Outline

Method of Least Squares

- 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*

Linear Least Squares

- For linear problems, we obtain *overdetermined* linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, with $m \times n$ matrix $\boldsymbol{A}$, $m > n$
- System is better written $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$, since equality is usually not exactly satisfiable when $m > n$
- Least squares solution $\boldsymbol{x}$ minimizes squared Euclidean norm of residual vector $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}$,

$$\min_{\boldsymbol{x}} \|\boldsymbol{r}\|_2^2 = \min_{\boldsymbol{x}} \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2$$

Least Squares Idea

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

$$\begin{aligned} \underline{y} := A\underline{x} &= \underline{a}_1 x_1 + \underline{a}_2 x_2 + \cdots + \underline{a}_n \underline{x}_n \approx \underline{b} \\ \underline{r} &:= \underline{b} - A\underline{x} = \underline{b} - \underline{y} \end{aligned}$$

Least squares:

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

This system is *overdetermined.*

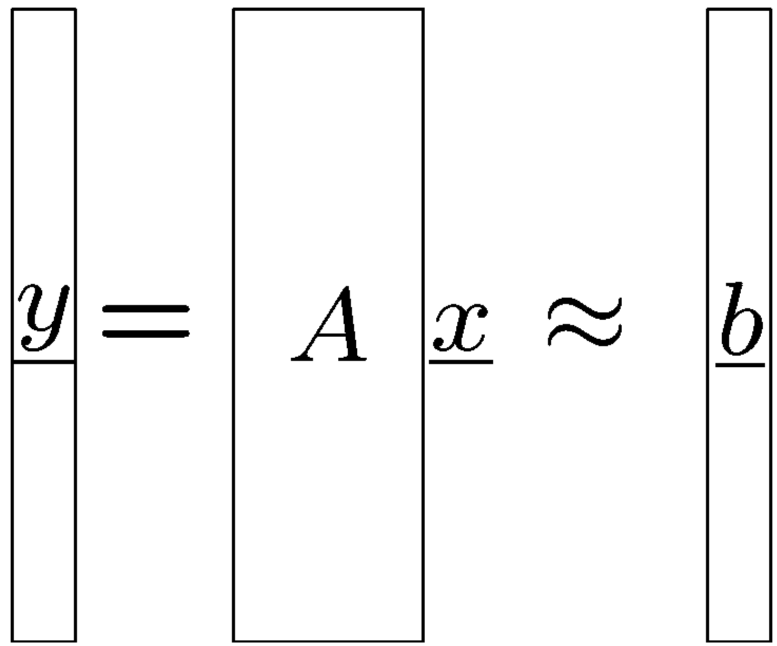
There are more equations than unknowns.

Least Squares Idea

With $m > n$, we have:

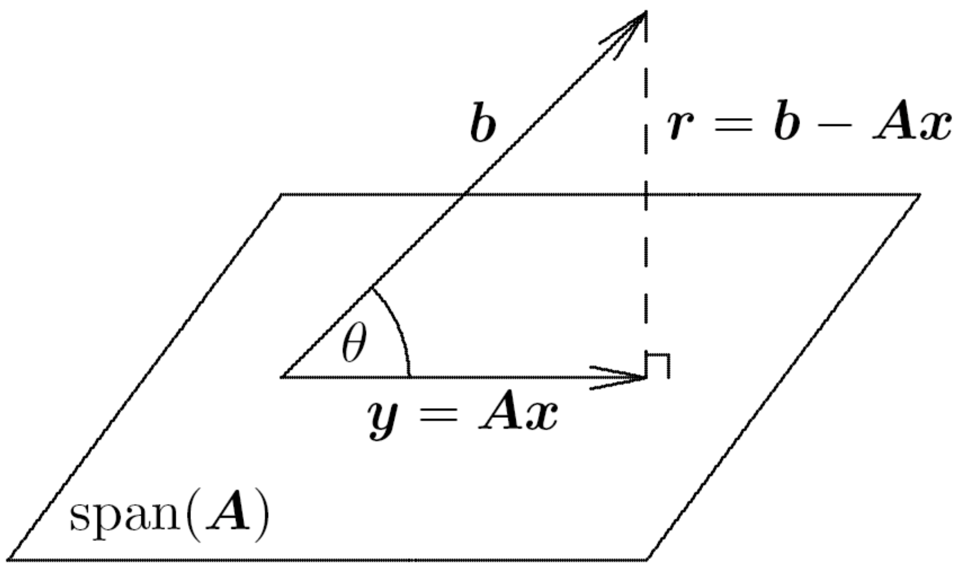
- Lots of data ($\underline{b} \in \mathbb{R}^m$)
- A few model parameters $(x_1, x_2, \ldots, x_n)$
- A few candidate basis vectors $(\underline{a}_1, \underline{a}_2, \ldots, \underline{a}_n)$
- Our estimate, $\underline{y} = A\underline{x}$

The matrix A is tall and thin.



Most Important Picture

Geometric relationships among $\boldsymbol{b}$, $\boldsymbol{r}$, and span$(\boldsymbol{A})$ are shown in diagram



- The vector ***y*** is the *orthogonal projection* of ***b*** onto span(***A***).

- The projection results in minimization of $\| \boldsymbol{r} \|_2$, which, as we shall see, is equivalent to having $\boldsymbol{r} := \boldsymbol{b} - \boldsymbol{Ax} \perp$ span(***A***)

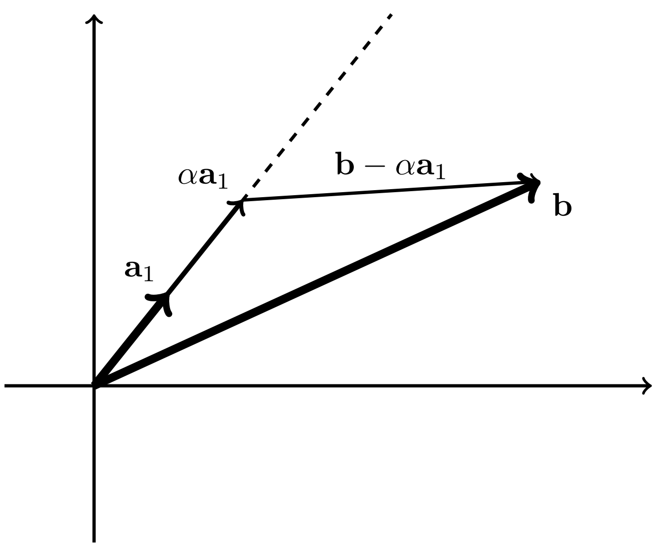
1D Projection

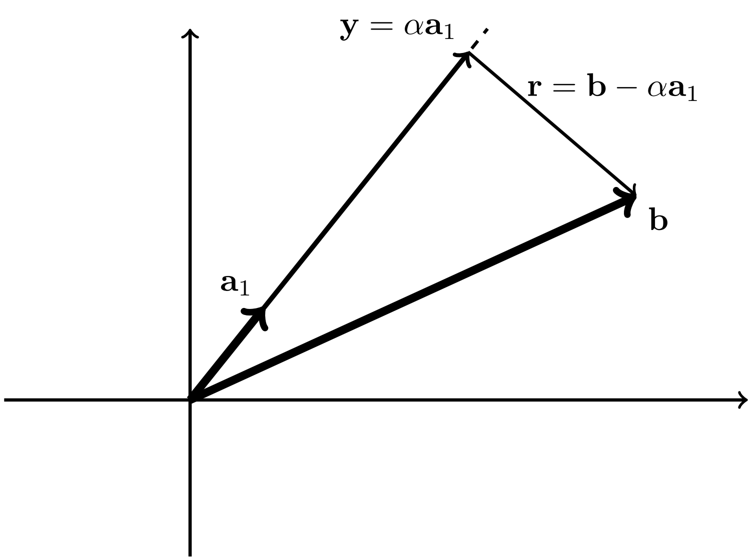
- Consider the 1D subspace of $\mathbb{R}^2$ spanned by $\mathbf{a}_1$:

$$\alpha \mathbf{a}_1 \in \text{span}\{\mathbf{a}_1\}.$$

- The *projection* of a point $\mathbf{b} \in \mathbb{R}^2$ onto span$\{\mathbf{a}_1\}$ is the point on the line $\mathbf{y} = \alpha \mathbf{a}_1$ that is closest to $\mathbf{b}$.

- To find the projection, we look for the value α that minimizes $||\mathbf{r}|| = ||\alpha \mathbf{a}_1 - \mathbf{b}||$ in the 2-norm. (Other norms are also possible.)





1D Projection

- Minimizing the square of the residual with respect to α, we have

$$\begin{aligned}
\frac{d}{d\alpha}\,||\mathbf{r}||^2 &= \\
&= \frac{d}{d\alpha}\,(\mathbf{b}\,-\,\alpha\mathbf{a}_1)^T\,(\mathbf{b}\,-\,\alpha\mathbf{a}_1) \\
&= \frac{d}{d\alpha}\,\left[\mathbf{b}^T\mathbf{b}\,+\,\alpha^2\,\mathbf{a}_1^T\mathbf{a}_1\,-\,2\alpha\,\mathbf{a}_1^T\mathbf{b}\right] \\
&= 2\alpha\,\mathbf{a}_1^T\mathbf{a}_1\,-\,2\,\mathbf{a}_1^T\mathbf{b}\;=\;0
\end{aligned}$$

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

$$\alpha \;=\; \frac{\mathbf{a}_1^T\mathbf{b}}{\mathbf{a}_1^T\mathbf{a}_1}, \quad\Longrightarrow\quad \mathbf{y}\;=\;\alpha\mathbf{a}_1\;=\;\frac{\mathbf{a}_1^T\mathbf{b}}{\mathbf{a}_1^T\mathbf{a}_1}\,\mathbf{a}_1.$$

- We see that $\mathbf{y}$ points in the direction of $\mathbf{a}_1$ and has magnitude that scales as $\mathbf{b}$ (but not with $\mathbf{a}_1$).

- Note that the numerator in the expression above can be zero; the denominator cannot unless $\mathbf{a}_1 = \mathbf{0}$.

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

$$\begin{aligned} ||\mathbf{r}||^2 &= ||A\mathbf{x} - \mathbf{b}||^2 \\ &= (A\mathbf{x} - \mathbf{b})^T (A\mathbf{x} - \mathbf{b}) \\ &= \mathbf{b}^T\mathbf{b} - \mathbf{b}^T A\mathbf{x} - (A\mathbf{x})^T \mathbf{b} + \mathbf{x}^T A^T A\mathbf{x} \\ &= \mathbf{b}^T\mathbf{b} + \mathbf{x}^T A^T A\mathbf{x} - 2\mathbf{x}^T A^T \mathbf{b}. \end{aligned}$$

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

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

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

- Differentiating with respect to x_i,

$$\frac{d}{dx_i}\left(\mathbf{x}^T A^T \mathbf{b}\right) \;=\; c_i = \left(A^T \mathbf{b}\right)_i \,, \qquad\qquad \text{and}$$

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

$$=\; 2 \sum_{j=1}^{n} H_{ij} \, x_j \;=\; 2 \left(H\mathbf{x}\right)_i \,.$$

Projection in Higher Dimensions

- From the preceding pages, the minimum is realized when

$$0 \;=\; \frac{d}{dx_i}\left(\mathbf{x}^T A^T A\mathbf{x} \;-\; 2\mathbf{x}^T A^T \mathbf{b}\right) \;=\; 2\left(A^T A\mathbf{x} \;-\; A^T\mathbf{b}\right)_i, \quad i = 1, \dots, n$$

- Or, in matrix form:

$$\mathbf{x} \;=\; \left(A^T A\right)^{-1} A^T \mathbf{b}.$$

- As in the 1D case, our *projection* is

$$\mathbf{y} \;=\; A\mathbf{x} \;=\; A\,\left(A^T A\right)^{-1} A^T \mathbf{b}.$$

- $\mathbf{y}$ has units and length that scale with $\mathbf{b}$, but it lies in the range of A.
- It is the projection of $\mathbf{b}$ onto $R(A)$.

Note: $(A^TA)^{-1}$ exists as long as the columns of A are independent.

Important Example: Weighted Least Squares

- Standard inner-product:

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

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

- Consider ***weighted*** inner-product:

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

$$W \ = \ \begin{bmatrix} w_1 & & & \\ & w_2 & & \\ & & \ddots & \\ & & & w_m \end{bmatrix}, \quad w_i > 0.$$

$$||\mathbf{r}||_w^2 \ = \ \sum_{i=1}^{m} w_i r_i^2 \ = \ \mathbf{r}^T W \mathbf{r},$$

- If we want to minimize in a weighted norm:

  Find $\mathbf{x} \in \mathbb{R}^n$ *such that* $||\mathbf{r}||_W^2$ *is minimized.*

- Require

$$
\begin{aligned}
\frac{d}{dx_i}\left[(\mathbf{b} - A\mathbf{x})^T\, W\, (\mathbf{b} - A\mathbf{x})\right]& \\
&= \frac{d}{dx_i}\left[\mathbf{b}^T W \mathbf{b} + \mathbf{x}^T A^T W A \mathbf{x} - \mathbf{x}^T A^T W \mathbf{b} - \mathbf{b}^T W A \mathbf{x}\right] \\
&= \frac{d}{dx_i}\left[\mathbf{x}^T A^T W A \mathbf{x} - 2\mathbf{x}^T A^T W \mathbf{b}\right] \\
&= 0.
\end{aligned}
$$

- Thus,

$$
\begin{aligned}
\mathbf{x} &= \left(A^T W A\right)^{-1} A^T W \mathbf{b}, \\
\mathbf{y} &= A\mathbf{x} = A\left(A^T W A\right)^{-1} A^T W \mathbf{b}, \approx \mathbf{b}.
\end{aligned}
$$

- $\mathbf{y}$ is the **weighted** least-squares approximation to $\mathbf{b}$.
- Works for ***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 $W\mathbf{b} = \mathbf{z}$.
- Linear system — $\mathbf{z}$ is right-hand side, ***known***.
  — $\mathbf{b}$ is ***unknown***.
- Want to find weighted least-squares fit, $\mathbf{y} \approx \mathbf{b}$, minimizing $||\,\mathbf{y} - \mathbf{b}\,||_W^2$ with $\mathbf{y} \in \mathcal{R}(A)$.
- Answer:

$$\begin{aligned} \mathbf{y} &= A\left(A^T W A\right)^{-1} A^T W \mathbf{b} \\ &= A\left(A^T W A\right)^{-1} A^T \mathbf{z} \\ &= A\mathbf{x} \end{aligned}$$

← *Here, we approximate* ***$\mathbf{b}=W^{-1}\mathbf{z}$*** *without knowing* ***$\mathbf{b}$***. *We only need matrix-vector products of the form* ***$W\mathbf{a}_j$*** *plus some means of effecting inversion of the small nxn matrix, A^TWA.*

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 $A = (\mathbf{a}_1 \, \mathbf{a}_2 \, \cdots \, \mathbf{a}_n)$, $n \ll m$, can form n vectors,

$$WA = (W\mathbf{a}_1 \, W\mathbf{a}_2 \, \cdots \, W\mathbf{a}_n),$$

and the *Gram* matrix, $\tilde{W} = A^T W A = [\mathbf{a}_i^T W \mathbf{a}_j]$, and solve

$$\tilde{W}\mathbf{x} \;=\; A^T\mathbf{z} \;=\; \left(\begin{array}{c} \mathbf{a}_1^T\mathbf{z} \\ \mathbf{a}_2^T\mathbf{z} \\ \vdots \\ \mathbf{a}_n^T\mathbf{z} \end{array}\right),$$

which requires solution of a small $n \times n$ system, $\tilde{W}$.

