ multiplications, and solving resulting symmetric linear system requires about $n^3/6$ multiplications.

- Solving least squares problem using Householder QR factorization requires about $mn^2 - n^3/3$ multiplications.

- If $m \approx n$, both methods require about the same amount of work.

- If $m \gg n$, Householder QR requires about twice as much work as normal equations.

- Cost of SVD is proportional to $mn^2 + n^3$, with proportionality constant ranging from 4 to 10, depending on algorithm used.
Comparison of Methods, continued

- Normal equations method produces solution whose relative error is proportional to $[\text{cond}(A)]^2$

- Required Cholesky factorization can be expected to break down if $\text{cond}(A) \approx 1/\sqrt{\epsilon_{\text{mach}}}$ or worse

- Householder method produces solution whose relative error is proportional to

$$
\text{cond}(A) + \|r\|_2 [\text{cond}(A)]^2
$$

which is best possible, since this is inherent sensitivity of solution to least squares problem

- Householder method can be expected to break down (in back-substitution phase) only if $\text{cond}(A) \approx 1/\epsilon_{\text{mach}}$ or worse
Householder is more accurate and more broadly applicable than normal equations.

These advantages may not be worth additional cost, however, when problem is sufficiently well conditioned that normal equations provide sufficient accuracy.

For rank-deficient or nearly rank-deficient problems, Householder with column pivoting can produce useful solution when normal equations method fails outright.

SVD is even more robust and reliable than Householder, but substantially more expensive.
Singular Value Decomposition

- Singular value decomposition (SVD) of $m \times n$ matrix $A$ has form
  \[ A = U \Sigma V^T \]
- where $U$ is $m \times m$ orthogonal matrix, $V$ is $n \times n$ orthogonal matrix, and $\Sigma$ is $m \times n$ diagonal matrix, with
  \[ \sigma_{ij} = \begin{cases} 0 & \text{for } i \neq j \\ \sigma_i & \text{for } i = j \end{cases} \]
- Diagonal entries $\sigma_i$, called singular values of $A$, are usually ordered so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$
- Columns $u_i$ of $U$ and $v_i$ of $V$ are called left and right singular vectors
SVD of Rectangular Matrix $A$

- $A = U \Sigma V^T$ is $m \times n$.
- $U$ is $m \times m$, orthogonal.
- $\Sigma$ is $m \times n$, diagonal, $\sigma_i > 0$.
- $V$ is $n \times n$, orthogonal.
Example: SVD

SVD of \( A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \) is given by \( U \Sigma V^T = \)

\[
\begin{bmatrix}
.141 & .825 & -.420 & -.351 \\
.344 & .426 & .298 & .782 \\
.547 & .0278 & .664 & -.509 \\
.750 & -.371 & -.542 & .0790 \\
\end{bmatrix}
\begin{bmatrix}
25.5 & 0 & 0 \\
0 & 1.29 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
.504 & .574 & .644 \\
-.761 & -.057 & .646 \\
.408 & -.816 & .408 \\
\end{bmatrix}
\]

In square matrix case, \( U \Sigma V^T \) closely related to eigenpair, \( X \Lambda X^{-1} \)
Applications of SVD

- **Minimum norm solution** to \( Ax \approx b \) is given by

\[
x = \sum_{\sigma_i \neq 0} \frac{u_i^T b}{\sigma_i} v_i
\]

For ill-conditioned or rank deficient problems, “small” singular values can be omitted from summation to stabilize solution.

- **Euclidean matrix norm**: \( \|A\|_2 = \sigma_{\text{max}} \)

- **Euclidean condition number of matrix**: \( \text{cond}(A) = \frac{\sigma_{\text{max}}}{\sigma_{\text{min}}} \)

- **Rank of matrix**: number of nonzero singular values
SVD for Linear Least Squares Problem:  \( A = U \Sigma V^T \)

\[
\begin{align*}
Ax & \approx b \\
U \Sigma V^T & \approx b \\
U^T U \Sigma V^T & \approx U^T b \\
\Sigma V^T & \approx U^T b \\
\begin{bmatrix} \tilde{R} \\ O \end{bmatrix} x & \approx \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \\
\tilde{R} x & = c_1 \\
x & = \sum_{j=1}^{n} v_j \frac{1}{\sigma_j} (c_1)_j = \sum_{j=1}^{n} v_j \frac{1}{\sigma_j} u_j^T b
\end{align*}
\]
SVD for Linear Least Squares Problem: \( A = U \Sigma V^T \)

- SVD can also handle the rank deficient case.
- If there are only \( k \) singular values \( \sigma_j > \epsilon \) then take only the first \( k \) contributions.