Using Least Squares to Solve Linear Systems

- Once we have $\mathbf{x}$,

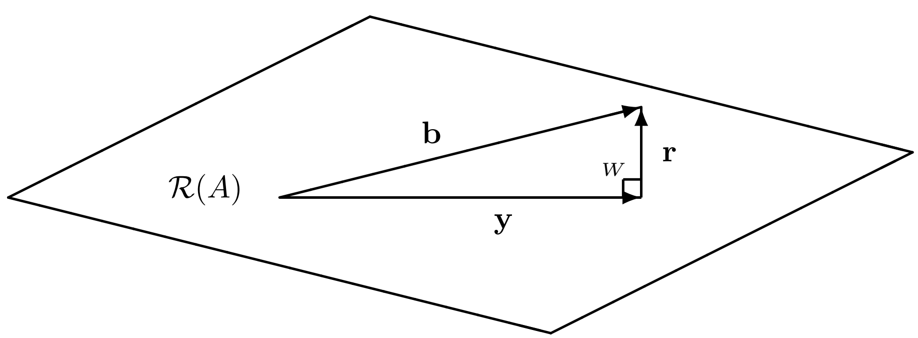
$$\mathbf{y} \;=\; A\mathbf{x} \;=\; \sum_{j=1}^{n} \mathbf{a}_j\, x_j \;\approx\; \mathbf{b} \;:=\; W^{-1}\mathbf{z}.$$

- So, *weighted inner-product* allows us to approximate $\mathbf{b}$, the solution to $W\mathbf{b} = \mathbf{z}$, without knowing $\mathbf{b}$!

- Approximate solution $\mathbf{y} \in \mathcal{R}(A) = \mathrm{span}\{\mathbf{a}_1\, \mathbf{a}_2\, \cdots\, \mathbf{a}_n\}$:

$$\mathbf{y} \;=\; A\left(A^T W A\right)^{-1} A^T \mathbf{z}$$

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



- $\mathbf{r}$ is W-orthogonal to $\mathcal{R}(A)$. ← ***$r^T WA = 0$.***

Using Least Squares to Solve Linear Systems

- Very often can have accurate approximations with $n \ll m$.
- In particular, if κ :=cond(W), and

$$\begin{aligned} \mathcal{R}(A) &= \text{span}\{W\mathbf{b}, W^2\mathbf{b}, \cdots, W^k\mathbf{b}\} \\ &= \text{span}\{\mathbf{z}, W\mathbf{z}, \cdots, W^{k-1}\mathbf{z}\}, \end{aligned}$$

  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 $W\mathbf{b} = \mathbf{z}$.

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.

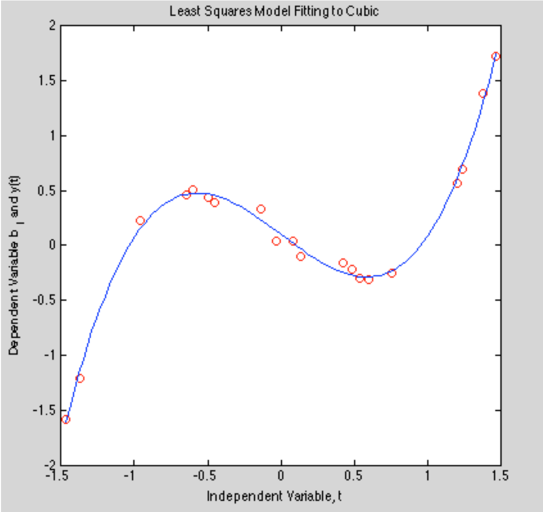
- We wish to fit a smooth model (blue curve) to the data so we can compactly describe (and perhaps integrate or differentiate) the functional relationship between b(t) and t.

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_js 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...)

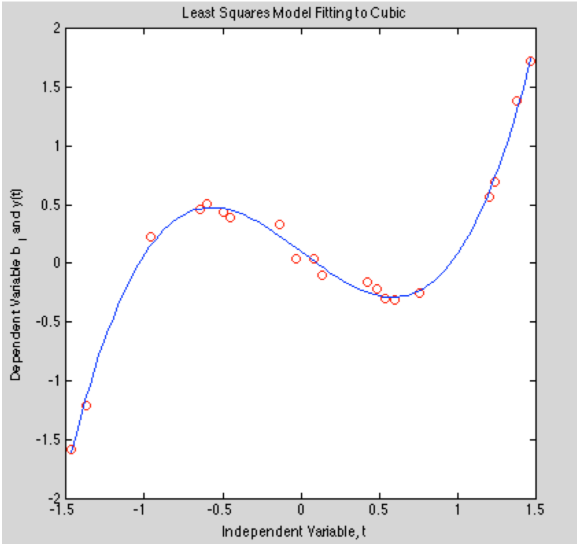
- We then set $\{\underline{a}_j\}_i = \phi_j(t_i)$ for each column $j = 1, \ldots, n$.

- We then solve the linear least squares problem: $\min ||\underline{b} - A\underline{x}||^2$

- Once we have the x_js, we can reconstruct the smooth function:

$$y(t) = \sum_{j=1}^{n} \phi_j(t) x_j$$

.



Matlab Example

```
% 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

```
# % 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)-0.5);

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(tt,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 $\boldsymbol{x}$ of parameters that gives "best fit" to model function $f(t, \boldsymbol{x})$,

$$\min_{\boldsymbol{x}} \sum_{i=1}^{m} (y_i - f(t_i, \boldsymbol{x}))^2$$

- Problem is *linear* if function f is linear in components of $\boldsymbol{x}$,

$$f(t, \boldsymbol{x}) = x_1 \phi_1(t) + x_2 \phi_2(t) + \cdots + x_n \phi_n(t)$$

  where functions ϕ_j depend only on t

- Problem can be written in matrix form as $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$, with $a_{ij} = \phi_j(t_i)$ and $b_i = y_i$

Data Fitting

- Polynomial fitting

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

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

- Fitting sum of exponentials

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

  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

$$\boldsymbol{A}\boldsymbol{x} = \begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ 1 & t_4 & t_4^2 \\ 1 & t_5 & t_5^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cong \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{bmatrix} = \boldsymbol{b}$$

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

Example, continued

- For data

t	-1.0	-0.5	0.0	0.5	1.0
y	1.0	0.5	0.0	0.5	2.0

  overdetermined 5×3 linear system is

$$\boldsymbol{Ax} = \begin{bmatrix} 1 & -1.0 & 1.0 \\ 1 & -0.5 & 0.25 \\ 1 & 0.0 & 0.0 \\ 1 & 0.5 & 0.25 \\ 1 & 1.0 & 1.0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \cong \begin{bmatrix} 1.0 \\ 0.5 \\ 0.0 \\ 0.5 \\ 2.0 \end{bmatrix} = \boldsymbol{b}$$

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

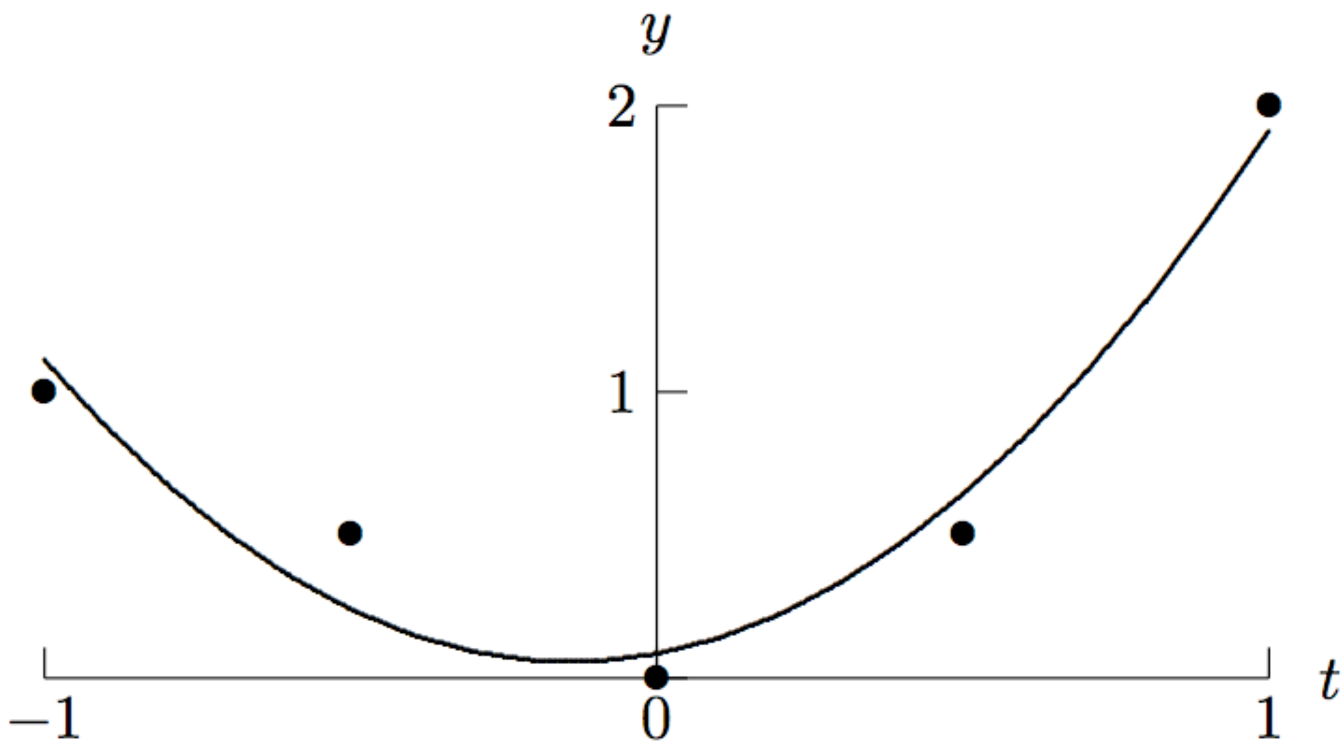
$$\boldsymbol{x} = \begin{bmatrix} 0.086 & 0.40 & 1.4 \end{bmatrix}^T$$

  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 $\boldsymbol{Ax} \cong \boldsymbol{b}$ *always* has solution
- Solution is *unique* if, and only if, columns of $\boldsymbol{A}$ are *linearly independent*, i.e., $\text{rank}(\boldsymbol{A}) = n$, where $\boldsymbol{A}$ is $m \times n$
- If $\text{rank}(\boldsymbol{A}) < n$, then $\boldsymbol{A}$ is *rank-deficient*, and solution of linear least squares problem is not unique
- For now, we assume $\boldsymbol{A}$ has full column rank n

Note: The minimizer, y, is unique.

Normal Equations

- To minimize squared Euclidean norm of residual vector

$$\begin{aligned}\|\boldsymbol{r}\|_2^2 &= \boldsymbol{r}^T\boldsymbol{r} = (\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x})^T(\boldsymbol{b}-\boldsymbol{A}\boldsymbol{x}) \\ &= \boldsymbol{b}^T\boldsymbol{b} - 2\boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{b} + \boldsymbol{x}^T\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}\end{aligned}$$

take derivative with respect to $\boldsymbol{x}$ and set it to $\boldsymbol{0}$,

$$2\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{A}^T\boldsymbol{b} = \boldsymbol{0}$$

which reduces to $n \times n$ linear system of *normal equations*

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$$

Orthogonality

- Vectors $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ are *orthogonal* if their inner product is zero, $\boldsymbol{v}_1^T \boldsymbol{v}_2 = 0$
- Space spanned by columns of $m \times n$ matrix $\boldsymbol{A}$, $\text{span}(\boldsymbol{A}) = \{\boldsymbol{Ax} : \boldsymbol{x} \in \mathbb{R}^n\}$, is of dimension at most n
- If $m > n$, $\boldsymbol{b}$ generally does not lie in $\text{span}(\boldsymbol{A})$, so there is no exact solution to $\boldsymbol{Ax} = \boldsymbol{b}$
- Vector $\boldsymbol{y} = \boldsymbol{Ax}$ in $\text{span}(\boldsymbol{A})$ closest to $\boldsymbol{b}$ in 2-norm occurs when residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{Ax}$ is *orthogonal* to $\text{span}(\boldsymbol{A})$,

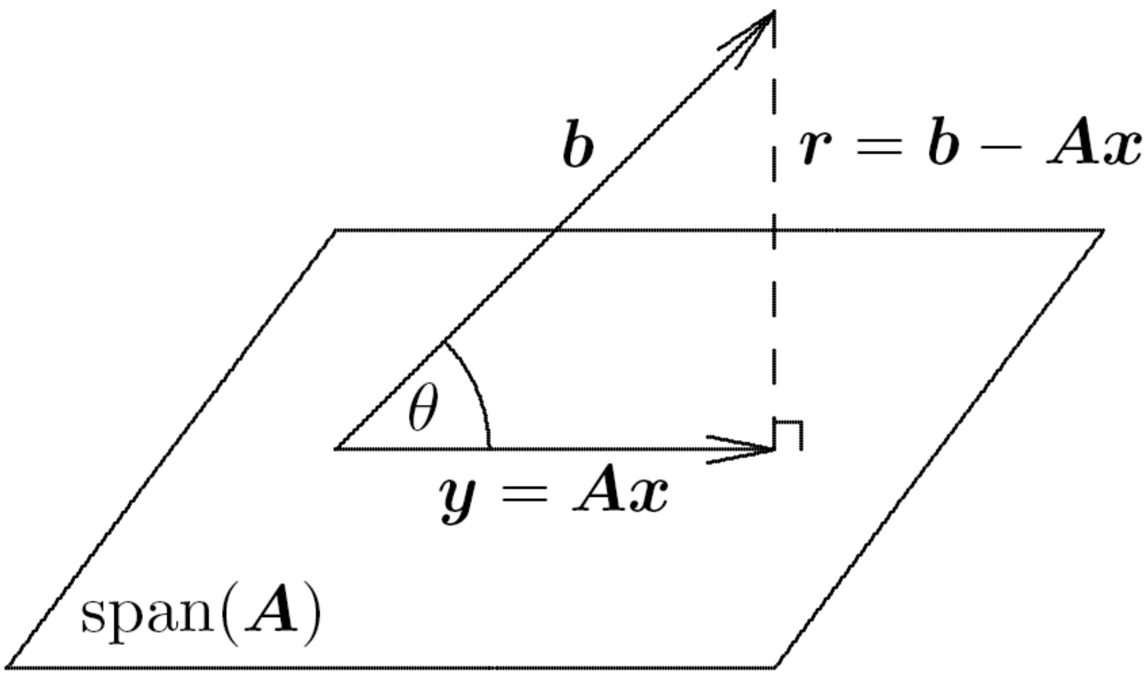
$$\boldsymbol{0} = \boldsymbol{A}^T \boldsymbol{r} = \boldsymbol{A}^T (\boldsymbol{b} - \boldsymbol{Ax})$$

again giving system of *normal equations*

$$\boldsymbol{A}^T \boldsymbol{Ax} = \boldsymbol{A}^T \boldsymbol{b}$$

Orthogonality, continued

- Geometric relationships among $\boldsymbol{b}$, $\boldsymbol{r}$, and span$(\boldsymbol{A})$ are shown in diagram



Orthogonal Projectors

- Matrix $\boldsymbol{P}$ is *orthogonal projector* if it is *idempotent* ($\boldsymbol{P}^2 = \boldsymbol{P}$) and *symmetric* ($\boldsymbol{P}^T = \boldsymbol{P}$)
- Orthogonal projector onto orthogonal complement span$(\boldsymbol{P})^\perp$ is given by $\boldsymbol{P}_\perp = \boldsymbol{I} - \boldsymbol{P}$
- For any vector $\boldsymbol{v}$,

$$\boldsymbol{v} = (\boldsymbol{P} + (\boldsymbol{I} - \boldsymbol{P}))\, \boldsymbol{v} = \boldsymbol{P}\boldsymbol{v} + \boldsymbol{P}_\perp \boldsymbol{v}$$

- For least squares problem $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$, if $\mathrm{rank}(\boldsymbol{A}) = n$, then

$$\boldsymbol{P} = \boldsymbol{A}(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$$

  is orthogonal projector onto span$(\boldsymbol{A})$, and

$$\boldsymbol{b} = \boldsymbol{P}\boldsymbol{b} + \boldsymbol{P}_\perp \boldsymbol{b} = \boldsymbol{A}\boldsymbol{x} + (\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}) = \boldsymbol{y} + \boldsymbol{r}$$

Pseudoinverse and Condition Number

- Nonsquare $m \times n$ matrix $\boldsymbol{A}$ has no inverse in usual sense
- If $\mathrm{rank}(\boldsymbol{A}) = n$, *pseudoinverse* is defined by

$$\boldsymbol{A}^+ = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$$

  and condition number by

$$\mathrm{cond}(\boldsymbol{A}) = \|\boldsymbol{A}\|_2 \cdot \|\boldsymbol{A}^+\|_2$$

- By convention, $\mathrm{cond}(\boldsymbol{A}) = \infty$ if $\mathrm{rank}(\boldsymbol{A}) < n$
- 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 $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$ is given by $\boldsymbol{x} = \boldsymbol{A}^+\boldsymbol{b}$

Sensitivity and Conditioning

- Sensitivity of least squares solution to $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$ depends on $\boldsymbol{b}$ as well as $\boldsymbol{A}$

- Define angle θ between $\boldsymbol{b}$ and $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ by

$$\cos(\theta) = \frac{\|\boldsymbol{y}\|_2}{\|\boldsymbol{b}\|_2} = \frac{\|\boldsymbol{A}\boldsymbol{x}\|_2}{\|\boldsymbol{b}\|_2}$$

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

$$\frac{\|\Delta\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \leq \text{cond}(\boldsymbol{A}) \, \frac{1}{\cos(\theta)} \, \frac{\|\Delta\boldsymbol{b}\|_2}{\|\boldsymbol{b}\|_2}$$

Sensitivity and Conditioning, contnued

- Similarly, for perturbation $\boldsymbol{E}$ in matrix $\boldsymbol{A}$,

$$\frac{\|\Delta \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \lessapprox \left([\mathrm{cond}(\boldsymbol{A})]^2 \tan(\theta) + \mathrm{cond}(\boldsymbol{A})\right) \frac{\|\boldsymbol{E}\|_2}{\|\boldsymbol{A}\|_2}$$

- Condition number of least squares solution is about $\mathrm{cond}(\boldsymbol{A})$ if residual is small, but can be squared or arbitrarily worse for large residual

Normal Equations Method

- If $m \times n$ matrix $\boldsymbol{A}$ has rank n, then symmetric $n \times n$ matrix $\boldsymbol{A}^T\boldsymbol{A}$ is positive definite, so its Cholesky factorization

$$\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$$

can be used to obtain solution $\boldsymbol{x}$ to system of normal equations

$$\boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^T\boldsymbol{b}$$

which has same solution as linear least squares problem $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$

- Normal equations method involves transformations

$$\underset{\boldsymbol{A}}{\text{rectangular}} \longrightarrow \underset{\boldsymbol{A}^T\boldsymbol{A}}{\text{square}} \longrightarrow \underset{\boldsymbol{L}\boldsymbol{L}^T}{\text{triangular}}$$

Spoiler: Normal Equations ***not*** Recommended

- 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; cheap if A is sparse)
  - 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

$$
\begin{aligned}
\boldsymbol{A}^T\boldsymbol{A} &= \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\ 1.0 & 0.25 & 0.0 & 0.25 & 1.0 \end{bmatrix} \begin{bmatrix} 1 & -1.0 & 1.0 \\ 1 & -0.5 & 0.25 \\ 1 & 0.0 & 0.0 \\ 1 & 0.5 & 0.25 \\ 1 & 1.0 & 1.0 \end{bmatrix} \\
&= \begin{bmatrix} 5.0 & 0.0 & 2.5 \\ 0.0 & 2.5 & 0.0 \\ 2.5 & 0.0 & 2.125 \end{bmatrix},
\end{aligned}
$$

$$
\boldsymbol{A}^T\boldsymbol{b} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ -1.0 & -0.5 & 0.0 & 0.5 & 1.0 \\ 1.0 & 0.25 & 0.0 & 0.25 & 1.0 \end{bmatrix} \begin{bmatrix} 1.0 \\ 0.5 \\ 0.0 \\ 0.5 \\ 2.0 \end{bmatrix} = \begin{bmatrix} 4.0 \\ 1.0 \\ 3.25 \end{bmatrix}
$$

Example, continued

- Cholesky factorization of symmetric positive definite matrix $\boldsymbol{A}^T\boldsymbol{A}$ gives

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

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

- Solving lower triangular system $\boldsymbol{L}\boldsymbol{z} = \boldsymbol{A}^T\boldsymbol{b}$ by forward-substitution gives $\boldsymbol{z} = \begin{bmatrix} 1.789 & 0.632 & 1.336 \end{bmatrix}^T$
- Solving upper triangular system $\boldsymbol{L}^T\boldsymbol{x} = \boldsymbol{z}$ by back-substitution gives $\boldsymbol{x} = \begin{bmatrix} 0.086 & 0.400 & 1.429 \end{bmatrix}^T$

Shortcomings of Normal Equations

- Information can be lost in forming $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{b}$
- For example, take

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

  where ϵ is positive number smaller than $\sqrt{\epsilon_{\text{mach}}}$
- Then in floating-point arithmetic

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

  which is singular
- Sensitivity of solution is also worsened, since

$$\text{cond}(\boldsymbol{A}^T\boldsymbol{A}) = [\text{cond}(\boldsymbol{A})]^2$$

- Avoid normal equations:
  - $A^T A\,\boldsymbol{x} = A^{\boldsymbol{T}}\boldsymbol{b}$
- *Instead, orthogonalize columns of A*
  - $A\boldsymbol{x} = QR\boldsymbol{x} \approx \boldsymbol{b}$
- *Columns of* $\boldsymbol{Q}$ *are orthonormal;* $\boldsymbol{R}$ *is upper triangular*
- *Since span(A)=span(Q), we get the same miminizer,* $\boldsymbol{y}$.

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:

$$\begin{aligned} \mathbf{r} \;&:=\; \mathbf{b} - A\mathbf{x} \;\perp\; \mathcal{R}(A) \\ ||\mathbf{r}||_2 \;&=\; ||\mathbf{b} - \mathbf{y}||_2 \;\leq\; ||\mathbf{b} - \mathbf{v}||_2 \;\; \forall\, \mathbf{v} \in \mathcal{R}(A). \end{aligned}$$

- This problem has solutions

$$\begin{aligned} \mathbf{x} \;&=\; \left(A^T A\right)^{-1} A^T \mathbf{b} \\ \mathbf{y} \;&=\; A\left(A^T A\right)^{-1} A^T \mathbf{b} \;=\; P\,\mathbf{b}, \end{aligned}$$

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

Observations

$$\left(A^T A\right)\,\mathbf{x} \;=\; A^T\,\mathbf{b} \;=\; \begin{pmatrix} \mathbf{a}_1^T\mathbf{b} \\ \mathbf{a}_2^T\mathbf{b} \\ \vdots \\ \mathbf{a}_n^T\mathbf{b} \end{pmatrix}$$

$$\left(A^T A\right) \;=\; \begin{pmatrix} \mathbf{a}_1^T\mathbf{a}_1 & \mathbf{a}_1^T\mathbf{a}_2 & \cdots & \mathbf{a}_1^T\mathbf{a}_n \\ \mathbf{a}_2^T\mathbf{a}_1 & \mathbf{a}_2^T\mathbf{a}_2 & \cdots & \mathbf{a}_2^T\mathbf{a}_n \\ \vdots & & & \vdots \\ \mathbf{a}_n^T\mathbf{a}_1 & \mathbf{a}_n^T\mathbf{a}_2 & \cdots & \mathbf{a}_n^T\mathbf{a}_n \end{pmatrix}.$$

Orthogonal Bases

- If the columns of A were *orthogonal*, such that $a_{ij} = \mathbf{a}_i^T \mathbf{a}_j = 0$ for $i \neq j$, then $A^T A$ is a diagonal matrix,

$$\left(A^T A\right) \;=\; \begin{pmatrix} \mathbf{a}_1^T \mathbf{a}_1 & & & \\ & \mathbf{a}_2^T \mathbf{a}_2 & & \\ & & \ddots & \\ & & & \mathbf{a}_n^T \mathbf{a}_n \end{pmatrix},$$

and the system is easily solved,

$$\mathbf{x} \;=\; \left(A^T A\right)^{-1} A^T \mathbf{b} \;=\; \begin{pmatrix} \frac{1}{\mathbf{a}_1^T \mathbf{a}_1} & & & \\ & \frac{1}{\mathbf{a}_2^T \mathbf{a}_2} & & \\ & & \ddots & \\ & & & \frac{1}{\mathbf{a}_n^T \mathbf{a}_n} \end{pmatrix} \begin{pmatrix} \mathbf{a}_1^T \mathbf{b} \\ \mathbf{a}_2^T \mathbf{b} \\ \vdots \\ \mathbf{a}_n^T \mathbf{b} \end{pmatrix}.$$

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

$$\mathbf{y} \;=\; \sum_{j=1}^{n} x_j \, \mathbf{a}_j \;=\; \sum_{j=1}^{n} \frac{\mathbf{a}_j^T \mathbf{b}}{\mathbf{a}_j^T \mathbf{a}_j} \, \mathbf{a}_j \,. \qquad (1)$$

- For *orthogonal* bases, (1) is the projection of $\mathbf{b}$ onto $\mathrm{span}\{\mathbf{a}_1, \, \mathbf{a}_2, \, \ldots, \, \mathbf{a}_n\}$.