\[
x = \sum_{j=1}^{k} v_j \frac{1}{\sigma_j} u_j^T b
\]
Pseudoinverse

- Define pseudoinverse of scalar $\sigma$ to be $1/\sigma$ if $\sigma \neq 0$, zero otherwise
- Define pseudoinverse of (possibly rectangular) diagonal matrix by transposing and taking scalar pseudoinverse of each entry
- Then *pseudoinverse* of general real $m \times n$ matrix $A$ is given by
  \[
  A^+ = V \Sigma^+ U^T
  \]
- Pseudoinverse always exists whether or not matrix is square or has full rank
- If $A$ is square and nonsingular, then $A^+ = A^{-1}$
- In all cases, minimum-norm solution to $Ax \approx b$ is given by $x = A^+ b$
Orthogonal Bases

- SVD of matrix, $A = U\Sigma V^T$, provides orthogonal bases for subspaces relevant to $A$

- Columns of $U$ corresponding to nonzero singular values form orthonormal basis for $\text{span}(A)$

- Remaining columns of $U$ form orthonormal basis for orthogonal complement $\text{span}(A)^\perp$

- Columns of $V$ corresponding to zero singular values form orthonormal basis for null space of $A$

- Remaining columns of $V$ form orthonormal basis for orthogonal complement of null space of $A$
Another way to write SVD is

\[ A = U\Sigma V^T = \sigma_1 E_1 + \sigma_2 E_2 + \cdots + \sigma_n E_n \]

with \( E_i = u_i v_i^T \).

- \( E_i \) has rank 1 and can be stored using only \( m + n \) storage locations.
- Product \( E_i x \) can be computed using only \( m + n \) multiplications.
- Condensed approximation to \( A \) is obtained by omitting from summation terms corresponding to small singular values.
- Approximation using \( k \) largest singular values is closest matrix of rank \( k \) to \( A \).
- Approximation is useful in image processing, data compression, information retrieval, cryptography, etc.
Low Rank Approximation to $A = U \Sigma V^T$

- Because of the diagonal form of $\Sigma$, we have

$$A = U \Sigma V^T = \sum_{j=1}^{n} u_j \sigma_j v_j^T$$

- A rank $k$ approximation to $A$ is given by

$$A \approx A_k := \sum_{j=1}^{k} u_j \sigma_j v_j^T$$

- $A_k$ is the best approximation to $A$ in the Frobenius norm,

$$||M||_F := \sqrt{m_{11}^2 + m_{21}^2 + \cdots + m_{mn}^2}$$
SVD for Image Compression

- If we view an image as an $m \times n$ matrix, we can use the SVD to generate a low-rank compressed version.

- Full image storage cost scales as $O(mn)$

- Compress image storage scales as $O(km) + O(kn)$, with $k < m$ or $n$.

\[ A \approx A_k := \sum_{j=1}^{k} u_j \sigma_j v_j^T \]
Image Compression

- If we view an image as an m x n matrix, we can use the SVD to generate a low-rank compressed version.

- Full image storage cost scales as \( O(mn) \)

- Compress image storage scales as \( O(km) + O(kn) \), with \( k < m \) or \( n \).

\[
A \approx A_k := \sum_{j=1}^{k} u_j \sigma_j v_j^T
\]
Image Compression

- If we view an image as an $m \times n$ matrix, we can use the SVD to generate a low-rank compressed version.

- Full image storage cost scales as $O(mn)$

- Compress image storage scales as $O(km) + O(kn)$, with $k < m$ or $n$.

$k=1$ \hspace{0.5cm} $k=2$ \hspace{0.5cm} $k=3$ (m=536, n=432)
```
[X,A]=imread('collins_img.gif');  [m,n]=size(X);
Xo=X;  imread(Xo,'oldfile.png')
whos
X=double(X);  [U,D,V] = svd(X);  % COMPUTE SVD

X = 0*X;
for k=1:min(m,n);  k
    X = X + U(:,k)*D(k,k)*V(:,k)';
    Xi = uint8(X);  imread(Xi,'newfile.png');  spy(Xi>100);
    pause
end;
```
Image Compression

Compressed image storage scales as $O(km) + O(kn)$, with $k < m$ or $n$.

$k=1$                      $k=2$                      $k=3$

$k=10$                     $k=20$                     $k=50$                     $(m=536, n=462)$
Low-Rank Approximations to Solutions of $Ax = b$

If $\sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_n$,  

$$x \approx \sum_{j=1}^{k} \sigma_j^+ v_j u_j^T b$$

Other functions, aside from the inverse of the matrix, can also be approximated in this way, at relatively low cost, once the SVD is known.