Orthonormal Bases

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

$$\mathbf{a}_i^T \mathbf{a}_j \;=\; \delta_{ij}, \text{ with } \delta_{ij} := \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \quad \text{the } \textit{Kronecker delta},$$

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

$$\mathbf{y} \;=\; A\,A^T\,\mathbf{b} \;=\; \sum_{j=1}^{n} \mathbf{a}_j \,\left(\mathbf{a}_j^T \mathbf{b}\right).$$

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

$$\phi_j(t) \;=\; \sqrt{2h}\,\sin j\,t_i\,, \quad t_i \;=\; i \cdot h, \;\; i = 1, \ldots, m; \quad h := \frac{\pi}{m+1}.$$

In this case,

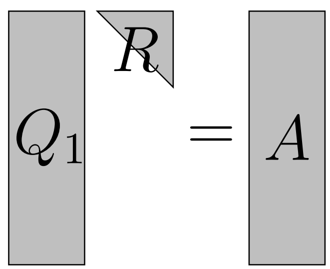
$$A \;=\; \left(\, \phi_1(t_i) \;\; \phi_2(t_i) \;\; \cdots \;\; \phi_n(t_i) \,\right),$$

$$A^T A \;=\; I.$$

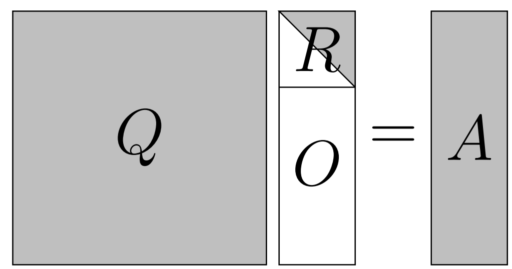
QR Factorization

- 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:

Reduced QR



Full QR



- 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,

$$
\begin{aligned}
\mathrm{span}\{\,\mathbf{a}_1\,\} &= \mathrm{span}\{\,\mathbf{q}_1\,\} \\
\mathrm{span}\{\,\mathbf{a}_1,\,\mathbf{a}_2\,\} &= \mathrm{span}\{\,\mathbf{q}_1,\,\mathbf{q}_2\,\} \\
\mathrm{span}\{\,\mathbf{a}_1,\,\mathbf{a}_2,\,\mathbf{a}_3\,\} &= \mathrm{span}\{\,\mathbf{q}_1,\,\mathbf{q}_2,\,\mathbf{q}_3\,\} \\
\mathrm{span}\{\,\mathbf{a}_1,\,\mathbf{a}_2,\,\ldots,\,\mathbf{a}_n\,\} &= \mathrm{span}\{\,\mathbf{q}_1,\,\mathbf{q}_2,\,\ldots,\,\mathbf{q}_n\,\}
\end{aligned}
$$

QR Factorization: Gram-Schmidt

- It's clear that the conditions

$$
\begin{aligned}
\operatorname{span}\{\,\mathbf{a}_1\,\} &= \operatorname{span}\{\,\mathbf{q}_1\,\} \\
\operatorname{span}\{\,\mathbf{a}_1,\,\mathbf{a}_2\,\} &= \operatorname{span}\{\,\mathbf{q}_1,\,\mathbf{q}_2\,\} \\
\operatorname{span}\{\,\mathbf{a}_1,\,\mathbf{a}_2,\,\mathbf{a}_3\,\} &= \operatorname{span}\{\,\mathbf{q}_1,\,\mathbf{q}_2,\,\mathbf{q}_3\,\} \\
\operatorname{span}\{\,\mathbf{a}_1,\,\mathbf{a}_2,\,\ldots,\,\mathbf{a}_n\,\} &= \operatorname{span}\{\,\mathbf{q}_1,\,\mathbf{q}_2,\,\ldots,\,\mathbf{q}_n\,\}
\end{aligned}
$$

  are equivalent to the equations

$$
\begin{aligned}
\mathbf{a}_1 &= \mathbf{q}_1 r_{11} \\
\mathbf{a}_2 &= \mathbf{q}_1 r_{12} + \mathbf{q}_2 r_{22} \\
\mathbf{a}_3 &= \mathbf{q}_1 r_{13} + \mathbf{q}_2 r_{23} + \mathbf{q}_3 r_{33} \\
\vdots &= \vdots \quad + \quad \cdots \\
\mathbf{a}_n &= \mathbf{q}_1 r_{1n} + \mathbf{q}_2 r_{2n} + \cdots + \mathbf{q}_n r_{nn} \\
\text{i.e.,} \qquad A &= QR
\end{aligned}
$$

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

Gram-Schmidt Orthogonalization

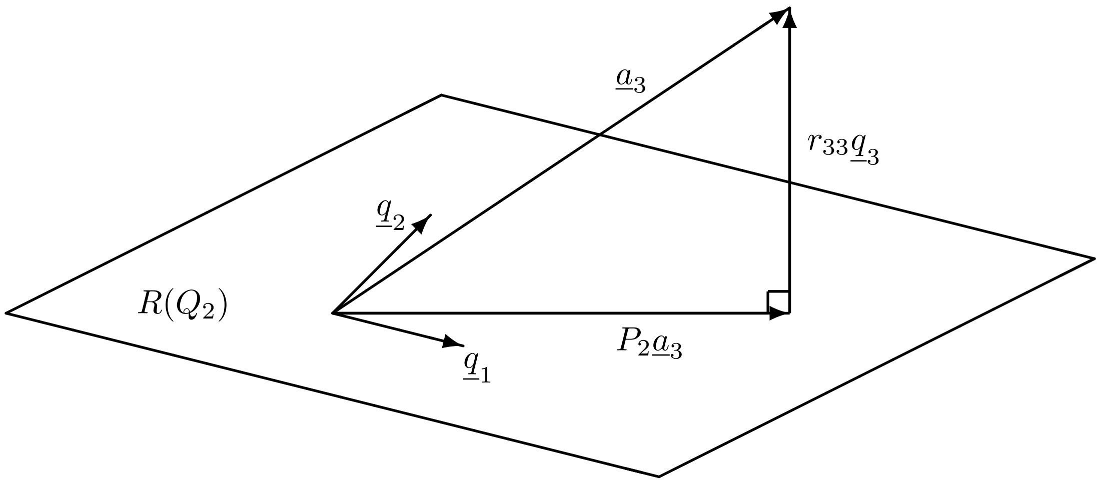
- The preceding relationship suggests the first algorithm.

$$\text{Let } Q_{j-1} := [\mathbf{q}_1 \, \mathbf{q}_2 \, \ldots \mathbf{q}_{j-1}], \; P_{j-1} := Q_j \, Q_{j-1}^T, \; P_{\perp,j-1} := I - P_{j-1}.$$

$$\begin{aligned}
\text{for } j &= 2, \ldots, n-1 \\
\mathbf{v}_j &= \mathbf{a}_j - P_{j-1} \, \mathbf{a}_j = (I - P_{j-1}) \, \mathbf{a}_j = P_{\perp,j-1} \, \mathbf{a}_j \\
\mathbf{q}_j &= \frac{\mathbf{v}_j}{||\mathbf{v}_j||} = \frac{P_{\perp,j-1} \mathbf{a}_j}{||P_{\perp,j-1} \mathbf{a}_j||} \\
\text{end}
\end{aligned}$$

- This is *Gram-Schmidt orthogonalization.*
- Each new vector $\mathbf{q}_j$ starts with $\mathbf{a}_j$ and subtracts off the projection onto $\mathcal{R}(Q_{j-1})$, followed by normalization.

Classical Gram-Schmidt Orthogonalization



$$
\begin{aligned}
P_2\underline{a}_3 &= Q_2 Q_2^T \underline{a}_3 \\
&= \underline{q}_1 \frac{\underline{q}_1^T \underline{a}_3}{\underline{q}_1^T \underline{q}_1} + \underline{q}_2 \frac{\underline{q}_2^T \underline{a}_3}{\underline{q}_2^T \underline{q}_2} \\
&= \underline{q}_1 \underline{q}_1^T \underline{a}_3 + \underline{q}_2 \underline{q}_2^T \underline{a}_3
\end{aligned}
$$

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: Classical vs. Modified

- We take a closer look at the projection step, $\mathbf{v}_j = \mathbf{a}_j \; - \; P_{j-1}\,\mathbf{a}_j$.
- The classical (unstable) GS projection is executed as

$$
\begin{aligned}
&\mathbf{v}_j \; = \; \mathbf{a}_j \\
&\text{for } k = 1, \ldots, j-1, \\
&\qquad \mathbf{v}_j \; = \; \mathbf{v}_j - \mathbf{q}_k \left(\mathbf{q}_k^T \mathbf{a}_j\right) \\
&\text{end}
\end{aligned}
$$

- The modified GS projection is executed as

$$
\begin{aligned}
&\mathbf{v}_j \; = \; \mathbf{a}_j \\
&\text{for } k = 1, \ldots, j-1, \\
&\qquad \mathbf{v}_j \; = \; \mathbf{v}_j - \mathbf{q}_k \left(\mathbf{q}_k^T \mathbf{v}_j\right) \\
&\text{end}
\end{aligned}
$$

Mathematical Difference Between CGS and MGS

- Let $\tilde{P}_k, := \mathbf{q}_k \mathbf{q}_k^T$ (This is an $m \times m$ matrix of what rank?)
- The CGS projection step amounts to

$$\begin{aligned} \mathbf{v}_j &= \mathbf{a}_j - \tilde{P}_1 \mathbf{a}_j - \tilde{P}_2 \mathbf{a}_j - \cdots - \tilde{P}_{j-1} \mathbf{a}_j \\ &= \mathbf{a}_j - \sum_{k=1}^{j-1} \tilde{P}_k \, \mathbf{a}_j . \end{aligned}$$

- The MGS projection step is equivalent to

$$\begin{aligned} \mathbf{v}_j &= \left(I - \tilde{P}_{j-1} \right) \left(I - \tilde{P}_{j-2} \right) \cdots \left(I - \tilde{P}_1 \right) \mathbf{a}_j \\ &= \prod_{k=1}^{j-1} \left(I - \tilde{P}_k \right) \mathbf{a}_j \end{aligned}$$

Note: $\tilde{P}_k \tilde{P}_j = 0$, *if* $k \neq j$.

Mathematical Difference Between CGS and MGS

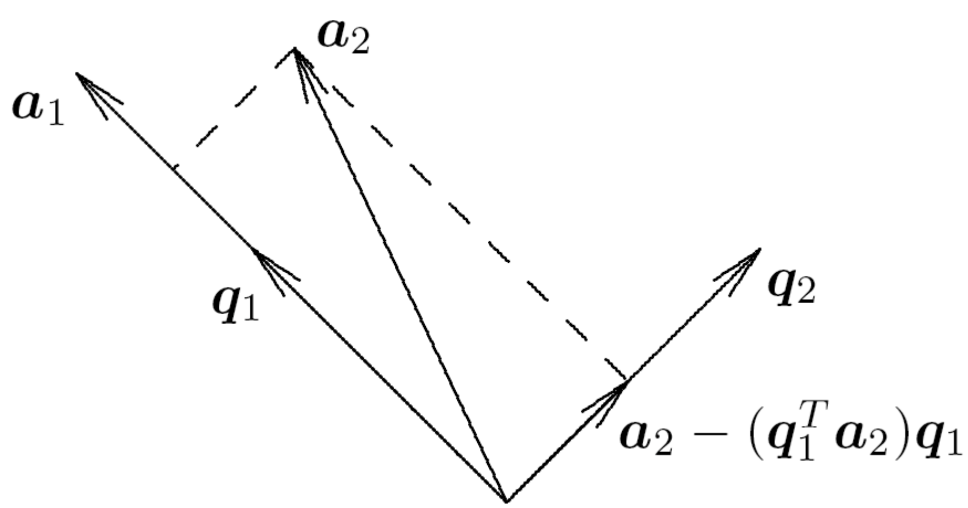
- Lack of associativity in floating point arithmetic drives the difference between CGS and MGS.
- Conceptually, MGS projects the remaining residual rather than the original $\mathbf{a}_j$.
- As we shall see, neither GS nor MGS are as robust as Householder transformations.
- Both, however, can be cleaned 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**.*

Gram-Schmidt Orthogonalization

- Given vectors $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$, we seek orthonormal vectors $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ 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



Gram-Schmidt Orthogonalization

- Process can be extended to any number of vectors $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k$, orthogonalizing each successive vector against all preceding ones, giving *classical Gram-Schmidt* procedure

for $k = 1$ **to** n
  $\boldsymbol{q}_k = \boldsymbol{a}_k$
  for $j = 1$ **to** $k - 1$
    $r_{jk} = \boldsymbol{q}_j^T \boldsymbol{a}_k$ ← *Coefficient involves original* $\boldsymbol{a}_k$
    $\boldsymbol{q}_k = \boldsymbol{q}_k - r_{jk} \boldsymbol{q}_j$
  end
  $r_{kk} = \|\boldsymbol{q}_k\|_2$
  $\boldsymbol{q}_k = \boldsymbol{q}_k / r_{kk}$
end

- Resulting $\boldsymbol{q}_k$ 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 $\boldsymbol{A}$, $\boldsymbol{Q}$, and $\boldsymbol{R}$, since original $\boldsymbol{a}_k$ are needed in inner loop, so $\boldsymbol{q}_k$ cannot overwrite columns of $\boldsymbol{A}$
- Both deficiencies are improved by *modified Gram-Schmidt* procedure, with each vector orthogonalized in turn against all *subsequent* vectors, so $\boldsymbol{q}_k$ can overwrite $\boldsymbol{a}_k$

Modified Gram-Schmidt QR Factorization

- Modified Gram-Schmidt algorithm

for $k = 1$ **to** n
  $r_{kk} = \|\boldsymbol{a}_k\|_2$
  $\boldsymbol{q}_k = \boldsymbol{a}_k / r_{kk}$
  for $j = k+1$ **to** n
    $r_{kj} = \boldsymbol{q}_k^T \boldsymbol{a}_j$ ← *Coefficient involves modified* $\boldsymbol{a}_j$
    $\boldsymbol{a}_j = \boldsymbol{a}_j - r_{kj}\boldsymbol{q}_k$
  end
end

Matlab Demo: house.m

Gram-Schmidt Examples

- Here we consider a matrix that is not well-conditioned.

Classical & Modified GS: Notes

```
%% Test several QR schemes

n=100;  format compact; format shorte

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);

for j=1:n-1; for i=j+2:n; A(i,j)=0; end;end; % Upper H

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;
```

Classical & Modified GS: Notes

```
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

```
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]
```

```
ans =
   5.9707e-05   5.9707e-05   6.4358e-10   6.4358e-10   2.2520e-15   2.1863e-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 $\boldsymbol{Q}$ is *orthogonal* if $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$
- Multiplication of vector by orthogonal matrix preserves Euclidean norm

$$\|\boldsymbol{Q}\boldsymbol{v}\|_2^2 = (\boldsymbol{Q}\boldsymbol{v})^T\boldsymbol{Q}\boldsymbol{v} = \boldsymbol{v}^T\boldsymbol{Q}^T\boldsymbol{Q}\boldsymbol{v} = \boldsymbol{v}^T\boldsymbol{v} = \|\boldsymbol{v}\|_2^2$$

- Thus, multiplying both sides of least squares problem by orthogonal matrix does not change its solution

Triangular Least Squares Problems

- 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

$$\begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix} \boldsymbol{x} \cong \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix}$$

  where $\boldsymbol{R}$ is $n \times n$ upper triangular and $\boldsymbol{b}$ is partitioned similarly
- Residual is

$$\|\boldsymbol{r}\|_2^2 = \|\boldsymbol{b}_1 - \boldsymbol{R}\boldsymbol{x}\|_2^2 + \|\boldsymbol{b}_2\|_2^2$$

Triangular Least Squares Problems, continued

- We have no control over second term, $\|\boldsymbol{b}_2\|_2^2$, but first term becomes zero if $\boldsymbol{x}$ satisfies $n \times n$ triangular system

$$\boldsymbol{R}\boldsymbol{x} = \boldsymbol{b}_1$$

  which can be solved by back-substitution

- Resulting $\boldsymbol{x}$ is least squares solution, and minimum sum of squares is

$$\|\boldsymbol{r}\|_2^2 = \|\boldsymbol{b}_2\|_2^2$$

- So our strategy is to transform general least squares problem to triangular form using orthogonal transformation so that least squares solution is preserved

QR Factorization

- Given $m \times n$ matrix $\boldsymbol{A}$, with $m > n$, we seek $m \times m$ orthogonal matrix $\boldsymbol{Q}$ such that

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

  where $\boldsymbol{R}$ is $n \times n$ and upper triangular

- Linear least squares problem $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$ is then transformed into triangular least squares problem

$$\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{x} = \boxed{\begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix} \boldsymbol{x} \cong \begin{bmatrix} \boldsymbol{c}_1 \\ \boldsymbol{c}_2 \end{bmatrix}} = \boldsymbol{Q}^T\boldsymbol{b}$$

  which has same solution, since

$$\|\boldsymbol{r}\|_2^2 = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|_2^2 = \|\boldsymbol{b} - \boldsymbol{Q}\begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix}\boldsymbol{x}\|_2^2 = \|\boldsymbol{Q}^T\boldsymbol{b} - \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix}\boldsymbol{x}\|_2^2$$

Orthogonal Bases

- If we partition $m \times m$ orthogonal matrix $\boldsymbol{Q} = [\boldsymbol{Q}_1\ \boldsymbol{Q}_2]$, where $\boldsymbol{Q}_1$ is $m \times n$, then

$$\boldsymbol{A} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{O}\end{bmatrix} = [\boldsymbol{Q}_1\ \boldsymbol{Q}_2]\begin{bmatrix}\boldsymbol{R}\\ \boldsymbol{O}\end{bmatrix} = \boldsymbol{Q}_1\boldsymbol{R}$$

  is called *reduced* QR factorization of $\boldsymbol{A}$

- Columns of $\boldsymbol{Q}_1$ are orthonormal basis for $\text{span}(\boldsymbol{A})$, and columns of $\boldsymbol{Q}_2$ are orthonormal basis for $\text{span}(\boldsymbol{A})^\perp$
- $\boldsymbol{Q}_1\boldsymbol{Q}_1^T$ is orthogonal projector onto $\text{span}(\boldsymbol{A})$
- Solution to least squares problem $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$ is given by solution to square system

$$\boldsymbol{Q}_1^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{R}\boldsymbol{x} = \boldsymbol{c}_1 = \boldsymbol{Q}_1^T\boldsymbol{b}$$

QR for Solving Least Squares

- Start with $\quad A\mathbf{x} \approx \mathbf{b}$

$$Q \begin{bmatrix} R \\ O \end{bmatrix} \mathbf{x} \approx \mathbf{b}$$

$$Q^T Q \begin{bmatrix} R \\ O \end{bmatrix} \mathbf{x} = \begin{bmatrix} R \\ O \end{bmatrix} \mathbf{x} \approx Q^T \mathbf{b} = [\, Q_1 \, Q_2]^T \mathbf{b} = \begin{bmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{bmatrix}.$$

- Define the residual, $\quad \mathbf{r} := \mathbf{b} - \mathbf{y} = \mathbf{b} - A\mathbf{x}$

$$\begin{aligned} ||\mathbf{r}|| &= ||\mathbf{b} - A\mathbf{x}|| \\ &= ||Q^T(\mathbf{b} - A\mathbf{x})|| \\ &= \left|\left| \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix} - \begin{pmatrix} R\mathbf{x} \\ O \end{pmatrix} \right|\right| \\ &= \left|\left| \begin{matrix} (\mathbf{c}_1 - R\mathbf{x}) \\ \mathbf{c}_2 \end{matrix} \right|\right| \end{aligned}$$

$$||\mathbf{r}||^2 = ||\mathbf{c}_1 - R\mathbf{x}||^2 + ||\mathbf{c}_2||^2$$

- Norm of residual is minimized when $\boxed{R\mathbf{x} = \mathbf{c}_1 = Q_1^T \mathbf{b},}$ and takes on value $||\mathbf{r}|| = ||\mathbf{c}_2||$.

QR Factorization and Least Squares Review

- Recall: $A\mathbf{x} \approx \mathbf{b}$.

$$A \;=\; QR \;\; or \;\; A \;=\; [Q_l \; Q_r] \begin{bmatrix} R \\ O \end{bmatrix},$$

  with $\tilde{Q} := [Q_l \; Q_r]$ *square.*

- If $\hat{Q}$ and $\tilde{Q}$ are $m \times m$ orthogonal matrices, then $\hat{Q}\tilde{Q}$ is also orthogonal.

- Least squares problem: *Find* $\mathbf{x}$ *such that*

$$\begin{aligned}
\mathbf{r} \;&:=\; (QR\mathbf{x} \,-\, \mathbf{b}) \perp \mathrm{range}(A) \,\equiv\, \mathrm{range}(Q). \\
0 \;&=\; Q^T\mathbf{r} \;=\; Q^TQR\mathbf{x} \,-\, Q^T\mathbf{b} \\
R\mathbf{x} \;&=\; Q^T\mathbf{b} \\
\mathbf{x} \;&=\; R^{-1}Q^T\mathbf{b}.
\end{aligned}$$

- Can solve least squares problem by finding $QR = A$.

Here, Q is the “reduced Q” matrix.

- Projection,

$$
\begin{aligned}
\mathbf{y} &= A\mathbf{x} \\
&= QR\mathbf{x} \\
&= QQ^T\mathbf{b} \\
&= Q(Q^TQ)^{-1}Q^T\mathbf{b} \\
&= \textit{projection onto } \mathcal{R}(Q).
\end{aligned}
$$

- Compare with normal equation approach:

$$
\begin{aligned}
\mathbf{y} &= A(A^TA)^{-1}A^T\mathbf{b} \\
&= \textit{projection onto } \mathcal{R}(A) \equiv \mathcal{R}(Q).
\end{aligned}
$$

- Here, QQ^T and $A(A^TA)^{-1}A^T$ are both *projectors*.
- QQ^T is generally better conditioned than the normal eqution approach.

Computing QR Factorization

- To compute QR factorization of $m \times n$ matrix $\boldsymbol{A}$, with $m > n$, we annihilate subdiagonal entries of successive columns of $\boldsymbol{A}$, 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

QR Householder Preliminaries

- Main idea of Householder is to apply successive simple orthogonal matrices that transform A into upper triangular form. (Similar to Gaussian elimination.)

- Key point is that product of square orthogonal matrices is also orthogonal, e.g., if $H_i^{-1} = H_i^T$, $i = 1, \ldots, m$, then

$$
\begin{aligned}
(H_2 H_1)^T (H_2 H_1) &= H_1^T H_2^T H_2 H_1 \\
&= H_1^T \left(H_2^T H_2 \right) H_1 \\
&= H_1^T H_1 \\
&= I
\end{aligned}
$$

- In the next slides, we start by looking at a single orthogonal matrix, H_i, denoted as a Householder transformation, H.

Householder Transformations

- *Householder transformation* has form

$$\boxed{\boldsymbol{H} = \boldsymbol{I} - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}}$$

  for nonzero vector $\boldsymbol{v}$
- $\boldsymbol{H}$ is orthogonal and symmetric: $\boldsymbol{H} = \boldsymbol{H}^T = \boldsymbol{H}^{-1}$
- Given vector $\boldsymbol{a}$, we want to choose $\boldsymbol{v}$ so that

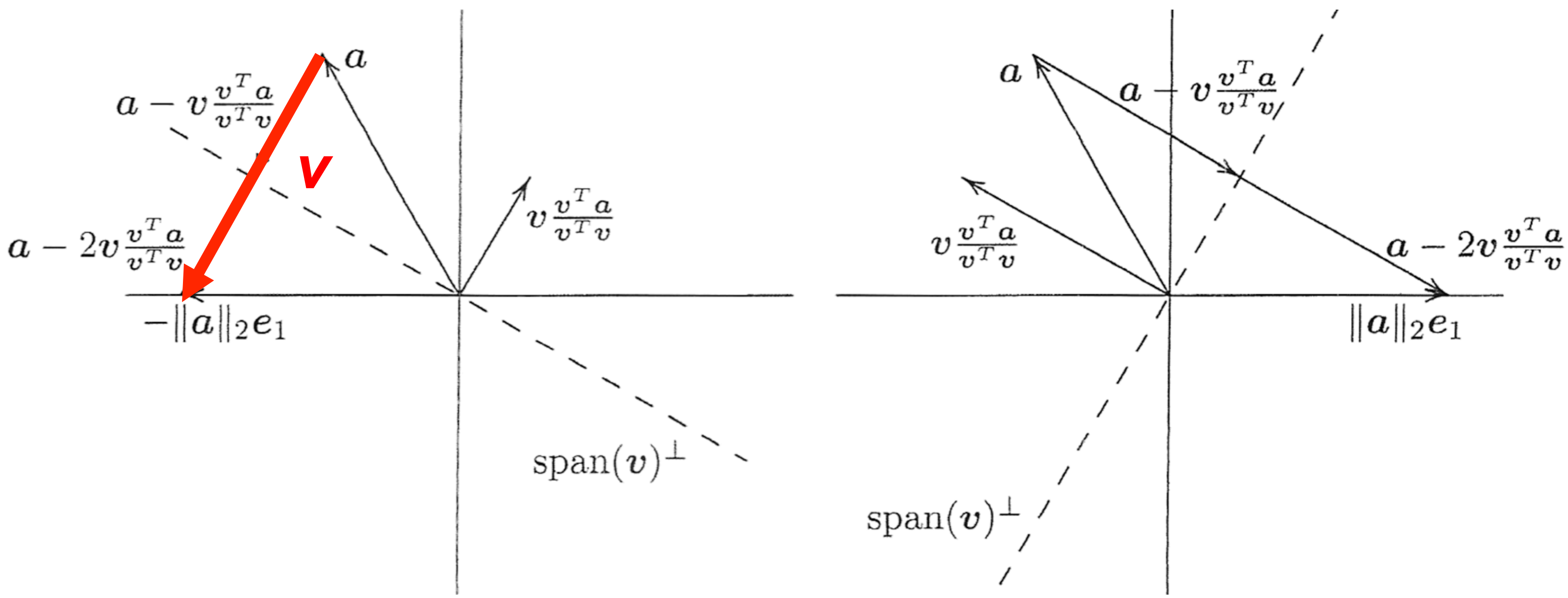
$$\boldsymbol{H}\boldsymbol{a} = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \boldsymbol{e}_1$$

- Substituting into formula for $\boldsymbol{H}$, we can take

$$\boxed{\boldsymbol{v} = \boldsymbol{a} - \alpha \boldsymbol{e}_1}$$

  and $\alpha = \pm\|\boldsymbol{a}\|_2$, with sign chosen to avoid cancellation

Householder Reflection



Recall, $I - \underline{v}(\underline{v}^T\underline{v})^{-1}\underline{v}^T$ is a projector onto $R^\perp(\underline{v})$.

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

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

Also, choose $\underline{v}$ to avoid cancellation in kth component.

Householder Derivation

$$
\begin{aligned}
H\mathbf{a} &= \mathbf{a} - 2\frac{\mathbf{v}^T\mathbf{a}}{\mathbf{v}^T\mathbf{v}} \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_m \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\
\mathbf{v} &= \mathbf{a} - \alpha\mathbf{e}_1 \longleftarrow \text{Choose } \alpha \text{ to avoid cancellation.} \\
\mathbf{v}^T\mathbf{a} &= \mathbf{a}^T\mathbf{a} - \alpha a_1, \qquad \mathbf{v}^T\mathbf{v} = \mathbf{a}^T\mathbf{a} - 2\alpha a_1 + \alpha^2 \\
H\mathbf{a} &= \mathbf{a} - 2\frac{\left(\mathbf{a}^T\mathbf{a} - \alpha a_1\right)}{\mathbf{a}^T\mathbf{a} - 2\alpha a_1 + \alpha^2}\left(\mathbf{a} - \alpha\mathbf{e}_1\right) \\
&= \mathbf{a} - 2\frac{||\mathbf{a}||^2 \pm ||\mathbf{a}||a_1}{2||\mathbf{a}||^2 \pm 2||\mathbf{a}||a_1}\left(\mathbf{a} - \alpha\mathbf{e}_1\right) \\
&= \mathbf{a} - \left(\mathbf{a} - \alpha\mathbf{e}_1\right) = \alpha\mathbf{e}_1.
\end{aligned}
$$

$$
\text{Choose} \qquad \alpha = -\mathrm{sign}(a_1)||\mathbf{a}|| = -\left(\frac{a_1}{|a_1|}\right)||\mathbf{a}||.
$$

Example: Householder Transformation

- If $\boldsymbol{a} = \begin{bmatrix} 2 & 1 & 2 \end{bmatrix}^T$, then we take

$$\boldsymbol{v} = \boldsymbol{a} - \alpha \boldsymbol{e}_1 = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix}$$

  where $\alpha = \pm\|\boldsymbol{a}\|_2 = \pm 3$

- Since a_1 is positive, we choose negative sign for α to avoid cancellation, so $\boldsymbol{v} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix}$

- To confirm that transformation works,

$$\boldsymbol{H}\boldsymbol{a} = \boldsymbol{a} - 2\frac{\boldsymbol{v}^T\boldsymbol{a}}{\boldsymbol{v}^T\boldsymbol{v}}\boldsymbol{v} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} - 2\,\frac{15}{30}\begin{bmatrix} 5 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$$

Householder QR Factorization

- To compute QR factorization of $\boldsymbol{A}$, 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 $\boldsymbol{H}$ to arbitrary vector $\boldsymbol{u}$,

$$\boldsymbol{H}\boldsymbol{u} = \left(\boldsymbol{I} - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right)\boldsymbol{u} = \boldsymbol{u} - \left(2\frac{\boldsymbol{v}^T\boldsymbol{u}}{\boldsymbol{v}^T\boldsymbol{v}}\right)\boldsymbol{v}$$

which is much cheaper than general matrix-vector multiplication and requires only vector $\boldsymbol{v}$, not full matrix $\boldsymbol{H}$

Householder QR Factorization, continued

- Process just described produces factorization

$$\boldsymbol{H}_n \cdots \boldsymbol{H}_1 \boldsymbol{A} = \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix}$$

  where $\boldsymbol{R}$ is $n \times n$ and upper triangular

- If $\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_n$, then $\boldsymbol{A} = \boldsymbol{Q} \begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix}$

- To preserve solution of linear least squares problem, right-hand side $\boldsymbol{b}$ is transformed by same sequence of Householder transformations

- Then solve triangular least squares problem $\begin{bmatrix} \boldsymbol{R} \\ \boldsymbol{O} \end{bmatrix} \boldsymbol{x} \cong \boldsymbol{Q}^T \boldsymbol{b}$

Householder QR Factorization, continued

- For solving linear least squares problem, product $\boldsymbol{Q}$ of Householder transformations need not be formed explicitly
- $\boldsymbol{R}$ can be stored in upper triangle of array initially containing $\boldsymbol{A}$
- Householder vectors $\boldsymbol{v}$ can be stored in (now zero) lower triangular portion of $\boldsymbol{A}$ (almost)
- Householder transformations most easily applied in this form anyway

Example: Householder QR Factorization

- For polynomial data-fitting example given previously, with

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

- Householder vector $\boldsymbol{v}_1$ for annihilating subdiagonal entries of first column of $\boldsymbol{A}$ is

$$\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -2.236 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3.236 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Example, continued

- Applying resulting Householder transformation $\boldsymbol{H}_1$ yields transformed matrix and right-hand side

$$\boldsymbol{H}_1\boldsymbol{A} = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & -0.191 & -0.405 \\ 0 & 0.309 & -0.655 \\ 0 & 0.809 & -0.405 \\ 0 & 1.309 & 0.345 \end{bmatrix}, \quad \boldsymbol{H}_1\boldsymbol{b} = \begin{bmatrix} -1.789 \\ -0.362 \\ -0.862 \\ -0.362 \\ 1.138 \end{bmatrix}$$

- Householder vector $\boldsymbol{v}_2$ for annihilating subdiagonal entries of second column of $\boldsymbol{H}_1\boldsymbol{A}$ is

$$\boldsymbol{v}_2 = \begin{bmatrix} 0 \\ -0.191 \\ 0.309 \\ 0.809 \\ 1.309 \end{bmatrix} - \begin{bmatrix} 0 \\ 1.581 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ -1.772 \\ 0.309 \\ 0.809 \\ 1.309 \end{bmatrix}$$

Example, continued

- Applying resulting Householder transformation $\boldsymbol{H}_2$ yields

$$\boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{A} = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & 1.581 & 0 \\ 0 & 0 & -0.725 \\ 0 & 0 & -0.589 \\ 0 & 0 & 0.047 \end{bmatrix}, \quad \boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{b} = \begin{bmatrix} -1.789 \\ 0.632 \\ -1.035 \\ -0.816 \\ 0.404 \end{bmatrix}$$

- Householder vector $\boldsymbol{v}_3$ for annihilating subdiagonal entries of third column of $\boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{A}$ is

$$\boldsymbol{v}_3 = \begin{bmatrix} 0 \\ 0 \\ -0.725 \\ -0.589 \\ 0.047 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0.935 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -1.660 \\ -0.589 \\ 0.047 \end{bmatrix}$$

Example, continued

- Applying resulting Householder transformation $\boldsymbol{H}_3$ yields

$$\boldsymbol{H}_3\boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{A} = \begin{bmatrix} -2.236 & 0 & -1.118 \\ 0 & 1.581 & 0 \\ 0 & 0 & 0.935 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \boldsymbol{H}_3\boldsymbol{H}_2\boldsymbol{H}_1\boldsymbol{b} = \begin{bmatrix} -1.789 \\ 0.632 \\ 1.336 \\ 0.026 \\ 0.337 \end{bmatrix}$$

- Now solve upper triangular system $\boldsymbol{R}\boldsymbol{x} = \boldsymbol{c}_1$ by back-substitution to obtain $\boldsymbol{x} = \begin{bmatrix} 0.086 & 0.400 & 1.429 \end{bmatrix}^T$

kth Householder Transformation (Reflection)

$$A_k = \begin{array}{c} \begin{array}{cccccc} & & & k\text{th column} & & \\ & & & \downarrow & & \end{array} \\ \begin{pmatrix} x & x & x & x & x & x \\ & x & x & x & x & x \\ & & x & x & x & x \\ & & & x & x & x \\ & & & x & x & x \\ & & & x & x & x \\ & & & x & x & x \\ & & & x & x & x \\ & & & x & x & x \end{pmatrix} \end{array} \begin{array}{l} \\ \\ \\ \\ \\ \longleftarrow k\text{th row} \\ \\ \\ \\ \\ \\ \end{array}$$

Note: $\qquad H_k \, \underline{a}_j = \underline{a}_j$ for $j < k$.

Successive Householder Transformations

$$
\underset{A}{\begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \\ x & x & x \end{bmatrix}}
\xrightarrow{H_1}
\underset{H_1A}{\begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} \\ \mathbf{0} & \mathbf{x} & \mathbf{x} \end{bmatrix}}
\xrightarrow{H_2}
\underset{H_2H_1A}{\begin{bmatrix} x & x & x \\ & \mathbf{x} & \mathbf{x} \\ & \mathbf{0} & \mathbf{x} \\ & \mathbf{0} & \mathbf{x} \\ & \mathbf{0} & \mathbf{x} \end{bmatrix}}
\xrightarrow{H_3}
\underset{H_3H_2H_1A}{\begin{bmatrix} x & x & x \\ & x & x \\ & & \mathbf{x} \\ & & \mathbf{0} \\ & & \mathbf{0} \end{bmatrix}}
$$

Householder Transformations

$$H_1\,A = \begin{pmatrix} x & x & x \\ & x & x \\ & x & x \\ & x & x \end{pmatrix}, \qquad H_1\,\mathbf{b} \longrightarrow \mathbf{b}^{(1)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}$$

$$H_2\,H_1\,A = \begin{pmatrix} x & x & x \\ & x & x \\ & & x \\ & & x \end{pmatrix}, \qquad H_2\,\mathbf{b}^{(1)} \longrightarrow \mathbf{b}^{(2)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}$$

$$H_3\,H_2\,H_1\,A = \begin{pmatrix} x & x & x \\ & x & x \\ & & x \\ & & \end{pmatrix}, \qquad H_3\,\mathbf{b}^{(2)} \longrightarrow \mathbf{b}^{(3)} = \begin{pmatrix} \mathbf{c}_1 \\ \mathbf{c}_2 \end{pmatrix}.$$

Questions: How does $H_3\,H_2\,H_1$ relate to Q or Q_1??

What is Q in this case?

Method 3: Givens Rotations

Givens Rotations

- *Givens rotations* introduce zeros one at a time
- Given vector $\begin{bmatrix} a_1 & a_2 \end{bmatrix}^T$, 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

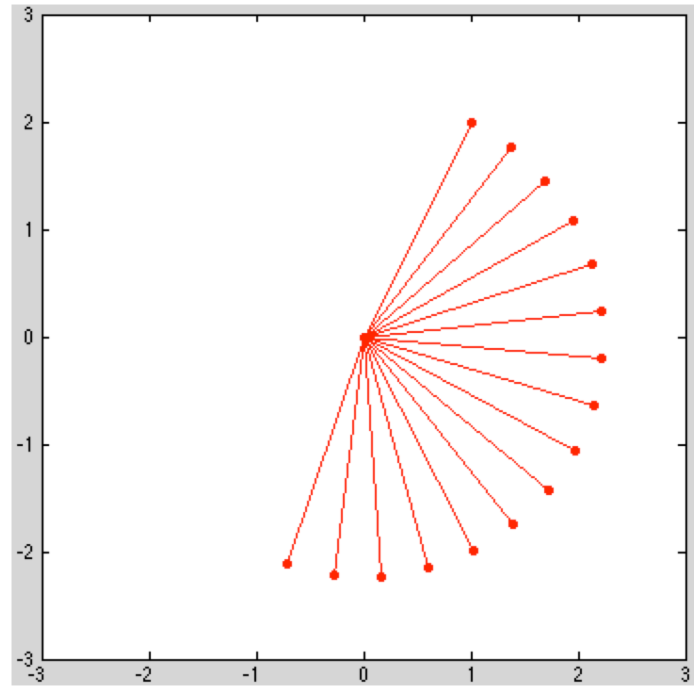
$$s = \frac{\alpha a_2}{a_1^2 + a_2^2} \qquad \text{and} \qquad c = \frac{\alpha a_1}{a_1^2 + a_2^2}$$

- Finally, $c^2 + s^2 = 1$, or $\alpha = \sqrt{a_1^2 + a_2^2}$, implies

$$\boxed{c = \frac{a_1}{\sqrt{a_1^2 + a_2^2}}} \qquad \text{and} \qquad \boxed{s = \frac{a_2}{\sqrt{a_1^2 + a_2^2}}}$$

2 x 2 Rotation Matrices

```
% 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 $\boldsymbol{a} = \begin{bmatrix} 4 & 3 \end{bmatrix}^T$
- 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

$$\boldsymbol{G} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix}$$

- To confirm that rotation works,

$$\boldsymbol{G}\boldsymbol{a} = \begin{bmatrix} 0.8 & 0.6 \\ -0.6 & 0.8 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

Givens QR Factorization

- More generally, to annihilate selected component of vector in n dimensions, rotate target component with another component

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

- 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

Givens Rotations

$$G_k = \begin{bmatrix} I & & \\ & G & \\ & & I \end{bmatrix}$$

- If G is a 2×2 block, G_k Selectively acts on two adjacent rows.
- The *full* rows.

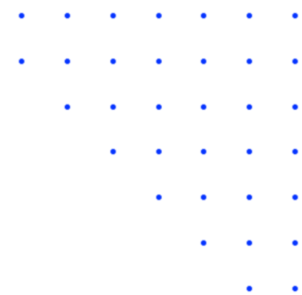
Givens QR Factorization

- 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

Givens QR

- 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* $i > j+1$

0.1967	0.2973	0.0899	0.3381	0.5261	0.3965	0.1279
0.0934	0.0620	0.0809	0.2940	0.7297	0.0616	0.5495
0	0.2982	0.7772	0.7463	0.7073	0.7802	0.4852
0	0	0.9051	0.0103	0.7814	0.3376	0.8905
0	0	0	0.0484	0.2880	0.6079	0.7990
0	0	0	0	0.6925	0.7413	0.7343
0	0	0	0	0	0.1048	0.0513



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

- Work for Givens is thus $O(n^2)$ instead of $O(n^3)$.

- Upper Hessenberg matrices arise when computing eigenvalues.

Extra Credit Question: What is cost of Householder in this case?

Successive Givens Rotations

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

$$G_1\,A = \begin{pmatrix} x & x & x \\ x & x & x \\ x & x & x \\ & x & x \end{pmatrix}, \qquad H_1\,\mathbf{b} \longrightarrow \mathbf{b}^{(1)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}$$

$$G_2\,G_1\,A = \begin{pmatrix} x & x & x \\ x & x & x \\ & x & x \\ & x & x \end{pmatrix}, \qquad G_2\,\mathbf{b}^{(1)} \longrightarrow \mathbf{b}^{(2)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}$$

$$G_3\,G_2\,G_1\,A = \begin{pmatrix} x & x & x \\ & x & x \\ & x & x \\ & x & x \end{pmatrix}, \qquad G_3\,\mathbf{b}^{(2)} \longrightarrow \mathbf{b}^{(3)} = \begin{pmatrix} x \\ x \\ x \\ x \end{pmatrix}$$

- **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?**

Rank Deficiency

- If $\mathrm{rank}(\boldsymbol{A}) < n$, then QR factorization still exists, but yields singular upper triangular factor $\boldsymbol{R}$, and multiple vectors $\boldsymbol{x}$ give minimum residual norm
- Common practice selects minimum residual solution $\boldsymbol{x}$ 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 3×2 matrix

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

- Computing QR factorization,

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

- $\boldsymbol{R}$ is extremely close to singular (exactly singular to 3-digit accuracy of problem statement)
- If $\boldsymbol{R}$ is used to solve linear least squares problem, result is highly sensitive to perturbations in right-hand side
- For practical purposes, $\mathrm{rank}(\boldsymbol{A}) = 1$ rather than 2, because columns are nearly linearly dependent

QR with Column Pivoting

- Instead of processing columns in natural order, select for reduction at each stage column of remaining unreduced submatrix having maximum Euclidean norm
- If $\mathrm{rank}(\boldsymbol{A}) = k < n$, 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

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

where $\boldsymbol{R}$ is $k \times k$, upper triangular, and nonsingular, and permutation matrix $\boldsymbol{P}$ performs column interchanges

QR with Column Pivoting, continued

- *Basic solution* to least squares problem $\boldsymbol{Ax} \cong \boldsymbol{b}$ can now be computed by solving triangular system $\boldsymbol{Rz} = \boldsymbol{c}_1$, where $\boldsymbol{c}_1$ contains first k components of $\boldsymbol{Q}^T\boldsymbol{b}$, and then taking

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

- *Minimum-norm solution* can be computed, if desired, at expense of additional processing to annihilate $\boldsymbol{S}$

- $\mathrm{rank}(\boldsymbol{A})$ 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 $\boldsymbol{A}^T\boldsymbol{A}$ requires about $n^2m/2$ 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 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 $[\mathrm{cond}(\boldsymbol{A})]^2$
- Required Cholesky factorization can be expected to break down if $\mathrm{cond}(\boldsymbol{A}) \approx 1/\sqrt{\epsilon_{\mathrm{mach}}}$ or worse
- Householder method produces solution whose relative error is proportional to

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

  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 $\mathrm{cond}(\boldsymbol{A}) \approx 1/\epsilon_{\mathrm{mach}}$ or worse

Comparison of Methods, continued

- 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 $\boldsymbol{A}$ has form

$$\boxed{\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T}$$

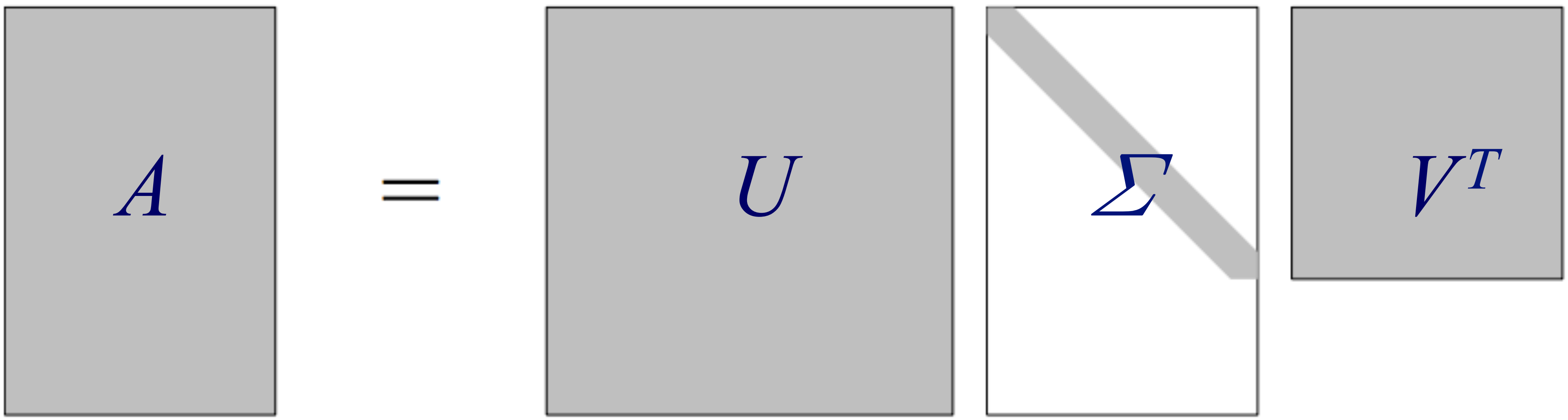
where $\boldsymbol{U}$ is $m \times m$ orthogonal matrix, $\boldsymbol{V}$ is $n \times n$ orthogonal matrix, and $\boldsymbol{\Sigma}$ is $m \times n$ diagonal matrix, with

$$\sigma_{ij} = \begin{cases} 0 & \text{for } i \neq j \\ \sigma_i \geq 0 & \text{for } i = j \end{cases}$$

- Diagonal entries σ_i, called *singular values* of $\boldsymbol{A}$, are usually ordered so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$

- Columns $\boldsymbol{u}_i$ of $\boldsymbol{U}$ and $\boldsymbol{v}_i$ of $\boldsymbol{V}$ 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.
- Σ is $m \times n$, diagonal, $\sigma_i > 0$.
- V is $n \times n$, orthogonal.

Example: SVD

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

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

In square matrix case, U Σ V^T closely related to eigenpair, X Λ X^{-1}

Applications of SVD

- *Minimum norm solution* to $\boldsymbol{Ax} \cong \boldsymbol{b}$ is given by

$$\boldsymbol{x} = \sum_{\sigma_i \neq 0} \frac{\boldsymbol{u}_i^T \boldsymbol{b}}{\sigma_i} \boldsymbol{v}_i$$

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

- *Euclidean matrix norm*: $\|\boldsymbol{A}\|_2 = \sigma_{\max}$

- *Euclidean condition number of matrix*: $\mathrm{cond}(\boldsymbol{A}) = \dfrac{\sigma_{\max}}{\sigma_{\min}}$

- *Rank of matrix*: number of nonzero singular values

SVD for Linear Least Squares Problem: $A = U\Sigma V^T$

$$
\begin{aligned}
A\underline{x} &\approx \underline{b} \\
U\Sigma V^T &\approx \underline{b} \\
U^T U\Sigma V^T &\approx U^T\underline{b} \\
\Sigma V^T &\approx U^T\underline{b} \\
\begin{bmatrix} \tilde{R} \\ O \end{bmatrix} \underline{x} &\approx \begin{pmatrix} \underline{c}_1 \\ \underline{c}_2 \end{pmatrix} \\
\tilde{R}\underline{x} &= \underline{c}_1
\end{aligned}
$$

$$
\underline{x} = \sum_{j=1}^{n} \underline{v}_j \frac{1}{\sigma_j} (\underline{c}_1)_j = \sum_{j=1}^{n} \underline{v}_j \frac{1}{\sigma_j} \underline{u}_j^T \underline{b}
$$

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.

$$\underline{x} = \sum_{j=1}^{k} \underline{v}_j \frac{1}{\sigma_j} \underline{u}_j^T \underline{b}$$

Pseudoinverse

- Define pseudoinverse of scalar σ 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 $\boldsymbol{A}$ is given by

$$\boldsymbol{A}^{+} = \boldsymbol{V}\boldsymbol{\Sigma}^{+}\boldsymbol{U}^{T}$$

- Pseudoinverse always exists whether or not matrix is square or has full rank
- If $\boldsymbol{A}$ is square and nonsingular, then $\boldsymbol{A}^{+} = \boldsymbol{A}^{-1}$
- In all cases, minimum-norm solution to $\boldsymbol{A}\boldsymbol{x} \cong \boldsymbol{b}$ is given by $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{b}$

Orthogonal Bases

- SVD of matrix, $\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$, provides orthogonal bases for subspaces relevant to $\boldsymbol{A}$
- Columns of $\boldsymbol{U}$ corresponding to nonzero singular values form orthonormal basis for span$(\boldsymbol{A})$
- Remaining columns of $\boldsymbol{U}$ form orthonormal basis for orthogonal complement span$(\boldsymbol{A})^\perp$
- Columns of $\boldsymbol{V}$ corresponding to zero singular values form orthonormal basis for null space of $\boldsymbol{A}$
- Remaining columns of $\boldsymbol{V}$ form orthonormal basis for orthogonal complement of null space of $\boldsymbol{A}$

Lower-Rank Matrix Approximation

- Another way to write SVD is

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^T = \sigma_1 \boldsymbol{E}_1 + \sigma_2 \boldsymbol{E}_2 + \cdots + \sigma_n \boldsymbol{E}_n$$

  with $\boldsymbol{E}_i = \boldsymbol{u}_i \boldsymbol{v}_i^T$
- $\boldsymbol{E}_i$ has rank 1 and can be stored using only $m+n$ storage locations
- Product $\boldsymbol{E}_i \boldsymbol{x}$ can be computed using only $m+n$ multiplications
- Condensed approximation to $\boldsymbol{A}$ 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 $\boldsymbol{A}$
- 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 Σ, we have

$$A = U\Sigma V^T = \sum_{j=1}^{n} \underline{u}_j \sigma_j \underline{v}_j^T$$

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

$$A \approx A_k := \sum_{j=1}^{k} \underline{u}_j \sigma_j \underline{v}_j^T$$

- 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 x n matrix, we can use the SVD to generate a low-rank compressed version.

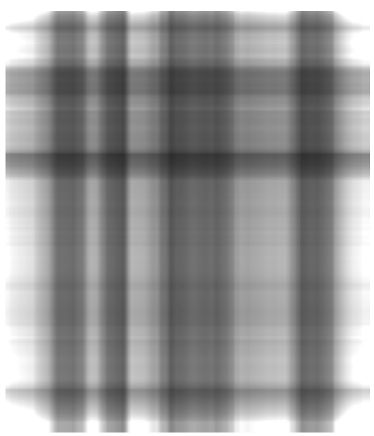
- Full image storage cost scales as O(mn)

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

$$A \approx A_k := \sum_{j=1}^{k} \underline{u}_j \sigma_j \underline{v}_j^T$$

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.



k=1

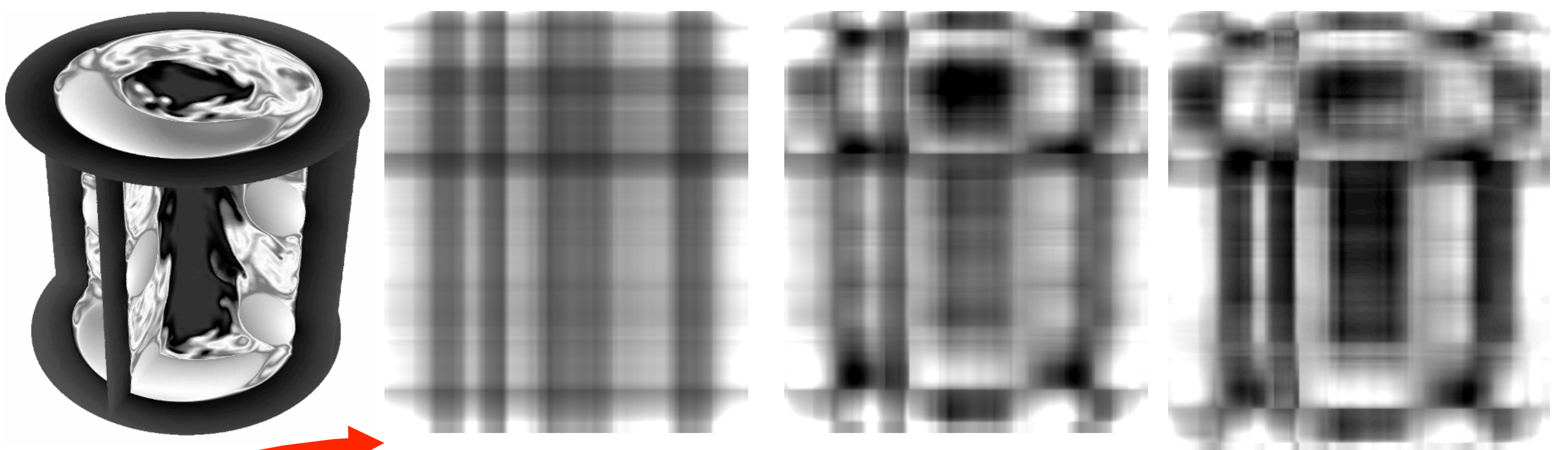
$$A \approx A_k := \sum_{j=1}^{k} \underline{u}_j \sigma_j \underline{v}_j^T$$

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.



k=1 k=2 k=3 (m=536,n=432)

Note: we don't store matrix – just vectors u_1 and v_1.

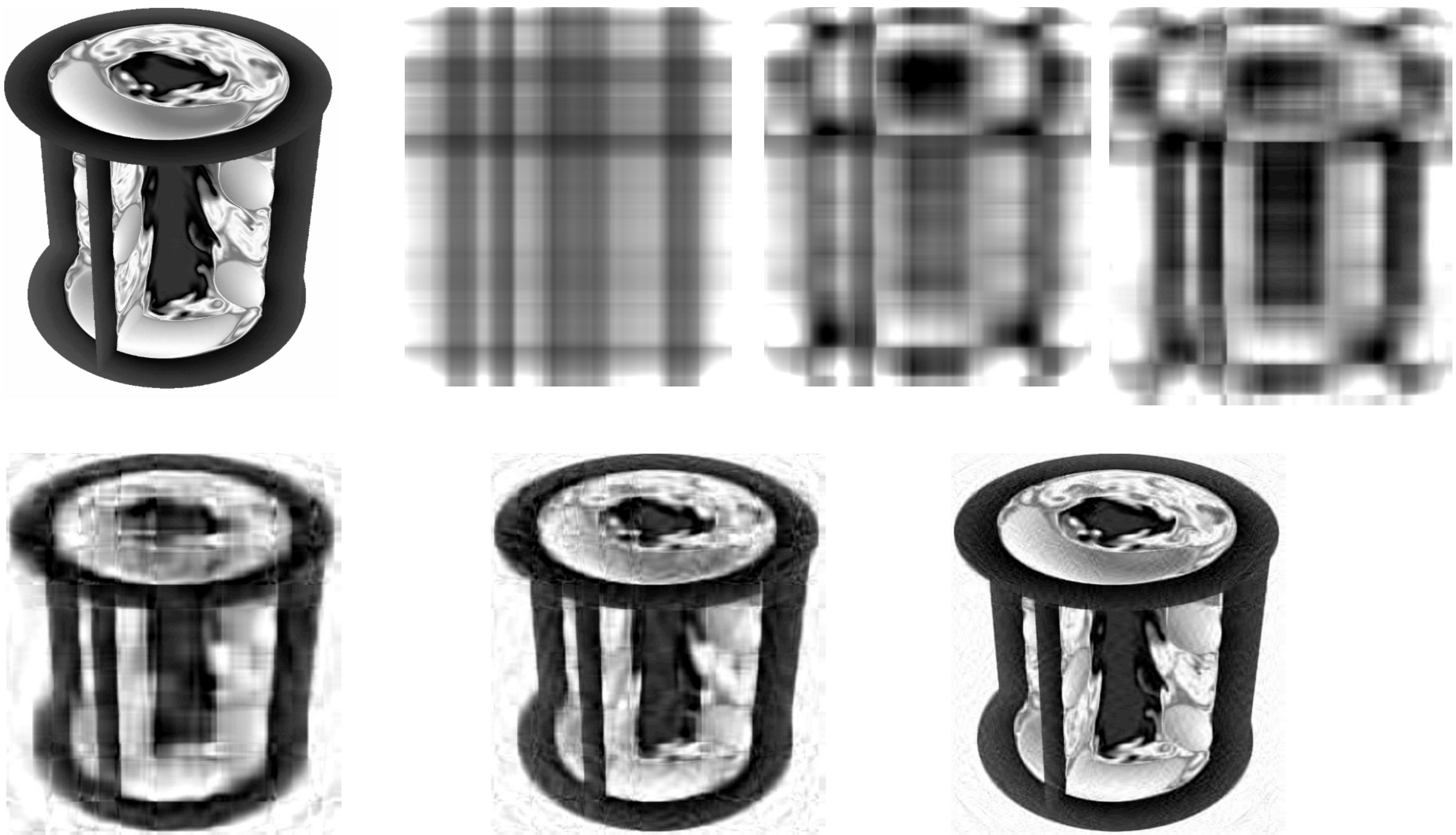
Matlab code

```
[X,A]=imread('collins_img.gif'); [m,n]=size(X);

Xo=X; imwrite(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); imwrite(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 A$\underline{x}$ = $\underline{b}$

$$\text{If } \sigma_1 \leq \sigma_2 \leq \cdots \leq \sigma_n,$$

$$\underline{x} \approx \textstyle\sum_{j=1}^{k} \sigma_j^{+} \underline{v}_j \underline{u}_j^T \underline{b}$$

- 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.

Eigenvalues, Projection, and Linear Systems: II

- Here, we tie Chapter 2 (linear systems) and 3 (projection) material in with the forthcoming chapter on eigenvalues.

- We start by reconsidering Jacobi iteration for the solution of $A\mathbf{x} = \mathbf{b}$:

$$\mathbf{x}_{k+1} \;=\; \mathbf{x}_k \;+\; D^{-1}(\mathbf{b} \;-\; A\mathbf{x}_k).$$

- For simplicity, assume $A_{ii} = 1$ and that A is SPD:

$$\begin{aligned}
\mathbf{x}_{k+1} \;&=\; \mathbf{x}_k \;+\; (\mathbf{b} \;-\; A\mathbf{x}_k) \\
&=\; \mathbf{x}_k \;+\; A(\mathbf{x} \;-\; \mathbf{x}_k) \\
&=\; \mathbf{x}_k \;+\; A\mathbf{e}_k.
\end{aligned}$$

- Subtract preceding expression from $\mathbf{x} \;=\; \mathbf{x}$:

$$\begin{aligned} \mathbf{e}_{k+1} &= \mathbf{e}_k - A\mathbf{e}_k \\ \mathbf{e}_k &= (I - A)^k \mathbf{e}_0. \qquad \textit{Error equation.} \end{aligned}$$

- Note that with $\mathbf{x}_0 = 0$, we have $\mathbf{e}_0 = \mathbf{x}$,$\mathbf{x}_1 = \mathbf{b}$, and

$$\mathbf{e}_k = (I - A)^k \mathbf{x}.$$

- We show that $\mathbf{x}_k$ is a polynomial of degree $k-1$ in A times $\mathbf{b}$:

$$
\begin{aligned}
\mathbf{x}_k &= \mathbf{x} - \mathbf{e}_k \\
&= \mathbf{x} - (I - A)^k \mathbf{x} \\
&= \mathbf{x} - (I - kA + \cdots + A^k)\mathbf{x} \\
&= \left(c_0 A + c_1 A^2 + \cdots + c_{k-1} A^k\right) \mathbf{x} \\
&= \left(c_0 I + c_1 A + \cdots + c_{k-1} A^{k-1}\right) A\mathbf{x} \\
&= \left(c_0 I + c_1 A + \cdots + c_{k-1} A^{k-1}\right) \mathbf{b} \\
&= P_{k-1}(A)\mathbf{b} \\
&\in \operatorname{span}(\mathbf{b},\, A\mathbf{b},\, \ldots,\, A^{k-1}\mathbf{b}) =: K_k(A;\mathbf{b}),
\end{aligned}
$$

  where $K_k(A;\mathbf{b})$ is the *Krylov subspace* associated with matrix A and vector $\mathbf{b}$.

- Look at the error behavior: $\mathbf{e}_k = (I - A)^k \mathbf{x}$.
- Assume A has an orthonormal set of eigenvectors spanning $\mathbb{R}^n$.
  - True if, say, A is symmetric.
  - Here, we'll further assume A is SPD such that $\lambda_i > 0$.
- Consider eigenvectors and eigenvalues $(\mathbf{s}_i, \lambda_i)$

$$A\mathbf{s}_i = \lambda_i \mathbf{s}_i \qquad \begin{cases} \text{Orthogonal:} & \mathbf{s}_i^T \mathbf{s}_j = 0, \; i \neq j \\ \text{Normalized:} & \mathbf{s}_i^T \mathbf{s}_i = 1. \end{cases}$$

- We have the matrix of eigenvectors $S = (\mathbf{s}_1 \, \mathbf{s}_2 \, \ldots \mathbf{s}_n)$ with $S^{-1} = S^T$
- Therefore S^{-1} exists.

Use of Eigenvector Decomposition

- For *any* $\mathbf{x} \in \mathbb{R}^n$, can find a decomposition of $\mathbf{x}$:

$$\mathbf{x} \;=\; \sum_{j=1}^{n} c_j \mathbf{s}_j.$$

- *Easy:*

$$\mathbf{s}_i^T \mathbf{x} \;=\; \sum_{j=1}^{n} c_j \mathbf{s}_i^T \mathbf{s}_j \;=\; c_i.$$

- In matrix form:

$$\mathbf{x} \;=\; S\mathbf{c}. \qquad\qquad \mathbf{c} \;=\; S^{-1}\mathbf{x} \;=\; S^T\mathbf{x}.$$

(Requires $S^T S = I$, which you, as a user, need to verify.)

- Returning to our error equation:

$$\begin{aligned}
\mathbf{e}_k &= (I - A)^k \mathbf{x} \\
&= \sum_{j=1}^{n} c_j (I - A)^k \mathbf{s}_j \\
&= \sum_{j=1}^{n} c_j (1 - \lambda_j)^k \mathbf{s}_j \\
&= \sum_{j=1}^{n} g_j \, c_j \, \mathbf{s}_j,
\end{aligned}$$

  where

$$g_j \;:=\; (1-\lambda_j)^k \;=\; g_k(\lambda_j) \;\in\; \mathbb{P}_k^1(\lambda_j).$$

- Here, we define $\mathbb{P}_k^1(\lambda)$ to be the space of polynomials of degree k in λ that take on the value 1 when λ=0.

- **Example:** 1D Poisson matrix:

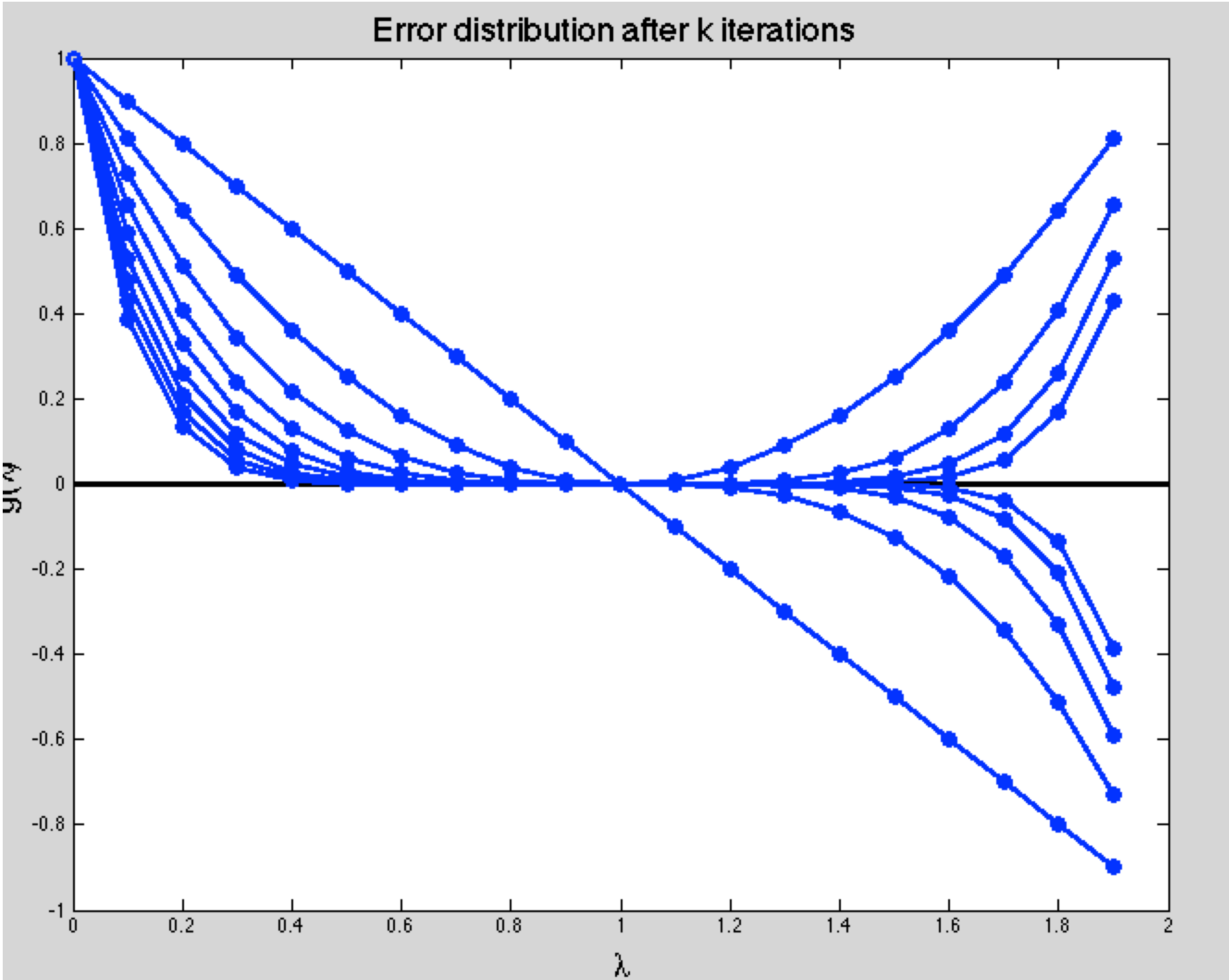
$$D^{-1}A\mathbf{u}\big|_i \;=\; \frac{h^2}{2}\left[\frac{1}{h^2}\left(-u_{i-1} \,+\, 2u_i \,-\, u_{i+1}\right)\right], \qquad\qquad h \;:=\; \frac{1}{n+1}.$$

- Eigenvalues:

$$\lambda_j \;=\; \frac{h^2}{2}\cdot\left[\frac{2}{h^2}\left(1 \,-\, \cos(\pi j h)\right)\right] \;\in\; \left(\frac{\pi^2 h^2}{2},\, 2-\frac{\pi h^2}{2}\right).$$

- Rate of contraction is

$$\rho \;=\; \max_j \;|g(\lambda_j)|\,.$$



Error distribution after k iterations
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
λ
g(λ)

- More generally, for well-chosen scaling matrix,

$$\rho \;=\; \frac{\kappa - 1}{\kappa + 1},$$

where κ is the condition number of the SPD matrix A:

$$\kappa \;=\; \frac{\lambda_{\max}}{\lambda_{\min}}.$$

Projection: Conjugate Gradient Iteration

- So far, we've established that Jacobi iteration gives a solution $\mathbf{x}_k \in \mathbb{P}_{k-1}(A)\mathbf{b} = K_k(A;\mathbf{b})$ with a rate of convergence (for this example) that scales like

$$\rho = \frac{\kappa - 1}{\kappa + 1},$$

  implying that the number of iterations is $O(\kappa)$.

- Conjugate gradients (CG) generates a solution $\mathbf{x}_k$ that is the *projection* of $\mathbf{x}$ (in the A-norm) onto the same approximation space, $K_k(A;\mathbf{b})$.

- It's not too difficult to show that

$$\frac{||\mathbf{e}_k||_A}{||\mathbf{x}||_A} \le 2\left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^k,$$

  which is decidedly faster than Jacobi iteration.

- CG is introduced in Chapter 6 for nonlinear optimization and in Chapter 11 for solving sparse linear systems.

- CG Algorithm:

$$
\begin{aligned}
\mathbf{p}_k &= \mathbf{r}_{k-1} - \sum_{j=1}^{k-1} \gamma_j \mathbf{p}_j, \qquad \text{(such that } \mathbf{p}_k \perp_A \mathbf{p}_j,\ j<k) \\
&= \mathbf{r}_{k-1} + \beta_k \mathbf{p}_{k-1}, \\
\mathbf{w}_k &= A\,\mathbf{p}_k \\
\mathbf{x}_k &= \mathbf{x}_{k-1} + \alpha \mathbf{p}_k, \qquad \alpha = \mathbf{r}_{k-1}^T \mathbf{r}_{k-1} / \mathbf{p}_k^T \mathbf{w}_k \\
\mathbf{r}_k &= \mathbf{r}_{k-1} - \alpha \mathbf{w}_k.
\end{aligned}
$$

- Error is bounded by maximum of *any* polynomial in $\mathbb{P}_k^1(\lambda)$, $\lambda \in [\lambda_1, \lambda_n]$